

# Introduction à ASP.NET MVC

## par l'exemple

serge.tahe at univ-angers.fr, novembre 2013  
rév. février 2017

Hébergé par [developpez.com](http://developpez.com) :



Droits d'usage :



# Table des matières

<b>1 LE COURS.....</b>	<b>9</b>
<b>  1.1 INTRODUCTION.....</b>	<b>9</b>
<b>1.1.1 LA PLACE DE ASP.NET MVC DANS UNE APPLICATION WEB.....</b>	<b>9</b>
<b>1.1.2 LE MODÈLE DE DÉVELOPPEMENT DE ASP.NET MVC.....</b>	<b>10</b>
<b>1.1.3 LES OUTILS UTILISÉS.....</b>	<b>11</b>
<b>1.1.4 LES EXEMPLES.....</b>	<b>13</b>
<b>  1.2 LES BASES DE LA PROGRAMMATION WEB.....</b>	<b>14</b>
<b>1.2.1 LES ÉCHANGES DE DONNÉES DANS UNE APPLICATION WEB AVEC FORMULAIRE.....</b>	<b>15</b>
<b>1.2.2 PAGES WEB STATIQUES, PAGES WEB DYNAMIQUES.....</b>	<b>16</b>
<b>1.2.2.1 Page statique HTML (HyperText Markup Language).....</b>	<b>16</b>
<b>1.2.2.2 Une page ASP.NET.....</b>	<b>18</b>
<b>1.2.2.3 Conclusion.....</b>	<b>21</b>
<b>1.2.3 SCRIPTS CÔTÉ NAVIGATEUR.....</b>	<b>21</b>
<b>1.2.4 LES ÉCHANGES CLIENT-SERVEUR.....</b>	<b>22</b>
<b>1.2.4.1 Le modèle OSI.....</b>	<b>23</b>
<b>1.2.4.2 Le modèle TCP/IP.....</b>	<b>24</b>
<b>1.2.4.3 Le protocole HTTP.....</b>	<b>26</b>
<b>1.2.4.4 Conclusion.....</b>	<b>30</b>
<b>1.2.5 LES BASES DU LANGAGE HTML.....</b>	<b>30</b>
<b>1.2.5.1 Un exemple.....</b>	<b>31</b>
<b>1.2.5.2 Un formulaire HTML.....</b>	<b>33</b>
<b>1.2.5.2.1 Le formulaire.....</b>	<b>35</b>
<b>1.2.5.2.2 Les champs de saisie texte.....</b>	<b>36</b>
<b>1.2.5.2.3 Les champs de saisie multilignes.....</b>	<b>36</b>
<b>1.2.5.2.4 Les boutons radio.....</b>	<b>37</b>
<b>1.2.5.2.5 Les cases à cocher.....</b>	<b>37</b>
<b>1.2.5.2.6 La liste déroulante (combo).....</b>	<b>37</b>
<b>1.2.5.2.7 Liste à sélection unique.....</b>	<b>38</b>
<b>1.2.5.2.8 Liste à sélection multiple.....</b>	<b>38</b>
<b>1.2.5.2.9 Bouton de type button.....</b>	<b>39</b>
<b>1.2.5.2.10 Bouton de type submit.....</b>	<b>39</b>
<b>1.2.5.2.11 Bouton de type reset.....</b>	<b>40</b>
<b>1.2.5.2.12 Champ caché.....</b>	<b>40</b>
<b>1.2.5.3 Envoi à un serveur Web par un client Web des valeurs d'un formulaire.....</b>	<b>40</b>
<b>1.2.5.3.1 Méthode GET.....</b>	<b>40</b>
<b>1.2.5.3.2 Méthode POST.....</b>	<b>43</b>
<b>1.2.6 CONCLUSION.....</b>	<b>44</b>
<b>  1.3 CONTRÔLEURS, ACTIONS, ROUTAGE.....</b>	<b>45</b>
<b>1.3.1 LA STRUCTURE D'UN PROJET ASP.NET MVC.....</b>	<b>45</b>
<b>1.3.2 LE ROUTAGE PAR DÉFAUT DES URL.....</b>	<b>48</b>
<b>1.3.3 CRÉATION D'UN CONTRÔLEUR ET D'UNE PREMIÈRE ACTION.....</b>	<b>50</b>
<b>1.3.4 ACTION AVEC UN RÉSULTAT DE TYPE [CONTENTRESULT] - 1.....</b>	<b>54</b>
<b>1.3.5 ACTION AVEC UN RÉSULTAT DE TYPE [CONTENTRESULT] - 2.....</b>	<b>55</b>
<b>1.3.6 ACTION AVEC UN RÉSULTAT DE TYPE [JSONRESULT].....</b>	<b>56</b>
<b>1.3.7 ACTION AVEC UN RÉSULTAT DE TYPE [STRING].....</b>	<b>57</b>
<b>1.3.8 ACTION AVEC UN RÉSULTAT DE TYPE [EMPTYRESULT].....</b>	<b>57</b>
<b>1.3.9 ACTION AVEC UN RÉSULTAT DE TYPE [REDIRECTRESULT] - 1.....</b>	<b>58</b>
<b>1.3.10 ACTION AVEC UN RÉSULTAT DE TYPE [REDIRECTRESULT] - 2.....</b>	<b>59</b>
<b>1.3.11 ACTION AVEC UN RÉSULTAT DE TYPE [REDIRECTToROUTERESULT].....</b>	<b>59</b>
<b>1.3.12 ACTION AVEC UN RÉSULTAT DE TYPE [VOID].....</b>	<b>60</b>
<b>1.3.13 UN SECOND CONTRÔLEUR.....</b>	<b>60</b>
<b>1.3.14 ACTION FILTRÉE PAR UN ATTRIBUT.....</b>	<b>62</b>
<b>1.3.15 RÉCUPÉRER LES ÉLÉMENTS D'UNE ROUTE.....</b>	<b>63</b>
<b>  1.4 LE MODÈLE D'UNE ACTION.....</b>	<b>65</b>
<b>1.4.1 INITIALISATION DES PARAMÈTRES DE L'ACTION.....</b>	<b>65</b>
<b>1.4.2 VÉRIFIER LA VALIDITÉ DES PARAMÈTRES DE L'ACTION.....</b>	<b>69</b>
<b>1.4.3 UNE ACTION À PLUSIEURS PARAMÈTRES.....</b>	<b>72</b>
<b>1.4.4 UTILISER UNE CLASSE COMME MODÈLE D'UNE ACTION.....</b>	<b>73</b>
<b>1.4.5 MODÈLE DE L'ACTION AVEC CONTRAINTES DE VALIDITÉ - 1.....</b>	<b>74</b>
<b>1.4.6 MODÈLE DE L'ACTION AVEC CONTRAINTES DE VALIDITÉ - 2.....</b>	<b>79</b>
<b>1.4.7 MODÈLE DE L'ACTION AVEC CONTRAINTES DE VALIDITÉ - 3.....</b>	<b>81</b>

<b>1.4.8</b>	MODÈLE D'ACTION DE TYPE TABLEAU OU LISTE.....	82
<b>1.4.9</b>	FILTRAGE D'UN MODÈLE D'ACTION.....	83
<b>1.4.10</b>	ÉTENDRE LE MODÈLE DE LIAISON DES DONNÉES.....	84
<b>1.4.11</b>	LIAISON TARDIVE DU MODÈLE DE L'ACTION.....	89
<b>1.4.12</b>	CONCLUSION.....	90
<b>1.5</b>	<b>LA VUE ET SON MODÈLE.....</b>	<b>92</b>
<b>1.5.1</b>	INTRODUCTION.....	92
<b>1.5.2</b>	UTILISER LE [VIEWBAG] POUR PASSER DES INFORMATIONS À LA VUE.....	95
<b>1.5.3</b>	UTILISER UN MODÈLE FORTEMENT TYPÉ POUR PASSER DES INFORMATIONS À LA VUE.....	96
<b>1.5.4</b>	[RAZOR] – PREMIERS PAS.....	102
<b>1.5.5</b>	FORMULAIRE – PREMIERS PAS.....	106
<b>1.5.6</b>	FORMULAIRE – UN EXEMPLE COMPLET.....	113
<b>1.5.6.1</b>	Le modèle de portée [Application].....	114
<b>1.5.6.2</b>	Le modèle de l'action [Action08Get].....	116
<b>1.5.6.3</b>	Le modèle de la vue [Formulaire].....	116
<b>1.5.6.4</b>	La vue [Formulaire].....	120
<b>1.5.6.5</b>	Traitement du POST du formulaire.....	122
<b>1.5.6.6</b>	Traitement des anomalies du POST.....	123
<b>1.5.7</b>	UTILISATION DE MÉTHODES SPÉCIALISÉES DANS LA GÉNÉRATION DE FORMULAIRE.....	125
<b>1.5.7.1</b>	Le nouveau formulaire.....	125
<b>1.5.7.2</b>	Les actions et le modèle.....	129
<b>1.5.8</b>	GÉNÉRATION D'UN FORMULAIRE À PARTIR DES MÉTADONNÉES DU MODÈLE.....	130
<b>1.5.8.1</b>	Le [POST] du formulaire.....	134
<b>1.5.8.2</b>	Propriété [Text].....	134
<b>1.5.8.3</b>	Propriété [MultiLineText].....	135
<b>1.5.8.4</b>	Propriété [Number].....	136
<b>1.5.8.5</b>	Propriété [Decimal].....	137
<b>1.5.8.6</b>	Propriété [Tel].....	138
<b>1.5.8.7</b>	Propriété [Date].....	138
<b>1.5.8.8</b>	Propriété [Time].....	139
<b>1.5.8.9</b>	Propriété [HiddenInput].....	140
<b>1.5.8.10</b>	Propriété [Boolean].....	141
<b>1.5.8.11</b>	Propriété [Email].....	142
<b>1.5.8.12</b>	Propriété [Url].....	142
<b>1.5.8.13</b>	Propriété [Password].....	143
<b>1.5.8.14</b>	Propriété [Currency].....	144
<b>1.5.8.15</b>	Propriété [CreditCard].....	145
<b>1.5.9</b>	VALIDATION D'UN FORMULAIRE.....	145
<b>1.5.9.1</b>	Validation côté serveur.....	145
<b>1.5.9.2</b>	Validation côté client.....	150
<b>1.5.10</b>	GESTION DES LIENS DE NAVIGATION ET D'ACTION.....	153
<b>1.6</b>	<b>INTERNATIONALISATION DES VUES.....</b>	<b>156</b>
<b>1.6.1</b>	LOCALISATION DES NOMBRES RÉELS.....	156
<b>1.6.2</b>	GÉRER UNE CULTURE.....	159
<b>1.6.3</b>	INTERNATIONALISER LE MODÈLE DE VUE [VIEWMODEL14].....	161
<b>1.6.4</b>	INTERNATIONALISER LA VUE [ACTION14GET.CSHMTL].....	167
<b>1.6.5</b>	EXEMPLES D'EXÉCUTION.....	173
<b>1.6.6</b>	INTERNATIONALISATION DES DATES.....	173
<b>1.6.7</b>	CONCLUSION.....	179
<b>1.7</b>	<b>AJAXIFICATION D'UNE APPLICATION ASP.NET MVC.....</b>	<b>180</b>
<b>1.7.1</b>	LA PLACE D'AJAX DANS UNE APPLICATION WEB.....	180
<b>1.7.2</b>	RUDIMENTS DE JQUERY ET DE JAVASCRIPT.....	181
<b>1.7.3</b>	MISE À JOUR D'UNE PAGE AVEC UN FLUX HTML.....	184
<b>1.7.3.1</b>	Les vues.....	184
<b>1.7.3.2</b>	Le contrôleur, les actions, le modèle, la vue.....	185
<b>1.7.3.3</b>	L'action [Action01Post].....	189
<b>1.7.3.4</b>	La vue [Action01Error].....	190
<b>1.7.3.5</b>	La vue [Action01Success].....	191
<b>1.7.3.6</b>	Gestion de la session.....	191
<b>1.7.3.7</b>	Gestion de l'image d'attente.....	192
<b>1.7.3.8</b>	Gestion du lien [Calculer].....	192
<b>1.7.4</b>	MISE À JOUR D'UNE PAGE HTML AVEC UN FLUX JSON.....	194
<b>1.7.4.1</b>	L'action [Action02Get].....	194

<u><a href="#">1.7.4.2</a></u> L'action [Action02Post].....	195
<u><a href="#">1.7.4.3</a></u> Le code Javascript côté client.....	196
<u><a href="#">1.7.4.4</a></u> Le lien [Calculer].....	198
<u><a href="#">1.7.5</a></u> APPLICATION WEB À PAGE UNIQUE.....	200
<u><a href="#">1.7.6</a></u> APPLICATION WEB À PAGE UNIQUE ET VALIDATION CÔTÉ CLIENT.....	204
<u><a href="#">1.7.6.1</a></u> Les vues de l'exemple.....	204
<u><a href="#">1.7.6.2</a></u> Le modèle des vues.....	209
<u><a href="#">1.7.6.3</a></u> Les données de portée [Session].....	209
<u><a href="#">1.7.6.4</a></u> L'action serveur [Action05Get].....	211
<u><a href="#">1.7.6.5</a></u> L'action client [Calculer].....	211
<u><a href="#">1.7.6.6</a></u> L'action client [Effacer].....	215
<u><a href="#">1.7.6.7</a></u> L'action client [Retour aux Saisies].....	216
<u><a href="#">1.7.7</a></u> RENDRE ACCESIBLE SUR INTERNET UNE APPLICATION ASP.NET.....	218
<u><a href="#">1.7.8</a></u> GÉNÉRATION D'UNE APPLICATION NATIVE POUR ANDROID À PARTIR D'UNE APPLICATION À PAGE UNIQUE APU.....	218
<b><u><a href="#">1.8 CONCLUSION INTERMÉDIAIRE</a></u></b> .....	<b>219</b>
<b><u><a href="#">2 ETUDE DE CAS N° 1 : GESTION BASIQUE DE SALAIRES.</a></u></b> .....	<b>220</b>
<u><a href="#">2.1 INTRODUCTION</a></u> .....	220
<u><a href="#">2.2 LE PROBLÈME À RÉSOUTRE</a></u> .....	220
<u><a href="#">2.3 ARCHITECTURE DE L'APPLICATION</a></u> .....	222
<u><a href="#">2.4 LA BASE DE DONNÉES</a></u> .....	223
<u><a href="#">2.5 MODE DE CALCUL DU SALAIRE D'UNE ASSISTANTE MATERNELLE</a></u> .....	225
<u><a href="#">2.6 LE PROJET VISUAL STUDIO DE LA COUCHE [WEB]</a></u> .....	226
<b><u><a href="#">2.7 ÉTAPE 1 – MISE EN PLACE DE COUCHE [MÉTIER] SIMULÉE</a></u></b> .....	<b>227</b>
<u><a href="#">2.7.1</a></u> LA SOLUTION VISUAL STUDIO DE L'APPLICATION COMPLÈTE.....	227
<u><a href="#">2.7.2</a></u> L'INTERFACE DE LA COUCHE [MÉTIER].....	228
<u><a href="#">2.7.3</a></u> LES ENTITÉS DE LA COUCHE [MÉTIER].....	230
<u><a href="#">2.7.4</a></u> LA CLASSE [PAMEXCEPTION].....	233
<u><a href="#">2.7.5</a></u> IMPLÉMENTATION DE LA COUCHE [MÉTIER].....	233
<u><a href="#">2.7.6</a></u> LE TEST CONSOLE DE LA COUCHE [MÉTIER].....	235
<b><u><a href="#">2.8 ÉTAPE 2 : MISE EN PLACE DE L'APPLICATION WEB</a></u></b> .....	<b>237</b>
<b><u><a href="#">2.9 ÉTAPE 3 : MISE EN PLACE DU MODÈLE APU</a></u></b> .....	<b>242</b>
<u><a href="#">2.9.1</a></u> LES OUTILS DU DÉVELOPPEUR JAVASCRIPT.....	243
<u><a href="#">2.9.2</a></u> UTILISATION D'UNE VUE PARTIELLE POUR AFFICHER LE FORMULAIRE.....	244
<u><a href="#">2.9.3</a></u> L'APPEL AJAX [FAIRESIMULATION].....	244
<u><a href="#">2.9.4</a></u> L'APPEL AJAX [ENREGISTRERSIMULATION].....	246
<u><a href="#">2.9.5</a></u> L'APPEL AJAX [VOIRSIMULATIONS].....	247
<u><a href="#">2.9.6</a></u> L'APPEL AJAX [RETOURFORMULAIRE].....	248
<u><a href="#">2.9.7</a></u> L'APPEL AJAX [TERMINERSESSION].....	248
<u><a href="#">2.9.8</a></u> LA FONCTION JS [EFFACERSIMULATION].....	248
<u><a href="#">2.9.9</a></u> GESTION DE LA NAVIGATION ENTRE ÉCRANS.....	249
<b><u><a href="#">2.10 ÉTAPE 4 : ÉCRITURE DE L'ACTION SERVEUR [INDEX]</a></u></b> .....	<b>252</b>
<u><a href="#">2.10.1</a></u> LE MODÈLE DU FORMULAIRE.....	253
<u><a href="#">2.10.2</a></u> LE MODÈLE DE L'APPLICATION.....	254
<u><a href="#">2.10.3</a></u> LE CODE DE L'ACTION [INDEX].....	256
<u><a href="#">2.10.4</a></u> LE MODÈLE DE LA VUE [INDEX.CSHTML].....	256
<u><a href="#">2.10.5</a></u> LES VUES [INDEX.CSHTML] ET [FORMULAIRE.CSHTML].....	258
<u><a href="#">2.10.6</a></u> TEST DE L'ACTION [INDEX].....	258
<b><u><a href="#">2.11 ÉTAPE 5 : MISE EN PLACE DE LA VALIDATION DES SAISIES</a></u></b> .....	<b>259</b>
<u><a href="#">2.11.1</a></u> LE PROBLÈME.....	259
<u><a href="#">2.11.2</a></u> SAISIE DES NOMBRES RÉELS AU FORMAT FRANÇAIS.....	262
<u><a href="#">2.11.3</a></u> VALIDATION DU FORMULAIRE PAR LE LIEN JAVASCRIPT [FAIRE LA SIMULATION].....	263
<b><u><a href="#">2.12 ÉTAPE 6 : FAIRE UNE SIMULATION</a></u></b> .....	<b>264</b>
<u><a href="#">2.12.1</a></u> LE PROBLÈME.....	264
<u><a href="#">2.12.2</a></u> ÉCRITURE DE LA VUE [SIMULATION.CSHTML].....	264
<u><a href="#">2.12.3</a></u> CALCUL DU SALAIRE RÉEL.....	267
<u><a href="#">2.12.4</a></u> GESTION DES ERREURS.....	269
<b><u><a href="#">2.13 ÉTAPE 7 : MISE EN PLACE D'UNE SESSION UTILISATEUR</a></u></b> .....	<b>274</b>
<b><u><a href="#">2.14 ÉTAPE 8 : ENREGISTRER UNE SIMULATION</a></u></b> .....	<b>275</b>
<u><a href="#">2.14.1</a></u> LE PROBLÈME.....	275
<u><a href="#">2.14.2</a></u> ÉCRITURE DE L'ACTION SERVEUR [ENREGISTRERSIMULATION].....	277
<u><a href="#">2.14.3</a></u> ÉCRITURE DE LA VUE PARTIELLE [SIMULATIONS.CSHTML].....	277
<b><u><a href="#">2.15 ÉTAPE 9 : RETOURNER AU FORMULAIRE DE SAISIE</a></u></b> .....	<b>278</b>
<u><a href="#">2.15.1</a></u> LE PROBLÈME.....	278

<b>2.15.2</b> ÉCRITURE DE L'ACTION SERVEUR [FORMULAIRE].....	279
<b>2.15.3</b> MODIFICATION DE LA FONCTION JAVASCRIPT [RETOURFORMULAIRE].....	279
<b>2.16</b> ÉTAPE 10 : VOIR LA LISTE DES SIMULATIONS.....	<b>280</b>
<b>2.16.1</b> LE PROBLÈME.....	280
<b>2.16.2</b> ÉCRITURE DE L'ACTION SERVEUR [VOIRSIMULATIONS].....	280
<b>2.17</b> ÉTAPE 11 : TERMINER LA SESSION.....	<b>281</b>
<b>2.17.1</b> LE PROBLÈME.....	281
<b>2.17.2</b> ÉCRITURE DE L'ACTION SERVEUR [TERMINERSESSION].....	281
<b>2.17.3</b> MODIFICATION DE LA FONCTION JAVASCRIPT [TERMINERSESSION].....	282
<b>2.18</b> ÉTAPE 12 : EFFACER LA SIMULATION.....	<b>282</b>
<b>2.18.1</b> LE PROBLÈME.....	282
<b>2.18.2</b> ÉCRITURE DE L'ACTION CLIENT [EFFACERSIMULATION].....	282
<b>2.19</b> ÉTAPE 13 : RETIRER UNE SIMULATION.....	<b>282</b>
<b>2.19.1</b> LE PROBLÈME.....	282
<b>2.19.2</b> ÉCRITURE DE L'ACTION CLIENT [RETRIERSIMULATION].....	283
<b>2.19.3</b> ÉCRITURE DE L'ACTION SERVEUR [RETRIERSIMULATION].....	283
<b>2.20</b> ÉTAPE 14 : AMÉLIORATION DE LA MÉTHODE D'INITIALISATION DE L'APPLICATION.....	<b>284</b>
<b>2.20.1</b> AJOUT DES RÉFÉRENCES [SPRING] AU PROJET WEB.....	285
<b>2.20.2</b> CONFIGURATION DE [WEB.CONFIG].....	286
<b>2.20.3</b> MODIFICATION DE [APPLICATION_START].....	287
<b>2.20.4</b> GÉRER UNE ERREUR D'INITIALISATION DE L'APPLICATION.....	287
<b>2.21</b> OÙ EN SOMMES – NOUS ?.....	<b>290</b>
<b>2.22</b> ÉTAPE 15 : MISE EN PLACE DE LA COUCHE ENTITY FRAMEWORK 5.....	<b>291</b>
<b>2.22.1</b> LA BASE DE DONNÉES.....	291
<b>2.22.2</b> LE PROJET VISUAL STUDIO.....	295
<b>2.22.3</b> AJOUT DES RÉFÉRENCES NÉCESSAIRES AU PROJET.....	296
<b>2.22.4</b> LES ENTITÉS ENTITY FRAMEWORK.....	297
<b>2.22.5</b> CONFIGURATION DE L'ORM EF5.....	300
<b>2.22.6</b> TEST DE LA COUCHE [EF5].....	302
<b>2.22.7</b> DLL DE LA COUCHE [EF5].....	304
<b>2.23</b> ÉTAPE 16 : MISE EN PLACE DE LA COUCHE [DAO].....	<b>304</b>
<b>2.23.1</b> L'INTERFACE DE LA COUCHE [DAO].....	304
<b>2.23.2</b> LE PROJET VISUAL STUDIO.....	305
<b>2.23.3</b> AJOUT DES RÉFÉRENCES NÉCESSAIRES AU PROJET.....	305
<b>2.23.4</b> IMPLÉMENTATION DE LA COUCHE [DAO].....	306
<b>2.23.5</b> CONFIGURATION DE LA COUCHE [DAO].....	308
<b>2.23.6</b> TEST DE LA COUCHE [DAO].....	308
<b>2.23.7</b> DLL DE LA COUCHE [DAO].....	309
<b>2.24</b> ÉTAPE 17 : MISE EN PLACE DE LA COUCHE [MÉTIER].....	<b>309</b>
<b>2.24.1</b> L'INTERFACE DE LA COUCHE [MÉTIER].....	309
<b>2.24.2</b> LE PROJET VISUAL STUDIO.....	309
<b>2.24.3</b> AJOUT DES RÉFÉRENCES NÉCESSAIRES AU PROJET.....	310
<b>2.24.4</b> IMPLÉMENTATION DE LA COUCHE [MÉTIER].....	310
<b>2.24.5</b> CONFIGURATION DE LA COUCHE [MÉTIER].....	311
<b>2.24.6</b> TEST DE LA COUCHE [MÉTIER].....	312
<b>2.24.7</b> DLL DE LA COUCHE [MÉTIER].....	313
<b>2.25</b> ÉTAPE 18 : MISE EN PLACE DE LA COUCHE [WEB].....	<b>313</b>
<b>2.25.1</b> LE PROJET VISUAL STUDIO.....	314
<b>2.25.2</b> AJOUT DES RÉFÉRENCES NÉCESSAIRES AU PROJET.....	315
<b>2.25.3</b> IMPLÉMENTATION DE LA COUCHE [WEB].....	315
<b>2.25.4</b> CONFIGURATION DE LA COUCHE [WEB].....	316
<b>2.25.5</b> TEST DE LA COUCHE [WEB].....	316
<b>2.26</b> ÉTAPE 19 : RENDRE ACCESSIBLE SUR INTERNET UNE APPLICATION ASP.NET.....	<b>321</b>
<b>2.27</b> ÉTAPE 20 : GÉNÉRATION D'UNE APPLICATION NATIVE POUR ANDROID.....	<b>332</b>
<b>2.27.1</b> L'ARCHITECTURE DE L'APPLICATION.....	332
<b>2.27.2</b> REFACTORISATION DU PROJET [PAM-WEB-02].....	334
<b>2.27.3</b> TEST DU PROJET REFACTORISÉ.....	337
<b>2.27.4</b> CRÉATION DU BINAIRE ANDROID.....	339
<b>2.28</b> ANNEXES.....	<b>343</b>
<b>2.28.1</b> INSTALLATION DU GESTIONNAIRE D'ÉMULATEURS GENYMOTION.....	343
<b>2.29</b> CONCLUSION.....	<b>345</b>
<b>3</b> ÉTUDE DE CAS N° 2 : GESTION DE BILANS DE FORMATION.....	<b>346</b>
<b>3.1</b> LES COMPÉTENCES MISES EN OEUVRE.....	346

<b>3.2 LE PROBLÈME.....</b>	<b>346</b>
<b>3.3 L'ARCHITECTURE DE L'APPLICATION.....</b>	<b>347</b>
<b>3.4 LA BASE DE DONNÉES.....</b>	<b>347</b>
<b>3.5 LES COUCHES [DAO, ENTITY FRAMEWORK].....</b>	<b>351</b>
<b>3.5.1 INSTALLATION ET TESTS.....</b>	<b>351</b>
<b>3.5.2 L'INTERFACE DE LA COUCHE [DAO].....</b>	<b>352</b>
<b>3.5.3 VERSIONS COURTES ET LONGUES DES ENTITÉS.....</b>	<b>357</b>
<b>3.6 LA COUCHE [WEB / ASP.NET MVC].....</b>	<b>359</b>
<b>3.6.1 L'IMPLÉMENTATION DE DÉMARRAGE.....</b>	<b>359</b>
<b>3.6.2 CONFIGURATION DE L'APPLICATION.....</b>	<b>361</b>
<b>3.6.3 INITIALISATION DE L'APPLICATION.....</b>	<b>362</b>
<b>3.6.4 LE MODÈLE DE PORTÉE [APPLICATION].....</b>	<b>363</b>
<b>3.6.5 LE ROUTAGE DES URL.....</b>	<b>366</b>
<b>3.6.6 LA PAGE [TEST.CSHTML] ET SON MODÈLE [TESTMODEL].....</b>	<b>366</b>
<b>3.6.7 LE PATRON DES VUES.....</b>	<b>367</b>
<b>3.6.8 L'ACTION [TEST].....</b>	<b>369</b>
<b>3.6.9 LE BEAN [SESSIONMODEL].....</b>	<b>370</b>
<b>3.7 IMPLÉMENTATION DE L'APPLICATION.....</b>	<b>370</b>
<b>3.7.1 ÉTAPE 1.....</b>	<b>370</b>
<b>3.7.2 ÉTAPE 2.....</b>	<b>376</b>
<b>3.7.3 ÉTAPE 3.....</b>	<b>377</b>
<b>3.7.4 ÉTAPE 4.....</b>	<b>377</b>
<b>3.7.5 ÉTAPE 5.....</b>	<b>377</b>
<b>3.7.6 ÉTAPE 6.....</b>	<b>378</b>
<b>3.7.7 ÉTAPE 7.....</b>	<b>379</b>
<b>3.7.8 ÉTAPE 8.....</b>	<b>381</b>
<b>3.7.9 ÉTAPE 9.....</b>	<b>382</b>
<b>3.7.10 ÉTAPE 10.....</b>	<b>384</b>
<b>3.7.11 ÉTAPE 11.....</b>	<b>385</b>
<b>3.7.12 ÉTAPE 12.....</b>	<b>385</b>
<b>3.7.13 ÉTAPE 13.....</b>	<b>386</b>
<b>3.7.14 ÉTAPE 14.....</b>	<b>387</b>
<b>4 ÉTUDE DE CAS N° 3 : APPLICATION E-COMMERCE.....</b>	<b>388</b>
<b>4.1 LES COMPÉTENCES MISES EN OEUVRE.....</b>	<b>388</b>
<b>4.2 LE PROBLÈME.....</b>	<b>388</b>
<b>4.3 PRÉSENTATION DES COUCHES BASSES.....</b>	<b>389</b>
<b>4.3.1 MISE EN PLACE DE L'ENVIRONNEMENT DE TRAVAIL.....</b>	<b>389</b>
<b>4.3.2 LA BASE DE DONNÉES.....</b>	<b>390</b>
<b>4.3.3 LES ENTITÉS MANIPULÉES PAR LES COUCHES BASSES.....</b>	<b>394</b>
<b>4.3.4 L'INTERFACE [IMETIER] DES COUCHES BASSES.....</b>	<b>400</b>
<b>4.3.5 LA CONFIGURATION DES COUCHES BASSES.....</b>	<b>401</b>
<b>4.3.6 VÉRIFICATION DU FONCTIONNEMENT DES COUCHES BASSES.....</b>	<b>402</b>
<b>4.4 ECRITURE DE LA COUCHE WEB / MVC.....</b>	<b>405</b>
<b>4.4.1 LES VUES DE L'APPLICATION.....</b>	<b>406</b>
<b>4.4.2 LE PROJET VISUAL STUDIO.....</b>	<b>408</b>
<b>4.4.3 CONFIGURATION.....</b>	<b>409</b>
<b>4.4.4 LA CLASSE [APPLICATIONMODEL].....</b>	<b>410</b>
<b>4.4.5 LA CLASSE [SESSIONMODEL].....</b>	<b>412</b>
<b>4.4.6 LES CLASSES [PARENTVIEWMODEL] ET [PARENTMODELBINDER].....</b>	<b>412</b>
<b>4.4.7 INITIALISATION DE L'APPLICATION.....</b>	<b>413</b>
<b>4.4.8 ROUTAGE DE L'APPLICATION.....</b>	<b>414</b>
<b>4.4.9 LE MODÈLE DES VUES.....</b>	<b>414</b>
<b>4.4.10 LA PAGE D'ACCUEIL [CATEGORIES.CSHTML].....</b>	<b>417</b>
<b>4.4.11 LA PAGE DES PRODUITS VENDUS [PRODUCTS.CSHTML].....</b>	<b>420</b>
<b>4.4.12 LA MÉTHODE [ADDTOCART] DU CONTRÔLEUR.....</b>	<b>422</b>
<b>4.4.13 LA PAGE DU PANIER [CART.CSHTML].....</b>	<b>425</b>
<b>4.4.14 LE LIEN [CONTINUER LES ACHATS].....</b>	<b>427</b>
<b>4.4.15 LA MÉTHODE [UPDATEQUANTITY] DU CONTRÔLEUR.....</b>	<b>428</b>
<b>4.4.16 LA PAGE DE PAIEMENT [CHECKOUT.CSHTML].....</b>	<b>430</b>
<b>4.4.17 LA MÉTHODE [CHECKOUT] DU CONTRÔLEUR.....</b>	<b>432</b>
<b>4.4.18 LA PAGE DE CONFIRMATION [CONFIRMATION.CSHTML].....</b>	<b>433</b>
<b>4.4.19 LE LIEN [VIDER LE CHARIOT].....</b>	<b>435</b>
<b>5 ÉTUDE DE CAS N° 4 : GESTION DE RENDEZ-VOUS.....</b>	<b>436</b>

<b>5.1 INTRODUCTION.....</b>	436
<b>5.1.1 LE PROJET EXISTANT.....</b>	436
<b>5.1.2 MÉTHODES DE TRAVAIL.....</b>	438
<b>5.2 LES VUES DE L'APPLICATION WEB.....</b>	438
<b>5.3 TRAVAIL À FAIRE.....</b>	441
<b>6 ÉTUDE DE CAS N° 5 : GESTION BASIQUE DE NOTES D'ÉLÈVES.....</b>	442
<b>6.1 LES COMPÉTENCES MISES EN OEUVRE.....</b>	442
<b>6.2 LE PROBLÈME.....</b>	442
<b>6.3 L'ARCHITECTURE DE L'APPLICATION WEB.....</b>	443
<b>6.4 LES PROJETS VISUAL STUDIO DE L'APPLICATION.....</b>	444
<b>6.5 LA BASE DE DONNÉES.....</b>	445
<b>6.6 LA COUCHE [MÉTIER] ET LES ENTITÉS MANIPULÉES PAR CELLE-CI.....</b>	448
<b>6.7 IMPLÉMENTATION DE LA COUCHE [WEB].....</b>	452
<b>6.7.1 LE PROJET VISUAL STUDIO.....</b>	452
<b>6.7.2 CONFIGURATION DU PROJET WEB.....</b>	453
<b>6.7.3 INITIALISATION DE L'APPLICATION.....</b>	454
<b>6.7.4 LE MODÈLE DE PORTÉE [APPLICATION].....</b>	456
<b>6.7.5 LE MODÈLE DE PORTÉE [SESSION].....</b>	456
<b>6.7.6 LE ROUTAGE DE L'APPLICATION.....</b>	457
<b>6.8 L'URL [/ECOLE/Exo00].....</b>	457
<b>6.9 L'URL [/ECOLE/Exo01].....</b>	459
<b>6.9.1 INTRODUCTION.....</b>	459
<b>6.9.2 GESTION DU LIEN [AFFICHER LES ÉLÈVES].....</b>	462
<b>6.10 L'URL [/ECOLE/Exo02].....</b>	464
<b>7 ÉTUDE DE CAS N° 6 : CLIENT / SERVEUR JSON.....</b>	466
<b>7.1 LES COMPÉTENCES MISES EN OEUVRE.....</b>	466
<b>7.2 LE PROBLÈME.....</b>	466
<b>7.3 LA BASE DE DONNÉES.....</b>	467
<b>7.4 MISE EN PLACE DE L'ENVIRONNEMENT DE TRAVAIL.....</b>	470
<b>7.5 ECRITURE DES COUCHES BASSES.....</b>	471
<b>7.5.1 LE PROJET VISUAL STUDIO.....</b>	472
<b>7.5.2 LA COUCHE [ENTITY FRAMEWORK].....</b>	473
<b>7.5.2.1 Le code.....</b>	473
<b>7.5.2.2 Le test.....</b>	476
<b>7.5.2.2.1 Le code.....</b>	476
<b>7.5.2.2.2 La configuration.....</b>	478
<b>7.5.2.2.3 Les résultats.....</b>	479
<b>7.5.3 La couche [DAO].....</b>	479
<b>7.5.3.1 Le code.....</b>	480
<b>7.5.3.2 Le test.....</b>	485
<b>7.5.3.2.1 Le code.....</b>	485
<b>7.5.3.2.2 La configuration.....</b>	486
<b>7.5.3.2.3 Les résultats.....</b>	486
<b>7.5.4 La couche [MÉTIER].....</b>	487
<b>7.5.4.1 Le code.....</b>	487
<b>7.5.4.2 Le test console.....</b>	488
<b>7.5.4.2.1 Le code.....</b>	488
<b>7.5.4.2.2 La configuration.....</b>	489
<b>7.5.4.2.3 Les résultats.....</b>	490
<b>7.5.4.3 Le test NUnit.....</b>	490
<b>7.5.4.3.1 Le code.....</b>	490
<b>7.5.4.3.2 La configuration.....</b>	491
<b>7.5.5 CRÉATION DES DLL DES COUCHES BASSES.....</b>	493
<b>7.6 ECRITURE DE LA COUCHE WEB / JSON.....</b>	494
<b>7.6.1 ARCHITECTURE.....</b>	494
<b>7.6.2 LE PROJET VISUAL STUDIO.....</b>	495
<b>7.6.3 CONFIGURATION.....</b>	496
<b>7.6.4 LA CLASSE [APPLICATIONMODEL].....</b>	496
<b>7.6.5 INITIALISATION DE L'APPLICATION.....</b>	497
<b>7.6.6 LES URL EXPOSÉES PAR LE SERVICE WEB / JSON.....</b>	498
<b>7.6.7 LA CLASSE [APPLICATIONMODELBINDER].....</b>	499
<b>7.6.8 LA CLASSE [RESPONSE].....</b>	500
<b>7.6.9 LES MÉTHODES DU CONTRÔLEUR.....</b>	501

<u>7.6.9.1</u> L'URL [Afb/GetCategories].....	501
<u>7.6.9.2</u> Les autres URL.....	501
<b>7.7 ÉTUDE DU CLIENT WEB / JSON.....</b>	<b>503</b>
<u>7.7.1</u> ARCHITECTURE.....	503
<u>7.7.2</u> LE PROJET VISUAL STUDIO.....	503
<u>7.7.3</u> LES ENTITÉS.....	504
<u>7.7.4</u> IMPLÉMENTATION DE LA COUCHE [DAO].....	505
<u>7.7.5</u> TESTS DE LA COUCHE [DAO].....	506
<u>7.7.6</u> IMPLÉMENTATION DE LA COUCHE [MÉTIER].....	507
<u>7.7.7</u> TESTS DE LA COUCHE [MÉTIER].....	509
<b>7.8 CONCLUSION.....</b>	<b>511</b>

# 1 Le cours

## 1.1 Introduction

Nous nous proposons ici d'introduire à l'aide d'exemples les notions importantes de ASP.NET MVC, un framework Web .NET qui fournit un cadre pour développer des applications Web selon le modèle MVC (Modèle – Vue – Contrôleur).

La compréhension des exemples de ce document nécessite peu de pré-requis. Seules des connaissances de base du langage C# sont nécessaires. On pourra les trouver dans le document **Introduction au langage C#** à l'URL [<http://tahe.developpez.com/dotnet/csharp>]. Une étude de cas est proposée pour mettre en pratique les connaissances acquises sur ASP.NET MVC. Elle utilise l'ORM (Object Relational Mapper) Entity Framework. Cet ORM est présenté dans le document **Introduction à Entity Framework 5 Code First** disponible à l'URL [<http://tahe.developpez.com/dotnet/ef5cf>].

Les divers exemples de ce document sont disponibles à l'URL [<http://tahe.developpez.com/dotnet/aspnetmvc>] sous la forme d'un fichier *.zip* à télécharger.

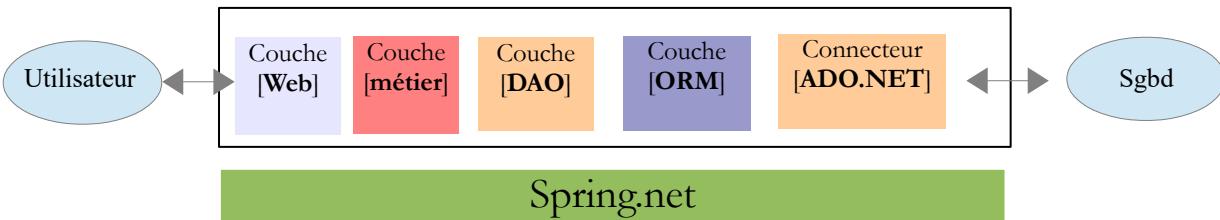
Ce document est incomplet à bien des égards. Pour approfondir ASP.NET MVC, on pourra utiliser les références suivantes :

- [ref1] : le livre "**Pro ASP.NET MVC 4**" écrit par **Adam Freeman** aux éditions **Apress**. C'est un excellent livre. Il serait disponible en ligne gratuitement et en français que le présent document n'aurait pas lieu d'être. Je recommande à tous ceux qui le peuvent de s'initier à ASP.NET MVC à l'aide de ce livre. Les codes source des exemples du livre sont disponibles gratuitement à l'URL [<http://www.apress.com/9781430242369>] ;
- [ref2] : le site du framework [<http://www.asp.net/mvc>]. On y trouvera tout le matériel nécessaire pour une auto-formation. Une version française est disponible à l'URL [<http://dotnet.developpez.com/mvc/>] ;
- le site de [developpez.com] consacré à ASP.NET [<http://dotnet.developpez.com/aspnet/>] .

Le document a été écrit de telle façon qu'il puisse être lu sans ordinateur sous la main. Aussi, donne-t-on beaucoup de copies d'écran.

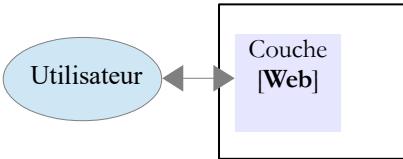
### 1.1.1 La place de ASP.NET MVC dans une application Web

Tout d'abord, situons ASP.NET MVC dans le développement d'une application Web. Le plus souvent, celle-ci sera bâtie sur une architecture multicouche telle que la suivante :

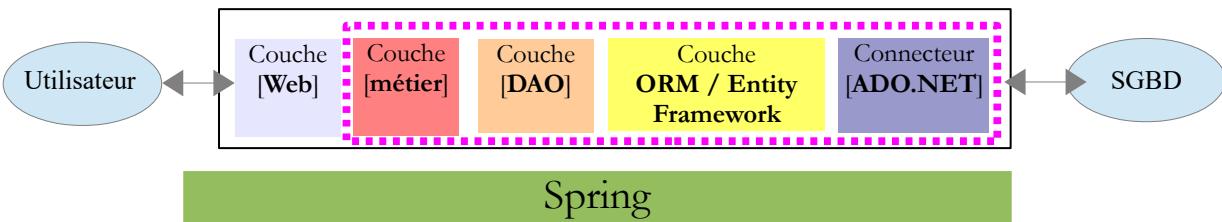


- la couche [Web] est la couche en contact avec l'utilisateur de l'application Web. Celui-ci interagit avec l'application Web au travers de pages Web visualisées par un navigateur. **C'est dans cette couche que se situe ASP.NET MVC et uniquement dans cette couche** ;
- la couche [métier] implémente les règles de gestion de l'application, tels que le calcul d'un salaire ou d'une facture. Cette couche utilise des données provenant de l'utilisateur via la couche [Web] et du SGBD via la couche [DAO] ;
- la couche [DAO] (Data Access Objects), la couche [ORM] (Object Relational Mapper) et le connecteur ADO.NET gèrent l'accès aux données du SGBD. La couche [ORM] fait un pont entre les objets manipulés par la couche [DAO] et les lignes et les colonnes des tables d'une base de données relationnelle. Deux ORM sont couramment utilisés dans le monde .NET, **NHibernate** (<http://sourceforge.net/projects/nhibernate/>) et **Entity Framework** (<http://msdn.microsoft.com/en-us/data/ef.aspx>) ;
- l'intégration des couches peut être réalisée par un conteneur d'injection de dépendances (Dependency Injection Container) tel que Spring (<http://www.springframework.net/>) ;

La plupart des exemples donnés dans la suite, n'utiliseront qu'une seule couche, la couche [Web] :

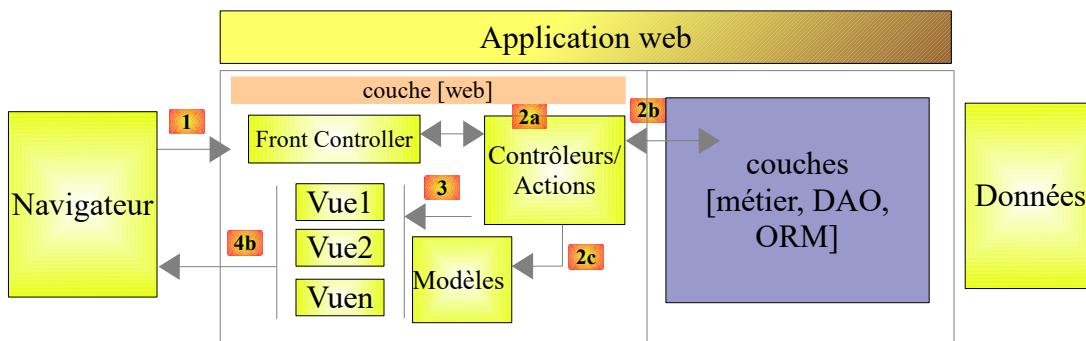


Ce document se terminera cependant par la construction d'une application Web multicouche :



### 1.1.2 Le modèle de développement de ASP.NET MVC

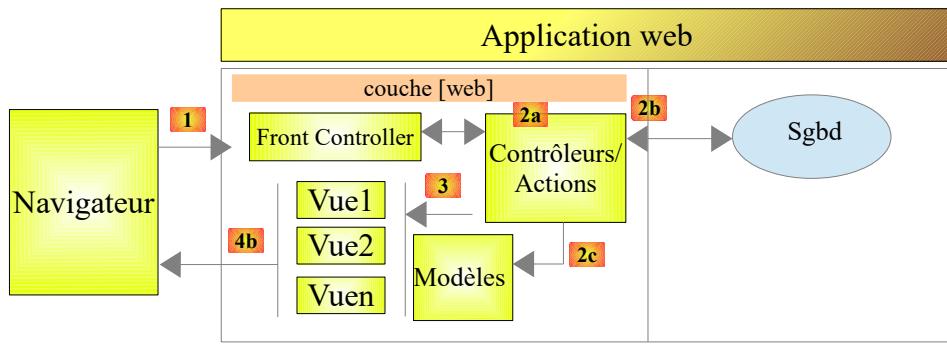
ASP.NET MVC implémente le modèle d'architecture dit MVC (Modèle – Vue – Contrôleur) de la façon suivante :



Le traitement d'une demande d'un client se déroule de la façon suivante :

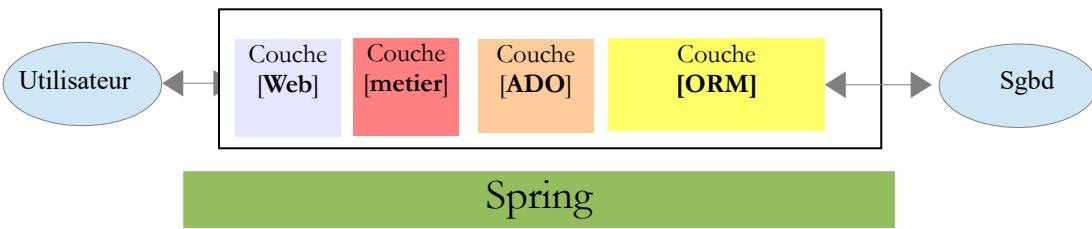
1. **demande** - les URL demandées sont de la forme `http:// machine:port/ contexte/ Contrôleur/ Action/ param1/ param2/ ...? p1=v1&p2=v2&...`. Le [Front Controller] utilise un fichier de configuration pour "router" la demande vers le bon contrôleur et la bonne action au sein de ce contrôleur. Pour cela, il utilise le chemin [Contrôleur/Action] de l'URL. Le reste de l'URL [/param1/param2/] sont des paramètres facultatifs qui seront transmis à l'action. Le **C** de MVC est ici la chaîne [Front Controller, Contrôleur, Action]. Si le chemin [Contrôleur/Action] n'aboutit pas à un contrôleur existant ou une action existante, le serveur Web répondra que l'URL demandée n'a pas été trouvée.
2. **traitement**
  - l'action choisie peut exploiter les paramètres *parami* que le [Front Controller] lui a transmis. Ceux-ci peuvent provenir de plusieurs sources :
    - du chemin [/param1/param2/...] de l'URL,
    - des paramètres [p1=v1&p2=v2] de l'URL,
    - de paramètres postés par le navigateur avec sa demande ;
  - dans le traitement de la demande de l'utilisateur, l'action peut avoir besoin de la couche [métier] [2b]. Une fois la demande du client traitée, celle-ci peut appeler diverses réponses. Un exemple classique est :
    - une page d'erreur si la demande n'a pu être traitée correctement
    - une page de confirmation sinon
  - l'action demande à une certaine vue de s'afficher [3]. Cette vue va afficher des données qu'on appelle le **modèle de la vue**. C'est le **M** de MVC. L'action va créer ce modèle M [2c] et demander à une vue V de s'afficher [3] ;
3. **réponse** - la vue V choisie utilise le modèle M construit par l'action pour initialiser les parties dynamiques de la réponse HTML qu'elle doit envoyer au client puis envoie cette réponse.

Maintenant, précisons le lien entre architecture Web MVC et architecture en couches. Selon la définition qu'on donne au **modèle**, ces deux concepts sont liés ou non. Prenons une application Web ASPNET MVC à une couche :



Si nous implémentons la couche [Web] avec ASP.NET MVC, nous aurons bien une architecture Web MVC mais pas une architecture multicouche. Ici, la couche [Web] s'occupera de tout : présentation, métier, accès aux données. Ce sont les actions qui feront ce travail.

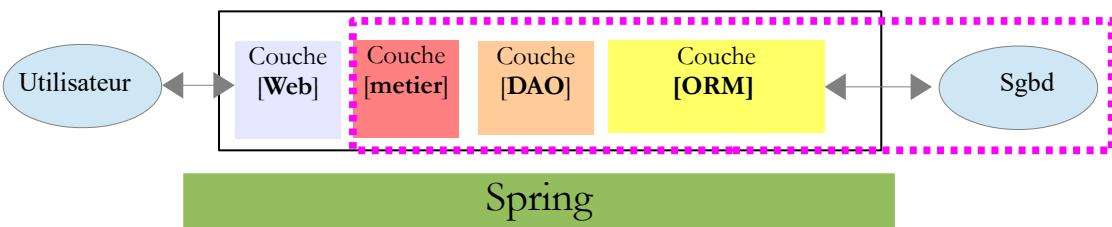
Maintenant, considérons une architecture Web multicouche :



La couche [Web] peut être implémentée sans framework et sans suivre le modèle MVC. On a bien alors une architecture multicouche mais la couche Web n'implémente pas le modèle MVC.

Ainsi, la couche [Web] ci-dessus peut être implémentée avec ASP.NET MVC et on a alors une architecture en couches avec une couche [Web] de type MVC. Ceci fait, on peut remplacer cette couche ASP.NET MVC par une couche ASP.NET classique (WebForms) tout en gardant le reste (métier, DAO, ORM) à l'**identique**. On a alors une architecture en couches avec une couche [Web] qui n'est plus de type MVC.

Dans MVC, nous avons dit que le modèle M était celui de la vue V, c.a.d. l'ensemble des données affichées par la vue V. Une autre définition du modèle M de MVC est donnée :



Beaucoup d'auteurs considèrent que ce qui est à droite de la couche [Web] forme le modèle M du MVC. Pour éviter les ambiguïtés on peut parler :

- du **modèle du domaine** lorsqu'on désigne tout ce qui est à droite de la couche [Web]
- du **modèle de la vue** lorsqu'on désigne les données affichées par une vue V

Dans la suite, le terme "modèle M" désignera exclusivement le **modèle d'une vue V**.

### 1.1.3 Les outils utilisés

Dans la suite, nous utiliserons (septembre 2013) les versions Express de Visual Studio 2012 :

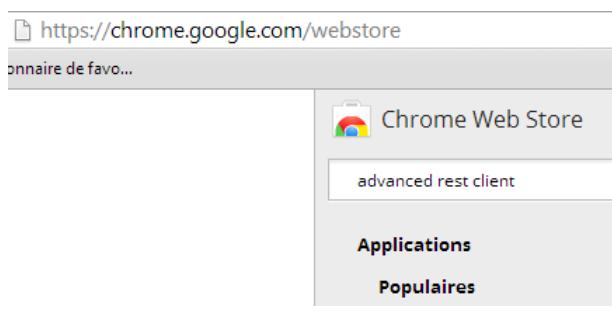
- [<http://www.microsoft.com/visualstudio/fra/products/visual-studio-express-for-windows-desktop>] pour les applications de bureau ;
- [<http://www.microsoft.com/visualstudio/fra/products/visual-studio-express-for-Web>] pour les applications Web.

Pour installer les dernières versions de ces produits, on pourra utiliser le produit [Web Platform Installer] de Microsoft (<http://www.microsoft.com/Web/downloads/platform.aspx>).

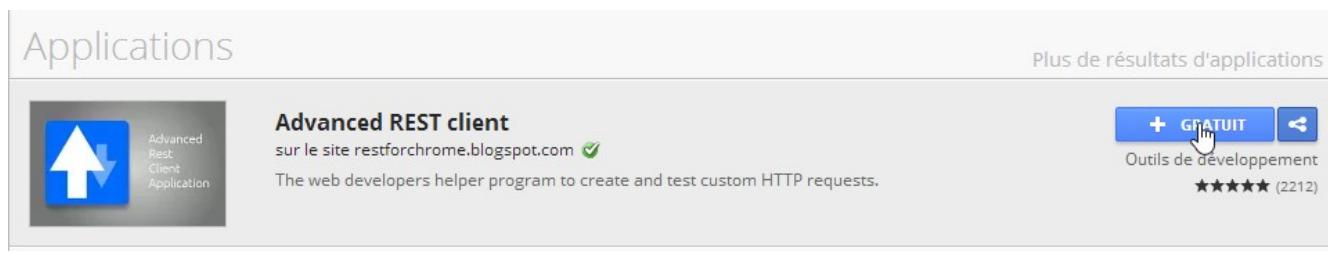
**Note :** vous aurez à adapter les copies d'écran Visual Studio 2012 qui sont présentées dans ce document à la version de Visual Studio que vous utiliserez.

Par ailleurs, on utilisera le navigateur **Chrome** de Google (<http://www.google.fr/intl/fr/chrome/browser/>). On lui ajoutera l'extension [Advanced Rest Client]. On pourra procéder ainsi :

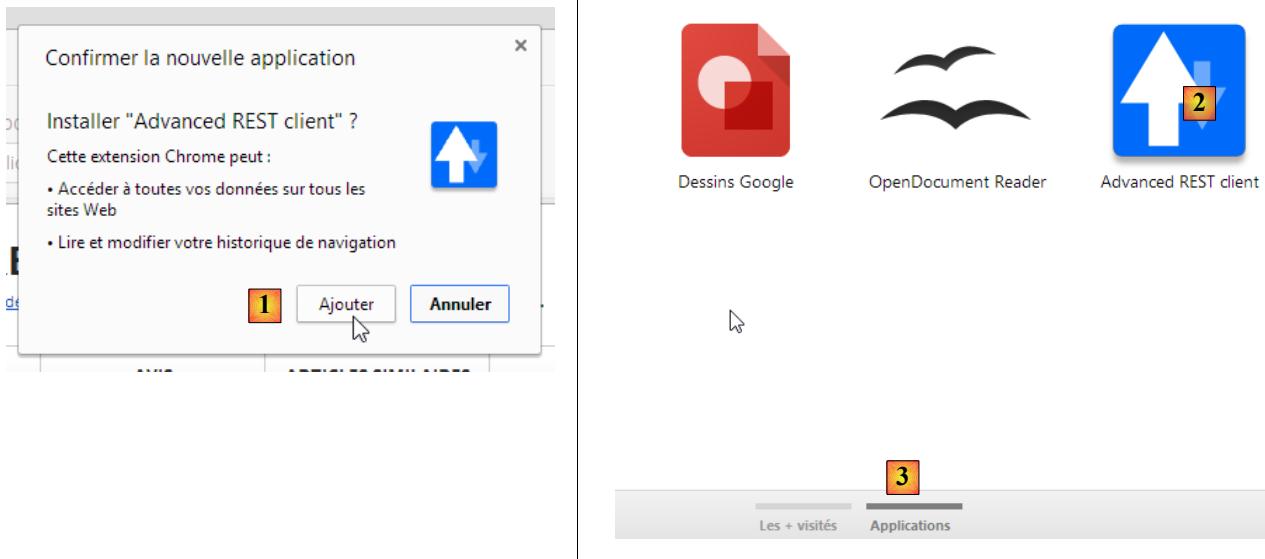
- aller sur le site de [Google Web store] (<https://chrome.google.com/webstore>) avec le navigateur Chrome ;
- chercher l'application [Advanced Rest Client] :



- l'application est alors disponible au téléchargement :



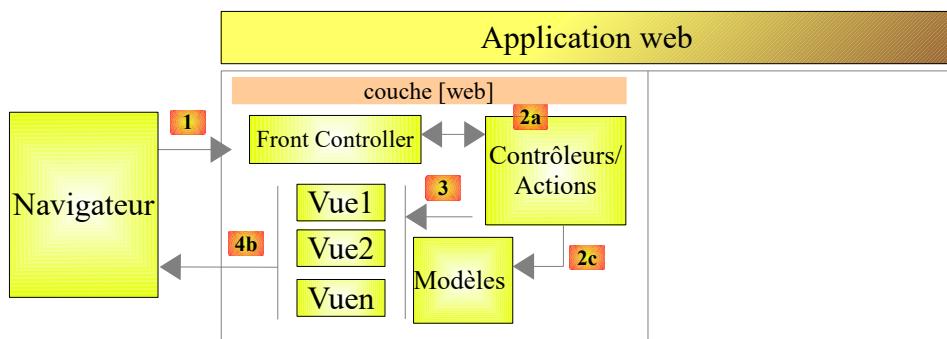
- pour l'obtenir, il vous faudra créer un compte Google. [Google Web Store] demande ensuite confirmation [1] :



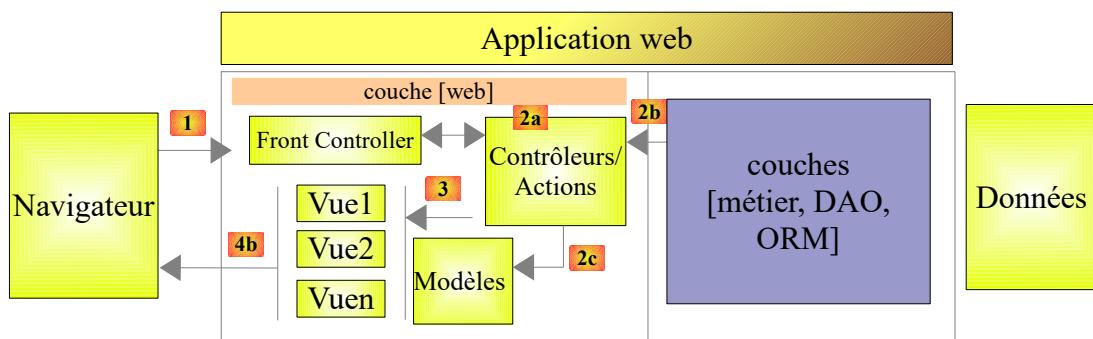
- en [2], l'extension ajoutée est disponible dans l'option [Applications] [3]. Cette option est affichée sur chaque nouvel onglet que vous créez (CTRL-T) dans le navigateur.

#### 1.1.4 Les exemples

La plupart des exemples d'apprentissage seront réduits à la seule couche Web :



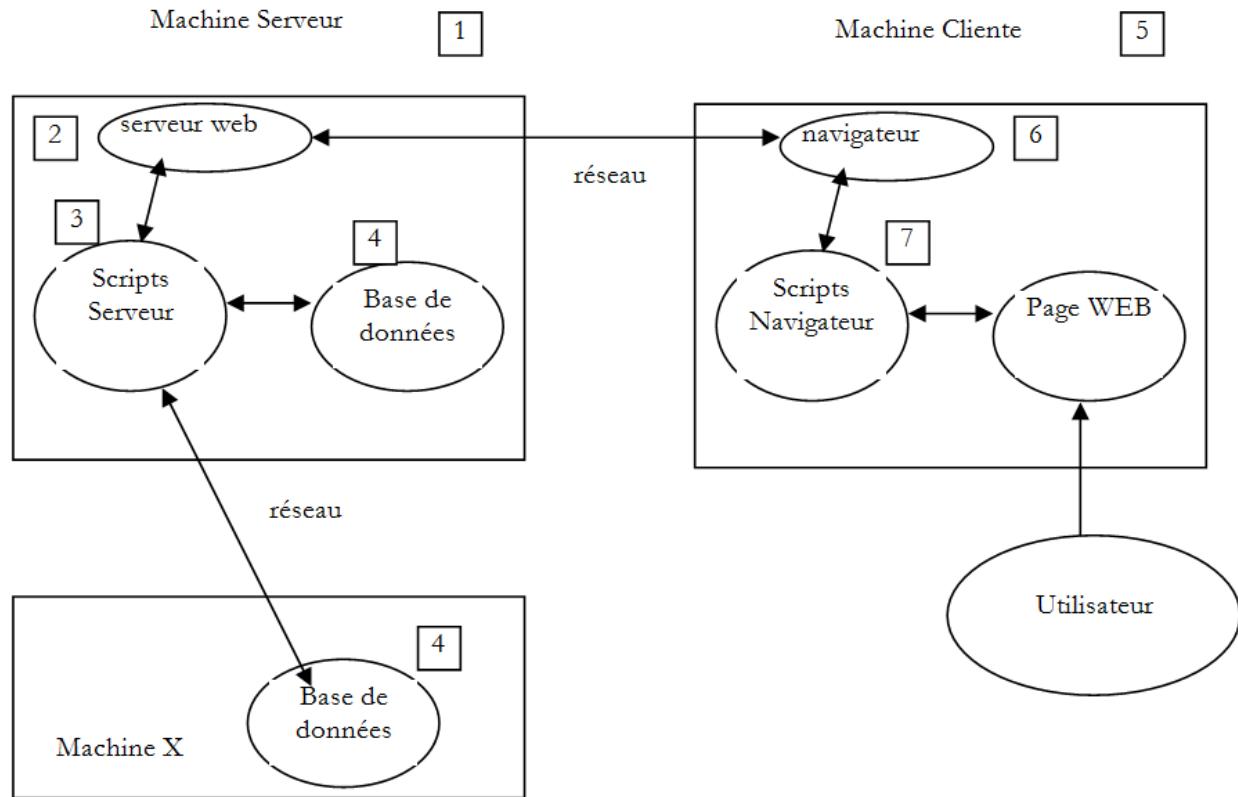
Une fois terminé cet apprentissage, nous présenterons une application Web multicouche :



## 1.2 Les bases de la programmation Web

Ce chapitre a pour but essentiel de faire découvrir les grands principes de la programmation Web qui sont indépendants de la technologie particulière utilisée pour les mettre en oeuvre. Il présente de nombreux exemples qu'il est conseillé de tester afin de "s'imprégner" peu à peu de la philosophie du développement Web. Le lecteur ayant déjà ces connaissances peut passer directement au chapitre suivant [page 45](#).

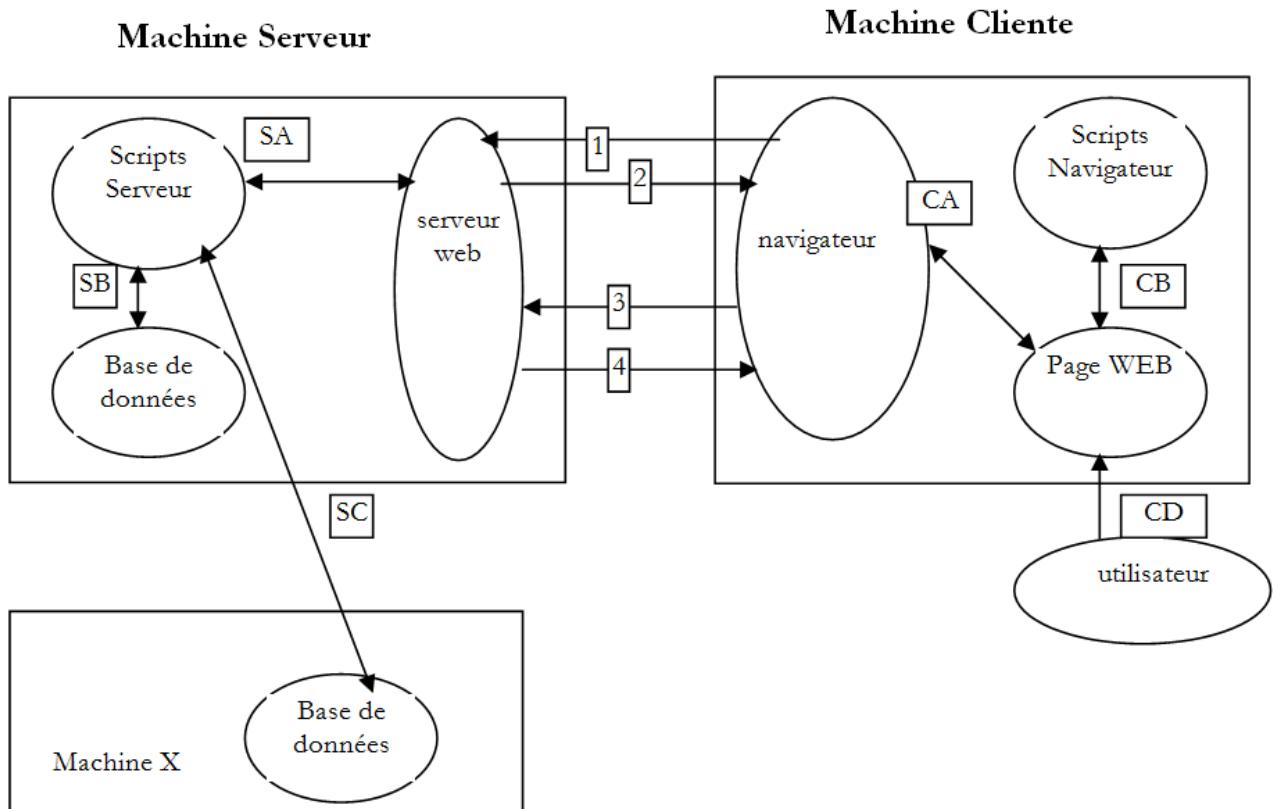
Les composantes d'une application Web sont les suivantes :



Numéro	Rôle	Exemples courants
1	OS Serveur	Unix, Linux, Windows
2	Serveur Web	Apache (Unix, Linux, Windows) IIS (Windows+plate-forme .NET) Node.js (Unix, Linux, Windows)
3	Codes exécutés côté serveur. Ils peuvent l'être par des modules du serveur ou par des programmes externes au serveur (CGI).	JAVASCRIPT (Node.js) PHP (Apache, IIS) JAVA (Tomcat, Websphere, JBoss, Weblogic, ...) C#, VB.NET (IIS)
4	Base de données - Celle-ci peut être sur la même machine que le programme qui l'exploite ou sur une autre via Internet.	Oracle (Linux, Windows) MySQL (Linux, Windows) Postgres (Linux, Windows) SQL Server (Windows)
5	OS Client	Unix, Linux, Windows
6	Navigateur Web	Chrome, Internet Explorer, Firefox,

- 7 Scripts exécutés côté client au sein du navigateur. Ces scripts n'ont aucun accès aux disques du poste client.

### 1.2.1 Les échanges de données dans une application Web avec formulaire



#### Numéro

#### Rôle

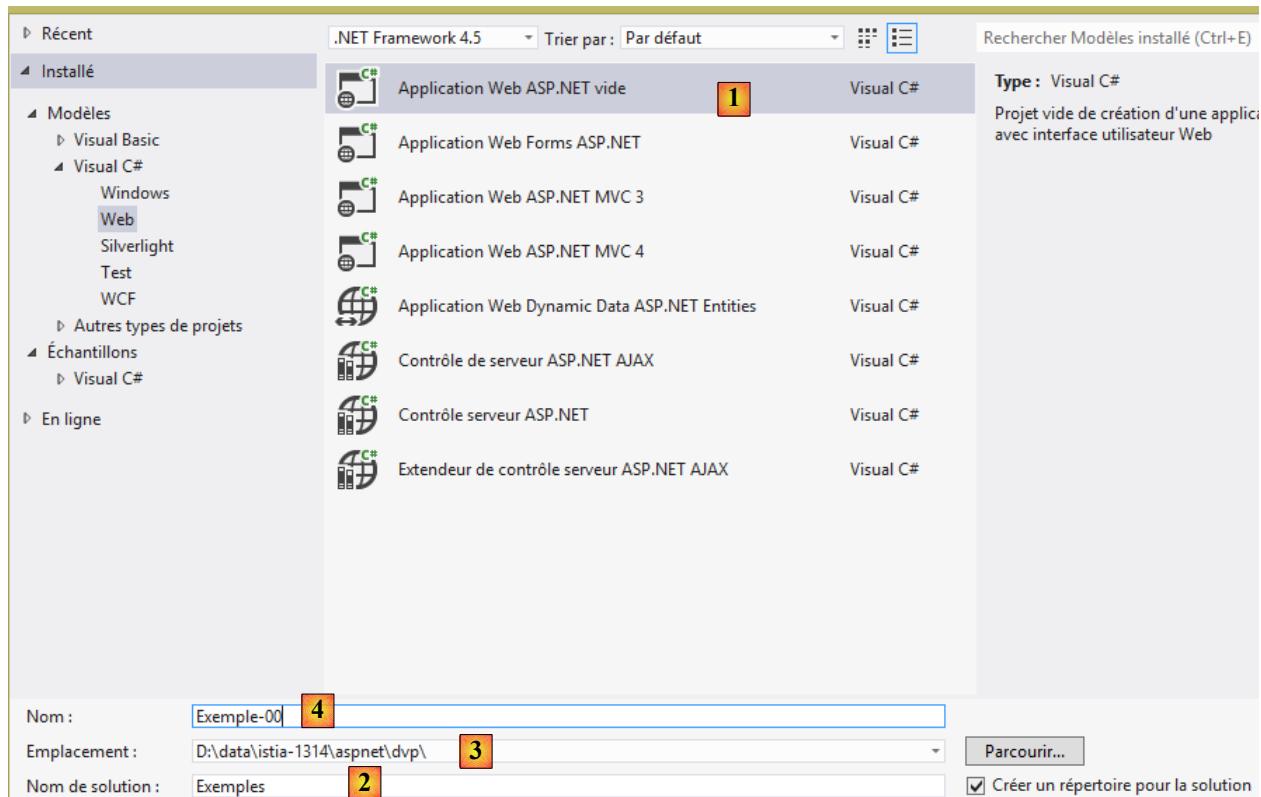
- Le navigateur demande une URL pour la 1ère fois (`http://machine/url`). Aucun paramètre n'est passé.
- Le serveur Web lui envoie la page Web de cette URL. Elle peut être statique ou bien dynamiquement générée par un script serveur (SA) qui a pu utiliser le contenu de bases de données (SB, SC). Ici, le script détectera que l'URL a été demandée sans passage de paramètres et générera la page Web initiale.  
Le navigateur reçoit la page et l'affiche (CA). Des scripts côté navigateur (CB) ont pu modifier la page initiale envoyée par le serveur. Ensuite par des interactions entre l'utilisateur (CD) et les scripts (CB) la page Web va être modifiée. Les formulaires vont notamment être remplis.
- L'utilisateur valide les données du formulaire qui doivent alors être envoyées au serveur Web. Le navigateur redemande l'URL initiale ou une autre selon les cas et transmet en même temps au serveur les valeurs du formulaire. Il peut utiliser pour ce faire deux méthodes appelées GET et POST. A réception de la demande du client, le serveur déclenche le script (SA) associé à l'URL demandée, script qui va détecter les paramètres et les traiter.
- Le serveur délivre la page Web construite par programme (SA, SB, SC). Cette étape est identique à l'étape 2 précédente. Les échanges se font désormais selon les étapes 2 et 3.

## 1.2.2 Pages Web statiques, Pages Web dynamiques

Une page statique est représentée par un fichier HTML. Une page dynamique est une page HTML générée "à la volée" par le serveur Web.

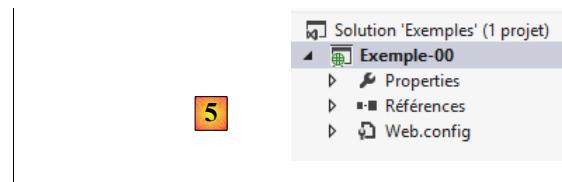
### 1.2.2.1 Page statique HTML (*HyperText Markup Language*)

Construisons un premier projet Web avec Visual Studio Express 2012. On utilise l'option [Fichier / Nouveau projet] :



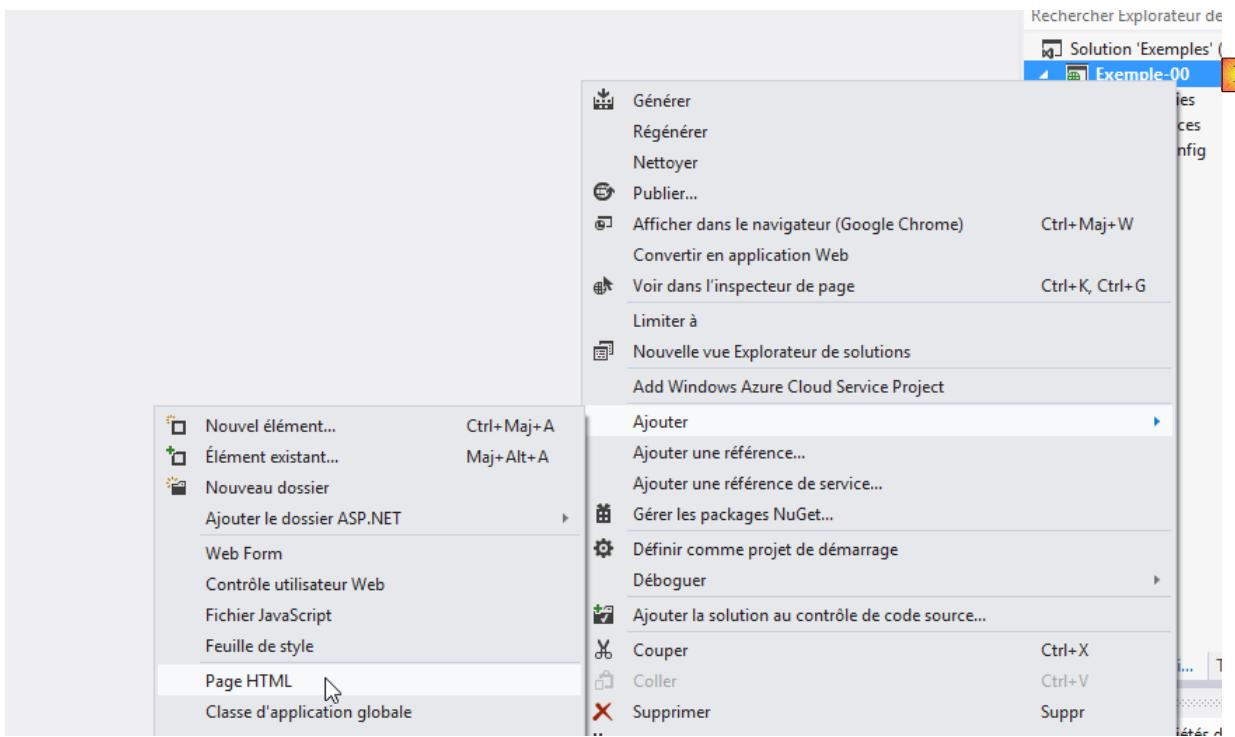
- en [1], on indique que l'on veut construire une application ASP.NET vide ;
- en [2], le nom de la solution Visual Studio. Tous les exemples de ce document seront dans la même solution ;
- en [3], le dossier parent de celui du projet qui va être créé ;
- en [4], le nom du projet.

On valide.

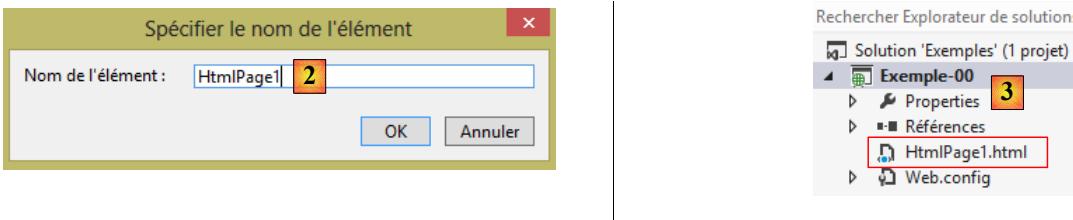


Le projet résultant est présenté en [5]. Nous allons l'utiliser pour illustrer les grands principes de la programmation Web.

Commençons par créer une page HTML statique :



- en [1], clic droit sur le projet puis suivre les options ;



- en [2], donner un nom à la page ;
- en [3], la page a été ajoutée.

Le contenu de la page créée est le suivant :

```

1. <!DOCTYPE html>
2. <html xmlns="http://www.w3.org/1999/xhtml">
3.   <head>
4.     <meta http-equiv="Content-Type" content="text/html; charset=utf-8"/>
5.     <title></title>
6.   </head>
7.   <body>
8.
9.   </body>
10.  </html>

```

- lignes 2-10 : le code est délimité par la balise racine <html> ;
- lignes 3-6 : la balise <head> délimite ce qu'on appelle l'entête de la page ;
- lignes 7-9 : la balise <body> délimite ce qu'on appelle le corps de la page.

Modifions ce code de la façon suivante :

```

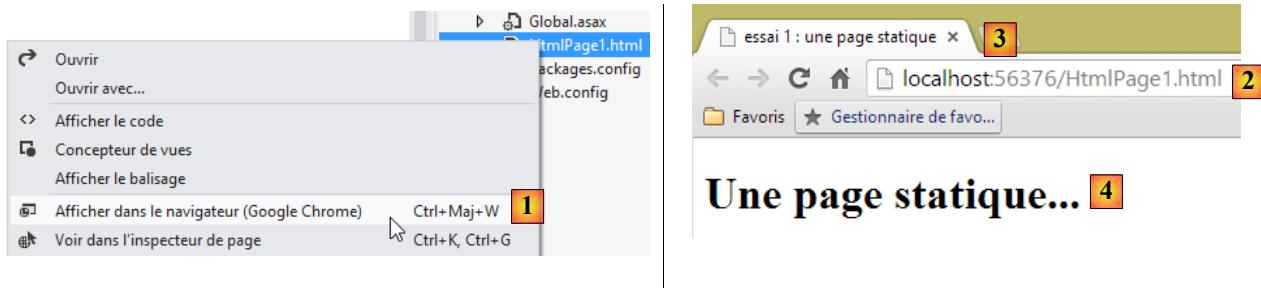
1. <!DOCTYPE html>
2. <html xmlns="http://www.w3.org/1999/xhtml">
3.   <head>
4.     <meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
5.     <title>essai 1 : une page statique</title>
6.   </head>
7.   <body>
8.     <h1>Une page statique...</h1>

```

```
9. </body>
10. </html>
```

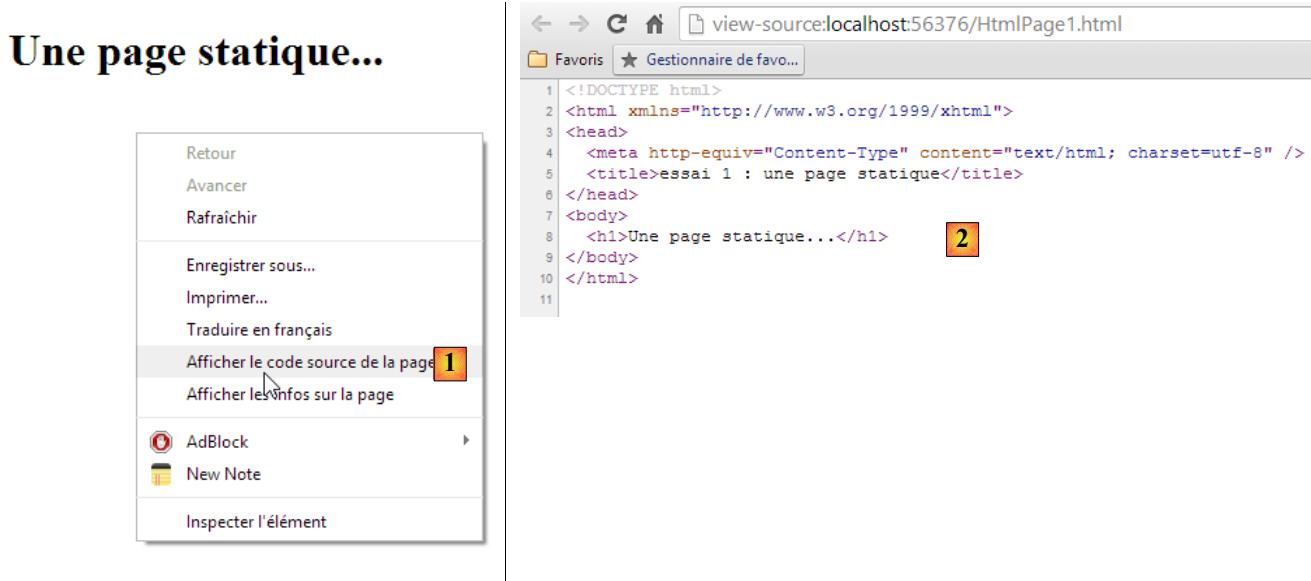
- ligne 5 : définit le titre de la page – sera affiché comme titre de la fenêtre du navigateur affichant la page ;
- ligne 8 : un texte en gros caractères (<h1>).

Visualisons cette page dans un navigateur :



- en [1], on demande la visualisation de la page ;
- en [2], l'URL de la page visualisée ;
- en [3], le titre de la fenêtre – a été fourni par la balise <title> de la page ;
- en [4], le corps de la page - a été fourni par la balise <h1>.

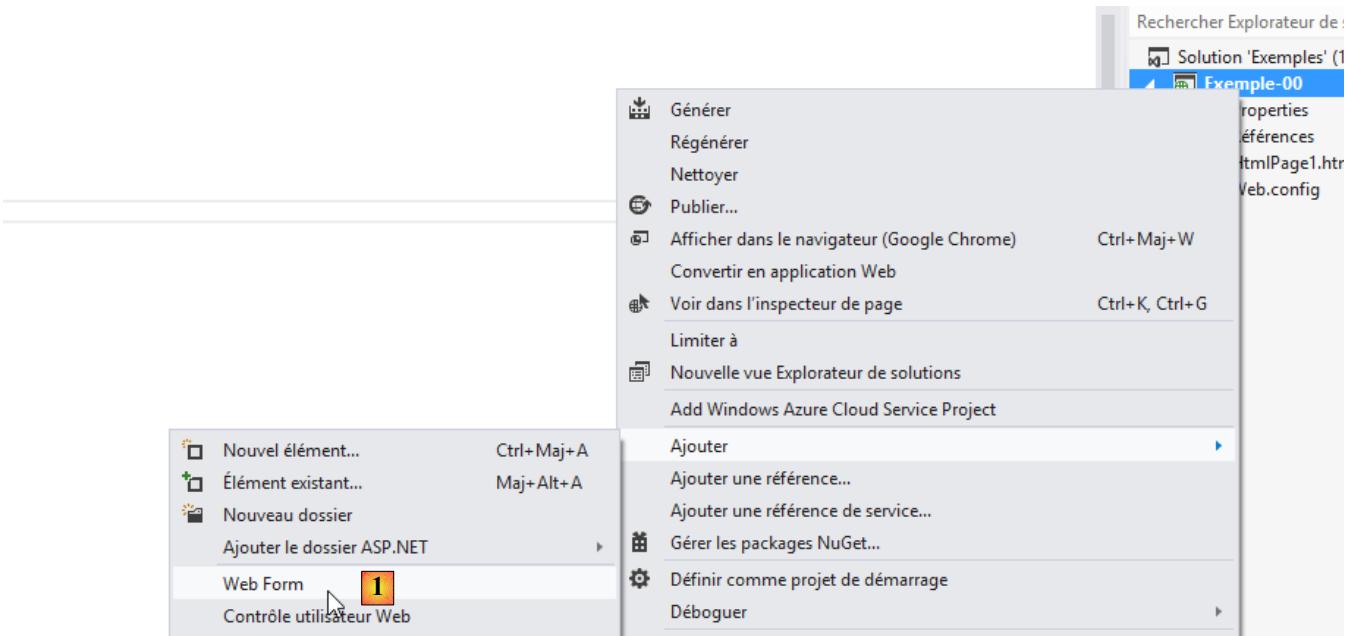
Regardons [1] le code HTML reçu par le navigateur :



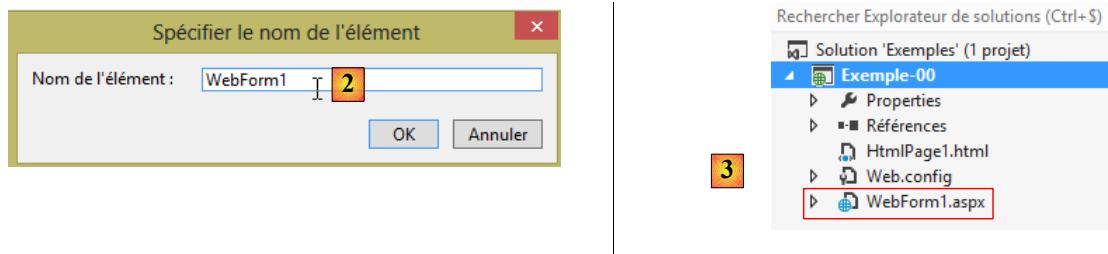
- en [2], le navigateur a reçu la page HTML que nous avions construite. Il l'a interprétée et en a fait un affichage graphique.

### 1.2.2.2 Une page ASP.NET

Créons maintenant une page ASP.NET. C'est une page HTML qui peut contenir du code exécuté côté serveur et qui génère certaines parties de la page. On suit une démarche analogue à celui de la création de la page HTML :



- en [1], une page ASP.NET est appelée une [Web Form] ;



- en [2], on donne un nom à la nouvelle page ;
- en [3], la page a été créée.

Le code de la page créée est le suivant :

```

1. <%@ Page Language="C#" AutoEventWireup="true" CodeBehind="WebForm1.aspx.cs" Inherits="Exemple_00.WebForm1" %>
2.
3. <!DOCTYPE html>
4.
5. <html xmlns="http://www.w3.org/1999/xhtml">
6. <head runat="server">
7. <meta http-equiv="Content-Type" content="text/html; charset=utf-8"/>
8. <title></title>
9. </head>
10. <body>
11.   <form id="form1" runat="server">
12.     <div>
13.
14.   </div>
15. </form>
16. </body>
17. </html>
```

On retrouve des balises HTML déjà rencontrées. Les balises qui ont l'attribut [runat= "server "] sont des balises qui vont être traitées par le serveur et transformées en balises HTML pures. Donc ce qu'on voit ci-dessus, n'est pas comme dans le cas de la page statique précédente le code HTML que va recevoir le navigateur. On parle alors de page dynamique : le flux HTML envoyé au serveur est produit par du code exécuté côté serveur. Modifions la page comme suit :

```

1. <%@ Page Language="C#" AutoEventWireup="true" CodeBehind="WebForm1.aspx.cs" Inherits="Exemple_00.WebForm1" %>
2.
3. <!DOCTYPE html>
```

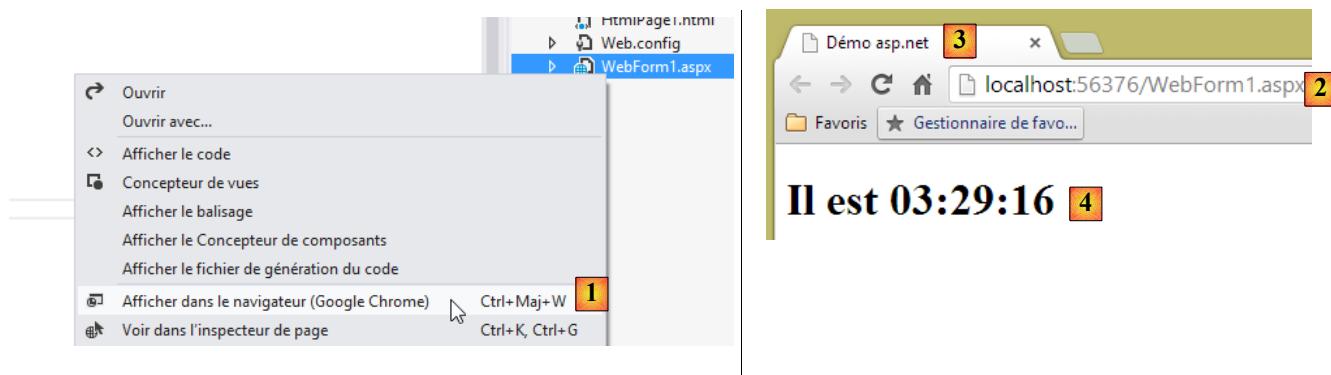
```

4. <html xmlns="http://www.w3.org/1999/xhtml">
5. <head runat="server">
6.   <meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
7.   <title>Démo asp.net</title>
8. </head>
9. <body>
10.  <form id="form1" runat="server">
11.    <div>
12.      <h1>Il est <% =DateTime.Now.ToString("hh:mm:ss") %></h1>
13.    </div>
14.  </form>
15. </body>
16. </html>

```

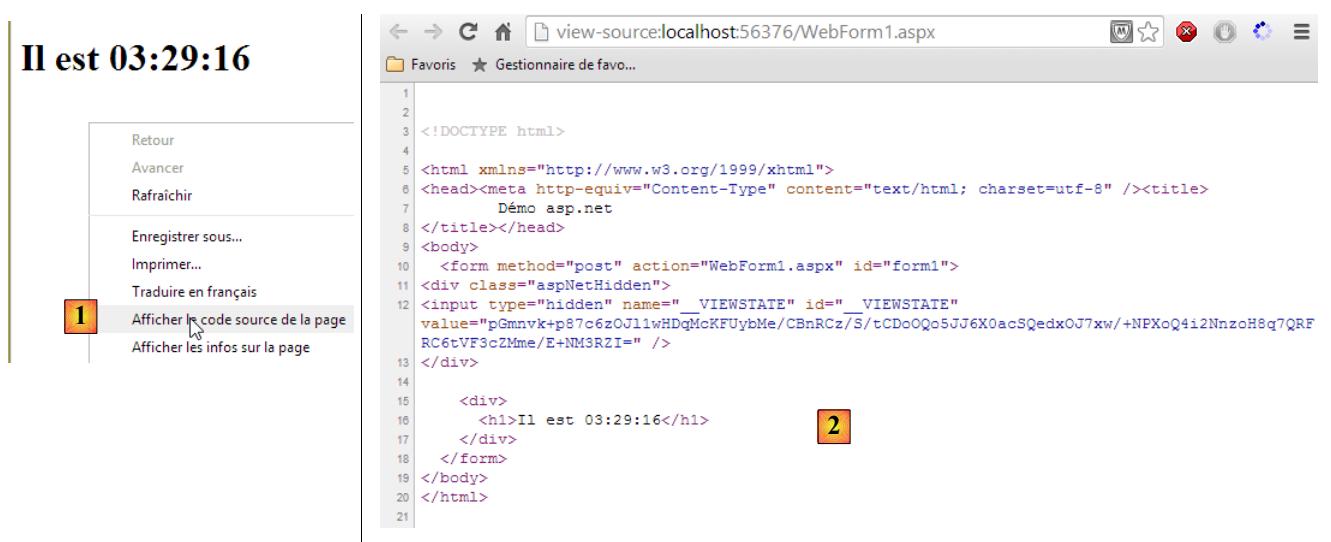
- ligne 8 : on donne un titre à la page ;
- ligne 13 : on affiche du texte généré par du code C#. Ce code est entre les balises <% %>. Ce code C# affiche l'heure courante sous la forme heures:minutes:secondes.

Affichons [1] cette page dans un navigateur :



- en [1], on demande la visualisation de la page ;
- en [2], l'URL de la page visualisée ;
- en [3], le titre de la fenêtre – a été fourni par la balise <title> de la page ;
- en [4], le corps de la page - a été fourni par la balise <h1>.

Si on rafraîchit la page (F5), nous obtenons un autre affichage (nouvelle heure) alors que l'URL ne change pas. C'est l'aspect dynamique de la page : son contenu peut changer au fil du temps. Regardons maintenant le code HTML reçu par le navigateur :



- en [1], on visualise le code source de la page ;

- en [2] : cette fois-ci le code HTML reçu n'est pas celui que nous avons construit mais celui qui a été généré par le serveur Web à partir des informations de notre page ASP.NET.

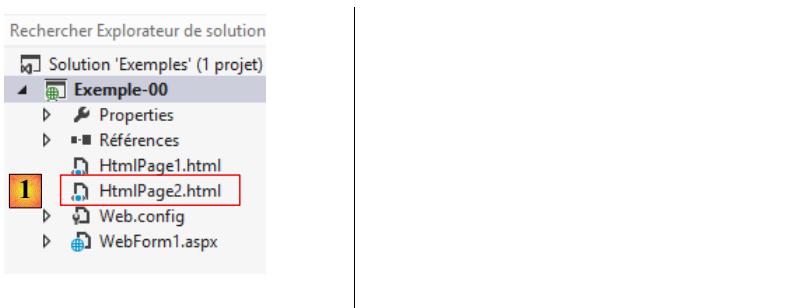
### 1.2.2.3 Conclusion

On retiendra de ce qui précède la nature fondamentalement différente des pages dynamiques et statiques.

## 1.2.3 Scripts côté navigateur

Une page HTML peut contenir des scripts qui seront exécutés par le navigateur. Le principal langage de script côté navigateur est actuellement (sept 2013) **Javascript**. Des centaines de bibliothèques ont été construites avec ce langage pour faciliter la vie du développeur.

Construisons une nouvelle page HTML [1] dans le projet déjà créé :



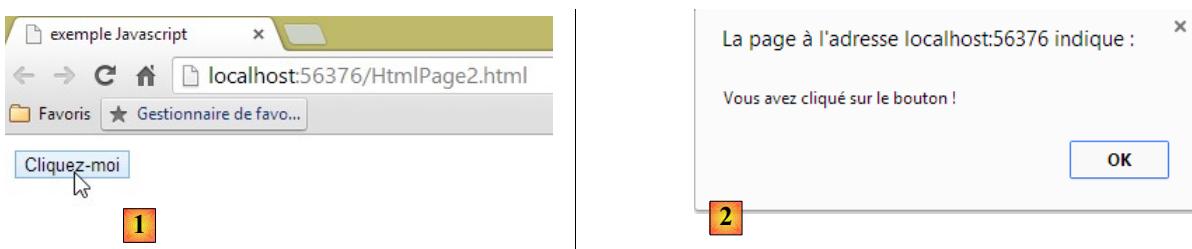
Editons le fichier [HtmlPage2.html] avec le contenu suivant :

```

1. <!DOCTYPE html>
2. <html xmlns="http://www.w3.org/1999/xhtml">
3. <head>
4.   <meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
5.   <title>exemple Javascript</title>
6.   <script type="text/javascript">
7.     function réagir() {
8.       alert("Vous avez cliqué sur le bouton !");
9.     }
10.  </script>
11. </head>
12. <body>
13.   <input type="button" value="Cliquez-moi" onclick="réagir()" />
14. </body>
15. </html>
```

- ligne 13 : définit un bouton (attribut **type**) avec le texte " Cliquez-moi " (attribut **value**). Lorsqu'on clique dessus, la fonction Javascript [réagir] est exécutée (attribut **onclick**) ;
- lignes 6-10 : un script Javascript ;
- lignes 7-9 : la fonction [réagir] ;
- ligne 8 : affiche une boîte de dialogue avec le message [Vous avez cliqué sur le bouton].

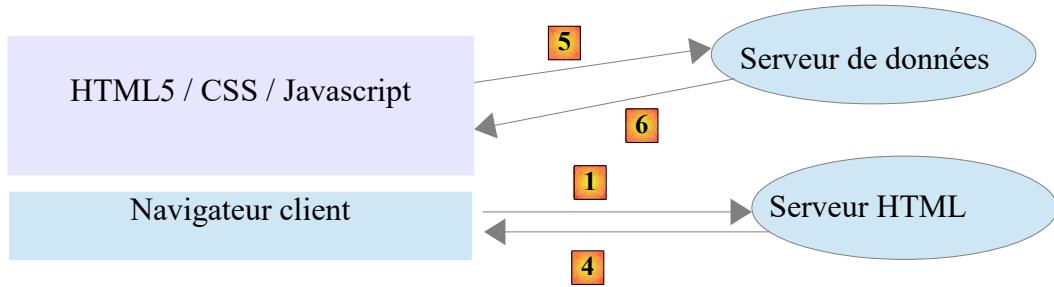
Visualisons la page dans un navigateur :



- en [1], la page affichée ;
- en [2], la boîte de dialogue lorsqu'on clique sur le bouton.

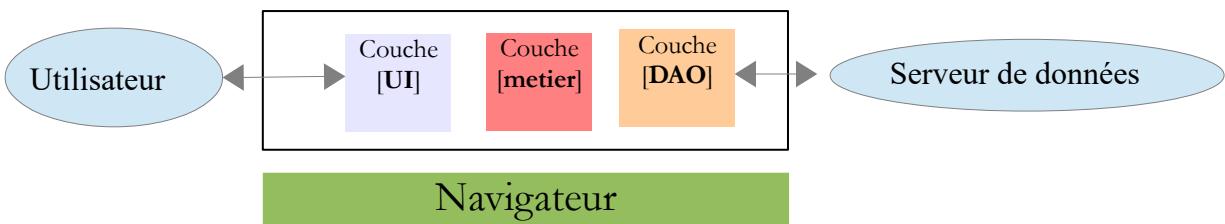
Lorsqu'on clique sur le bouton, il n'y a pas d'échanges avec le serveur. Le code Javascript est exécuté par le navigateur.

Avec les très nombreuses bibliothèques Javascript disponibles, on peut désormais embarquer de véritables applications sur le navigateur. On tend alors vers les architectures suivantes :



- 1-4 : le serveur HTML est un serveur de pages statiques HTML5 / CSS / Javascript ;
- 5-6 : les pages HTML5 / CSS / Javascript délivrées interagissent directement avec un serveur de données. Celui-ci délivre uniquement des données sans habillage HTML. C'est le Javascript qui les insère dans des pages HTML déjà présentes sur le navigateur.

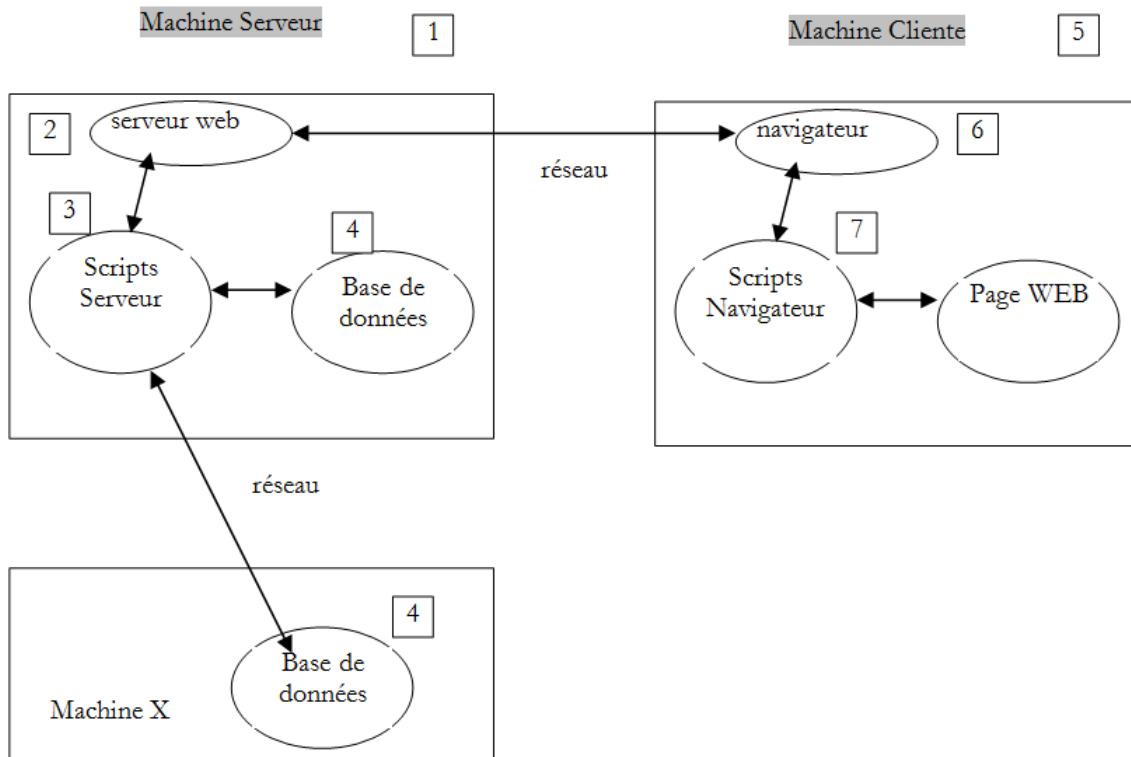
Dans cette architecture, le code Javascript peut devenir lourd. On cherche alors à la structurer en couches comme on le fait pour le code côté serveur :



- la couche [UI] est celle qui interagit avec l'utilisateur ;
- la couche [DAO] interagit avec le serveur de données ;
- la couche [métier] rassemble les procédures métier qui n'interagissent ni avec l'utilisateur, ni avec le serveur de données. Cette couche peut ne pas exister.

#### 1.2.4 Les échanges client-serveur

Revenons à notre schéma de départ qui illustre les acteurs d'une application Web :



Nous nous intéressons ici aux échanges entre la machine cliente et la machine serveur. Ceux-ci se font au travers d'un réseau et il est bon de rappeler la structure générale des échanges entre deux machines distantes.

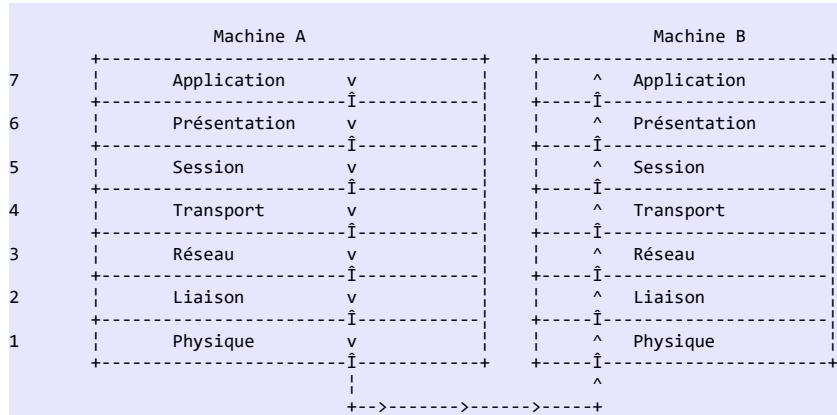
#### 1.2.4.1 Le modèle OSI

Le modèle de réseau ouvert appelé **OSI** (Open Systems Interconnection Reference Model) défini par l'**ISO** (International Standards Organisation) décrit un réseau idéal où la communication entre machines peut être représentée par un modèle à sept couches :



Chaque couche reçoit des services de la couche inférieure et offre les siens à la couche supérieure. Supposons que deux applications situées sur des machines A et B différentes veulent communiquer : elles le font au niveau de la couche *Application*. Elles n'ont pas besoin de connaître tous les détails du fonctionnement du réseau : chaque application remet l'information qu'elle souhaite transmettre à la couche du dessous : la couche *Présentation*. L'application n'a donc à connaître que les règles d'interfaçage avec la couche *Présentation*. Une fois l'information dans la couche *Présentation*, elle est passée selon d'autres règles à la couche *Session* et ainsi de suite, jusqu'à ce que l'information arrive sur le support physique et soit transmise physiquement à la machine destination. Là, elle subira le traitement inverse de celui qu'elle a subi sur la machine expéditeur.

A chaque couche, le processus expéditeur chargé d'envoyer l'information, l'envoie à un processus récepteur sur l'autre machine appartenant à la même couche que lui. Il le fait selon certaines règles que l'on appelle le **protocole** de la couche. On a donc le schéma de communication final suivant :

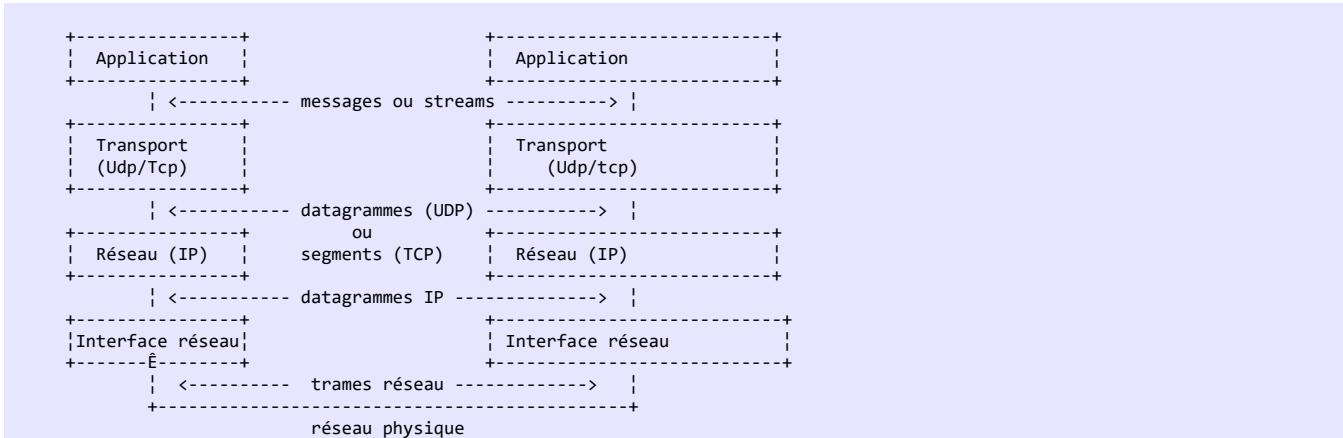


Le rôle des différentes couches est le suivant :

- Physique** Assure la transmission de bits sur un support physique. On trouve dans cette couche des équipements terminaux de traitement des données (E.T.T.D.) tels que terminal ou ordinateur, ainsi que des équipements de terminaison de circuits de données (E.T.C.D.) tels que modulateur/démodulateur, multiplexeur, concentrateur. Les points d'intérêt à ce niveau sont :
  - le choix du codage de l'information (analogique ou numérique)
  - le choix du mode de transmission (synchrone ou asynchrone).
- Liaison de données** Masque les particularités physiques de la couche Physique. Détecte et corrige les erreurs de transmission.
- Réseau** Gère le chemin que doivent suivre les informations envoyées sur le réseau. On appelle cela le *routage* : déterminer la route à suivre pour une information pour qu'elle arrive à son destinataire.
- Transport** Permet la communication entre deux applications alors que les couches précédentes ne permettaient que la communication entre machines. Un service fourni par cette couche peut être le multiplexage : la couche transport pourra utiliser une même connexion réseau (de machine à machine) pour transmettre des informations appartenant à plusieurs applications.
- Session** On va trouver dans cette couche des services permettant à une application d'ouvrir et de maintenir une session de travail sur une machine distante.
- Présentation** Elle vise à uniformiser la représentation des données sur les différentes machines. Ainsi des données provenant d'une machine A, vont être "habillées" par la couche *Présentation* de la machine A, selon un format standard avant d'être envoyées sur le réseau. Parvenues à la couche *Présentation* de la machine destinatrice B qui les reconnaîtra grâce à leur format standard, elles seront habillées d'une autre façon afin que l'application de la machine B les reconnaisse.
- Application** A ce niveau, on trouve les applications généralement proches de l'utilisateur telles que la messagerie électronique ou le transfert de fichiers.

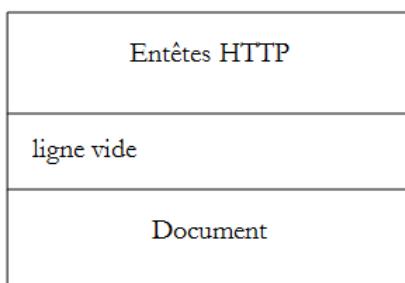
#### 1.2.4.2 Le modèle TCP/IP

Le modèle OSI est un modèle idéal. La suite de protocoles TCP/IP s'en approche sous la forme suivante :



- l'**interface réseau** (la carte réseau de l'ordinateur) assure les fonctions des couches 1 et 2 du modèle OSI
- la couche **IP** (Internet Protocol) assure les fonctions de la couche 3 (réseau)
- la couche **TCP** (Transfer Control Protocol) ou **UDP** (User Datagram Protocol) assure les fonctions de la couche 4 (transport). Le protocole TCP s'assure que les paquets de données échangés par les machines arrivent bien à destination. Si ce n'est pas le cas, il renvoie les paquets qui se sont égarés. Le protocole UDP ne fait pas ce travail et c'est alors au développeur d'applications de le faire. C'est pourquoi sur l'internet qui n'est pas un réseau fiable à 100%, c'est le protocole TCP qui est le plus utilisé. On parle alors de réseau **TCP-IP**.
- la couche **Application** recouvre les fonctions des niveaux 5 à 7 du modèle OSI.

Les applications Web se trouvent dans la couche *Application* et s'appuient donc sur les protocoles TCP-IP. Les couches *Application* des machines clientes et serveur s'échangent des messages qui sont confiées aux couches 1 à 4 du modèle pour être acheminées à destination. Pour se comprendre, les couches application des deux machines doivent "parler" un même langage ou protocole. Celui des applications Web s'appelle **HTTP** (HyperText Transfer Protocol). C'est un protocole de type texte, c.a.d. que les machines échangent des lignes de texte sur le réseau pour se comprendre. Ces échanges sont normalisés, c.-à-d. que le client dispose d'un certain nombre de messages pour indiquer exactement ce qu'il veut au serveur et ce dernier dispose également d'un certain nombre de messages pour donner au client sa réponse. Cet échange de messages a la forme suivante :



### Client --> Serveur

Lorsque le client fait sa demande au serveur Web, il envoie

1. des lignes de texte au format HTTP pour indiquer ce qu'il veut ;
2. une ligne vide ;
3. optionnellement un document.

### Serveur --> Client

Lorsque le serveur fait sa réponse au client, il envoie

1. des lignes de texte au format HTTP pour indiquer ce qu'il envoie ;
2. une ligne vide ;
3. optionnellement un document.

Les échanges ont donc la même forme dans les deux sens. Dans les deux cas, il peut y avoir envoi d'un document même s'il est rare qu'un client envoie un document au serveur. Mais le protocole HTTP le prévoit. C'est ce qui permet par exemple aux abonnés d'un fournisseur d'accès de télécharger des documents divers sur leur site personnel hébergé chez ce fournisseur d'accès. Les documents échangés peuvent être quelconques. Prenons un navigateur demandant une page Web contenant des images :

- le navigateur se connecte au serveur Web et demande la page qu'il souhaite. Les ressources demandées sont désignées de façon unique par des URL (Uniform Resource Locator). Le navigateur n'envoie que des entêtes HTTP et aucun document.
- le serveur lui répond. Il envoie tout d'abord des entêtes HTTP indiquant quel type de réponse il envoie. Ce peut être une erreur si la page demandée n'existe pas. Si la page existe, le serveur dira dans les entêtes HTTP de sa réponse qu'après ceux-ci il va envoyer un document **HTML** (HyperText Markup Language). Ce document est une suite de lignes de texte au format HTML. Un texte HTML contient des balises (marqueurs) donnant au navigateur des indications sur la façon d'afficher le texte.
- le client sait d'après les entêtes HTTP du serveur qu'il va recevoir un document HTML. Il va analyser celui-ci et s'apercevoir peut-être qu'il contient des références d'images. Ces dernières ne sont pas dans le document HTML. Il fait donc une nouvelle demande au même serveur Web pour demander la première image dont il a besoin. Cette demande est identique à celle faite en 1, si ce n'est que la resource demandée est différente. Le serveur va traiter cette demande en envoyant à son client l'image demandée. Cette fois-ci, dans sa réponse, les entêtes HTTP préciseront que le document envoyé est une image et non un document HTML.
- le client récupère l'image envoyée. Les étapes 3 et 4 vont être répétées jusqu'à ce que le client (un navigateur en général) ait tous les documents lui permettant d'afficher l'intégralité de la page.

#### 1.2.4.3 *Le protocole HTTP*

Découvrons le protocole HTTP sur des exemples. Que s'échangent un navigateur et un serveur Web ?

Le service Web ou service HTTP est un service TCP-IP qui travaille habituellement sur le port 80. Il pourrait travailler sur un autre port. Dans ce cas, le navigateur client serait obligé de préciser ce port dans l'URL qu'il demande. Une URL a la forme générale suivante :

**protocole://machine[:port]/chemin/infos**

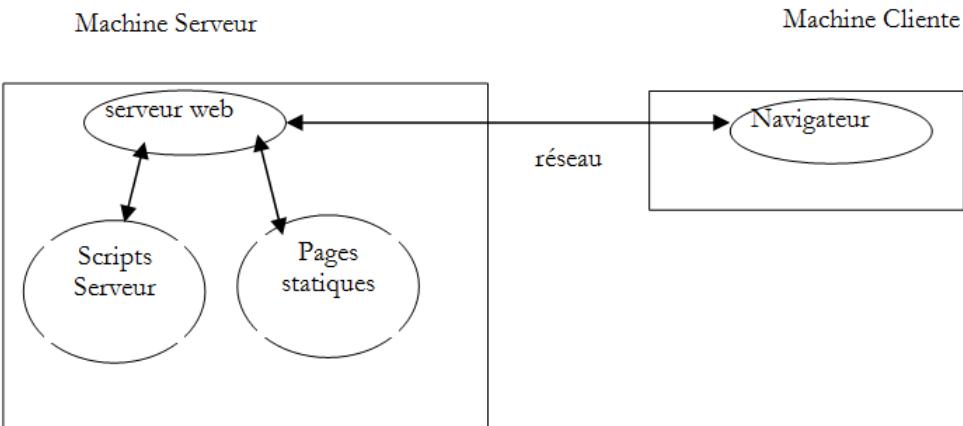
avec

<b>protocole</b>	http pour le service Web. Un navigateur peut également servir de client à des services ftp, news, telnet, ..
<b>machine</b>	nom de la machine où officie le service Web
<b>port</b>	port du service Web. Si c'est 80, on peut omettre le n° du port. C'est le cas le plus fréquent
<b>chemin</b>	chemin désignant la ressource demandée
<b>infos</b>	informations complémentaires données au serveur pour préciser la demande du client

Que fait un navigateur lorsqu'un utilisateur demande le chargement d'une URL ?

- il ouvre une communication TCP-IP avec la machine et le port indiqués dans la partie **machine[:port]** de l'URL. Ouvrir une communication TCP-IP, c'est créer un "tuyau" de communication entre deux machines. Une fois ce tuyau créé, toutes les informations échangées entre les deux machines vont passer dedans. La création de ce tuyau TCP-IP n'implique pas encore le protocole HTTP du Web.
- le tuyau TCP-IP créé, le client va faire sa demande au serveur Web et il va la faire en lui envoyant des lignes de texte (des commandes) au format HTTP. Il va envoyer au serveur la partie **chemin/infos** de l'URL
- le serveur lui répondra de la même façon et dans le même tuyau
- l'un des deux partenaires prendra la décision de fermer le tuyau. Cela dépend du protocole HTTP utilisé. Avec le protocole HTTP 1.0, le serveur ferme la connexion après chacune de ses réponses. Cela oblige un client qui doit faire plusieurs demandes pour obtenir les différents documents constituant une page Web à ouvrir une nouvelle connexion à chaque demande, ce qui a un coût. Avec le protocole HTTP/1.1, le client peut dire au serveur de garder la connexion ouverte jusqu'à ce qu'il lui dise de la fermer. Il peut donc récupérer tous les documents d'une page Web avec une seule connexion et fermer lui-même la connexion une fois le dernier document obtenu. Le serveur détectera cette fermeture et fermera lui aussi la connexion.

Pour découvrir les échanges entre un client et un serveur Web, nous allons utiliser l'extension [Advanced Rest Client] du navigateur Chrome que nous avons installée page 12. Nous serons dans la situation suivante :



Le serveur Web pourra être quelconque. Nous cherchons ici à découvrir les échanges qui vont se produire entre navigateur et le serveur Web. Précédemment, nous avons créé la page HTML statique suivante :

```

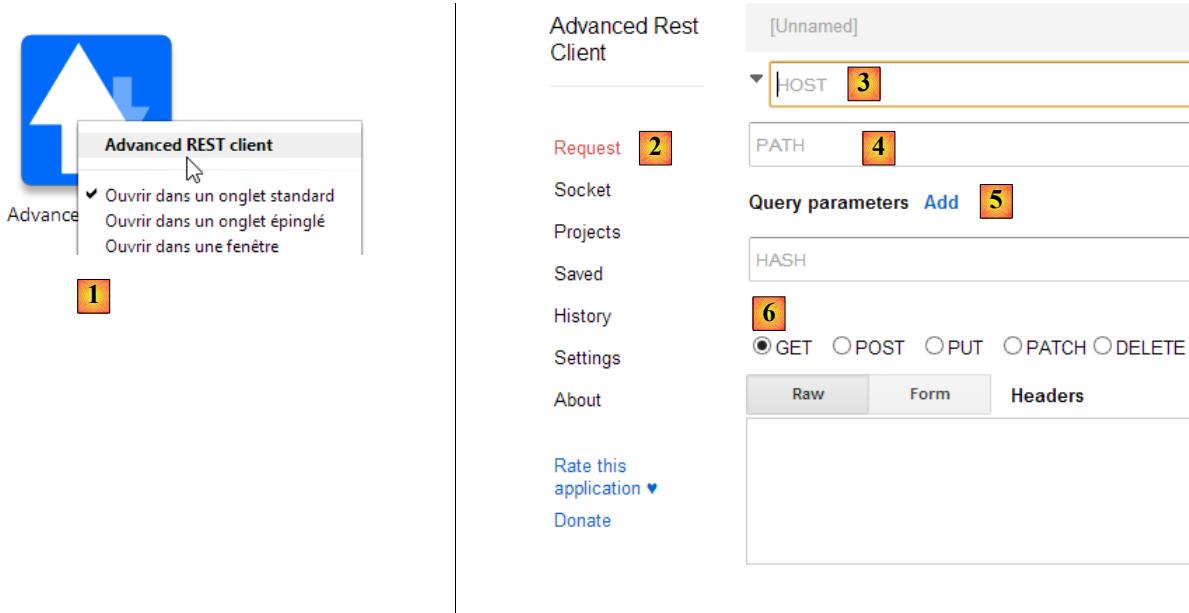
1.  <!DOCTYPE html>
2.  <html xmlns="http://www.w3.org/1999/xhtml">
3.  <head>
4.      <meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
5.      <title>essai 1 : une page statique</title>
6.  </head>
7.  <body>
8.      <h1>Une page statique...</h1>
9.  </body>
10. </html>

```

que nous visualisons dans un navigateur :

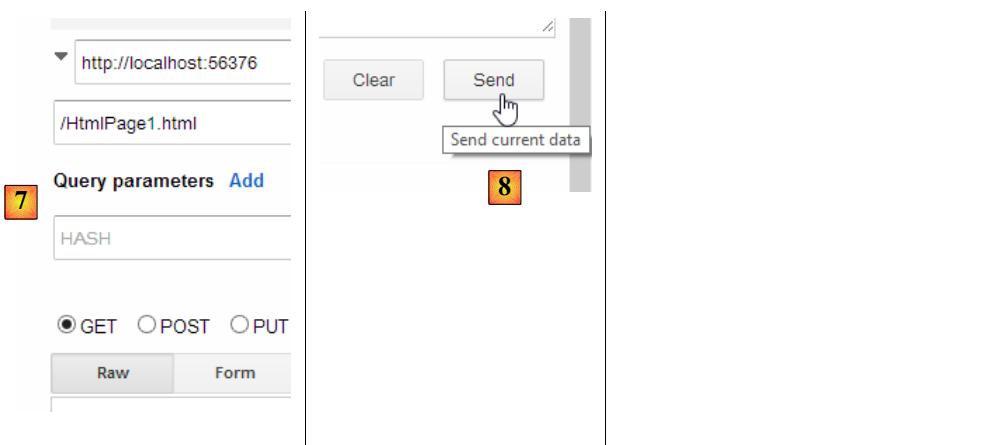


On voit que l'URL demandée est : `http://localhost:56376/HtmlPage1.html`. La machine du service Web est donc `localhost` (=machine locale) et le port 56376. Utilisons l'application [Advanced Rest Client] pour demander la même URL :



- en [1], on lance l'application (dans l'onglet [Applications] d'un nouvel onglet Chrome) ;
- en [2], on sélectionne l'option [Request] ;
- en [3], on précise le serveur interrogé : `http://localhost:56376`;
- en [4], on précise l'URL demandée : `/HtmlPage1.html` ;
- en [5], on ajoute d'éventuels paramètres à l'URL. Aucun ici ;
- en [6], on précise la commande HTTP utilisée pour la requête, ici GET.

Cela donne la requête suivante :



La requête ainsi préparée [7] est envoyée au serveur par [8]. La réponse obtenue est alors la suivante :

Status      **200 OK** Loading time: 13 ms

Request headers

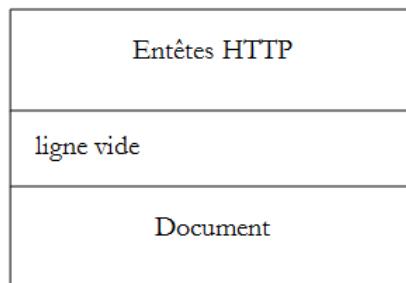
```
User-Agent: Mozilla/5.0 (Windows NT 6.2; WOW64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/29.0.1547.66 Safari/537.36
Content-Type: text/plain; charset=utf-8
Accept: */*
Accept-Encoding: gzip,deflate,sdch
Accept-Language: fr-FR,fr;q=0.8,en-US;q=0.6,en;q=0.4
```

Response headers

```
Accept-Ranges: bytes
Server: Microsoft-IIS/8.0
X-SourceFiles: =?UTF-8?B?RDpcZGF0YVxpc3RpYS0xMzE0XGFzcG5IdFxkdnBcRXhlbXBsZXNcRXhlbXBsZS0wMFxlG1sUGFnZTEuaHRtbA==?=
X-Powered-By: ASP.NET
Date: Wed, 18 Sep 2013 15:20:38 GMT
Content-Type: text/html
Content-Encoding: gzip
Last-Modified: Wed, 18 Sep 2013 13:13:19 GMT
ETag: "b474e0d770b4ce1:0"
Vary: Accept-Encoding
Content-Length: 313
```

Raw	Parsed	Response
<a href="#">Open output in new window</a>	<a href="#">Copy to clipboard</a>	<a href="#">Save as file</a>
<a href="#">Open in JSON tab</a>		
<pre>&lt;!DOCTYPE html&gt; &lt;html xmlns="http://www.w3.org/1999/xhtml"&gt; &lt;head&gt;   &lt;meta http-equiv="Content-Type" content="text/html; charset=utf-8" /&gt;   &lt;title&gt;essai 1 : une page statique&lt;/title&gt; &lt;/head&gt; &lt;body&gt;   &lt;h1&gt;Une page statique...&lt;/h1&gt; &lt;/body&gt; &lt;/html&gt;</pre>		
<a href="#">Code highlighting thanks to Code Mirror</a>		

Nous avons dit plus haut que les échanges client-serveur avaient la forme suivante :



- en [1], on voit les entêtes HTTP envoyés par le navigateur dans sa requête. Il n'avait pas de document à envoyer ;
- en [2], on voit les entêtes HTTP envoyés par le serveur en réponse. En [3], on voit le document qu'il a envoyé.

En [3], on reconnaît la page HTML statique que nous avons placée sur le serveur web.

Examinons la requête HTTP du navigateur :

```
1. GET /HtmlPage1.html HTTP/1.1
2. User-Agent: Mozilla/5.0 (Windows NT 6.2; WOW64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/29.0.1547.66 Safari/537.36
3. Content-Type: text/plain; charset=utf-8
4. Accept: /*
5. Accept-Encoding: gzip,deflate,sdch
6. Accept-Language: fr-FR,fr;q=0.8,en-US;q=0.6,en;q=0.4
```

- la ligne 1 n'a pas été affichée par l'application ;
- ligne 2 : le navigateur s'identifie avec l'entête [User-Agent] ;
- ligne 3 : le navigateur indique qu'il envoie au serveur un document texte (text/plain) au format UTF-8. En fait ici, le navigateur n'a envoyé aucun document ;
- ligne 4 : le navigateur indique qu'il accepte tout type de document en réponse ;
- ligne 5 : le navigateur précise les formats de document acceptés ;
- ligne 6 : le navigateur précise les langues qu'il souhaite par ordre de préférence.

Le serveur lui a répondu en envoyant les entêtes HTTP suivants :

```

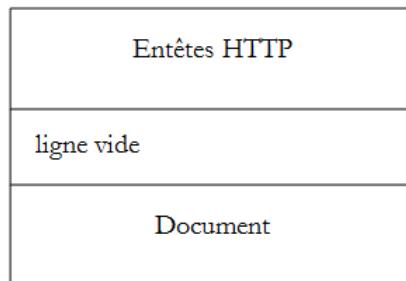
1. HTTP/1.1 304 Not Modified
2. Accept-Ranges: bytes
3. Server: Microsoft-IIS/8.0
4. X-SourceFiles: =?UTF-8?B?RDpcZGF0YVxpc3RpYS0xMzE0XGFzcG5ldFvkdnBcRXh1bXBsZXNcRXh1bXBsZS0wMFxIdG1sUGFnZTEuaHRtbA==?=?
5. X-Powered-By: ASP.NET
6. Date: Wed, 18 Sep 2013 15:33:53 GMT
7. Content-Type: text/html
8. Content-Encoding: gzip
9. Last-Modified: Wed, 18 Sep 2013 13:13:19 GMT
10. ETag: "b474e0d770b4ce1:0"
11. Vary: Accept-Encoding
12. Content-Length: 313

```

- ligne 1 : n'a pas été affichée par l'application ;
- ligne 3 : le serveur s'identifie, ici un serveur IIS de Microsoft ;
- ligne 5 : indique la technologie qui a générée la réponse, ici ASP.NET ;
- ligne 6 : date et heure de la réponse ;
- ligne 7 : la nature du document envoyé par le serveur. Ici un document HTML ;
- ligne 12 : la taille en octets du document HTML envoyé.

#### 1.2.4.4 Conclusion

Nous avons découvert la structure de la demande d'un client Web et celle de la réponse qui lui est faite par le serveur Web sur quelques exemples. Le dialogue se fait à l'aide du protocole HTTP, un ensemble de commandes au format texte échangées par les deux partenaires. La requête du client et la réponse du serveur ont la même structure suivante :



Les deux commandes usuelles pour demander une ressource sont GET et POST. La commande GET n'est pas accompagnée d'un document. La commande POST elle, est accompagnée d'un document qui est le plus souvent une chaîne de caractères rassemblant l'ensemble des valeurs saisies dans le formulaire. La commande HEAD permet de demander seulement les entêtes HTTP et n'est pas accompagnée de document.

A la demande d'un client, le serveur envoie une réponse qui a la même structure. La ressource demandée est transmise dans la partie [Document] sauf si la commande du client était HEAD, auquel cas seuls les entêtes HTTP sont envoyés.

### 1.2.5 Les bases du langage HTML

Un navigateur Web peut afficher divers documents, le plus courant étant le document HTML (HyperText Markup Language). Celui-ci est un texte formaté avec des balises de la forme `<balise>texte</balise>`. Ainsi le texte `<B>important</B>` affichera le texte important en gras. Il existe des balises seules telles que la balise `<br/>` qui affiche une ligne horizontale. Nous ne passerons pas en revue les balises que l'on peut trouver dans un texte HTML. Il existe de nombreux logiciels WYSIWYG permettant de construire une page Web sans écrire une ligne de code HTML. Ces outils génèrent automatiquement le code HTML d'une mise en page faite à l'aide de la souris et de contrôles prédéfinis. On peut ainsi insérer (avec la souris) dans la page un tableau puis consulter le code HTML généré par le logiciel pour découvrir les balises à utiliser pour définir un tableau dans une page Web. Ce n'est pas plus compliqué que cela. Par ailleurs, la connaissance du langage HTML est indispensable puisque les applications Web dynamiques doivent générer elles-mêmes le code HTML à envoyer aux clients Web. Ce code est généré par programme et il faut bien sûr savoir ce qu'il faut générer pour que le client ait la page Web qu'il désire.

Pour résumer, il n'est nul besoin de connaître la totalité du langage HTML pour démarrer la programmation Web. Cependant cette connaissance est nécessaire et peut être acquise au travers de l'utilisation de logiciels WYSIWYG de construction de pages Web tels que DreamWeaver et des dizaines d'autres. Une autre façon de découvrir les subtilités du langage HTML est de parcourir le Web et d'afficher le code source des pages qui présentent des caractéristiques intéressantes et encore inconnues pour vous.

### 1.2.5.1 Un exemple

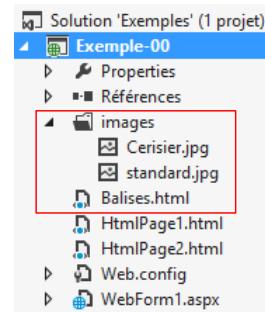
Considérons l'exemple suivant qui présente quelques éléments qu'on peut trouver dans un document Web tels que :

- un tableau ;
- une image ;
- un lien.

The screenshot shows a web browser window with the title 'balises'. The address bar says 'localhost:56376/Balises.html'. The page content includes a title 'Les balises HTML' and a 3x3 table:

Colonne 1	Colonne 2	Colonne 3
cellule(1,1)	cellule(1,2)	cellule(1,3)
cellule(2,1)	cellule(2,2)	cellule(2,3)

Below the table is a photograph of white cherry blossoms. To the left of the image is the caption 'Une image' and below it is the link 'le site de l'ISTIA [ici](#)'.



Un document HTML a la forme générale suivante :

```
<html>
  <head>
    <title>Un titre</title>
    ...
  </head>
  <body attributs>
    ...
  </body>
</html>
```

L'ensemble du document est encadré par les balises **<html>...</html>**. Il est formé de deux parties :

1. **<head>...</head>** : c'est la partie non affichable du document. Elle donne des renseignements au navigateur qui va afficher le document. On y trouve souvent la balise **<title>...</title>** qui fixe le texte qui sera affiché dans la barre de titre du navigateur. On peut y trouver d'autres balises notamment des balises définissant les mots clés du document, mot clés utilisés ensuite par les moteurs de recherche. On peut trouver également dans cette partie des scripts, écrits le plus souvent en javascript ou vbscript et qui seront exécutés par le navigateur.
2. **<body attributs>...</body>** : c'est la partie qui sera affichée par le navigateur. Les balises HTML contenues dans cette partie indiquent au navigateur la forme visuelle "souhaitée" pour le document. Chaque navigateur va interpréter ces balises à sa façon. Deux navigateurs peuvent alors visualiser différemment un même document Web. C'est généralement l'un des casse-têtes des concepteurs Web.

Le code HTML de notre document exemple est le suivant :

```
1.  <!DOCTYPE html>
2.  <html xmlns="http://www.w3.org/1999/xhtml">
3.  <head>
```

```

4.      <meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
5.      <title>balises</title>
6.    </head>
7.
8.    <body style="height: 400px; width: 400px; background-image: url(images/standard.jpg)">
9.      <h1 style="text-align: center">Les balises HTML</h1>
10.     <hr />
11.
12.    <table border="1">
13.      <thead>
14.        <tr>
15.          <th>Colonne 1</th>
16.          <th>Colonne 2</th>
17.          <th>Colonne 3</th>
18.        </tr>
19.      </thead>
20.      <tbody>
21.        <tr>
22.          <td>cellule(1,1)</td>
23.          <td style="width: 150px; text-align: center;">cellule(1,2)</td>
24.          <td>cellule(1,3)</td>
25.        </tr>
26.        <tr>
27.          <td>cellule(2,1)</td>
28.          <td>cellule(2,2)</td>
29.          <td>cellule(2,3)</td>
30.        </tr>
31.      </tbody>
32.    </table>
33.
34.    <table border="0">
35.      <tr>
36.        <td>Une image</td>
37.        <td>
38.          </td>
39.        </tr>
40.        <tr>
41.          <td>le site de l'ISTIA</td>
42.          <td><a href="http://istia.univ-angers.fr">ici</a></td>
43.        </tr>
44.      </table>
45.    </body>
46.  </html>

```

## Elément      balises et exemples HTML

### titre du document

<title>balises</title> (ligne 5)

le texte *balises* apparaîtra dans la barre de titre du navigateur qui affichera le document

### barre horizontale

<hr/> : affiche un trait horizontal (ligne 10)

### tableau

<table attributs>....</table> : pour définir le tableau (lignes 12, 32)

<thead>...</thead> : pour définir les entêtes des colonnes (lignes 13, 19)

<tbody>...</tbody> : pour définir le contenu du tableau (ligne 20, 31)

<tr attributs>...</tr> : pour définir une ligne (lignes 21, 25)

<td attributs>...</td> : pour définir une cellule (ligne 22)

### exemples :

<table border="1">...</table> : l'attribut border définit l'épaisseur de la bordure du tableau

<td style="width: 150px; text-align: center;">cellule(1,2)</td> : définit une cellule dont le contenu sera **cellule(1,2)**. Ce contenu sera centré horizontalement (text-align :center). La cellule aura une largeur de 150 pixels (width :150px)

### image

 (ligne 38) : définit une image sans bordure (border=0") dont le fichier source est /images/cerisier.jpg sur le serveur Web (src="/images/cerisier.jpg"). Ce lien se trouve sur un document Web obtenu avec l'URL <http://localhost:port/html/balises.htm>. Aussi, le navigateur demandera-t-il l'URL <http://localhost:port/images/cerisier.jpg> pour avoir l'image référencée ici.

### lien

<a href="http://istia.univ-angers.fr">ici</a> (ligne 42) : fait que le texte *ici* sert de lien vers l'URL <http://istia.univ-angers.fr>.

fond de page

`<body style="height:400px; width:400px; background-image:url(images/standard.jpg)">`  
(ligne 8) : indique que l'image qui doit servir de fond de page se trouve à l'URL /images/standard.jpg du serveur Web. Dans le contexte de notre exemple, le navigateur demandera l'URL `http://localhost:port/images/standard.jpg` pour obtenir cette image de fond. Par ailleurs, le corps du document sera affiché dans un rectangle de hauteur 400 pixels et de largeur 400 pixels.

On voit dans ce simple exemple que pour construire l'intégralité du document, le navigateur doit faire trois requêtes au serveur :

1. `http://localhost:port/html/balises.htm` pour avoir le source HTML du document
2. `http://localhost:port/images/cerisier.jpg` pour avoir l'image `cerisier.jpg`
3. `http://localhost:port/images/standard.jpg` pour obtenir l'image de fond `standard.jpg`

### 1.2.5.2 Un formulaire HTML

L'exemple suivant présente un formulaire :

Etes-vous marié(e)  Oui  Non

Cases à cocher  1  2  3

Champ de saisie qqs mots

Mot de passe .....

Boîte de saisie ligne1  
ligne2

combo choix2

liste à choix simple liste1  
liste2  
liste3  
liste4

liste à choix multiple liste1  
liste2  
liste3  
liste4

bouton Effacer

envoyer Envoyer

rétablissement Rétablir

Le code HTML produisant cet affichage est le suivant :

```
1. <!DOCTYPE html>
2. <html xmlns="http://www.w3.org/1999/xhtml">
3. <head>
4.   <meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
5.   <title>formulaire</title>
6.   <script type="text/javascript">
7.     function effacer() {
8.       alert("Vous avez cliqué sur le bouton Effacer");
9.     }
10.  </script>
11. </head>
```

```

12.
13. <body style="height: 400px; width: 400px; background-image: url(images/standard.jpg)">
14.   <h1 style="text-align: center">Formulaire HTML</h1>
15.   <form method="post" action="FormulairePost.aspx">
16.     <table border="0">
17.       <tr>
18.         <td>Etes-vous marié(e)</td>
19.         <td>
20.           <input type="radio" value="Oui" name="R1" />Oui
21.           <input type="radio" name="R1" value="non" checked="checked" />Non
22.         </td>
23.       </tr>
24.       <tr>
25.         <td>Cases à cocher</td>
26.         <td>
27.           <input type="checkbox" name="C1" value="un" />1
28.           <input type="checkbox" name="C2" value="deux" checked="checked" />2
29.           <input type="checkbox" name="C3" value="trois" />3
30.         </td>
31.       </tr>
32.       <tr>
33.         <td>Champ de saisie</td>
34.         <td>
35.           <input type="text" name="txtSaisie" size="20" value="qqq mots" />
36.         </td>
37.       </tr>
38.       <tr>
39.         <td>Mot de passe</td>
40.         <td>
41.           <input type="password" name="txtMdp" size="20" value="unMotDePasse" />
42.         </td>
43.       </tr>
44.       <tr>
45.         <td>Boîte de saisie</td>
46.         <td>
47.           <textarea rows="2" name="areaSaisie" cols="20">
48. ligne1
49. ligne2
50. ligne3
51. </textarea>
52.         </td>
53.       </tr>
54.       <tr>
55.         <td>combo</td>
56.         <td>
57.           <select size="1" name="cmbValeurs">
58.             <option value="1">choix1</option>
59.             <option selected="selected" value="2">choix2</option>
60.             <option value="3">choix3</option>
61.           </select>
62.         </td>
63.       </tr>
64.       <tr>
65.         <td>liste à choix simple</td>
66.         <td>
67.           <select size="3" name="lst1">
68.             <option selected="selected" value="1">liste1</option>
69.             <option value="2">liste2</option>
70.             <option value="3">liste3</option>
71.             <option value="4">liste4</option>
72.             <option value="5">liste5</option>
73.           </select>
74.         </td>
75.       </tr>
76.       <tr>
77.         <td>liste à choix multiple</td>
78.         <td>
79.           <select size="3" name="lst2" multiple="multiple">
80.             <option value="1" selected="selected">liste1</option>
81.             <option value="2">liste2</option>
82.             <option selected="selected" value="3">liste3</option>
83.             <option value="4">liste4</option>
84.             <option value="5">liste5</option>
85.           </select>
86.         </td>
87.       </tr>
88.       <tr>
89.         <td>bouton</td>
90.         <td>
91.           <input type="button" value="Effacer" name="cmdEffacer" onclick="effacer()" />
92.         </td>
93.       </tr>
94.       <tr>
95.         <td>envoyer</td>
96.         <td>
97.           <input type="submit" value="Envoyer" name="cmdEnvoyer" />
98.         </td>

```

```

99.      </tr>
100.     <tr>
101.       <td>r  tablir</td>
102.       <td>
103.         <input type="reset" value="R  tablir" name="cmdR  tablir" />
104.       </td>
105.     </tr>
106.   </table>
107.   <input type="hidden" name="secret" value="uneValeur" />
108. </form>
109. </body>
110. </html>

```

L'association contr  le visuel <--> balise HTML est le suivant :

Contr��le	balise HTML
formulaire	<form method="post" action="...">
champ de saisie	<input type="text" name="txtSaisie" size="20" value="qq s mots" />
champ de saisie cach��e	<input type="password" name="txtMdp" size="20" value="unMotDePasse" />
champ de saisie multilignes	<textarea rows="2" name="areaSaisie" cols="20"> ligne1 ligne2 ligne3</textarea>
boutons radio	<input type="radio" value="Oui" name="R1" />Oui <input type="radio" name="R1" value="non" checked="checked" />Non
cases �� cocher	<input type="checkbox" name="C1" value="un" />1 <input type="checkbox" name="C2" value="deux" checked="checked" />2 <input type="checkbox" name="C3" value="trois" />3
Combo	<select size="1" name="cmbValeurs"> <option value="1">choix1</option> <option selected="selected" value="2">choix2</option> <option value="3">choix3</option></select>
liste �� s��lection unique	<select size="3" name="lst1"> <option selected="selected" value="1">liste1</option> <option value="2">liste2</option> <option value="3">liste3</option> <option value="4">liste4</option> <option value="5">liste5</option></select>
liste �� s��lection multiple	<select size="3" name="lst2" multiple="multiple"> <option value="1">liste1</option> <option value="2">liste2</option> <option selected="selected" value="3">liste3</option> <option value="4">liste4</option> <option value="5">liste5</option></select>
bouton de type submit	<input type="submit" value="Envoyer" name="cmdRenvoyer" />
bouton de type reset	<input type="reset" value="R��tablir" name="cmdR��tablir" />
bouton de type button	<input type="button" value="Effacer" name="cmdEffacer" onclick="effacer()" />

Passons en revue ces diff  rentes balises :

### 1.2.5.2.1 Le formulaire

```
formulaire <form method="post" action="FormulairePost.aspx">
```

balise HTML <form name="..." method="..." action="...">>...</form>

attributs name="frmexemple" : nom du formulaire

method="..." : méthode utilisée par le navigateur pour envoyer au serveur Web les valeurs récoltées dans le formulaire

action="..." : URL à laquelle seront envoyées les valeurs récoltées dans le formulaire.

Un formulaire Web est entouré des balises <form>...</form>. Le formulaire peut avoir un nom (*name*="xx"). C'est le cas pour tous les contrôles qu'on peut trouver dans un formulaire. Le but d'un formulaire est de rassembler des informations données par l'utilisateur au clavier/souris et d'envoyer celles-ci à une URL de serveur Web. Laquelle ? Celle référencée dans l'attribut *action*="URL". Si cet attribut est absent, les informations seront envoyées à l'URL du document dans lequel se trouve le formulaire. Un client Web peut utiliser deux méthodes différentes appelées POST et GET pour envoyer des données à un serveur web. L'attribut *method*="métode", avec *method* égal à GET ou POST, de la balise <form> indique au navigateur la méthode à utiliser pour envoyer les informations recueillies dans le formulaire à l'URL précisée par l'attribut *action*="URL". Lorsque l'attribut *method* n'est pas précisé, c'est la méthode GET qui est prise par défaut.

### 1.2.5.2.2 Les champs de saisie texte

champ de saisie <input type="text" name="txtSaisie" size="20" value="qqz mots" />

<input type="password" name="txtMdp" size="20" value="unMotDePasse" />

Champ de saisie

Mot de passe

balise HTML <input type="..." name="..." size=".." value=".."/>

La balise **input** existe pour divers contrôles. C'est l'attribut *type* qui permet de différencier ces différents contrôles entre eux.

attributs type="text" : précise que c'est un champ de saisie

type="password" : les caractères présents dans le champ de saisie sont remplacés par des caractères \*. C'est la seule différence avec le champ de saisie normal. Ce type de contrôle convient pour la saisie des mots de passe.

size="20" : nombre de caractères visibles dans le champ - n'empêche pas la saisie de davantage de caractères

name="txtSaisie" : nom du contrôle

value="qqz mots" : texte qui sera affiché dans le champ de saisie.

### 1.2.5.2.3 Les champs de saisie multilignes

champ de saisie multilignes <textarea rows="2" name="areaSaisie" cols="20">  
ligne1  
ligne2  
ligne3</textarea>

Boîte de saisie

**balise HTML** <textarea ...>texte</textarea>

**attributs**

- affiche une zone de saisie multilignes avec au départ **texte** dedans
- rows="2"** : nombre de lignes
- cols="20"** : nombre de colonnes
- name="areaSaisie"** : nom du contrôle

#### 1.2.5.2.4 Les boutons radio

**boutons radio**

```
<input type="radio" value="Oui" name="R1" />Oui
<input type="radio" name="R1" value="non" checked="checked" />Non
```

Etes-vous marié(e)  Oui  Non

**balise HTML** <input type="radio" attribut2="valeur2" ....>texte

**attributs**

- affiche un bouton radio avec **texte** à côté.
- name="radio"** : nom du contrôle. Les boutons radio portant le même nom forment un groupe de boutons exclusifs les uns des autres : on ne peut cocher que l'un d'eux.
- value="valeur"** : valeur affectée au bouton radio. Il ne faut pas confondre cette valeur avec le texte affiché à côté du bouton radio. Celui-ci n'est destiné qu'à l'affichage.
- checked= "checked"** : si ce mot clé est présent, le bouton radio est coché, sinon il ne l'est pas.

#### 1.2.5.2.5 Les cases à cocher

**cases à cocher**

```
<input type="checkbox" name="C1" value="un" />1
<input type="checkbox" name="C2" value="deux" checked="checked" />2
<input type="checkbox" name="C3" value="trois" />3
```

Cases à cocher  1  2  3

**balise HTML** <input type="checkbox" attribut2="valeur2" ....>texte

**attributs**

- affiche une case à cocher avec **texte** à côté.
- name="C1"** : nom du contrôle. Les cases à cocher peuvent porter ou non le même nom. Les cases portant le même nom forment un groupe de cases associées.
- value="valeur"** : valeur affectée à la case à cocher. Il ne faut pas confondre cette valeur avec le texte affiché à côté du bouton radio. Celui-ci n'est destiné qu'à l'affichage.
- checked= "checked"** : si ce mot clé est présent, la case à cocher est cochée, sinon elle ne l'est pas.

#### 1.2.5.2.6 La liste déroulante (combo)

**Combo**

```
<select size="1" name="cmbValeurs">
  <option value="1">choix1</option>
  <option selected="selected" value="2">choix2</option>
  <option value="3">choix3</option>
</select>
```

combo

balise HTML

```
<select size=".." name="..">
    <option [selected="selected"] value="v">...</option>
    ...
</select>
```

attributs

affiche dans une liste les textes compris entre les balises `<option>...</option>`  
**name="cmbValeurs"** : nom du contrôle.

**size="1"** : nombre d'éléments de liste visibles. `size="1"` fait de la liste l'équivalent d'un combobox.

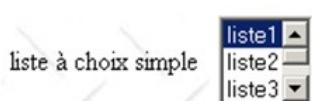
**selected="selected"** : si ce mot clé est présent pour un élément de liste, ce dernier apparaît sélectionné dans la liste. Dans notre exemple ci-dessus, l'élément de liste `choix2` apparaît comme l'élément sélectionné du combo lorsque celui-ci est affiché pour la première fois.

**value="v"** : si l'élément est sélectionné par l'utilisateur, c'est cette valeur `[v]` qui est postée au serveur. En l'absence de cet attribut, c'est le texte affiché et sélectionné qui est posté au serveur.

### 1.2.5.2.7 Liste à sélection unique

liste à sélection unique

```
<select size="3" name="Lst1">
    <option selected="selected" value="1">liste1</option>
    <option value="2">liste2</option>
    <option value="3">liste3</option>
    <option value="4">liste4</option>
    <option value="5">liste5</option>
</select>
```



balise HTML

```
<select size=".." name="..">
    <option [selected="selected"]>...</option>
    ...
</select>
```

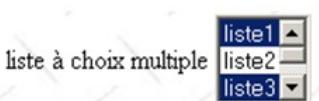
attributs

affiche dans une liste les textes compris entre les balises `<option>...</option>`  
les mêmes que pour la liste déroulante n'affichant qu'un élément. Ce contrôle ne diffère de la liste déroulante précédente que par son attribut `size>1`.

### 1.2.5.2.8 Liste à sélection multiple

liste à sélection unique

```
<select size="3" name="lst2" multiple="multiple">
    <option value="1" selected="selected">liste1</option>
    <option value="2">liste2</option>
    <option selected="selected" value="3">liste3</option>
    <option value="4">liste4</option>
    <option value="5">liste5</option>
</select>
```



balise HTML      <select size=".." name=".." multiple="multiple">  
                   <option [selected="selected"]>...</option>  
                   ...  
                   </select>

attributs      affiche dans une liste les textes compris entre les balises <option>...</option>  
                   multiple : permet la sélection de plusieurs éléments dans la liste. Dans l'exemple ci-dessus, les éléments *liste1* et *liste3* sont tous deux sélectionnés.

### 1.2.5.2.9 Bouton de type button

bouton de type  
button      <input type="button" value="Effacer" name="cmdEffacer" onclick="effacer()" />



balise HTML      <input type="button" value="..." name="..." onclick="effacer() ..../>

attributs      type="button" : définit un contrôle bouton. Il existe deux autres types de bouton, les types *submit* et *reset*.

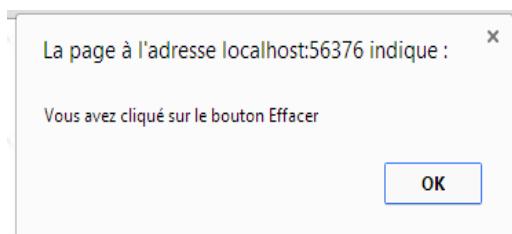
value="Effacer" : le texte affiché sur le bouton

onclick="fonction()" : permet de définir une fonction à exécuter lorsque l'utilisateur clique sur le bouton. Cette fonction fait partie des scripts définis dans le document Web affiché. La syntaxe précédente est une syntaxe *javascript*. Si les scripts sont écrits en *vbscript*, il faudrait écrire onclick="fonction" sans les parenthèses. La syntaxe devient identique s'il faut passer des paramètres à la fonction : onclick="fonction(val1, val2,...)"

Dans notre exemple, un clic sur le bouton *Effacer* appelle la fonction javascript *effacer* suivante :

```
<script type="text/javascript">
  function effacer() {
    alert("Vous avez cliqué sur le bouton Effacer");
  }
</script>
```

La fonction *effacer* affiche un message :



### 1.2.5.2.10 Bouton de type submit

bouton de type  
submit      <input type="submit" value="Envoyer" name="cmdRenvoyer" />



balise HTML      <input type="submit" value="Envoyer" name="cmdRenvoyer" />

**attributs**

**type="submit"** : définit le bouton comme un bouton d'envoi des données du formulaire au serveur Web. Lorsque le client va cliquer sur ce bouton, le navigateur va envoyer les données du formulaire à l'URL définie dans l'attribut **action** de la balise **<form>** selon la méthode définie par l'attribut **method** de cette même balise.  
**value="Envoyer"** : le texte affiché sur le bouton

### 1.2.5.2.11 Bouton de type reset

**bouton de type reset**

```
<input type="reset" value="Rétablir" name="cmdRétablir" />
```



**balise HTML**

```
<input type="reset" value="Rétablir" name="cmdRétablir"/>
```

**attributs**

**type="reset"** : définit le bouton comme un bouton de réinitialisation du formulaire. Lorsque le client va cliquer sur ce bouton, le navigateur va remettre le formulaire dans l'état où il l'a reçu.  
**value="Rétablir"** : le texte affiché sur le bouton

### 1.2.5.2.12 Champ caché

**champ caché**

```
<input type="hidden" name="secret" value="uneValeur" />
```

**balise HTML**

```
<input type="hidden" name="..." value="..."/>
```

**attributs**

**type="hidden"** : précise que c'est un champ caché. Un champ caché fait partie du formulaire mais n'est pas présenté à l'utilisateur. Cependant, si celui-ci demandait à son navigateur l'affichage du code source, il verrait la présence de la balise **<input type="hidden" value="...">** et donc la valeur du champ caché.

**value="uneValeur"** : valeur du champ caché.

Quel est l'intérêt du champ caché ? Cela peut permettre au serveur Web de garder des informations au fil des requêtes d'un client. Considérons une application d'achats sur le Web. Le client achète un premier article *art1* en quantité *q1* sur une première page d'un catalogue puis passe à une nouvelle page du catalogue. Pour se souvenir que le client a acheté *q1* articles *art1*, le serveur peut mettre ces deux informations dans un champ caché du formulaire Web de la nouvelle page. Sur cette nouvelle page, le client achète *q2* articles *art2*. Lorsque les données de ce second formulaire vont être envoyées au serveur (submit), celui-ci va non seulement recevoir l'information (*q2,art2*) mais aussi (*q1,art1*) qui fait partie également partie du formulaire en tant que champ caché. Le serveur Web va alors mettre dans un nouveau champ caché les informations (*q1,art1*) et (*q2,art2*) et envoyer une nouvelle page de catalogue. Et ainsi de suite.

## 1.2.5.3 Envoi à un serveur Web par un client Web des valeurs d'un formulaire

Nous avons dit dans l'étude précédente que le client Web disposait de deux méthodes pour envoyer à un serveur Web les valeurs d'un formulaire qu'il a affiché : les méthodes GET et POST. Voyons sur un exemple la différence entre les deux méthodes.

### 1.2.5.3.1 Méthode GET

Faisons un premier test, où dans le code HTML du document, la balise **<form>** est définie de la façon suivante :

```
<form method="get" action="FormulaireGet.aspx">
```

The left side shows a web browser window with the URL `localhost:56376/HtmlFormulaire.html`. The page displays an HTML form with various input fields: a radio button group ('Etes-vous marié(e)'), a checkbox group ('Cases à cocher'), a text input ('Champ de saisie'), a password input ('Mot de passe'), a dropdown menu ('Boîte de saisie'), a select dropdown ('combo'), a single-select dropdown ('liste à choix simple'), a multiple-select dropdown ('liste à choix multiple'), a button ('bouton'), and two submit buttons ('Envoyer' and 'Rétablir'). A red box labeled [1] highlights the 'Envoyer' button.

The right side shows a Windows File Explorer window for a project named 'Exemple-00'. It lists files like 'Cerisier.jpg', 'standard.jpg', 'Balises.html', 'FormulaireGet.aspx' (which has a red box labeled [2]), 'FormulairePost.aspx', 'HtmlFormulaire.html', 'HtmlPage1.html', 'HtmlPage2.html', 'Web.config', and 'WebForm1.aspx'.

Lorsque l'utilisateur va cliquer sur le bouton [1], les valeurs saisies dans le formulaire vont être envoyées à la page ASP.NET [2]. Celle-ci ne fait rien de ces paramètres et renvoie une page vide. On veut seulement savoir comment le navigateur transmet les valeurs saisies au serveur web. Pour cela, nous allons utiliser un outil de débogage disponible dans Chrome. On l'active en tapant CTRL-I (majuscule) [3] :

The screenshot shows the Network tab of the Chrome DevTools developer console. The tab bar is highlighted with a red box. Below it is a table with columns: Name, Method, Status, Type, and Initiator. There are no entries in the table at this point.

Comme nous nous intéressons aux échanges réseau entre le navigateur et le serveur web, nous activons ci-dessus l'onglet [Network] puis nous cliquons sur le bouton [Envoyer] du formulaire. Celui-ci est un bouton de type [submit] à l'intérieur d'une balise [form]. Le navigateur réagit au clic en demandant l'URL [FormulaireGet.aspx] indiquée dans l'attribut [action] de la balise [form], avec la méthode GET indiquée dans l'attribut [method]. Nous obtenons alors les informations suivantes :

The screenshot shows the Network tab of the Chrome DevTools developer console with a single entry. The request details are as follows:

- Request URL:** `http://localhost:56376/FormulaireGet.aspx?R1=non&C2=deux&txtSaisie=qqs+mots&txtMdp=unMotDePasse&areaSaisie=ligne1%0D%0Aligne2%0D%0Aligne3%0D%0A&cmbValeurs=2&lst1=1&lst2=1&lst3=3&cmdRenoyer=Envoyer&secret=uneValeur`
- Method:** GET
- Status:** 200 OK
- Type:** text/html
- Size:** 1.1 KB (868 B)
- Time:** 6 ms (5 ms)

La copie d'écran ci-dessus nous montre l'URL demandée par le navigateur à l'issue du clic sur le bouton [envoyer]. Il demande bien l'URL prévue [FormulaireGet.aspx] mais derrière il rajoute des informations qui sont les valeurs saisies dans le formulaire. Pour avoir plus d'informations, nous cliquons sur le lien ci-dessus :

Request URL: http://localhost:56376/FormulaireGet.aspx?R1=Envoyer&secret=uneValeur  
envoyer=Envoyer&secret=uneValeur  
Request Method: GET  
Status Code: 200 OK

**Request Headers** view source [3]  
 Accept: text/html,application/xhtml+xml,application/xml  
 Accept-Encoding: gzip,deflate,sdch  
 Accept-Language: fr-FR,fr;q=0.8,en-US;q=0.6,en;q=0.4  
 Connection: keep-alive [2]  
 Host: localhost:56376  
 Referer: http://localhost:56376/HtmlFormulaire.html  
 User-Agent: Mozilla/5.0 (Windows NT 6.2; WOW64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/38.0.2125.111 Safari/537.36

**Query String Parameters** view source view URL encoded  
R1: Oui

**Request Headers** view parsed  
 GET /FormulaireGet.aspx?R1=Oui&C1=un&C2=deux&txtSaisie=programmation+web&txtMdp=ceciestsecret&areaSaisie=les+bases+de+la%0D%0Aprogrammation+web&cmbValeurs=3&lst1=3&lst2=1&lst3=3&cmdRenvoyer=Envoyer&secret=uneValeur HTTP/1.1  
 Host: localhost:56376 [4]  
 Connection: keep-alive  
 Accept: text/html,application/xhtml+xml,application/xml  
 User-Agent: Mozilla/5.0 (Windows NT 6.2; WOW64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/38.0.2125.111 Safari/537.36  
 Accept-Encoding: gzip,deflate,sdch  
 Accept-Language: fr-FR,fr;q=0.8,en-US;q=0.6,en;q=0.4

Ci-dessus [1, 2], nous voyons les entêtes HTTP envoyés par le navigateur. Ils ont été ici mis en forme. Pour voir le texte brut de ces entêtes, nous suivons le lien [view source] [3, 4]. Le texte complet est le suivant :

```

1. GET /FormulaireGet.aspx?R1=Oui&C1=un&C2=deux&txtSaisie=programmation+web&txtMdp=ceciestsecret&areaSaisie=les+bases+de+la%0D%0Aprogrammation+web&cmbValeurs=3&lst1=3&lst2=1&lst3=3&cmdRenvoyer=Envoyer&secret=uneValeur HTTP/1.1
2. Host: localhost:56376
3. Connection: keep-alive
4. Pragma: no-cache
5. Cache-Control: no-cache
6. Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8
7. User-Agent: Mozilla/5.0 (Windows NT 6.3; WOW64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/38.0.2125.111 Safari/537.36
8. Referer: http://localhost:56376/HtmlFormulaire.html
9. Accept-Encoding: gzip,deflate,sdch
10. Accept-Language: fr-FR,fr;q=0.8,en-US;q=0.6,en;q=0.4
  
```

Nous retrouvons des éléments déjà rencontrés précédemment. D'autres apparaissent pour la première fois :

**Connection: keep-alive** le client demande au serveur de ne pas fermer la connexion après sa réponse. Cela lui permettra d'utiliser la même connexion pour une demande ultérieure. La connexion ne reste pas ouverte indéfiniment. Le serveur la fermera après un trop long délai d'inutilisation.

**Referer** l'URL qui était affichée dans le navigateur lorsque la nouvelle demande a été faite.

La nouveauté est ligne 1 dans les informations qui suivent l'URL. On constate que les choix faits dans le formulaire se retrouvent dans l'URL. Les valeurs saisies par l'utilisateur dans le formulaire ont été passées dans la commande *GET URL?param1=valeur1&param2=valeur2&... HTTP/1.1* où les *parami* sont les noms (attribut *name*) des contrôles du formulaire Web et *valeuri* les valeurs qui leur sont associées. Nous présentons ci-dessous un tableau à trois colonnes :

- colonne 1 : reprend la définition d'un contrôle HTML de l'exemple ;
- colonne 2 : donne l'affichage de ce contrôle dans un navigateur ;
- colonne 3 : donne la valeur envoyée au serveur par le navigateur pour le contrôle de la colonne 1 sous la forme qu'elle a dans la requête GET de l'exemple.

contrôle HTML	visuel	valeur(s) renvoyée(s)
<input type="radio" value="Oui" name="R1"/>Oui	Etes-vous marié(e) <input checked="" type="radio"/> Oui <input type="radio"/> Non	R1=Oui
<input type="radio" name="R1" value="non" checked="checked"/>Non		- la valeur de l'attribut <i>value</i> du bouton radio coché par l'utilisateur.
<input type="checkbox" name="C1" value="un"/>1	Cases à cocher <input checked="" type="checkbox"/> 1 <input checked="" type="checkbox"/> 2 <input type="checkbox"/> 3	C1=un
<input type="checkbox" name="C2" value="deux" checked="checked"/>2		C2=deux
<input type="checkbox" name="C3" value="trois"/>3		
<input type="text" name="txtSaisie" size="20" value="qqq mots"/>	Champ de saisie <input type="text" value="programmation web"/>	txtSaisie=programmation+Web

- valeurs des attributs *value* des cases cochées par l'utilisateur

- texte tapé par l'utilisateur dans le champ de saisie. Les espaces ont été remplacés

```
<input type="password" name="txtMdp" size="20" value="unMotDePasse"/>
```

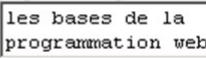
Mot de passe



par le signe +

```
<textarea rows="2" name="areaSaisie" cols="20">  
ligne1  
ligne2  
ligne3  
</textarea>
```

Boîte de saisie



```
les bases de la  
programmation web
```

```
<select size="1" name="cmbValeurs">  
<option value='1'>choix1</option>  
<option selected="selected" value='2'>choix2</option>  
<option value='3'>choix3</option>  
</select>  
<select size="3" name="lst1">  
<option selected="selected" value='1'>liste1</option>  
<option value='2'>liste2</option>  
<option value='3'>liste3</option>  
<option value='4'>liste4</option>  
<option value='5'>liste5</option>  
</select>  
<select size="3" name="lst2" multiple="multiple">  
<option selected="selected" value='1'>liste1</option>  
<option value='2'>liste2</option>  
<option selected="selected" value='3'>liste3</option>  
<option value='4'>liste4</option>  
<option value='5'>liste5</option>  
</select>  
<input type="submit" value="Envoyer!" name="cmdRenvoyer" />
```

combo



liste à choix simple



```
liste1  
liste2  
liste3
```

liste à choix multiple



```
liste1  
liste2  
liste3
```

```
<input type="hidden" name="secret" value="uneValeur"/>
```

### 1.2.5.3.2 Méthode POST

Nous changeons le document HTML pour que le navigateur utilise maintenant la méthode POST pour envoyer les valeurs du formulaire au serveur Web :

```
<form method="post" action="FormulairePost.aspx">
```

Nous remplissons le formulaire tel que pour la méthode GET et nous transmettons les paramètres au serveur avec le bouton [Envoyer]. Comme il a été fait au paragraphe précédent page 40, nous avons accès dans Chrome aux entêtes HTTP de la requête envoyée par le navigateur :

**txtMdp=ceciestsecret**

- texte tapé par l'utilisateur dans le champ de saisie

**areaSaisie=les+bases+de+la%0D%0A  
programmation+Web**

- texte tapé par l'utilisateur dans le champ de saisie. %OD%OA est la marque de fin de ligne. Les espaces ont été remplacés par le signe +

**cmbValeurs=3**

- attribut [value] de l'élément sélectionné par l'utilisateur

**lst1=3**

- attribut [value] de l'élément sélectionné par l'utilisateur

**lst2=1**

**lst2=3**

- attributs [value] des éléments sélectionnés par l'utilisateur

**cmdRenvoyer=Envoyer**

- nom et attribut *value* du bouton qui a servi à envoyer les données du formulaire au serveur

**secret=uneValeur**

- attribut *value* du champ caché

```

1. POST /FormulairePost.aspx HTTP/1.1
2. Host: localhost:56376
3. Connection: keep-alive
4. Content-Length: 195
5. Pragma: no-cache
6. Cache-Control: no-cache
7. Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8
8. Origin: http://localhost:56376
9. User-Agent: Mozilla/5.0 (Windows NT 6.3; WOW64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/38.0.2125.111 Safari/537.36
10. Content-Type: application/x-www-form-urlencoded
11. Referer: http://localhost:56376/HtmlFormulaire.html
12. Accept-Encoding: gzip,deflate
13. Accept-Language: fr-FR,fr;q=0.8,en-US;q=0.6,en;q=0.4
14.
15. R1=Oui&C1=un&C2=deux&txtSaisie=programmation+web&txtMdp=unMotDePasse&areaSaisie=les+bases+de+la%0D%0Aprogrammation+web%0D%0A&cmbValeurs=3&lst1=3&lst2=1&lst2=3&cmdRenvoyer=Envoyer&secret=uneValeur

```

Des nouveautés apparaissent dans la requête HTTP du client :

#### POST URL HTTP/1.1

la requête GET a laissé place à une requête POST. Les paramètres ne sont plus présents dans cette première ligne de la requête. On peut constater qu'ils sont maintenant placés (ligne 14) derrière la requête HTTP après une ligne vide. Leur encodage est identique à celui qu'ils avaient dans la requête GET.

#### Content-Length

nombre de caractères "postés", c.a.d. le nombre de caractères que devra lire le serveur Web après avoir reçu les entêtes HTTP pour récupérer le document que lui envoie le client. Le document en question est ici la liste des valeurs du formulaire.

#### Content-type

précise le type du document que le client enverra après les entêtes HTTP. Le type [application/x-www-form-urlencoded] indique que c'est un document contenant des valeurs de formulaire.

Il y a deux méthodes pour transmettre des données à un serveur Web : GET et POST. Y-a-t-il une méthode meilleure que l'autre ? Nous avons vu que si les valeurs d'un formulaire étaient envoyées par le navigateur avec la méthode GET, le navigateur affichait dans son champ *Adresse* l'URL demandée sous la forme *URL?param1=val1&param2=val2*.... On peut voir cela comme un avantage ou un inconvénient :

- un avantage si on veut permettre à l'utilisateur de placer cette URL paramétrée dans ses liens favoris ;
- un inconvénient si on ne souhaite pas que l'utilisateur ait accès à certaines informations du formulaire tels, par exemple, les champs cachés.

Par la suite, nous utiliserons quasi exclusivement la méthode POST dans nos formulaires.

## 1.2.6 Conclusion

Ce chapitre a présenté différents concepts de base du développement Web :

- les échanges client-serveur via le protocole HTTP ;
- la conception d'un document à l'aide du langage HTML ;
- la conception de formulaires de saisie.

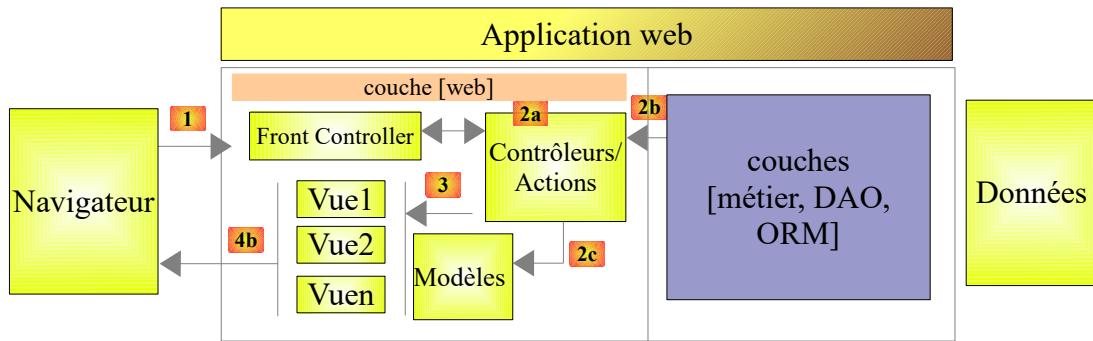
Nous avons pu voir sur un exemple comment un client pouvait envoyer des informations au serveur Web. Nous n'avons pas présenté comment le serveur pouvait

- récupérer ces informations ;
- les traiter ;
- envoyer au client une réponse dynamique dépendant du résultat du traitement.

C'est le domaine de la programmation Web, domaine que nous abordons dans le chapitre suivant avec la présentation de la technologie ASP.NET MVC.

## 1.3 Contrôleurs, Actions, Routage

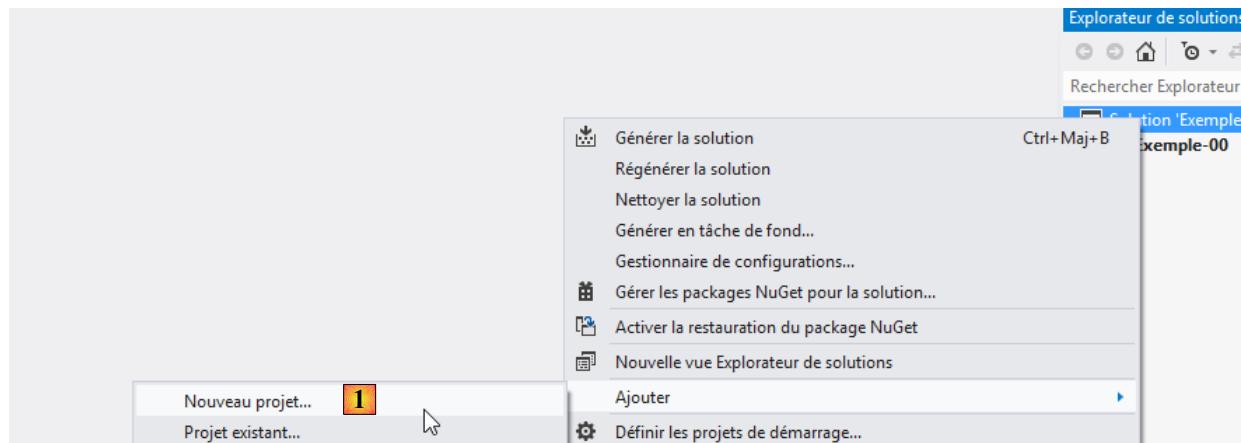
Considérons l'architecture d'une application ASP.NET MVC :

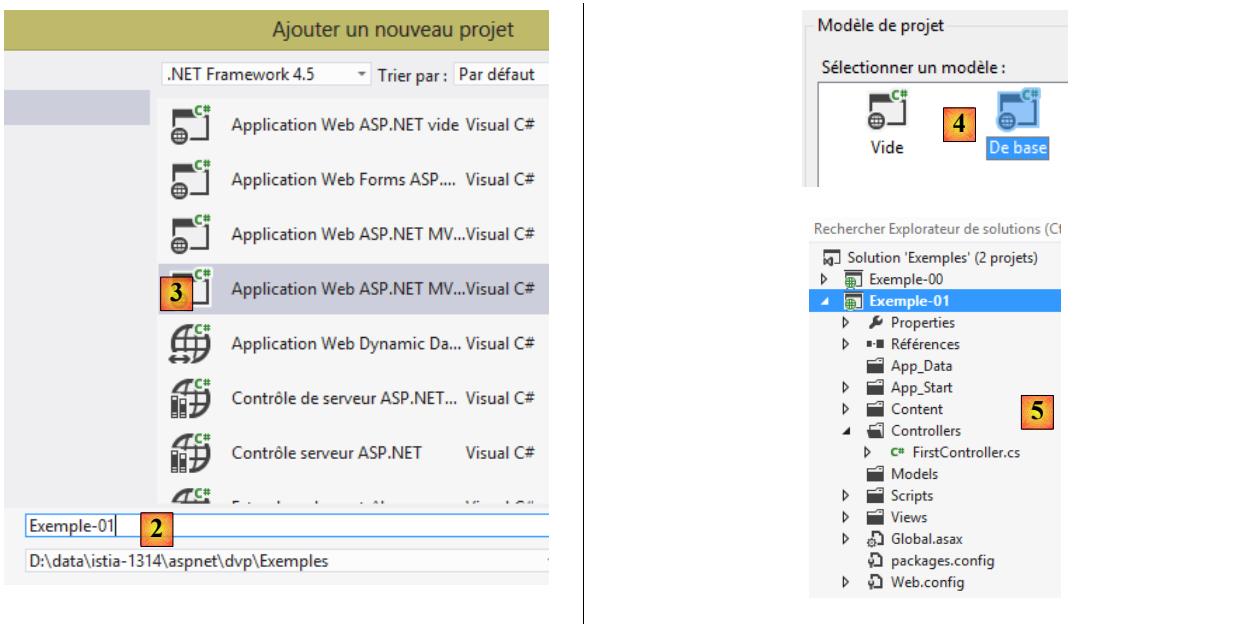


Dans ce chapitre, nous regardons le processus qui amène la requête [1] au contrôleur et à l'action [2a] qui vont la traiter, un mécanisme qu'on appelle le routage. Nous présentons par ailleurs les différentes réponses [3] que peut faire une action au navigateur. Ce peut être autre chose qu'une vue V [4b].

### 1.3.1 La structure d'un projet ASP.NET MVC

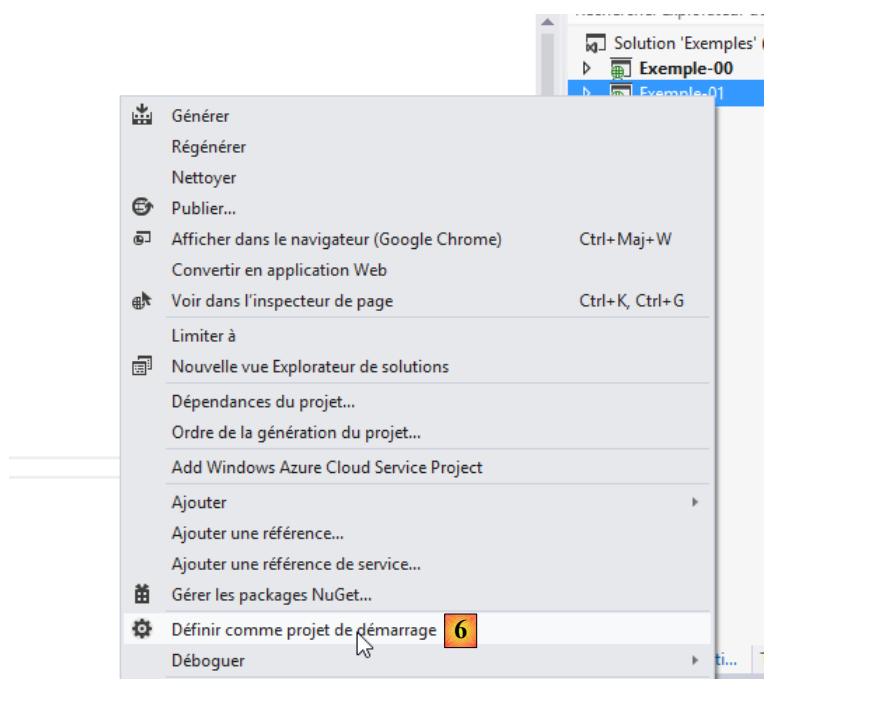
Construisons un premier projet ASP.NET MVC avec Visual Studio Express 2012. Nous allons l'ajouter [1] à la solution utilisée dans le chapitre précédent :





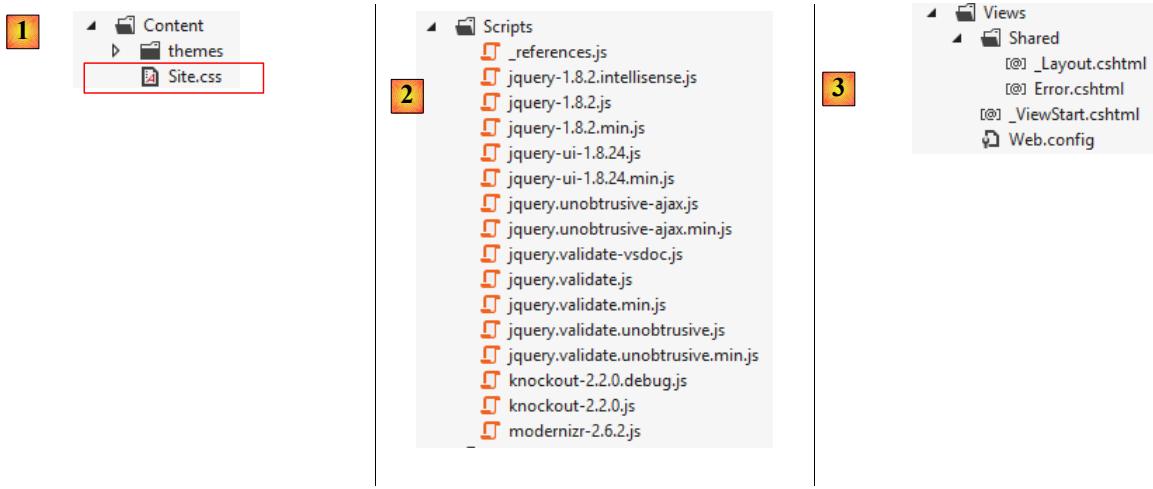
- en [2], le nom du nouveau projet ;
- en [3, 4], nous choisissons un projet ASP.NET MVC de base. Ce modèle nous fournit une application Web vide avec cependant toutes les ressources (DLL, bibliothèques javascript, ...) pour travailler.

Le projet résultant est présenté en [5]. On en fera [6] le projet de démarrage de la solution :



On notera en [5] les points suivants :

- l'architecture du projet reflète son modèle MVC :
  - les contrôleurs **C** seront placés dans le dossier [Controllers],
  - les modèles de données **M** seront placés dans le dossier [Models],
  - les vues **V** seront placées dans le dossier [Views],



- en [1], le fichier [Site.css] sera le fichier CSS par défaut de l'application ;
- en [2], un certain nombre de bibliothèques Javascript sont mises à notre disposition ;
- en [3], on trouve trois vues particulières : **\_ViewStart**, **\_Layout**, **Error**.

Le fichier [**\_ViewStart**] est le suivant :

```
1. @{
2.     Layout = "~/Views/Shared/_Layout.cshtml";
3. }
```

- ligne 1 : le caractère @ annonce une séquence C# dans la vue. En effet, on peut inclure du code C# dans une vue ;
- ligne 2 : définit une variable *Layout* qui définit la vue parente de toutes les vues. Elle correspond à la page maître du framework ASP.NET classique.

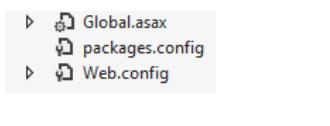
Le fichier [**\_Layout**] est le suivant :

```
1. <!DOCTYPE html>
2. <html>
3. <head>
4.     <meta charset="utf-8" />
5.     <meta name="viewport" content="width=device-width" />
6.     <title>@ViewBag.Title</title>
7.     @Styles.Render("~/Content/css")
8.     @Scripts.Render("~/bundles/modernizr")
9. </head>
10. <body>
11.     @RenderBody()
12.
13.     @Scripts.Render("~/bundles/jquery")
14.     @RenderSection("scripts", required: false)
15. </body>
16. </html>
```

Lorsqu'une vue du dossier [Views] sera rendue, son corps sera rendu par la ligne 11 ci-dessus. Cela signifie que la vue n'a pas à inclure les balises <html>, <head>, <body>. Celles-ci sont fournies par le fichier ci-dessus. Celui-ci est pour l'instant assez hermétique. Simplifions-le :

```
1. <!DOCTYPE html>
2. <html>
3. <head>
4.     <meta charset="utf-8" />
5.     <meta name="viewport" content="width=device-width" />
6.     <title>Tutoriel ASP.NET MVC</title>
7. </head>
8. <body>
9.     <h2>Tutoriel ASP.NET MVC</h2>
10.    @RenderBody()
11. </body>
12. </html>
```

- ligne 6 : le titre commun à toutes les vues ;
- ligne 9 : l'entête commun à toutes les vues ;
- ligne 10 : le contenu propre à la vue affichée.



- [Web.config] est le fichier de configuration de l'application Web. Il est complexe. On sera amené à le modifier lorsqu'on utilisera le framework [Spring.net] dans une architecture multicouche.
- [Global.asax] contient le code exécuté au démarrage de l'application. En général, ce code exploite les différents fichiers de configuration de l'application dont [Web.config].

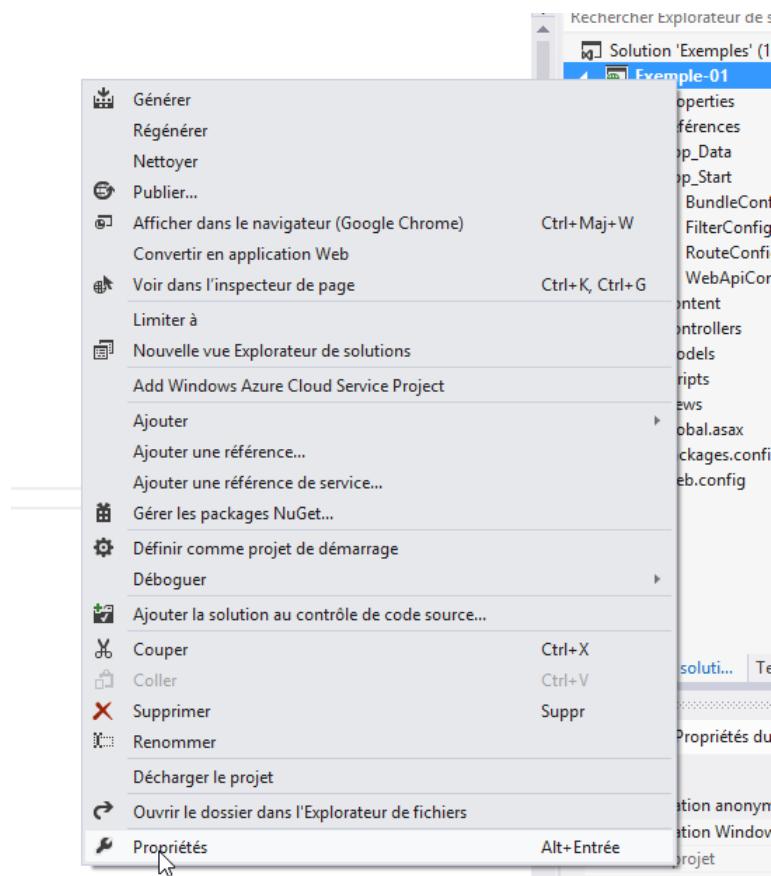
### 1.3.2 Le routage par défaut des URL

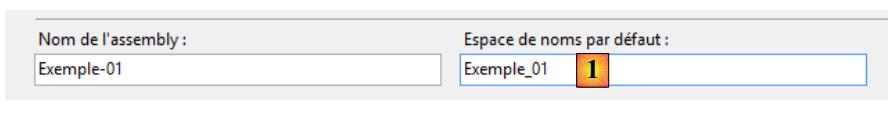
Le code de [Global.asax] est pour l'instant le suivant :

```

1.  using System.Web.Http;
2.  using System.Web.Mvc;
3.  using System.Web.Optimization;
4.  using System.Web.Routing;
5.
6.  namespace Exemple_01
7.  {
8.
9.      public class MvcApplication : System.Web.HttpApplication
10.     {
11.         protected void Application_Start()
12.         {
13.             ...
14.         }
15.     }
16. }
```

- ligne 6, le *namespace* de la classe. Il vient directement du nom du projet et est présent dans les propriétés du projet :





On fixe en [1], l'espace de noms par défaut. Il est alors utilisé par défaut pour toutes les classes qui seront créées dans le projet.

Revenons au code de [Global.asax] :

```

1.  using System.Web.Http;
2.  using System.Web.Mvc;
3.  using System.Web.Optimization;
4.  using System.Web.Routing;
5.
6.  namespace Exemple_01
7.  {
8.
9.      public class MvcApplication : System.Web.HttpApplication
10.     {
11.         protected void Application_Start()
12.         {
13.             ...
14.         }
15.     }
16. }
```

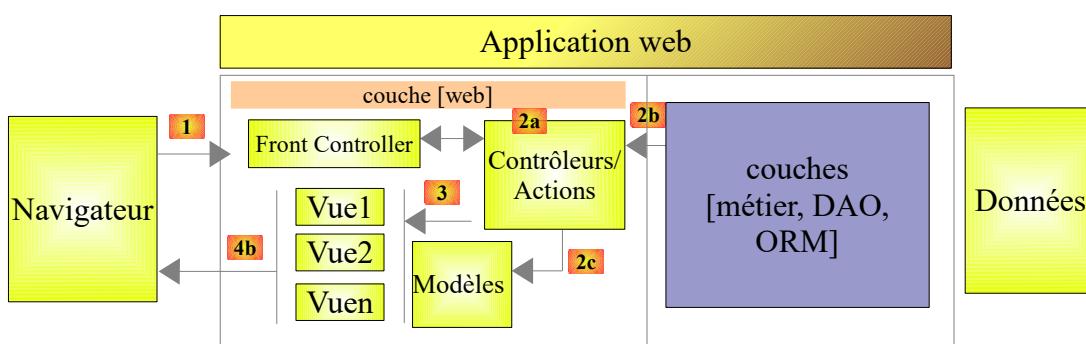
- ligne 9 : la classe [MvcApplication] dérive de la classe [HttpApplication]. Le nom [MvcApplication] peut être changé ;
- ligne 11 : la méthode [Application\_Start] est la méthode exécutée au démarrage de l'application Web. Elle n'est exécutée qu'une fois. C'est là qu'on initialise l'application.

Le code de [Application\_Start] est actuellement le suivant :

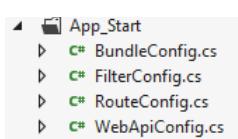
```

1.      protected void Application_Start()
2.      {
3.          AreaRegistration.RegisterAllAreas();
4.
5.          WebApiConfig.Register(GlobalConfiguration.Configuration);
6.          FilterConfig.RegisterGlobalFilters(GlobalFilters.Filters);
7.          RouteConfig.RegisterRoutes(RouteTable.Routes);
8.          BundleConfig.RegisterBundles(BundleTable.Bundles);
9.      }
```

Pour l'instant, on n'a pas besoin de comprendre l'intégralité de ce code. Les lignes 5-8 définissent les routes acceptées par l'application Web. Revenons à l'architecture d'une application ASP.NET MVC :



Nous avons expliqué que le [Front Controller] devait router une URL vers l'action chargée de la traiter. Une route sert à faire le lien entre un modèle d'URL et une action. Ces routes sont définies dans le dossier [App\_Start] du projet par les classes [WebApiConfig, FilterConfig, RouteConfig, BundleConfig] :



Seule la classe [RouteConfig] nous intéresse pour l'instant :

```
1.  using System.Web.Mvc;
2.  using System.Web.Routing;
3.
4.  namespace Exemple_01
5.  {
6.      public class RouteConfig
7.      {
8.          public static void RegisterRoutes(RouteCollection routes)
9.          {
10.              routes.IgnoreRoute("{resource}.axd/{*pathInfo}");
11.
12.              routes.MapRoute(
13.                  name: "Default",
14.                  url: "{controller}/{action}/{id}",
15.                  defaults: new { controller = "Home", action = "Index", id = UrlParameter.Optional }
16.              );
17.          }
18.      }
19.  }
```

Les lignes 12-16 définissent la forme des URL acceptées par l'application. On appelle cela une **route**. Il peut y avoir plusieurs routes possibles (= plusieurs modèles d'URL possibles). Elles se distinguent entre-elles par leur nom (ligne 13). La forme des URL de la route est définie ligne 14. Ici, l'URL aura trois composantes :

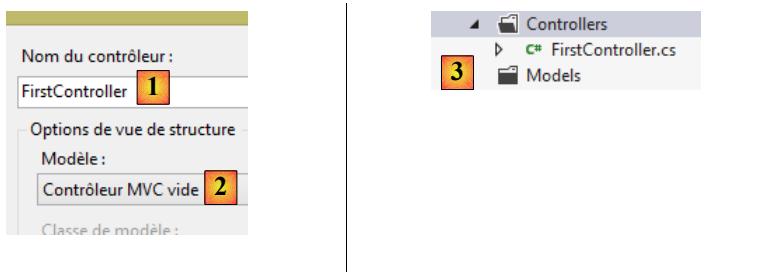
- **{controller}** : le nom d'une classe dérivée de [Controller]. Elle sera cherchée dans le dossier [Controllers] du projet. Par convention, si l'URL est /X/Y/Z, le contrôleur chargé de traiter cette URL sera la classe **XController**. Le suffixe **Controller** est ajouté au nom du contrôleur présent dans l'URL ;
- **{action}** : le nom d'une méthode dans le contrôleur désigné ci-dessus. C'est elle qui va recevoir les paramètres qui accompagnent l'URL et qui va les traiter. Cette méthode peut rendre divers résultats :
  - **void** : l'action va construire elle-même la réponse au navigateur client
  - **String** : l'action renvoie au client une chaîne de caractères ;
  - **ViewResult** : renvoie une vue au client ;
  - **PartialViewResult** : renvoie une vue partielle ;
  - **EmptyResult** : une réponse vide est envoyée au client ;
  - **RedirectResult** : demande au client de se rediriger vers une URL
  - **RedirectToRouteResult** : idem mais l'URL est construite à partir de routes de l'application ;
  - **JsonResult** : envoie une réponse JSON
  - **JavaScriptResult** : renvoie un code Javascript au client ;
  - **ContentResult** : renvoie un flux HTML au client sans passer par une vue ;
  - **FileContentResult** : renvoie un fichier au client ;
  - **FileStreamResult** : idem mais par une autre voie ;
  - **FilePathResult** : ...
- **{id}** : un paramètre qui sera transmis à l'action. Pour cela l'action devra avoir un paramètre nommé **id**.

La ligne 15 définit des valeurs par défaut lorsque l'URL n'a pas la forme attendue **/{controller}/{action}/{id}**. Elle indique également que le paramètre **{id}** dans l'URL est facultatif. Voici une liste d'URL incomplètes et l'URL complétée avec les valeurs par défaut :

URL d'origine	URL complétée
/	/Home/Index
/Do	/Do/Index
/Do/Something	/Do/Something
/Do/Something/4	/Do/Something/4
/Do/Something/x/y/z	URL non routée

### 1.3.3 Crédation d'un contrôleur et d'une première action

Créons un premier contrôleur :



- en [1] donnez le nom du contrôleur avec son suffixe [Controller] ;
- en [2], créez un contrôleur MVC vide ;
- en [3], il a été créé.

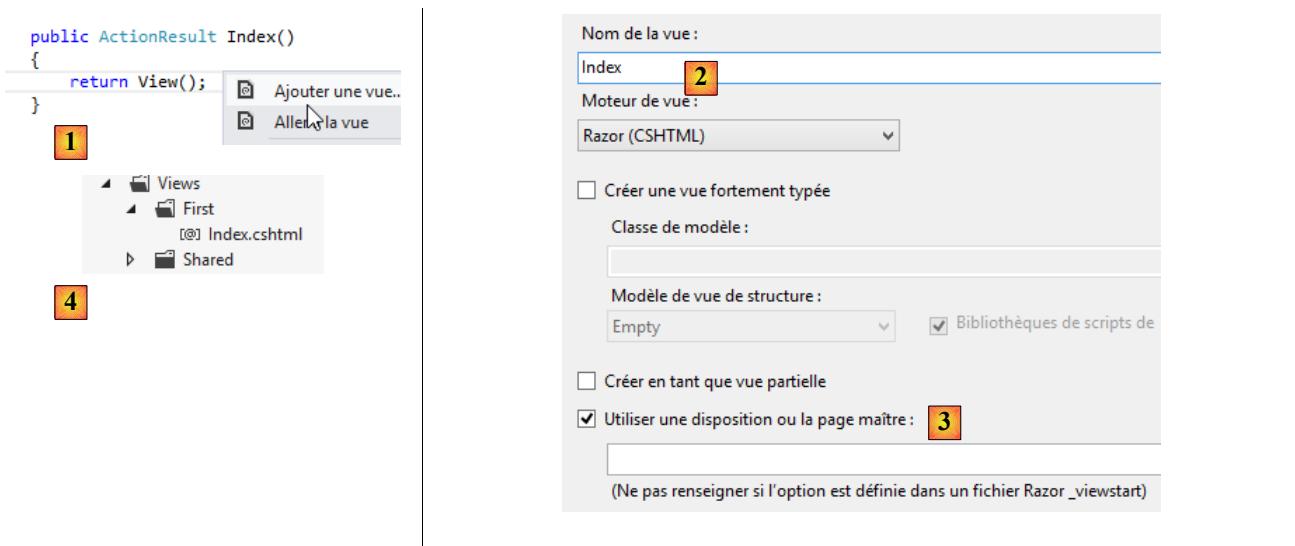
Le code de [FirstController] est le suivant :

```

1.  using System;
2.  using System.Collections.Generic;
3.  using System.Linq;
4.  using System.Web;
5.  using System.Web.Mvc;
6.
7.  namespace Exemple_01.Controllers
8.  {
9.      public class FirstController : Controller
10.     {
11.         //
12.         // GET: /First/
13.
14.         public ActionResult Index()
15.         {
16.             return View();
17.         }
18.
19.     }
20. }
```

- ligne 7 : un *namespace* par défaut a été généré ;
- ligne 9 : la classe [FirstController] dérive de la classe [System.Web.Mvc.Controller] ;
- lignes 14-17 : une action [Index] a été générée par défaut. Elle est **publique**. C'est important, sinon elle ne sera pas trouvée. Elle rend un type [ActionResult] qui est une classe abstraite dont dérivent la plupart des résultats que rend habituellement une action. C'est un type "fourre-tout" qu'on peut préciser en le remplaçant par le nom réel du type rendu ;
- ligne 16 : la méthode ne fait rien. Elle se contente de rendre une vue, donc un type [ViewResult]. Le nom de la vue n'est pas précisé. Dans ce cas, le framework cherche dans le dossier [Views / First] une vue portant le nom de l'action, donc ici : [Index.cshtml].

Créons la vue [Index.cshtml] :



- en [1], on clique droit dans le code de l'action et on prend l'option [Ajouter une vue] ;
- en [2], l'assistant propose une vue portant le nom de l'action. C'est ce qu'on veut ici ;
- en [3], par défaut l'utilisation de la page maître [\_Layout.cshtml] est proposée ;
- en [4], une fois validé, l'assistant crée la vue dans un sous-dossier du dossier [Views] portant le nom du contrôleur (First).

Le code généré pour la vue [Index] est le suivant :

```
1. @{
2.     ViewBag.Title = "Index";
3. }
4.
5. <h2>Index</h2>
```

- lignes 1-3 : un code C# qui définit une variable ;
- ligne 5 : une balise HTML.

On remplace tout le code précédent par celui-ci :

```
<strong>Vue [Index]...</strong>
```

Résumons :

- nous avons un contrôleur **C** appelé [First] ;
- nous avons une action appelée [Index] qui demande l'affichage d'une vue appelée [Index] ;
- nous avons la vue **V** [Index].

Nous pouvons appeler l'action [Index] de deux façons :

- /First/Index ;
- /First, car [Index] est également l'action par défaut dans les routes.

Exécutons l'application (CTRL-F5). Nous obtenons la page suivante :



## Erreur du serveur dans l'application '/'.

*La ressource est introuvable.*

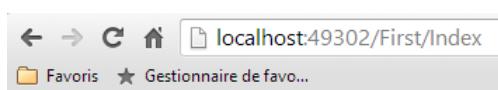
**Description :** HTTP 404. La ressource recherchée (ou l'une de ses dépendances) a peut-être été

**URL demandée:** /

**Informations sur la version :** Version Microsoft .NET Framework :4.0.30319; Version ASP.

En [1], l'URL demandée a été `http://localhost:49302`. Il n'y a pas de chemin. On sait que notre routeur attend des URL de la forme `/controller/{action}/{id}`. Ces éléments étant absents, les valeurs par défaut sont utilisées. L'URL devient `http://localhost:49302/Home/Index`. Le contrôleur [Home] n'existe pas. Donc l'URL est rejetée.

Essayons maintenant l'URL `http://localhost:49302/First/Index` en la tapant directement dans le navigateur :



## Tutoriel ASP.NET MVC [2]

Vue [Index]... [1]

La page ci-dessus a été générée par l'action [Index] du contrôleur [First]. La page générée par cette action est la vue [Index] dont le code était le suivant :

```
<strong>Vue [Index]...</strong>
```

Elle génère la partie [1]. La partie [2] elle, provient de la page maître [`_Layout`] que nous avons définie un peu plus tôt :

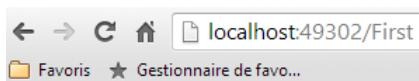
```
1.  <!DOCTYPE html>
2.  <html>
3.  <head>
4.    <meta charset="utf-8" />
5.    <meta name="viewport" content="width=device-width" />
6.    <title>Tutoriel ASP.NET MVC</title>
7.  </head>
8.  <body>
9.    <h2>Tutoriel ASP.NET MVC</h2>
10.   @RenderBody()
11. </body>
12. </html>
```

La ligne 9 a généré la partie [2] de la page. La vue [Index] n'intervient elle, qu'en ligne 10.

Si on affiche le code source de la page reçue, on retrouve bien la page [Index] incluse (ligne 10 ci-dessous) dans la page [`_Layout`] :

```
1.  <!DOCTYPE html>
2.  <html>
3.  <head>
4.    <meta charset="utf-8" />
5.    <meta name="viewport" content="width=device-width" />
6.    <title>Tutoriel ASP.NET MVC</title>
7.  </head>
8.  <body>
9.    <h2>Tutoriel ASP.NET MVC</h2>
10.   <strong>Vue [Index]...</strong>
11. </body>
12. </html>
```

Essayons maintenant l'URL [/First] :



## Tutoriel ASP.NET MVC

Vue [Index]...

L'URL [/First] était incomplète. Elle a été complétée avec les valeurs par défaut de la route et est devenue [/First/Index]. On obtient donc le même résultat que précédemment.

### 1.3.4 Action avec un résultat de type [ContentResult] - 1

Créons une nouvelle action dans le contrôleur [First] :

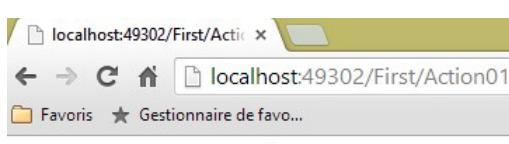
```
1.  using System.Text;
2.  using System.Web.Mvc;
3.
4.  namespace Exemple_01.Controllers
5.  {
6.      public class FirstController : Controller
7.      {
8.          // Index
9.          public ViewResult Index()
10.         {
11.             return View();
12.         }
13.         // Action01
14.         public ContentResult Action01()
15.         {
16.             return Content("<h1>Action [Action01]</h1>", "text/plain", Encoding.UTF8);
17.         }
18.     }
19. }
```

La nouvelle action est définie lignes 14-18. Elle se contente de rendre une chaîne de caractères à l'aide de la méthode [Content] (ligne 16) de la classe [Controller] (ligne 6). Les paramètres de la méthode sont :

1. la chaîne de caractères de la réponse ;
2. un indicateur de la nature du texte envoyé : "text/plain", "text/html", "text/xml", ... Cet indicateur est appelé type MIME ([http://fr.wikipedia.org/wiki>Type MIME](http://fr.wikipedia.org/wiki>Type_MIME)) ;
3. le troisième paramètre permet de préciser le type d'encodage utilisé pour le texte.

Plutôt que d'utiliser le type abstrait [ActionResult], nos méthodes précisent le type réel rendu (lignes 9 et 14).

Demandons l'URL [/First/Action01]. On obtient la page suivante :



Regardons le code source de la page reçue :

```
1.  <h1>Action [Action01]</h1>
```

Le navigateur n'a reçu que le texte envoyé par l'action et rien d'autre. Ce mode est intéressant lorsqu'on veut demander au serveur Web des données pures sans l'habillage HTML autour. On notera ci-dessus, que le navigateur n'a pas interprété la balise HTML <h1>. Pour comprendre pourquoi, regardons dans Chrome, les échanges HTTP :

Le navigateur a envoyé les entêtes HTTP suivants :

```

1. GET /First/Action01 HTTP/1.1
2. Host: localhost:49302
3. Connection: keep-alive
4. Cache-Control: max-age=0
5. Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
6. User-Agent: Mozilla/5.0 (Windows NT 6.2; WOW64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/29.0.1547.76 Safari/537.36
7. Accept-Encoding: gzip,deflate,sdch
8. Accept-Language: fr-FR,fr;q=0.8,en-US;q=0.6,en;q=0.4

```

Le serveur lui a répondu avec les entêtes suivants :

```

1. HTTP/1.1 200 OK
2. Cache-Control: private
3. Content-Type: text/plain; charset=utf-8
4. Content-Encoding: gzip
5. Vary: Accept-Encoding
6. Server: Microsoft-IIS/8.0
7. X-AspNetMvc-Version: 4.0
8. X-AspNet-Version: 4.0.30319
9. X-SourceFiles: =?UTF-8?B?RDpcZGF0YVxpc3RpYS0xMzE0XGFzcG5ldFxkdnBcRXh1bXBsZXNcRXh1bXBsZS0wMVxGaXJzdFxBY3Rpb24wMQ==?=?
10. X-Powered-By: ASP.NET
11. Date: Mon, 23 Sep 2013 15:22:33 GMT
12. Content-Length: 141

```

- ligne 3 : fixe la nature du document. On retrouve les attributs fixés dans la méthode [Action01]. C'est parce qu'on lui a dit que le document était de type "text/plain" et non "text/html" que le navigateur n'a pas interprété la balise <h1> qui était dans le document reçu.

### 1.3.5 Action avec un résultat de type [ContentResult] - 2

Considérons la troisième action suivante :

```

1. using System.Text;
2. using System.Web.Mvc;
3.
4. namespace Exemple_01.Controllers
5. {
6.     public class FirstController : Controller
7.     {
8.         ...
9.         // Action02
10.        public ContentResult Action02()
11.        {
12.            string data = "<action><name>Action02</name><description>renvoie un texte XML</description></action>";
13.            return Content(data, "text/xml", Encoding.UTF8);
14.        }
15.    }
16. }

```

- ligne 12 : on définit un texte XML ;
- ligne 13 : on l'envoie au navigateur en précisant que c'est du XML avec le type MIME " text/xml ".

Dans le navigateur, on obtient la page suivante :



Regardons dans Chrome la réponse HTTP du serveur :

```

1. HTTP/1.1 200 OK
2. Cache-Control: private
3. Content-Type: text/xml; charset=utf-8
4. Content-Encoding: gzip
5. Vary: Accept-Encoding
6. Server: Microsoft-IIS/8.0
7. X-AspNetMvc-Version: 4.0

```

```

8. X-AspNet-Version: 4.0.30319
9. X-SourceFiles: =?UTF-8?B?RDpcZGF0YVxpc3RpYS0xMzE0XGFzcG5ldFxdnBcRXh1bXBsZXNcRXh1bXBsZS0wMVxGaXJzdFxBY3Rpb24wMg==?=
10. X-Powered-By: ASP.NET
11. Date: Mon, 23 Sep 2013 15:29:34 GMT
12. Content-Length: 176

```

- ligne 3 : fixe la nature du document. On retrouve les attributs fixés dans la méthode [Action02] ;
- ligne 12 : la taille en octets du document envoyé par le serveur.

Le document envoyé par le serveur est celui-ci (copie Chrome) :

```

<action><name>Action02</name><description>renvoie un texte XML</description></action>

```

### 1.3.6 Action avec un résultat de type JsonResult

Ajoutons l'action suivante au contrôleur [First] :

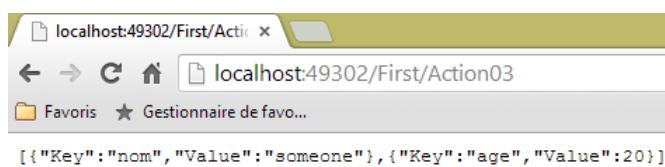
```

1.     // Action03
2.     public JsonResult Action03()
3.     {
4.         dynamic personne = new ExpandoObject();
5.         personne.nom = "someone";
6.         personne.age = 20;
7.         return Json(personne, JsonRequestBehavior.AllowGet);
8.     }

```

- ligne 4 : une variable de type *dynamic*. A l'exécution, on peut librement ajouter des propriétés à une telle variable. La propriété est créée en même temps qu'elle est initialisée ;
- lignes 5-6 : on initialise deux propriétés *nom* et *age* ;
- ligne 7 : on retourne la représentation JSON (Javascript Object Notation) de l'objet. Le JSON permet de sérialiser un objet en chaîne de caractères et inversement de déserialiser une chaîne en objet. C'est une alternative à la sérialisation / déserialisation XML ;
- ligne 2 : l'action retourne un type *[JsonResult]*. Ce type ne peut être retourné que pour une requête POST. Si on veut la retourner pour une méthode GET, il faut donner au second paramètre du constructeur de la class *Json* (ligne 7), la valeur *JsonRequestBehavior.AllowGet*.

Lorsqu'on demande l'URL [/First/Action03], le navigateur affiche la chose suivante :



La réponse HTTP du serveur est elle, la suivante :

```

1. HTTP/1.1 200 OK
2. Cache-Control: private
3. Content-Type: application/json; charset=utf-8
4. Server: Microsoft-IIS/8.0
5. X-AspNetMvc-Version: 4.0
6. X-AspNet-Version: 4.0.30319
7. X-SourceFiles: =?UTF-8?B?RDpcZGF0YVxpc3RpYS0xMzE0XGFzcG5ldFxdnBcRXh1bXBsZXNcRXh1bXBsZS0wMVxGaXJzdFxBY3Rpb24wMw==?=
8. X-Powered-By: ASP.NET
9. Date: Mon, 23 Sep 2013 15:48:53 GMT
10. Content-Length: 58

```

- ligne 3 : précise que le document envoyé est du JSON ;

- ligne 10 : le document envoyé fait 58 octets. C'est le suivant :

```
[{"Key": "nom", "Value": "someone"}, {"Key": "age", "Value": 20}]
```

L'élément dynamique [personne] est vu par JSON comme un tableau de dictionnaires où chaque dictionnaire :

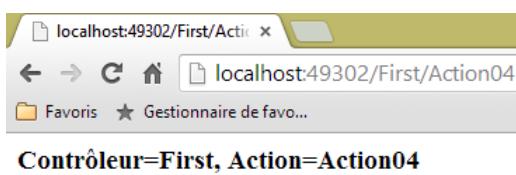
- correspond à un champ de la variable [personne] ;
- a deux clés "Key" et "Value". "Key" a pour valeur associée le nom du champ et "Value" a pour valeur la valeur du champ.

### 1.3.7 Action avec un résultat de type [string]

Ajoutons l'action suivante au contrôleur [First] :

```
1. // Action04
2. public string Action04()
3. {
4.     return "<h3>Contrôleur=First, Action=Action04</h3>";
5. }
```

Lorsque nous demandons l'URL [/First/Action04] avec Chrome, on obtient la réponse suivante :



On voit que la balise <h3> a été interprétée. Regardons la réponse HTTP du serveur :

```
1. HTTP/1.1 200 OK
2. Cache-Control: private
3. Content-Type: text/html; charset=utf-8
4. Content-Encoding: gzip
5. Vary: Accept-Encoding
6. Server: Microsoft-IIS/8.0
7. X-AspNetMvc-Version: 4.0
8. X-AspNet-Version: 4.0.30319
9. X-SourceFiles: =?UTF-8?B?RDpcZGF0YVxpc3RpYS0xMzE0XGFzcG5ldFxdnBcRXh1bXBsZXNcRXh1bXBsZS0wMVxGaXJzdFxBY3Rpb24wNA==?=
10. X-Powered-By: ASP.NET
11. Date: Tue, 24 Sep 2013 07:49:00 GMT
12. Content-Length: 156
```

et le document suivant :

```
<h3>Contrôleur=First, Action=Action04</h3>
```

On voit ligne 3, que le serveur a indiqué envoyer du texte au format HTML. C'est pourquoi le navigateur a interprété la balise <h3>. Lorsqu'on veut envoyer du texte pur, il est donc préférable de rendre un [ContentResult] plutôt qu'un [string]. Le [ContentResult] nous permet en effet de préciser un type MIME "text/plain" pour indiquer qu'on envoie du texte non formaté, donc non susceptible d'interprétation de la part du navigateur.

### 1.3.8 Action avec un résultat de type [EmptyResult]

Soit la nouvelle action suivante :

```
1. // Action05
2. public EmptyResult Action05()
3. {
4.     return new EmptyResult();
5. }
```

L'action se contente de rendre un type [EmptyResult]. Dans ce cas, le serveur envoie une réponse vide au client comme le montre sa réponse HTTP :

```
1. HTTP/1.1 200 OK
2. Cache-Control: private
3. Server: Microsoft-IIS/8.0
4. X-AspNetMvc-Version: 4.0
5. X-AspNet-Version: 4.0.30319
6. X-SourceFiles: =?UTF-8?B?RDpcZGF0YVxpc3RpYS0xMzE0XGFzcG5ldFxdnBcRXh1bXBsZXNcRXh1bXBsZS0wMVxGaXJzdFxBY3Rpb24wNQ==?=
```

```
7. X-Powered-By: ASP.NET
8. Date: Tue, 24 Sep 2013 08:11:12 GMT
9. Content-Length: 0
```

- ligne 9 : le serveur indique à son client qu'il lui envoie un document vide.

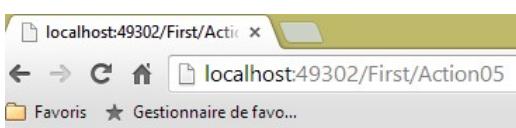
### 1.3.9 Action avec un résultat de type [RedirectResult] - 1

Soit la nouvelle action suivante :

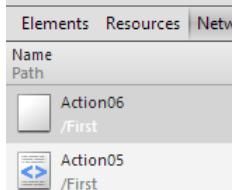
```
1. // Action06
2. public RedirectResult Action06()
3. {
4.     return new RedirectResult("/First/Action05");
5. }
```

L'action rend un type [RedirectResult]. Ce type permet d'envoyer au client un ordre de redirection vers l'URL paramètre du constructeur (ligne 4). Le client va alors émettre une nouvelle requête GET vers [/First/Action05]. Le client fait donc deux requêtes au total.

Le navigateur affiche le résultat de la seconde requête :



Examinons la réponse HTTP du serveur dans Chrome :



On voit ci-dessus, les deux requêtes du navigateur. Examinons la première requête [Action06]. La réponse HTTP du serveur est la suivante :

```
1. HTTP/1.1 302 Found
2. Cache-Control: private
3. Content-Type: text/html; charset=utf-8
4. Location: /First/Action05
5. Server: Microsoft-IIS/8.0
6. X-AspNetMvc-Version: 4.0
7. X-AspNet-Version: 4.0.30319
8. X-SourceFiles: =?UTF-8?B?RDpcZGF0YVxpc3RpYS0xMzE0XGFzcG5ldFxkdnBcRXh1bXBsZXNcRXh1bXBsZS0wMVxGaXJzdFxBY3Rpb24wNg==?=
9. X-Powered-By: ASP.NET
10. Date: Tue, 24 Sep 2013 08:16:59 GMT
11. Content-Length: 132
```

- ligne 1 : le serveur a répondu avec un code 302 Found. Pour l'instant, c'était 200 OK qui veut dire que le document demandé a été trouvé. Le code 302 indique qu'une redirection est demandée. L'adresse de redirection est donnée par la ligne 4. On retrouve là l'adresse de redirection que nous avions donnée dans le code de l'action ;
- ligne 11 : le serveur indique qu'avec sa réponse HTTP, il envoie un document de type text/html (ligne 3) de 132 octets (ligne 11). Lorsqu'on examine dans Chrome la réponse à la requête [Action06], elle est vide comme on pouvait s'y attendre. Il y a probablement une explication mais je ne la connais pas.

A cause de la redirection, le navigateur fait une nouvelle requête GET vers l'URL précisée ligne 4 ci-dessus comme on peut le voir dans Chrome ligne 1 ci-dessous :

```
1. GET /First/Action05 HTTP/1.1
2. Host: localhost:49302
3. Connection: keep-alive
4. Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
5. User-Agent: Mozilla/5.0 (Windows NT 6.2; WOW64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/29.0.1547.76 Safari/537.36
6. Accept-Encoding: gzip,deflate,sdch
7. Accept-Language: fr-FR,fr;q=0.8,en-US;q=0.6,en;q=0.4
```

### 1.3.10 Action avec un résultat de type [RedirectResult] - 2

Soit la nouvelle action suivante :

```
1. // Action07
2. public RedirectResult Action07()
3. {
4.     return new RedirectResult("/First/Action05",true);
5. }
```

Ligne 4, on a ajouté un second paramètre au constructeur du type [RedirectResult]. C'est un booléen qui par défaut est à *false*. Lorsqu'on le passe à *true*, on modifie la réponse HTTP envoyée au client. Elle devient :

```
HTTP/1.1 301 Moved Permanently
```

Ainsi le code de réponse envoyé au client est maintenant *301 Moved Permanently*. La redirection se passe comme précédemment mais on indique que cette redirection est permanente. Cela permet aux moteurs de recherche de remplacer dans leurs résultats l'ancienne URL par la nouvelle.

### 1.3.11 Action avec un résultat de type [RedirectToRouteResult]

Soit la nouvelle action suivante :

```
1. // Action08
2. public RedirectToRouteResult Action08()
3. {
4.     return new RedirectToRouteResult("Default",new RouteValueDictionary(new
5.     {controller="First",action="Action05"}));
}
```

- ligne 2 : l'action rend un type [RedirectToRouteResult]. Ce type permet de rediriger un client vers une URL spécifiée non pas par une chaîne de caractères comme précédemment mais par une route.

Les routes sont définies dans [App\_Start/RouteConfig]. Il n'y en a qu'une actuellement :

```
1. routes.MapRoute(
2.     name: "Default",
3.     url: "{controller}/{action}/{id}",
4.     defaults: new { controller = "Home", action = "Index", id = UrlParameter.Optional }
5. );
```

- ligne 4, on demande au client de se rediriger vers la route nommée [Default] avec la variable **controller**=First et la variable **action**=Action05. Le système de routage va alors générer l'URL de redirection */First/Action05*. C'est ce que montre la réponse HTTP du serveur :

```
1. HTTP/1.1 302 Found
2. Cache-Control: private
3. Content-Type: text/html; charset=utf-8
4. Location: /First/Action05
5. Server: Microsoft-IIS/8.0
6. X-AspNetMvc-Version: 4.0
7. X-AspNet-Version: 4.0.30319
8. X-SourceFiles: =?UTF-8?B?RDpcZGF0YVxpc3RpYS0xMzE0XGFzcG5ldFxkdnBcRXh1bXBsZXNcRXh1bXBsZS0wMVxGaXJzdFxBY3RpB24wOA==?=
9. X-Powered-By: ASP.NET
10. Date: Tue, 24 Sep 2013 08:58:19 GMT
11. Content-Length: 132
```

- ligne 1 : la redirection ;
- ligne 4 : l'adresse de redirection générée par le système de routage des URL.

### 1.3.12 Action avec un résultat de type [void]

Soit la nouvelle action suivante :

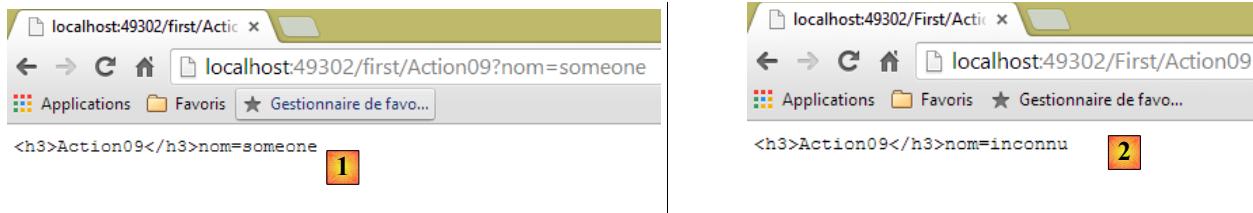
```
1. // Action09
2. public void Action09()
3. {
4.     string nom = Request.QueryString["nom"] ?? "inconnu";
5.     Response.AddHeader("Content-Type", "text/plain");
6.     Response.Write(string.Format("<h3>Action09</h3>nom={0}", nom));
7. }
```

- ligne 2 : l'action ne rend aucun résultat. Elle écrit elle-même dans le flux de la réponse envoyée au client ;
- ligne 4 : on récupère un éventuel paramètre nommé [nom] dans la requête. Celle-ci est accessible via la propriété [Request] du contrôleur [Controller] dont hérite le contrôleur [First]. Le paramètre [nom] passé sous la forme [/First/Action09?nom=quelquechose], est disponible dans **Request.QueryString["nom"]**. La syntaxe de la ligne 4 est équivalente à :

```
string nom=Request.QueryString["nom"];
if(nom==null){
    nom="inconnu";
}
```

- ligne 5 : la réponse envoyée au client est accessible via la propriété [Response] du contrôleur [Controller] dont hérite le contrôleur [First] ;
- ligne 5 : on fixe l'entête HTTP [Content-Type] qui indique la nature du document que le serveur s'apprête à envoyer au client. Ici "text/plain" indique que le document est du texte pur qui ne doit pas être interprété par le navigateur ;
- ligne 6 : on écrit dans le flux de la réponse, une chaîne de caractères. On y a inclus des balises HTML qui ne doivent pas être interprétées par le navigateur puisque celui-ci aura reçu auparavant l'entête HTTP [Content-Type : text/plain"]. C'est ce qu'on veut vérifier.

Compilons le projet et demandons l'URL [/First/Action09?nom=someone ] [1] puis l'URL [/First/Action09 ] [2] :



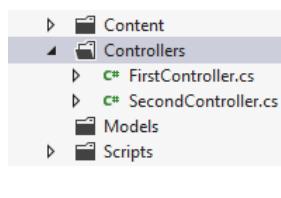
Regardons maintenant dans Chrome la réponse HTTP du serveur :

```
1. HTTP/1.1 200 OK
2. Cache-Control: private
3. Content-Type: text/plain; charset=utf-8
4. ...
5. Content-Length: 144
```

- ligne 3 : on retrouve l'entête HTTP que nous avions nous-même fixé dans le code de l'action.

### 1.3.13 Un second contrôleur

Créons dans le projet un second contrôleur. On suivra la méthode décrite au paragraphe 1.3.1, page 45. On l'appellera [Second].



Son code généré est le suivant :

```
1. namespace Exemple_01.Controllers
```

```

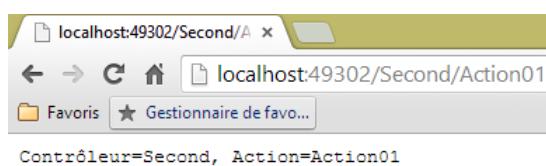
2. {
3.     public class SecondController : Controller
4.     {
5.         //
6.         // GET: /Second/
7.
8.         public ActionResult Index()
9.         {
10.             return View();
11.         }
12.
13.     }
14. }
```

Modifions-le de la façon suivante :

```

1. using System.Text;
2. using System.Web.Mvc;
3.
4. namespace Exemple_01.Controllers
5. {
6.     public class SecondController : Controller
7.     {
8.         // /Second/Action01
9.         public ContentResult Action01()
10.        {
11.            return Content("Contrôleur=Second, Action=Action01", "text/plain", Encoding.UTF8);
12.        }
13.
14.    }
15. }
```

Puis demandons l'URL [/Second/Action01] avec un navigateur. Nous obtenons la réponse suivante :



Cette URL a été demandée avec une commande HTTP GET comme le montrent les logs HTTP de la requête dans Chrome :

```
GET /Second/Action01 HTTP/1.1
```

L'URL peut être demandée également avec une commande HTTP POST. Pour le montrer, utilisons de nouveau l'application [Advanced Rest Client] :



- en [1], on lance l'application (dans l'onglet [Applications] d'un nouvel onglet Chrome) ;
- en [2], on sélectionne l'option [Request] ;
- en [3], on précise l'URL demandée ;
- en [4], on indique que l'URL doit être demandée avec un POST ;

On active les logs de Chrome par (CTRL-I) afin d'avoir la réponse HTTP du serveur. Lorsqu'on exécute [Send] la requête précédente, les échanges HTTP sont les suivants :

Le navigateur envoie la requête suivante :

```

1. POST /Second/Action01 HTTP/1.1
2. Host: localhost:49302
3. Connection: keep-alive
4. Content-Length: 0
5. Origin: chrome-extension://hgmlloofddfdnphfgcellkdfbfbjeloo
6. User-Agent: Mozilla/5.0 (Windows NT 6.2; WOW64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/29.0.1547.76 Safari/537.36
7. Content-Type: application/x-www-form-urlencoded
8. Accept: */
9. Accept-Encoding: gzip,deflate,sdch
10. Accept-Language: fr-FR,fr;q=0.8,en-US;q=0.6,en;q=0.4

```

- ligne 1 : l'URL est bien demandée avec un POST ;
- ligne 4 : la taille en octets des éléments postés. Il n'y en a pas ici.

La réponse HTTP du serveur est elle la suivante :

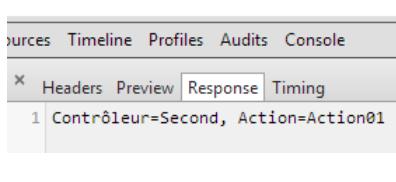
```

1. HTTP/1.1 200 OK
2. Cache-Control: private
3. Content-Type: text/plain; charset=utf-8
4. Content-Encoding: gzip
5. Vary: Accept-Encoding
6. Server: Microsoft-IIS/8.0
7. X-AspNetMvc-Version: 4.0
8. X-AspNet-Version: 4.0.30319
9. X-SourceFiles: =?UTF-8?B?RDpcZGF0YVxpc3RpYS0xMzE0XGFzcG5ldFxkdnBcRXh1bXBsZXNcRXh1bXBsZS0wMVxTZWNvbmRcQWN0aW9uMDE=?=
10. X-Powered-By: ASP.NET
11. Date: Tue, 24 Sep 2013 10:47:59 GMT
12. Content-Length: 148

```

- ligne 3 : le serveur envoie un document texte non formaté (plain) ;
- ligne 12 : de 148 caractères.

Le document envoyé est le suivant :



On obtient le même document qu'avec le GET.

### 1.3.14 Action filtrée par un attribut

Créons la nouvelle action suivante :

```

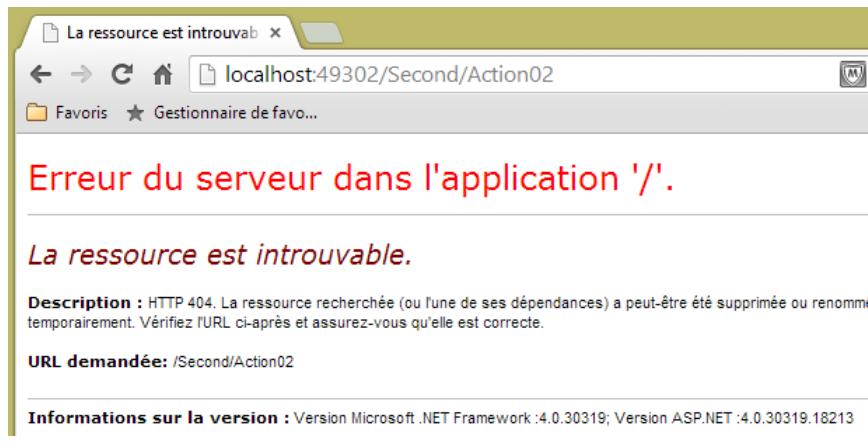
1.      // /Second/Action02
2.      [HttpPost]
3.      public ContentResult Action02()
4.      {
5.          return Content("Contrôleur=Second, Action=Action02", "text/plain", Encoding.UTF8);
6.      }

```

L'action [Action02] est analogue à l'action [Action01] mais on indique qu'elle n'est accessible que par une commande HTTP POST (ligne 2). D'autres attributs sont utilisables :

- `HttpGet` ne sert que la commande GET
- `HttpHead` ne sert que la commande HEAD
- `HttpOptions` ne sert que la commande OPTIONS
- `HttpPut` ne sert que la commande PUT
- `HttpDelete` ne sert que la commande DELETE

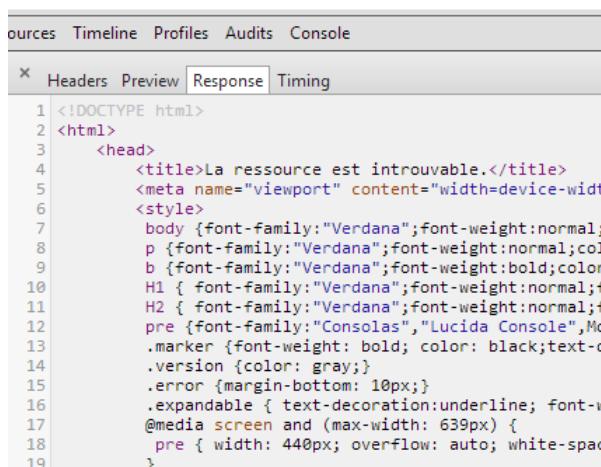
Demandons l'URL [/Second/Action02] directement dans le navigateur. Elle est alors demandée par un GET. Le navigateur affiche alors la réponse suivante :



La réponse HTTP du serveur a été la suivante :

1. HTTP/1.1 404 Not Found
2. ...
3. Content-Length: 3807

- ligne 1 : le code HTTP **404 Not Found** indique que le serveur n'a pas trouvé le document demandé. Ici, l'action [Action02] n'a pu servir la requête GET car elle ne sert que les commandes POST ;
  - ligne 3 : la taille du document renvoyé. C'est la page qui a été affichée par le navigateur :



### 1.3.15 Récupérer les éléments d'une route

Dans les deux actions décrites précédemment, on écrivait quelque chose du genre :

```
1.     public ContentResult Action02()
2.     {
3.         return Content("Contrôleur=Second, Action=Action02", "text/plain", Encoding.UTF8);
4.     }
```

Les noms du contrôleur et de l'action étaient écrits en dur dans le code. Si on change ces noms, le code n'est plus correct. On peut avoir accès au contrôleur et à l'action de la façon suivante :

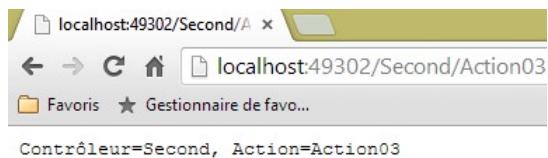
```
1.     // /Second/Action03
2.     public ContentResult Action03()
3.     {
4.         string texte = string.Format("Contrôleur={0}, Action={1}", RouteData.Values["controller"],
    RouteData.Values["action"]);
5.         return Content(texte, "text/plain", Encoding.UTF8);
6.     }
```

La route définie dans [App\_Start/RouteConfig] est la suivante :

```
1.     routes.MapRoute(
2.         name: "Default",
3.         url: "{controller}/{action}/{id}",
4.         defaults: new { controller = "Home", action = "Index", id = UrlParameter.Optional }
5.     );
```

Ligne 3, les trois éléments de la route peuvent être obtenus par **RouteData.Values["élément"]** avec **élément** dans [controller, action, id].

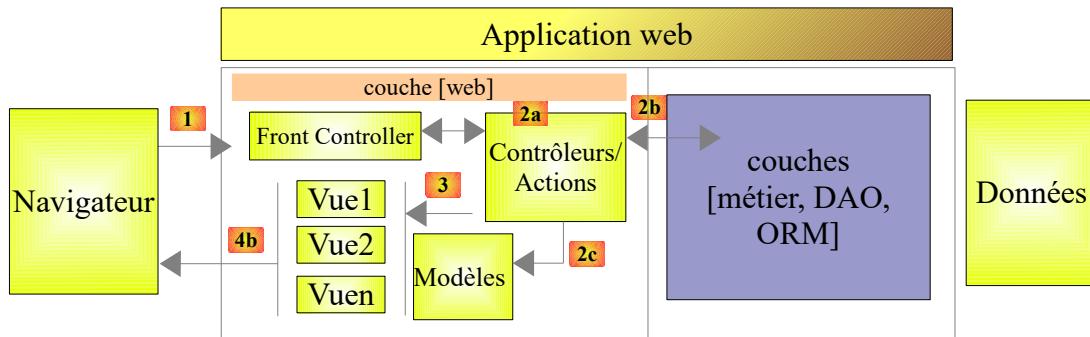
Demandons l'URL [http://localhost:49302/Second/Action03] :



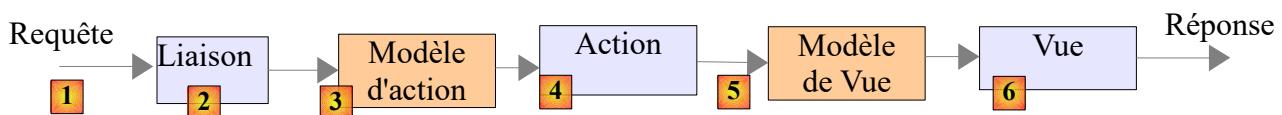
Nous avons bien récupéré et le nom du contrôleur et le nom de l'action.

## 1.4 Le modèle d'une action

Revenons à l'architecture d'une application ASP.NET MVC :



Dans le chapitre précédent, nous avons regardé le processus qui amène la requête [1] au contrôleur et à l'action [2a] qui vont la traiter, un mécanisme qu'on appelle le routage. Nous avons présenté par ailleurs les différentes réponses que peut faire une action au navigateur. Nous avons pour l'instant présenté des actions qui n'exploitaient pas la requête qui leur était présentée. Une requête [1] transporte avec elle diverses informations que ASP.NET MVC présente [2a] à l'action sous forme d'un **modèle**. On ne confondra pas ce terme avec le modèle **M** d'une vue **V** [2c] qui est produit par l'action :



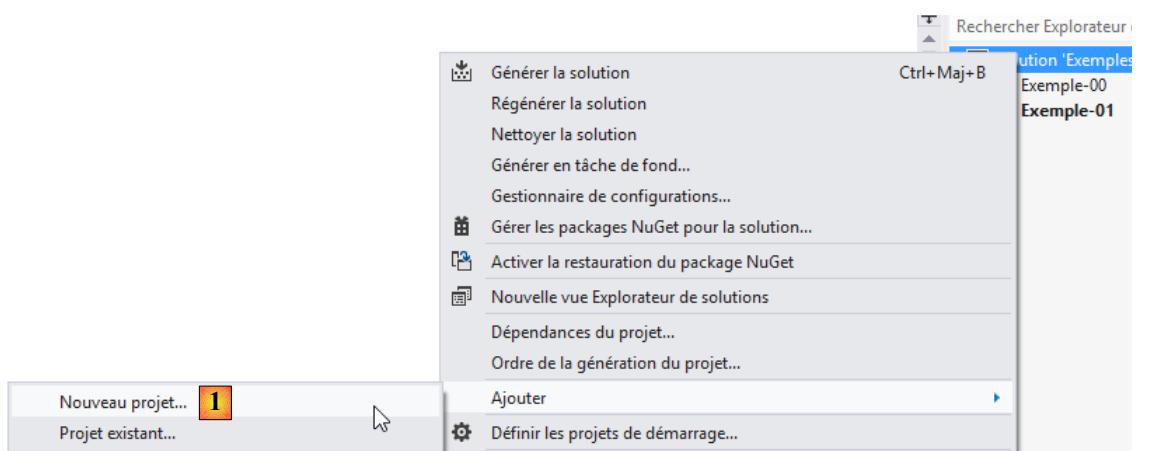
- la requête HTTP du client arrive en [1] ;
- en [2], les informations contenues dans la requête vont être transformées en modèle d'action [3], une classe souvent mais pas forcément, qui servira d'entrée à l'action [4] ;
- en [4], l'action, à partir de ce modèle, va générer une réponse. Celle-ci aura deux composantes : une vue V [6] et le modèle M de cette vue [5] ;
- la vue V [6] va utiliser son modèle M [5] pour générer la réponse HTTP destinée au client.

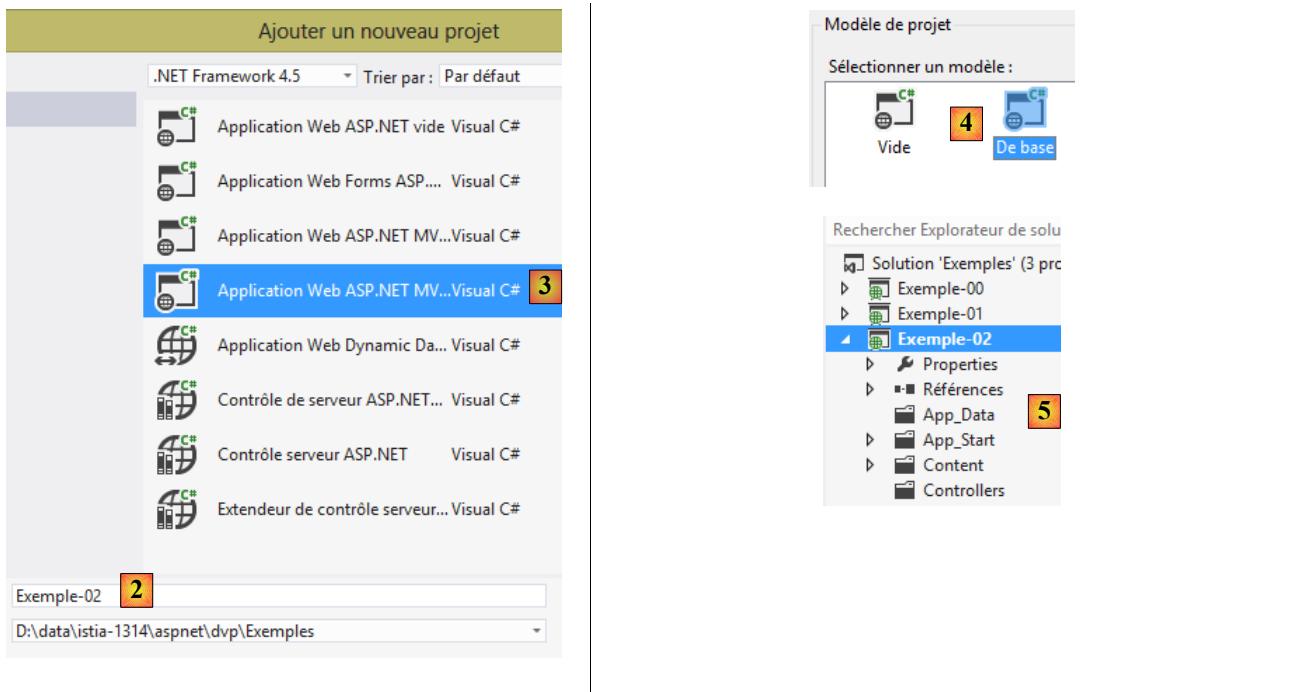
Dans le modèle **MVC**, l'action [4] fait partie du **C** (contrôleur), le modèle de la vue [5] est le **M** et la vue [6] est le **V**.

Ce chapitre étudie les mécanismes de liaison entre les informations transportées par la requête, qui sont par nature des chaînes de caractères et le modèle de l'action qui peut être une classe avec des propriétés de divers types.

### 1.4.1 Initialisation des paramètres de l'action

Nous ajoutons [1] à la solution existante un nouveau projet ASP.NET MVC de base :

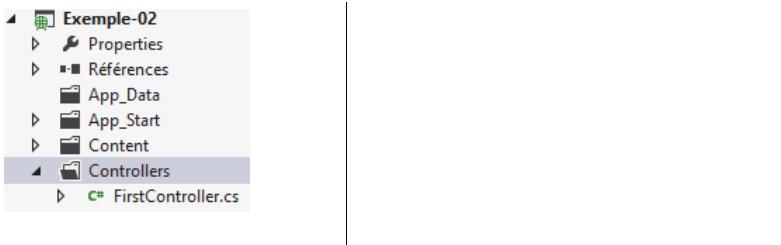




- en [2], le nom du nouveau projet ;
- en [3, 4], nous choisissons un projet ASP.NET MVC de base ;
- en [5], le nouveau projet.

On fera du nouveau projet, le projet de démarrage de la solution (cf page 46).

Comme il a été fait au paragraphe 1.3.1, page 45, nous créons un contrôleur nommé [First] [1] :



Dans ce contrôleur, nous créons l'action [Action01] suivante :

```

1. using System.Web.Mvc;
2.
3. namespace Exemple_02.Controllers
4. {
5.     public class FirstController : Controller
6.     {
7.         // Action01
8.         public ContentResult Action01(string nom)
9.         {
10.             return Content(string.Format("Contrôleur=First, Action=Action01, nom={0}", nom));
11.         }
12.     }
13. }
14. }
```

La nouveauté réside à la ligne 8 : la méthode [Action01] a un paramètre. On s'intéresse dans ce chapitre aux différentes façons d'initialiser les paramètres d'une action. Le paramètre [nom] ci-dessus est initialisé dans l'ordre avec les valeurs suivantes :

`Request.Form["nom"]` un paramètre nommé [nom] envoyé par une commande POST

`RouteData.Values["nom"]` un élément d'URL nommé [nom]

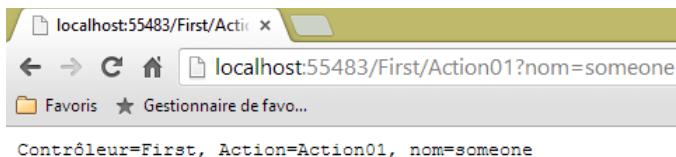
```
Request.QueryString["nom"]
```

 un paramètre nommé [nom] envoyé par une commande GET

```
Request.Files["nom"]
```

 un fichier uploadé nommé [nom]

Examinons ces différents cas. Demandons directement dans le navigateur l'URL [/First/Action01?nom=someone]. On obtient la réponse suivante :



La requête HTTP du navigateur a été la suivante :

1. GET /First/Action01?nom=someone HTTP/1.1
2. Host: localhost:55483
3. ...

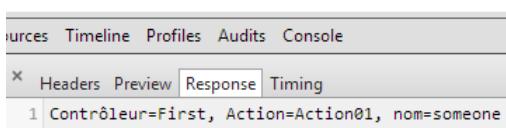
- ligne 1 : la requête est un GET. L'URL demandée embarque le paramètre [nom]. Côté serveur, la requête arrive à l'action [Action01] qui a la signature suivante :

```
public ContentResult Action01(string nom)
```

Pour donner une valeur au paramètre nom, ASP.NET MVC essaie successivement et dans l'ordre les valeurs `Request.Form["nom"]`, `RouteData.Values["nom"]`, `Request.QueryString["nom"]`, `Request.Files["nom"]`. Il s'arrête dès qu'il a trouvé une valeur. Le paramètre [nom] embarqué dans l'URL du GET a été placé par le framework dans `Request.QueryString["nom"]`. C'est avec cette valeur [someone] que va être initialisé le paramètre [nom] de [Action01]. Ensuite le code de [Action01] s'exécute :

```
return Content(string.Format("Contrôleur=First, Action=Action01, nom={0}", nom), "text/plain", Encoding.UTF8);
```

Ce code fournit la réponse envoyée au client :

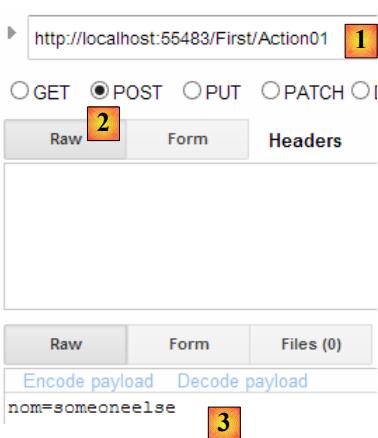


**Note** : le mécanisme de liaison des paramètres est **insensible à la casse**. Ainsi si notre action est définie comme :

```
public ContentResult Action01(string NOM)
```

et que le paramètre passé est [?NoM=zébulon], la liaison aura bien lieu. Le paramètre [NOM] de [Action01] recevra la valeur [zébulon].

Maintenant, demandons la même URL avec un POST. Pour cela, nous utilisons l'application [Advanced Rest Client] :



- en [1], l'URL demandée ;
- en [2], la commande POST sera utilisée ;
- en [3], les paramètres du POST.

Envoyons cette requête et regardons les logs HTTP. La requête HTTP est la suivante :

- en [1], le POST ;
- en [2], les paramètres du POST. Techniquement, ils ont été envoyés derrière les entêtes HTTP après la ligne vide signalant la fin de ces entêtes ;
- en [3], la réponse obtenue. On récupère bien le paramètre [nom] du POST. Dans les valeurs essayées pour le paramètre nom `Request.Form["nom"]`, `RouteData.Values["nom"]`, `Request.QueryString["nom"]`, `Request.Files["nom"]`, c'est la première qui a marché.

Maintenant, modifions la route par défaut dans [App\_Start/RouteConfig]. Actuellement, cette route est la suivante :

```

1.     routes.MapRoute(
2.         name: "Default",
3.         url: "{controller}/{action}/{id}",
4.         defaults: new { controller = "Home", action = "Index", id = UrlParameter.Optional }
5.     );

```

Changeons-la en :

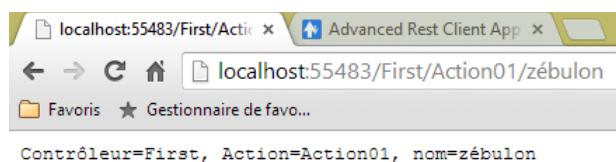
```

1.     routes.MapRoute(
2.         name: "Default",
3.         url: "{controller}/{action}/{nom}",
4.         defaults: new { controller = "Home", action = "Index", nom = UrlParameter.Optional }
5.     );

```

- ligne 3, nous avons nommé [nom] le troisième élément d'une route ;
- ligne 4, cet élément est déclaré optionnel.

Maintenant, recompilons l'application et demandons l'URL [/First/Action01/zébulon] directement dans le navigateur. Nous obtenons, la réponse suivante :



Dans les valeurs essayées pour le paramètre nom `Request.Form["nom"]`, `RouteData.Values["nom"]`, `Request.QueryString["nom"]`, `Request.Files["nom"]`, c'est la deuxième qui a marché.

Faisons la même requête avec un POST et [Advanced Rest Client] :

A screenshot of the Advanced Rest Client application. The top navigation bar shows the URL 'http://localhost:55483/First/Action01/zébulon' with a red box labeled '1' over it. Below the URL, there are radio buttons for GET, POST, PUT, PATCH, and DELETE, with 'POST' selected. Underneath the URL, there are three tabs: 'Raw', 'Form', and 'Headers', with 'Form' selected. In the 'Form' tab, there is a single field 'nom=someoneelse' with a red box labeled '2' over it. At the bottom of the client interface, there is a 'Payload' section with tabs for 'Raw', 'Form', 'Files (0)', and 'Payload', with 'Payload' selected. The payload area contains the same 'nom=someoneelse' value. To the right of the client interface, there is a 'Response' panel with tabs for 'Raw', 'Parsed', and 'Response', with 'Response' selected. The response content shows the route information: 'Contrôleur=First, Action=Action01, nom=someoneelse' with a red box labeled '3' over it.

- en [1], nous avons donné une valeur à l'élément {nom} de la route ;
- en [2], on ajoute un paramètre [nom] à la requête postée ;
- la réponse obtenue est en [3].

Dans les valeurs essayées pour le paramètre [nom] `Request.Form["nom"]`, `RouteData.Values["nom"]`, `Request.QueryString["nom"]`, `Request.Files["nom"]`, deux convenaient, les deux premières. C'est la première qui a été utilisée.

#### 1.4.2 Vérifier la validité des paramètres de l'action

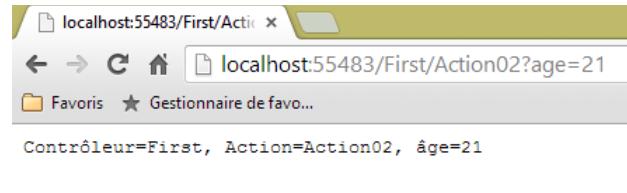
Si une action a un paramètre nommé [p], ASP.NET MVC va essayer de lui affecter l'une des valeurs `Request.Form["p"]`, `RouteData.Values["p"]`, `Request.QueryString["p"]`, `Request.Files["p"]`. Les trois premières valeurs sont des chaînes de caractères. Si le paramètre [p] n'est pas de type [string], des problèmes peuvent survenir.

Créons la nouvelle action suivante :

```
1. // Action02
2. public ContentResult Action02(int age)
3. {
4.     string texte = string.Format("Contrôleur={0}, Action={1}, âge={2}", RouteData.Values["controller"],
    RouteData.Values["action"], age);
5.     return Content(texte, "text/plain", Encoding.UTF8);
6. }
```

- ligne 2, l'action [Action02] admet un paramètre nommé [age] de type **int**. Il faudra que la chaîne de caractères récupérée soit convertible en **int**.

Demandons l'URL [<http://localhost:55483/First/Action02?age=21>]. On obtient la page suivante :



Demandons l'URL [http://localhost:55483/First/Action02?age=21x]. On obtient la page suivante :

**Le dictionnaire de paramètres contient une entrée « Exemple\_02.Controllers.FirstController ». Un pa Nom du paramètre : parameters**

**Description :** Une exception non gérée s'est produite au moment de l'exécution de la requête \

**Détails de l'exception:** System.ArgumentException: Le dictionnaire de paramètres contient paramètre facultatif.  
Nom du paramètre : parameters

Cette fois-ci, on a récupéré une page d'erreur. Il est intéressant de regarder les entêtes HTTP envoyés par le serveur dans ce cas :

```
1. HTTP/1.1 500 Internal Server Error
2. ...
3. Content-Type: text/html; charset=utf-8
4. ...
5. Content-Length: 12438
```

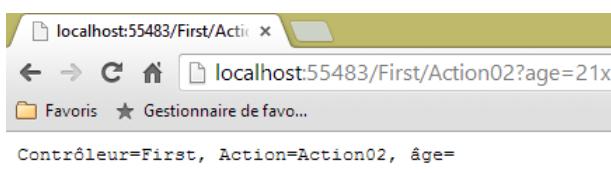
- ligne 1 : le serveur a répondu avec un code [500 Internal Server Error] et a envoyé une page HTML (ligne 3) de 12438 octets (ligne 5) pour expliquer les raisons possibles de cette erreur.

Créons maintenant l'action [Action03] suivante :

```
1. // Action03
2. public ContentResult Action03(int? age)
3. {
4.     ...
5. }
```

[Action03] est identique à [Action02] si ce n'est qu'on a changé le type du paramètre [age] en **int?**, ce qui signifie **entier ou null**.

Demandons l'URL [http://localhost:55483/First/Action03?age=21x]. On obtient la page suivante :



ASP.NET MVC n'a pas réussi à convertir [21x] en type **int**. Il a alors affecté la valeur **null** au paramètre [age] comme l'autorise son type **int?**. Il est cependant possible de savoir si le paramètre a pu recevoir une valeur de la requête ou non.

Nous construisons la nouvelle action [Action04] suivante :

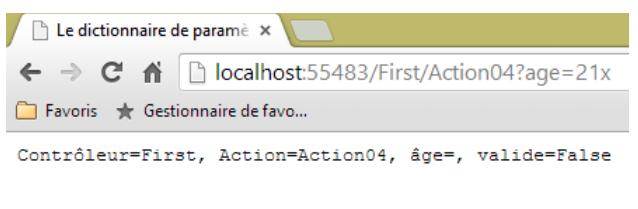
```

1.      // Action04
2.      public ContentResult Action04(int? age)
3.      {
4.          bool valide = ModelState.IsValid;
5.          string texte = string.Format("Contrôleur={0}, Action={1}, âge={2}, valide={3}", RouteData.Values["controller"],
6.              RouteData.Values["action"], age, valide);
7.          return Content(texte, "text/plain", Encoding.UTF8);
7.      }

```

- ligne 2 : on a gardé le type [int?]. Cela permet en particulier à la requête de ne pas fournir le paramètre [age] qui reçoit alors la valeur **null** ;
- ligne 4 : on teste si le modèle de l'action est valide. Le modèle de l'action est formé de l'ensemble de ses paramètres, ici [age]. Le modèle est valide si tous les paramètres ont pu obtenir une valeur de la requête ou bien la valeur **null** si le type du paramètre le permet ;
- ligne 5 : on ajoute la valeur de la variable [valide] dans le text envoyé au client.

Demandons l'URL [<http://localhost:55483/First/Action04?age=21x>]. On obtient la page suivante :



ASP.NET MVC n'a pas réussi à convertir [21x] en type **int**. Il a alors affecté la valeur **null** au paramètre [age] comme l'autorise son type **int?**. Mais il y a eu des erreurs de conversion comme le montre la valeur de [valide].

Il est possible d'avoir le message d'erreur associé à une conversion ratée. Examinons la nouvelle action suivante :

```

1.      // Action05
2.      public ContentResult Action05(int? age)
3.      {
4.          string erreurs = getErrorMessagesFor(ModelState);
5.          string texte = string.Format("Contrôleur={0}, Action={1}, âge={2}, valide={3}, erreurs={4}",
6.              RouteData.Values["controller"], RouteData.Values["action"], age, ModelState.IsValid, erreurs);
7.          return Content(texte, "text/plain", Encoding.UTF8);
7.      }

```

La nouveauté est ligne 4. On y appelle une méthode privée [getErrorMessagesFor] à laquelle on passe l'état du modèle de l'action. Elle rend une chaîne de caractères rassemblant les messages de toutes les erreurs qui se sont produites. Cette méthode est la suivante :

```

1.  private string getErrorMessagesFor(ModelStateDictionary état)
2.  {
3.      List<String> erreurs = new List<String>();
4.      string messages = string.Empty;
5.      if (!état.IsValid)
6.      {
7.          foreach (ModelState modèle in état.Values)
8.          {
9.              foreach (ModelError erreur in modèle.Errors)
10.              {
11.                  erreurs.Add(getErrorMessageFor(erreur));
12.              }
13.          }
14.          foreach (string message in erreurs)
15.          {
16.              messages += string.Format("[{0}]", message);
17.          }
18.      }
19.      return messages;
20.  }

```

- ligne 1 : le paramètre effectif [ModelState] passé à la méthode est de type [ModelStateDictionary] ;
- ligne 3 : une liste de messages d'erreurs, vide au départ ;
- ligne 5 : on teste si l'état passé en paramètre est valide ou non. Si non, alors on va agréger tous les messages d'erreur en une unique chaîne de caractères ;
- ligne 7 : le type [ModelStateDictionary] a une propriété [Values] qui est une collection de types [ModelState]. Il y a un [ModelState] par élément du modèle. Par exemple :

- **ModelState["age"]** : l'état du modèle de l'action pour le paramètre [age],
- **ModelState["age"].Errors** : la collection d'erreurs pour ce paramètre. Les erreurs sont de type [ModelError],
- **ModelState["age"].Errors[i].ErrorMessage** : l'éventuel message d'erreur n° i pour le paramètre [age] du modèle
- **ModelState["age"].Errors[i].Exception** : l'exception de l'erreur n° i de la collection des erreurs sur le paramètre [age],
- **ModelState["age"].Errors[i].Exception.InnerException** : la cause de cette exception,
- **ModelState["age"].Errors[i].Exception.InnerException.Message** : le message de la cause de l'exception ;
- ligne 9 : on parcourt la collection [Errors] d'un [ModelState] particulier ;
- ligne 11 : on récupère le message d'erreur d'un [ModelError] particulier et on l'ajoute à la liste des messages d'erreurs de la ligne 3 ;
- lignes 14-17 : on agrège les éléments de la liste des messages d'erreurs en une unique chaîne de caractères.

La méthode [getErrorMessageFor] de la ligne 11 est la suivante :

```

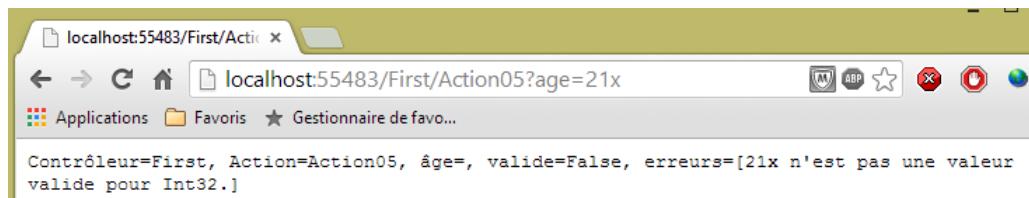
1.     private string getErrorMessageFor(ModelError error)
2.     {
3.         if (error.ErrorMessage != null && error.ErrorMessage.Trim() != string.Empty)
4.         {
5.             return error.ErrorMessage;
6.         }
7.         if (error.Exception != null && error.Exception.InnerException == null && error.Exception.Message != string.Empty)
8.         {
9.             return error.Exception.Message;
10.        }
11.        if (error.Exception != null && error.Exception.InnerException != null && error.Exception.InnerException.Message != string.Empty)
12.        {
13.            return error.Exception.InnerException.Message;
14.        }
15.        return string.Empty;
16.    }

```

- ligne 1 : on reçoit un type [ModelError] qui encapsule une erreur sur l'un des éléments du modèle de l'action. On va chercher le message d'erreur dans trois endroits différents :
  - dans [ModelError].ErrorMessage, lignes 3-6 ;
  - dans [ModelError].Exception.Message, lignes 7-10 ;
  - dans [ModelError].Exception.InnerException.Message, lignes 11-14 ;

Durant les tests, on remarque que le message d'erreur est trouvé dans ces trois endroits selon la nature de l'élément du modèle. Il doit y avoir une règle qui permette d'obtenir à coup sûr le message d'erreur associé à un élément du modèle, mais je ne la connais pas. Je le cherche donc aux différents endroits où je peux le trouver et ce dans un certain ordre. Dès qu'un message non vide a été trouvé, il est retourné.

Demandons l'URL [<http://localhost:55483/First/Action05?age=21x>]. On obtient la page suivante :



### 1.4.3 Une action à plusieurs paramètres

Considérons la nouvelle action suivante :

```

1.     // Action06
2.     public ContentResult Action06(double? poids, int? age)
3.     {
4.         string erreurs = getErrorMessagesFor(ModelState);

```

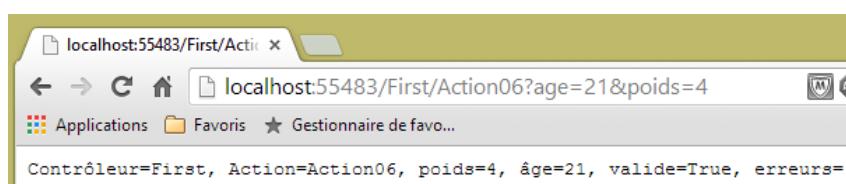
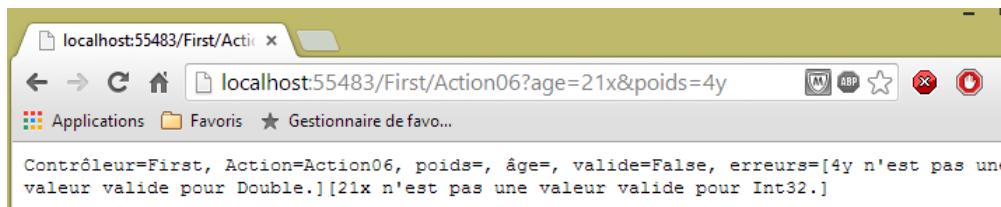
```

5.     string texte = string.Format("Contrôleur={0}, Action={1}, poids={2}, âge={3}, valide={4}, erreurs={5}",
6.         RouteData.Values["controller"], RouteData.Values["action"], poids, age, ModelState.IsValid, erreurs);
7.     return Content(texte, "text/plain", Encoding.UTF8);

```

- ligne 2 : nous avons deux paramètres [poids] et [age].

Les règles décrites précédemment s'appliquent maintenant aux deux paramètres. Voici quelques exemples d'exécution :



#### 1.4.4 Utiliser une classe comme modèle d'une action

Définissons une classe qui sera le modèle d'une action. Nous la mettons dans le dossier [Models] [1].



Son code sera le suivant :

```

1. namespace Exemple_02.Models
2. {
3.     public class ActionModel01
4.     {
5.         public double? Poids { get; set; }
6.         public int? Age { get; set; }
7.     }
8. }

```

Notre classe a comme propriétés automatiques, les deux paramètres [Poids] et [Age] étudiés précédemment. Cette classe sera le paramètre d'entrée de l'action [Action07] :

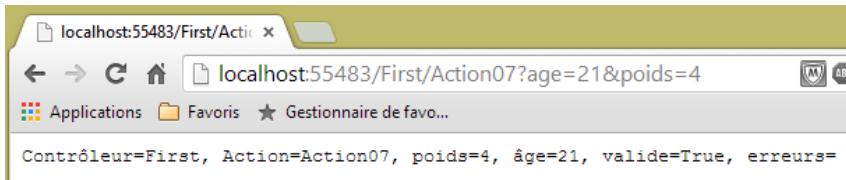
```

1. // Action07
2. public ContentResult Action07(ActionModel01 modèle)
3. {
4.     string erreurs = getErrorMessagesFor(ModelState);
5.     string texte = string.Format("Contrôleur={0}, Action={1}, poids={2}, âge={3}, valide={4}, erreurs={5}",
6.         RouteData.Values["controller"], RouteData.Values["action"], modèle.Poids, modèle.Age, ModelState.IsValid, erreurs);
7.     return Content(texte, "text/plain", Encoding.UTF8);
}

```

- ligne 2 : le modèle de l'action est une instance de type [ActionModel01].

Reprenons les mêmes deux exemples que précédemment :



On remarquera que la liaison des paramètres n'est pas sensible à la casse. Les paramètres de la requête étaient [age] et [poids]. Ils ont alimenté les propriétés [Age] et [Poids] de la classe [ModelAction01].

Par ailleurs, nous avons jusqu'à maintenant utilisé des requêtes HTTP [GET]. Montrons que les requêtes [POST] ont le même comportement. Pour cela utilisons de nouveau l'application [Advanced Rest Client] :

- en [1], l'URL demandée ;
- en [2], elle le sera par une commande POST ;
- en [3], les paramètres du POST.

On obtient la même réponse qu'avec le GET :

#### 1.4.5 Modèle de l'action avec contraintes de validité - 1

Avec le modèle précédent :

```

1.  namespace Exemple_02.Models
2.  {
3.      public class ActionModel01
4.      {
5.          public double? Poids { get; set; }
6.          public int? Age { get; set; }
7.      }
8.  }

```

les paramètres [poids] et [age] peuvent être absents de la requête. Dans ce cas, les propriétés [Poids] et [Age] reçoivent la valeur [null] et aucune erreur n'est signalée. On pourrait vouloir transformer le modèle comme suit :

```

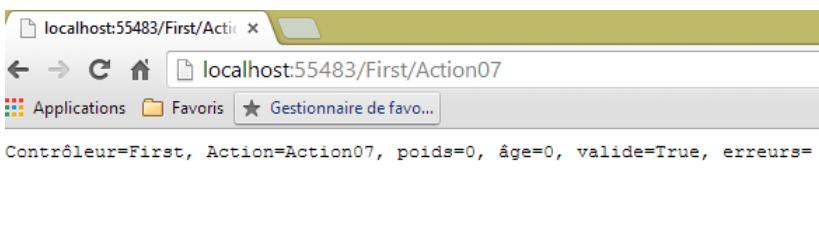
1.  namespace Exemple_02.Models
2.  {
3.      public class ActionModel01
4.      {

```

```

5.     public double Poids { get; set; }
6.     public int Age { get; set; }
7.   }
8. }
```

Lignes 5 et 6, les propriétés [Poids] et [Age] ne peuvent plus avoir la valeur [null]. Voyons ce qui se passe avec ce nouveau modèle lorsque les paramètres [poids] et [age] sont absents de la requête.

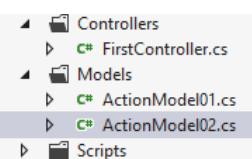


Il n'y a pas eu d'erreurs et les propriétés [Poids] et [Age] ont gardé leur valeur d'initialisation : 0. ASP.NET MVC :

- a créé une instance du modèle par un *new ActionModel01*. C'est là que les propriétés [Poids] et [Age] ont reçu leur valeur 0 ;
- n'a affecté aucune valeur à ces deux propriétés car il n'y avait aucun paramètre portant leur nom.

Le premier modèle nous permet de vérifier l'absence d'un paramètre : la propriété correspondante a alors la valeur [null]. Le second ne nous le permet pas. Il est possible d'ajouter d'autres contraintes de validation que le simple type des paramètres. Nous allons maintenant les présenter.

Considérons le nouveau modèle d'action suivant :



```

1. using System.ComponentModel.DataAnnotations;
2. namespace Exemple_02.Models
3. {
4.   public class ActionModel02
5.   {
6.     [Required]
7.     [Range(1, 200)]
8.     public double? Poids { get; set; }
9.     [Required]
10.    [Range(1, 150)]
11.    public int? Age { get; set; }
12.  }
13. }
```

- ligne 6 : indique que le champ [Poids] est requis ;
- ligne 7 : indique que le champ [Poids] doit être dans l'intervalle [1,200] ;
- ligne 9 : indique que le champ [Age] est requis ;
- ligne 7 : indique que le champ [Age] doit être dans l'intervalle [1,150] ;

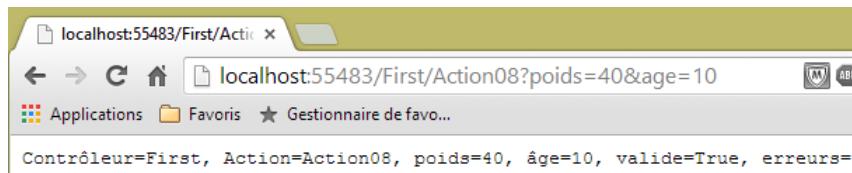
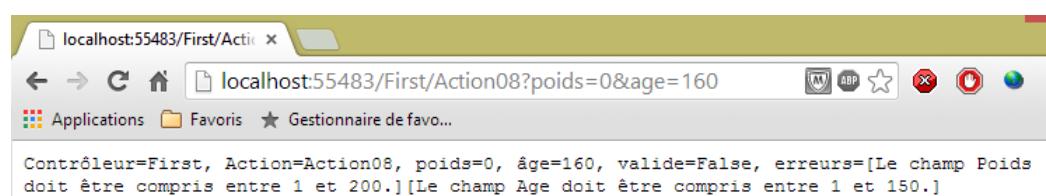
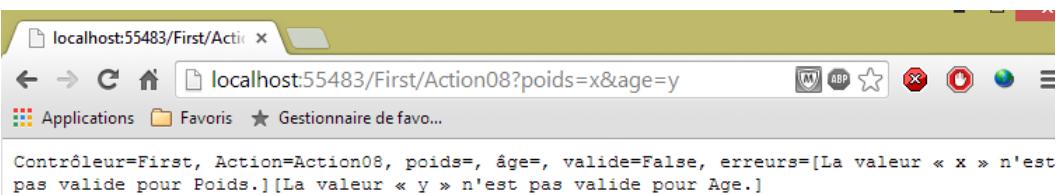
L'action utilisant ce modèle sera l'action [Action08] suivante :

```

1. // Action08
2. public ContentResult Action08(ActionModel02 modèle)
3. {
4.   string erreurs = getErrorMessagesFor(ModelState);
5.   string texte = string.Format("Contrôleur={0}, Action={1}, poids={2}, âge={3}, valide={4}, erreurs={5}",
6.     RouteData.Values["controller"], RouteData.Values["action"], modèle.Poids, modèle.Age, ModelState.IsValid, erreurs);
7.   return Content(texte, "text/plain", Encoding.UTF8);
}
```

- ligne 2 : l'action reçoit une instance du modèle [ActionModel02] ;

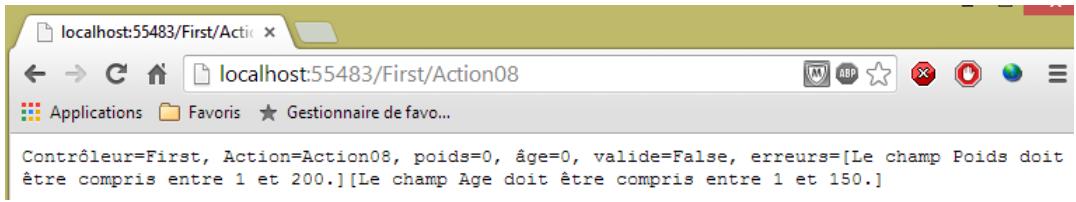
Faisons quelques tests :



Les erreurs sont bien détectées. Maintenant, faisons évoluer le modèle comme suit :

```
1. using System.ComponentModel.DataAnnotations;
2. namespace Exemple_02.Models
3. {
4.     public class ActionModel02
5.     {
6.         [Required]
7.         [Range(1, 200)]
8.         public double Poids { get; set; }
9.         [Required]
10.        [Range(1, 150)]
11.        public int Âge { get; set; }
12.    }
13. }
```

Lignes 8 et 11, les propriétés ne peuvent plus avoir la valeur [null]. Compilons et refaisons le test sans paramètres :

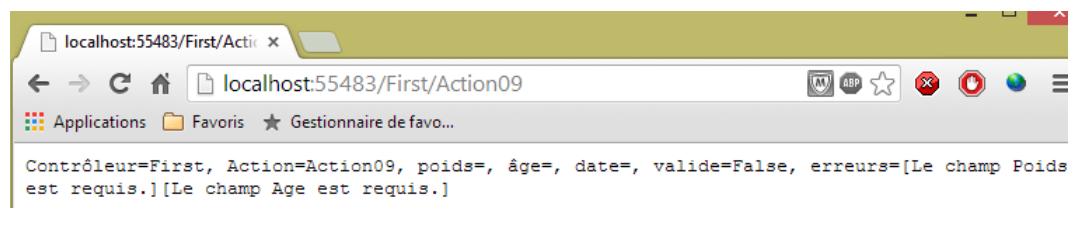


L'absence de paramètres a fait que les propriétés [Poids] et [Age] ont gardé leur valeur acquise lors de l'instanciation du modèle : 0. La validation intervient ensuite. L'attribut [Required] est alors satisfait. On voit que le message d'erreur ci-dessus est celui de l'attribut [Range]. Donc pour vérifier la présence d'un paramètre, il faut que la propriété associée soit *nullable*, c-à-d puisse recevoir la valeur *null*.

Revenons au modèle [ActionModel02] initial et considérons une action dont le modèle est constitué d'une instance [ActionModel02] et d'un type [DateTime] *nullable*:

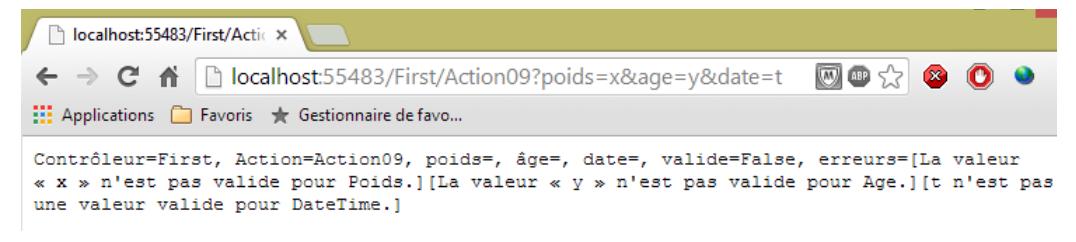
```
1.      // Action09
2.      public ContentResult Action09(ActionModel02 modèle, DateTime? date)
3.      {
4.          string erreurs = getErrorMessagesFor(ModelState);
5.          string texte = string.Format("Contrôleur={0}, Action={1}, poids={2}, âge={3}, date={4}, valide={5},"
6.              erreurs={6}", RouteData.Values["controller"], RouteData.Values["action"], modèle.Poids, modèle.Age, date,
7.              ModelState.IsValid, erreurs);
7.          return Content(texte, "text/plain", Encoding.UTF8);
    }
```

Faisons quelques tests :

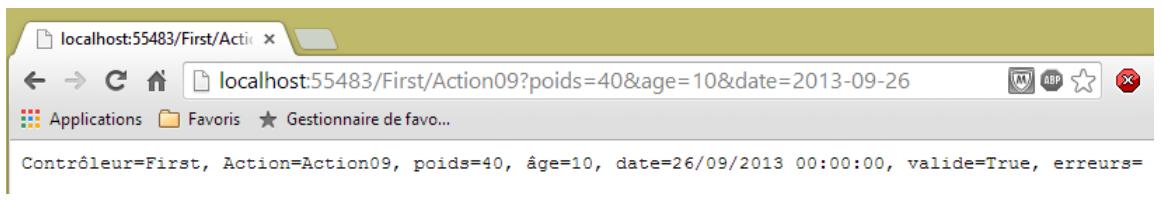


On n'a pas passé de paramètres à l'action. Les attributs [Required] des propriétés [Poids] et [Age] ont joué leur rôle. La date elle, a reçu la valeur *null* et aucune erreur n'a été signalée.

On passe maintenant des paramètres invalides :



On passe maintenant des valeurs valides :



Examinons d'autres contraintes de validité. Le nouveau modèle d'action est le suivant :

```

1.  using System.ComponentModel.DataAnnotations;
2.  namespace Exemple_02.Models
3.  {
4.      public class ActionModel03
5.      {
6.          [Required(ErrorMessage = "Le paramètre email est requis")]
7.          [EmailAddress(ErrorMessage = "Le paramètre email n'a pas un format valide")]
8.          public string Email { get; set; }
9.
10.         [Required(ErrorMessage = "Le paramètre jour est requis")]
11.         [RegularExpression(@"^\d{1,2}$", ErrorMessage = "Le paramètre jour doit avoir 1 ou 2 chiffres")]
12.         public string Jour { get; set; }
13.
14.         [Required(ErrorMessage = "Le paramètre info1 est requis")]
15.         [MaxLength(4, ErrorMessage = "Le paramètre info1 ne peut avoir plus de 4 caractères")]
16.         public string Info1 { get; set; }
17.
18.         [Required(ErrorMessage = "Le paramètre info2 est requis")]
19.         [MinLength(2, ErrorMessage = "Le paramètre info2 ne peut avoir moins de 2 caractères")]
20.         public string Info2 { get; set; }
21.
22.         [Required(ErrorMessage = "Le paramètre info3 est requis")]
23.         [MinLength(4, ErrorMessage = "Le paramètre info3 doit avoir 4 caractères exactement")]
24.         [MaxLength(4, ErrorMessage = "Le paramètre info3 doit avoir 4 caractères exactement")]
25.         public string Info3 { get; set; }
26.     }
27. }
```

- ligne 6 : l'attribut [Required] avec cette fois, un message d'erreur que nous fixons nous mêmes ;
- ligne 7 : l'attribut [EmailAddress] demande à ce que le champ [Email] contienne une adresse électronique de format valide ;
- ligne 11 : l'attribut [RegularExpression] demande à ce que le champ [Jour] contienne une chaîne d'un ou deux chiffres. Le premier paramètre est l'expression régulière que doit vérifier le champ ;
- ligne 15 : l'attribut [MaxLength] demande à ce que le champ [Info1] ait au plus 4 caractères ;
- ligne 19 : l'attribut [MinLength] demande à ce que le champ [Info2] ait au moins 2 caractères ;
- lignes 23-24 : les attributs [MaxLength] et [MinLength] combinés demandent à ce que le champ [Info3] ait exactement 4 caractères ;

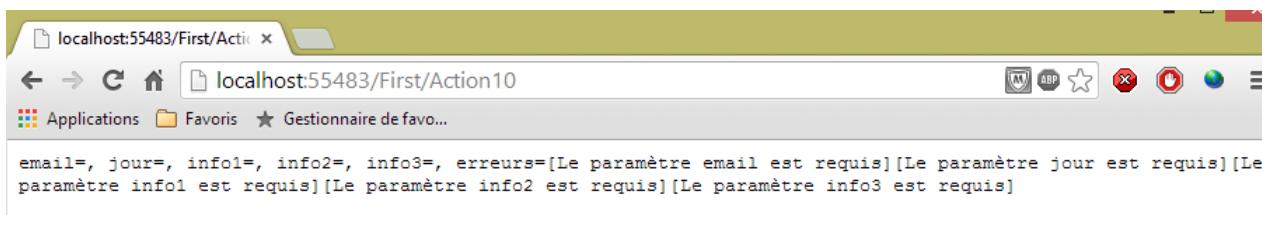
L'action [Action10] utilisera ce modèle :

```

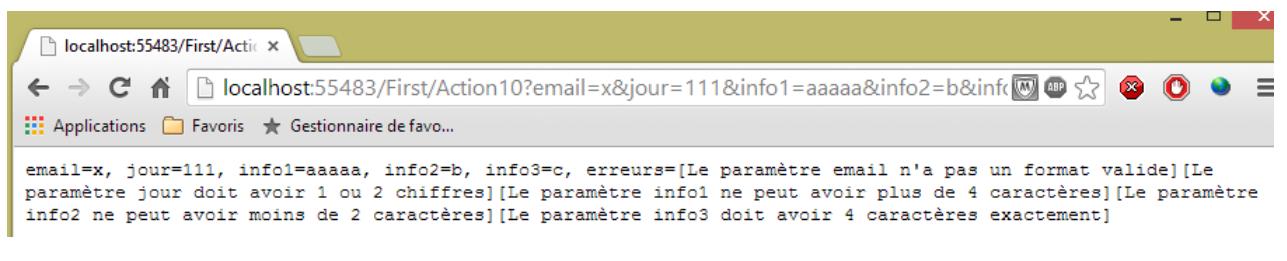
1.  // Action10
2.  public ContentResult Action10(ActionModel03 modèle)
3.  {
4.      string erreurs = getErrorMessagesFor ModelState);
5.      string texte = string.Format("email={0}, jour={1}, info1={2}, info2={3}, info3={4}, erreurs={5}",
6.          modèle.Email, modèle.Jour, modèle.Info1, modèle.Info2, modèle.Info3, erreurs);
7.      return Content(texte, "text/plain", Encoding.UTF8);
8.  }
```

Faisons quelques tests avec cette action.

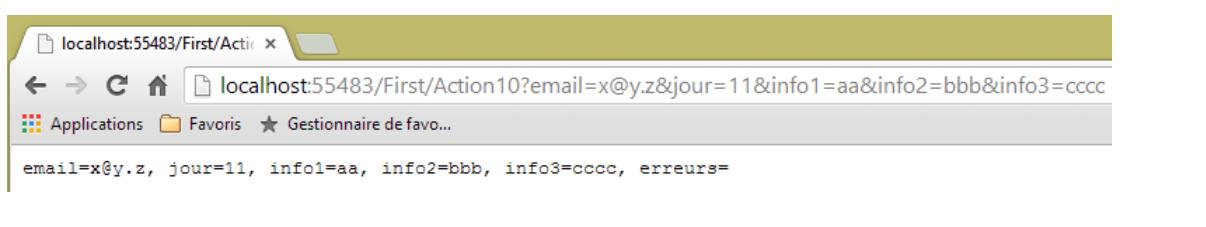
Tout d'abord sans paramètres :



Puis avec des paramètres invalides :

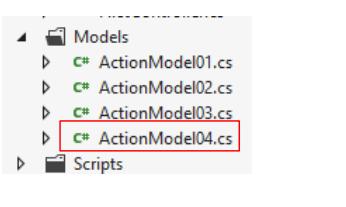


Puis avec des paramètres valides :



## 1.4.6 Modèle de l'action avec contraintes de validité - 2

Nous présentons d'autres contraintes d'intégrité. Le nouveau modèle de l'action sera la classe [ActionModel04] suivante :



```
1. using System.ComponentModel.DataAnnotations;
2.
3. namespace Exemple_02.Models
4. {
5.     public class ActionModel04
6.     {
7.         [Required(ErrorMessage="Le paramètre url est requis")]
8.         [Url(ErrorMessage="URL invalide")]
9.         public string Url { get; set; }
10.        [Required(ErrorMessage = "Le paramètre info1 est requis")]
11.        public string Info1 { get; set; }
12.        [Required(ErrorMessage = "Le paramètre info2 est requis")]
13.        [Compare("Info1",ErrorMessage="Les paramètres info1 et info2 doivent être identiques")]
14.        public string Info2 { get; set; }
15.        [Required(ErrorMessage = "Le paramètre cc est requis")]
16.        [CreditCard(ErrorMessage = "Le paramètre cc n'est pas un n° de carte de crédit valide")]
17.        public string Cc { get; set; }
18.    }
```

19. }

- ligne 8 : demande à ce que le champ annoté soit une URL valide ;
- ligne 13 : demande à ce que les propriétés [Info1] et [Info2] aient la même valeur ;
- ligne 16 : demande à ce que le champ annoté soit un n° de carte de crédit valide.

L'action utilisant ce modèle sera la suivante :

```
1. // Action11
2. public ContentResult Action11(ActionModel04 modèle)
3. {
4.     string erreurs = getErrorMessagesFor(ModelState);
5.     string texte = string.Format("URL={0}, Info1={1}, Info2={2}, CC={3}, erreurs={4}",
6.         modèle.Url, modèle.Info1, modèle.Info2, modèle.Cc, erreurs);
7.     return Content(texte, "text/plain", Encoding.UTF8);
8. }
```

Pour tester l'action [Action11], nous utilisons l'application [Advanced Rest Client] :

The screenshot shows the Advanced Rest Client interface. At the top, the URL is `http://localhost:55483/First/Action11` (1). Below it, the method is set to `POST` (2). The `Headers` tab is selected (3). In the `Payload` section, there are four fields: `url` (value: `http://machine.com:124/un/chemin`), `info1` (value: `xx`), `info2` (value: `xx`), and `cc` (value: `0123012301230123`) (4). The `Raw` tab shows the URL-encoded payload: `url=http%3A%2F%2Fmachine.com%3A124%2Fun%2Fchemin&info1=xx&info2=xx&cc=0123012301230123` (5).

- en [1], l'URL de l'action [Action11] ;
- en [2], cette URL sera demandée avec un POST ;
- en [3], on sélectionne l'onglet [Form] ;
- en [4], les valeurs des quatre paramètres attendus. Cette initialisation est une facilité offerte par [ARC]. Les paramètres réellement envoyés peuvent être vus dans l'onglet [Raw] [5] ;

The screenshot shows the Advanced Rest Client interface with the `Raw` tab selected (6). The payload is the same as in the previous screenshot: `url=http%3A%2F%2Fmachine.com%3A124%2Fun%2Fchemin&info1=xx&info2=xx&cc=0123012301230123`.

- en [6], les paramètres du POST.

Pour cette requête, on reçoit la réponse suivante :

Raw    Parsed    Response

[Open output in new window](#) [Copy to clipboard](#) [Save as file](#) [Open in JSON tab](#)

URL=http://machine.com:124/un/chemin, Info1=xx, Info2=xx, CC=0123012301230123, erreurs=[Le paramètre cc n'est pas un n° de carte de crédit valide]

Passons des paramètres invalides :

▶ <http://localhost:55483/First/Action11>

GET  POST  PUT  PATCH

Raw    Form    Headers

Add new header

Raw    Form    Files (0)

Add new value Values from here will be URL encoded

url	x
info1	y
info2	z
cc	t

application/x-www-form-urlencoded ▾

Nous obtenons alors la réponse suivante :

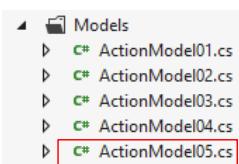
Raw    Parsed    Response

[Open output in new window](#) [Copy to clipboard](#) [Save as file](#) [Open in JSON tab](#)

URL=x, Info1=y, Info2=z, CC=t, erreurs=[URL invalide][Les paramètres info1 et info2 doivent être identiques][Le paramètre cc n'est pas un n° de carte de crédit valide]

#### 1.4.7 Modèle de l'action avec contraintes de validité - 3

Parfois les contraintes d'intégrité disponibles ne suffisent pas. On peut alors en créer soit même. On peut notamment utiliser un modèle implémentant l'interface [IValidatableObject]. Dans ce cas, on ajoute nos propres vérifications du modèle dans la méthode [Validate] de cette interface. Voyons un exemple. Le nouveau modèle de l'action sera la classe [ActionModel05] suivante :



```

1.  using System.Collections.Generic;
2.  using System.ComponentModel.DataAnnotations;
3.
4.  namespace Exemple_02.Models
5.  {
6.      public class ActionModel05 : IValidatableObject

```

```

7.      {
8.          [Required(ErrorMessage = "Le paramètre taux est requis")]
9.          public double? Taux { get; set; }
10.         public IEnumerable<ValidationResult> Validate(ValidationContext validationContext)
11.         {
12.             List<ValidationResult> résultats = new List<ValidationResult>();
13.             bool ok = Taux < 4.2 || Taux > 6.7;
14.             if (!ok)
15.             {
16.                 résultats.Add(new ValidationResult("Le paramètre taux doit être < 4.2 ou > 6.7", new string[] { "Taux" }));
17.             }
18.             return résultats;
19.         }
20.     }
21. }
```

- ligne 6 : le modèle implémente l'interface [IValidatableObject] ;
- ligne 10 : la méthode [Validate] de cette interface. Elle rend une collection d'éléments de type [ValidationResult]. Ce type encapsule les erreurs que l'on veut signaler ;
- ligne 9 : un taux valide est un taux <4.2 ou > 6.7 ;
- ligne 12 : on crée une liste vide d'éléments de type [ValidationResult] ;
- ligne 13 : on vérifie la validité de la propriété [Taux] ;
- lignes 14-17 : si la propriété [Taux] est invalide, alors on ajoute un élément de type [ValidationResult] dans la liste des résultats. Le premier paramètre est un message d'erreur. Le second paramètre, facultatif, est une collection des propriétés concernées par cette erreur.

L'action utilisant ce modèle sera la suivante :

```

1.    // Action12
2.    public ContentResult Action12(ActionModel05 modèle)
3.    {
4.        string erreurs = getErrorMessagesFor(ModelState);
5.        string texte = string.Format("taux={0}, erreurs={1}", modèle.Taux, erreurs);
6.        return Content(texte, "text/plain", Encoding.UTF8);
7.    }
```

Voici un exemple d'exécution :



## 1.4.8 Modèle d'action de type Tableau ou Liste

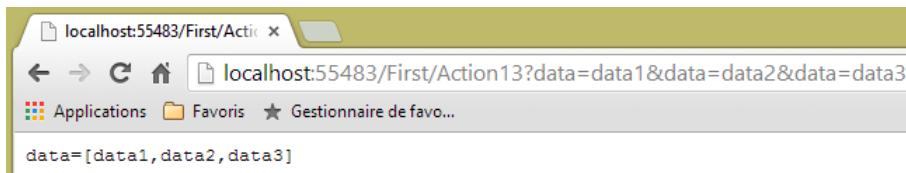
Considérons l'action [Action13] suivante :

```

1.    // Action13
2.    public ContentResult Action13(string[] data)
3.    {
4.        string strData = "";
5.        if (data != null & data.Length != 0)
6.        {
7.            strData = string.Join(", ", data);
8.        }
9.        string texte = string.Format("data=[{0}]", strData);
10.       return Content(texte, "text/plain", Encoding.UTF8);
11.    }
```

- ligne 2 : le modèle de l'action est constitué d'un tableau de [string]. Il nous permet de récupérer un paramètre nommé [data] qui peut être présent plusieurs fois dans les paramètres de la requête comme dans [?data=data1&data=data2&data=data3]. Les différents paramètres [data] de la requête vont alimenter le tableau [data] du modèle de l'action. Ce cas se rencontre avec les listes à choix multiple. Le navigateur envoie alors les différentes valeurs sélectionnées par l'utilisateur, avec le même nom de paramètre.

Voici un exemple :



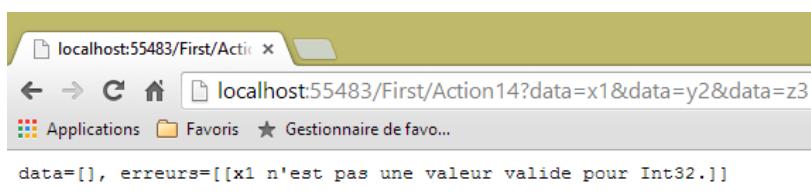
Le modèle peut être également une liste :

```
1. // Action14
2. public ContentResult Action14(List<int> data)
3. {
4.     string erreurs = getErrorMessagesFor(ModelState);
5.     string strData = "";
6.     if (data != null && data.Count != 0)
7.     {
8.         strData = string.Join(", ", data);
9.     }
10.    string texte = string.Format("data=[{0}], erreurs=[{1}]", strData, erreurs);
11.    return Content(texte, "text/plain", Encoding.UTF8);
12. }
```

Le modèle est ici une liste d'entiers (ligne 2). Voici une première exécution :



et une seconde :



## 1.4.9 Filtrage d'un modèle d'action

Parfois nous disposons d'un modèle mais nous souhaitons que seuls certains éléments du modèle soient initialisés par la requête HTTP. Considérons le modèle d'action [ActionModel06] suivant :

```
1. using System.ComponentModel.DataAnnotations;
2. using System.Web.Mvc;
3.
4. namespace Exemple_02.Models
5. {
6.     [Bind(Exclude = "Info2")]
7.     public class ActionModel06
8.     {
9.         [Required(ErrorMessage = "Le paramètre [info1] est requis")]
10.        public string Info1 { get; set; }
11.
12.        public string Info2 { get; set; }
13.    }
14. }
```

- lignes 9-10 : le paramètre [info1] est obligatoire ;
- ligne 6 : le paramètre [info2] de la ligne 12 est exclu de la liaison de la requête HTTP à son modèle.

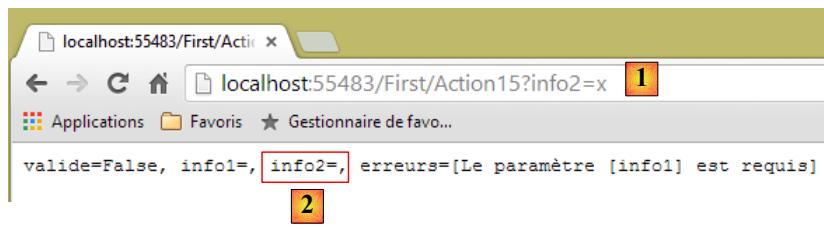
L'action sera la suivante [Action15] :

```

1.  // Action15
2.  public ContentResult Action15(ActionModel06 modèle)
3.  {
4.      string erreurs = getErrorMessagesFor ModelState;
5.      string texte = string.Format("valide={0}, info1={1}, info2={2}, erreurs={3}", ModelState.IsValid, modèle.Info1,
    modèle.Info2, erreurs);
6.      return Content(texte, "text/plain", Encoding.UTF8);
7.  }

```

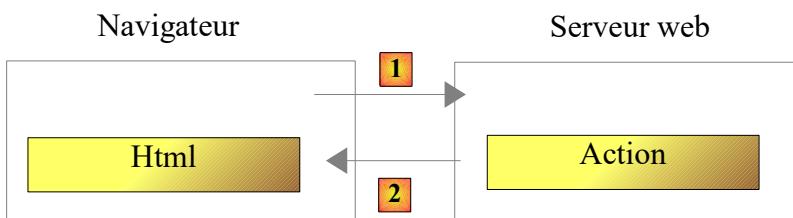
Voici un exemple d'exécution :



- en [1] : on passe le paramètre [info2] dans l'URL ;
- en [2] : la propriété [Info2] du modèle de l'action est restée vide.

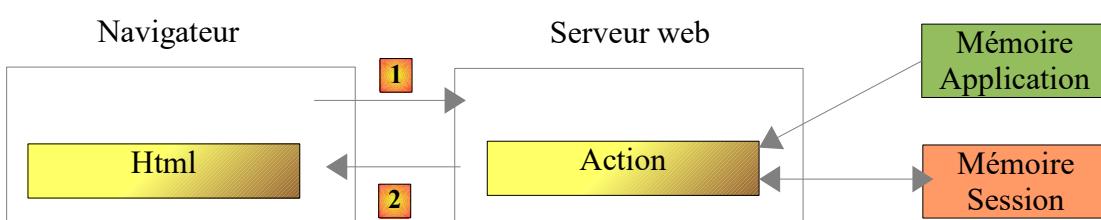
#### 1.4.10 Étendre le modèle de liaison des données

Revenons sur l'architecture d'exécution d'une action :



La classe de l'action est instanciée au début de la requête du client et détruite à la fin de celle-ci. Aussi ne peut-elle servir à mémoriser des données entre deux requêtes même si elle est appelée de façon répétée. On peut vouloir mémoriser deux types de données :

- des données **partagées par tous les utilisateurs** de l'application web. Ce sont en général des données en lecture seule. Trois fichiers sont utilisés pour mettre en oeuvre ce partage de données :
  - [Web.Config] : le fichier de configuration de l'application
  - [Global.asax, Global.asax.cs] : permettent de définir une classe, appelée classe globale d'application, dont la durée de vie est celle de l'application, ainsi que des gestionnaires pour certains événements de cette même application.
- La classe globale d'application permet de définir des données qui seront disponibles pour toutes les requêtes de tous les utilisateurs.
- des données partagées par les requêtes **d'un même client**. Ces données sont mémorisées dans un objet appelé **Session**. On parle alors de **session client** pour désigner la mémoire du client. Toutes les requêtes d'un client ont accès à cette session. Elles peuvent y stocker et y lire des informations.



Ci-dessus, nous montrons les types de mémoire auxquels a accès une action :

- la mémoire de l'application qui contient la plupart du temps des données en lecture seule et qui est accessible à tous les utilisateurs ;
- la mémoire d'un utilisateur particulier, ou session, qui contient des données en lecture / écriture et qui est accessible aux requêtes successives d'**un même utilisateur** ;
- non représentée ci-dessus, il existe une mémoire de requête, ou contexte de requête. La requête d'un utilisateur peut être traitée par plusieurs actions successives. Le contexte de la requête permet à une action 1 de transmettre de l'information à une action 2.

Regardons un premier exemple mettant en lumière ces différentes mémoires :

Tout d'abord, nous modifions le fichier [Web.config] du projet [Exemple-02] de la façon suivante :

```
1.  <appSettings>
2.    <add key="webpages:Version" value="2.0.0.0" />
3.    ...
4.    <add key="infoAppli1" value="infoAppli1"/>
5.  </appSettings>
```

Nous rajoutons la ligne 4 qui à la clé [infoAppli1] associe la valeur [infoAppli1]. Ce sera notre donnée de portée [Application] : elle sera accessible à toutes les requêtes de tous les utilisateurs.

Ensuite nous modifions la méthode [Application\_Start] du fichier [Global.asax]. Cette méthode s'exécute une unique fois au démarrage de l'application. C'est là qu'il faut exploiter le fichier [Web.config] :

```
1.  protected void Application_Start()
2.  {
3.    AreaRegistration.RegisterAllAreas();
4.
5.    WebApiConfig.Register(GlobalConfiguration.Configuration);
6.    FilterConfig.RegisterGlobalFilters(GlobalFilters.Filters);
7.    RouteConfig.RegisterRoutes(RouteTable.Routes);
8.    BundleConfig.RegisterBundles(BundleTable.Bundles);
9.    // initialisation application
10.   Application["infoAppli1"] = ConfigurationManager.AppSettings["infoAppli1"];
11. }
```

Nous rajoutons la ligne 10. Elle fait deux choses :

- elle récupère la valeur de la clé [infoAppli1] dans le fichier [Web.config] au moyen de la classe [System.Configuration.ConfigurationManager] ;
- elle l'enregistre dans le dictionnaire [HttpApplication.Application], associée à la clé [infoAppli1]. Toutes les actions ont accès à ce dictionnaire.

Dans le même fichier [Gloabal.asax], on ajoute la méthode [Session\_Start] suivante :

```
1.  protected void Session_Start()
2.  {
3.    // initialisation compteur
4.    Session["compteur"] = 0;
5. }
```

La méthode [Session\_Start] est exécutée pour tout nouvel utilisateur. Qu'est-ce qu'un nouvel utilisateur ? Un utilisateur est "suivi" par un jeton de session. Ce jeton est :

- créé par le serveur web et envoyé au nouvel utilisateur dans les entêtes HTTP de la première réponse qui lui est faite ;
- renvoyé par le navigateur de l'utilisateur à chaque nouvelle requête qu'il fait. Cela permet au serveur de reconnaître l'utilisateur et de gérer une mémoire pour lui qu'on appelle la session de l'utilisateur.

Le serveur web reconnaît qu'il a affaire à un nouvel utilisateur lorsque celui-ci ne lui envoie pas de jeton de session. Le serveur lui en crée alors un.

Ligne 4 ci-dessus, on met dans la session de l'utilisateur un compteur qui sera incrémenté à chaque requête de cet utilisateur. Cela illustrera la mémoire associée à un utilisateur. La classe [Session] s'utilise comme un dictionnaire (ligne 4).

Ceci fait, nous écrivons l'action [Action16] suivante :

```
1.  // Action16
2.  public ContentResult Action16()
3.  {
4.    // on récupère le contexte de la requête HTTP
5.    HttpContextBase contexte = ControllerContext.HttpContext;
6.    // on récupère les infos de portée Application
```

```

7.     string infoAppli1 = contexte.Application["infoAppli1"] as string;
8.     // et celles de portée Session
9.     int? compteur = contexte.Session["compteur"] as int?;
10.    compteur++;
11.    contexte.Session["compteur"] = compteur;
12.    // la réponse au client
13.    string texte = string.Format("infoAppli1={0}, compteur={1}", infoAppli1, compteur);
14.    return Content(texte, "text/plain", Encoding.UTF8);
15. }

```

- ligne 5 : on récupère le contexte de la requête HTTP en cours de traitement. Ce contexte va nous donner accès aux données de portée [Application] et [Session] ;
- ligne 7 : on récupère l'information de portée [Application] ;
- ligne 9 : on récupère le compteur dans la session ;
- lignes 10-11 : il est incrémenté puis remis dans la session ;
- lignes 13-14 : les deux informations sont envoyées au client.

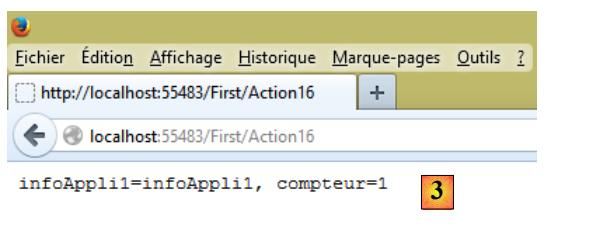
Voici des exemples d'exécution :

[Action16] est demandée une première fois [1] puis la page est rafraîchie [F5] deux fois [2] :



En [2], le client a fait au total trois requêtes. A chaque fois, il a pu récupérer le compteur mis à jour par la précédente requête.

Pour simuler un second utilisateur, nous utilisons un deuxième navigateur pour demander la même URL :



En [3], le second utilisateur récupère bien la même information de portée [Application] mais a son propre compteur de portée [Session].

Revenons au code de l'action [Action16] :

```

1. // Action16
2. public ContentResult Action16()
3. {
4.     // on récupère le contexte de la requête HTTP
5.     HttpContextBase contexte = ControllerContext.HttpContext;
6.     // on récupère les infos de portée Application
7.     string infoAppli1 = contexte.Application["infoAppli1"] as string;
8.     // et celles de portée Session
9.     int? compteur = contexte.Session["compteur"] as int?;
10.    compteur++;
11.    contexte.Session["compteur"] = compteur;
12.    // la réponse au client
13.    string texte = string.Format("infoAppli1={0}, compteur={1}", infoAppli1, compteur);
14.    return Content(texte, "text/plain", Encoding.UTF8);
15. }

```

L'un des buts du framework ASP.NET MVC est de rendre les contrôleurs et les actions testables de façon isolée sans serveur web. Or on voit ligne 5, que le contexte de la requête HTTP est nécessaire pour récupérer les informations de portée [Application] et de portée [Session]. On se propose de créer une nouvelle action [Action17] qui recevrait les données de portée [Application] et [Session] en paramètres :

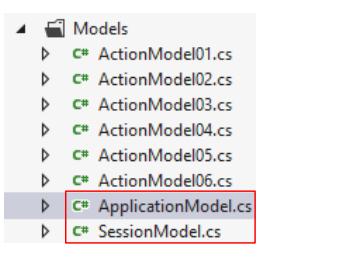
```

1.    // Action17
2.    public ContentResult Action17(ApplicationModel applicationData, SessionModel sessionData)
3.    {
4.        // on récupère les infos de portée Application
5.        string infoAppli1 = applicationData.InfoAppli1;
6.        // et celles de portée Session
7.        int compteur = sessionData.Compteur++;
8.        // la réponse au client
9.        string texte = string.Format("infoAppli1={0}, compteur={1}", infoAppli1, compteur);
10.       return Content(texte, "text/plain", Encoding.UTF8);
11.    }

```

Le code ne comporte désormais plus de dépendances envers la requête HTTP. Elle peut donc être testée isolément d'un serveur web.

Voyons comment y arriver. Tout d'abord, il nous faut créer les classes [ApplicationModel] et [SessionModel] qui vont encapsuler respectivement les données de portée [Application] et [Session]. Ce sont les suivantes :



```

1. namespace Exemple_02.Models
2. {
3.     public class ApplicationModel
4.     {
5.         public string InfoAppli1 { get; set; }
6.     }
7. }

1. namespace Exemple_02.Models
2. {
3.     public class SessionModel
4.     {
5.         public int Compteur { get; set; }
6.         public SessionModel()
7.         {
8.             Compteur = 0;
9.         }
10.    }
11. }

```

Ensuite, il nous faut modifier les méthodes [Application\_Start] et [Session\_Start] du fichier [Global.asax] :

```

1. public class MvcApplication : System.Web.HttpApplication
2. {
3.     protected void Application_Start()
4.     {
5.         AreaRegistration.RegisterAllAreas();
6.
7.         WebApiConfig.Register(GlobalConfiguration.Configuration);
8.         FilterConfig.RegisterGlobalFilters(GlobalFilters.Filters);
9.         RouteConfig.RegisterRoutes(RouteTable.Routes);
10.        BundleConfig.RegisterBundles(BundleTable.Bundles);
11.        // initialisation application - cas 1
12.        Application["infoAppli1"] = ConfigurationManager.AppSettings["infoAppli1"];
13.        // initialisation application - cas 2
14.        ApplicationModel data=new ApplicationModel();
15.        data.InfoAppli1=ConfigurationManager.AppSettings["infoAppli1"];
16.        Application["data"] = data;
17.    }
18.
19.    protected void Session_Start()
20.    {
21.        // initialisation compteur - cas 1
22.        Session["compteur"] = 0;
23.        // initialisation compteur - cas 2
24.        Session["data"] = new SessionModel();
25.    }
26. }

```

- ligne 14 : une instance de [ApplicationModel] est créée ;
- ligne 15 : elle est initialisée ;
- ligne 16 : et placée dans le dictionnaire de [Application], associée à la clé [data]. [Application] est une propriété de la classe [HttpApplication] de la ligne 1 ;
- ligne 24 : une instance de [SessionModel] est créée et placée dans le dictionnaire de [Session], associée à la clé [data]. [Session] est une propriété de la classe [HttpApplication] de la ligne 1 ;

Si on s'en tient à ce que nous avons vu jusqu'à maintenant, la signature

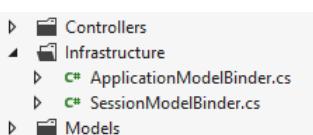
```
public ContentResult Action17(ApplicationModel applicationData, SessionModel sessionData)
```

signifie que la requête HTTP traitée par l'action devra comporter des paramètres nommés [applicationData] et [sessionData]. Ce ne sera pas le cas. Nous devons créer un nouveau modèle de liaison de données pour que lorsqu'une action reçoit en paramètre un type :

- [ApplicationModel], la donnée de portée [Application] et de clé [data] lui soit fournie ;
- [SessionModel], la donnée de portée [Session] et de clé [data] lui soit fournie.

Il faut pour cela créer des classes implémentant l'interface [IModelBinder].

Nous commençons par créer un dossier [Infrastructure] dans le projet [Exemple-02] :



Nous y créons la classe [ApplicationModelBinder] suivante :

```
1. using System.Web.Mvc;
2.
3. namespace Exemple_02.Infrastructure
4. {
5.     public class ApplicationModelBinder : IModelBinder
6.     {
7.         public object BindModel(ControllerContext controllerContext, ModelBindingContext bindingContext)
8.         {
9.             // on rend les données de portée [Application]
10.            return controllerContext.HttpContext.Application["data"];
11.        }
12.    }
13. }
```

- ligne 5 : la classe implémente l'interface [IModelBinder]. Pour comprendre son code, il faut savoir qu'elle sera appelée à chaque fois qu'une action aura un paramètre de type [ApplicationModel]. Cette liaison [ApplicationModel] --> [ApplicationModelBinder] sera faite au démarrage de l'application, dans la méthode [Application\_Start] de [Global.asax] ;
- ligne 7 : l'unique méthode de l'interface [IModelBinder] ;
- ligne 7 : le paramètre de type [ControllerContext] nous donne accès à la requête HTTP en cours de traitement ;
- ligne 7 : le paramètre de type [ModelBindingContext] nous donne accès à des informations sur le modèle à construire, ici le type [ApplicationModel] ;
- ligne 7 : le résultat de [BindModel] est l'objet qui va être affecté au paramètre lié, ici un paramètre de type [ApplicationModel] ;
- ligne 10 : nous nous contentons de rendre l'objet de portée [Application] et de clé [data].

La classe [SessionModelBinder] suit le même schéma :

```
1. using System.Web.Mvc;
2.
3. namespace Exemple_02.Infrastructure
4. {
5.     public class SessionModelBinder : IModelBinder
6.     {
7.         public object BindModel(ControllerContext controllerContext, ModelBindingContext bindingContext)
8.         {
9.             // on rend les données de portée [Session]
10.            return controllerContext.HttpContext.Session["data"];
11.        }
12.    }
13. }
```

```
12.    }
13. }
```

Il ne nous reste plus qu'à associer chacun des modèles [XModel] à son *binder* [XModelBinder]. Ceci est fait dans la méthode [Application\_Start] de [Global.asax] :

```
1.     protected void Application_Start()
2.     {
3.     ....
4.         // initialisation application - cas 2
5.         ApplicationModel data=new ApplicationModel();
6.         data.InfoAppli1=ConfigurationManager.AppSettings["infoAppli1"];
7.         Application["data"] = data;
8.         // model binders
9.         ModelBinders.Binders.Add(typeof(ApplicationModel), new ApplicationModelBinder());
10.        ModelBinders.Binders.Add(typeof(SessionModel), new SessionModelBinder());
11.    }
```

- ligne 9 : lorsqu'une action aura un paramètre de type [ApplicationModel], la méthode [ApplicationModelBinder.Bind] sera appelée. On sait qu'elle rend la donnée de portée [Application] associée à la clé [data] ;
- ligne 10 : idem pour le type [SessionModel].

Revenons à notre action [Action17] :

```
1.     // Action17
2.     public ContentResult Action17(ApplicationModel applicationData, SessionModel sessionData)
3.     {
4.         // on récupère les infos de portée Application
5.         string infoAppli1 = applicationData.InfoAppli1;
6.         // et celles de portée Session
7.         sessionData.Compteur++;
8.         int compteur = sessionData.Compteur;
9.         // la réponse au client
10.        string texte = string.Format("infoAppli1={0}, compteur={1}", infoAppli1, compteur);
11.        return Content(texte, "text/plain", Encoding.UTF8);
12.    }
```

- ligne 2 : lorsque [Action17] sera appelée, elle recevra pour
  - premier paramètre : la donnée de portée [Application] associée à la clé [data],
  - second paramètre : la donnée de portée [Session] associée à la clé [data] ;

Ces deux données peuvent être aussi complexes que l'on veut et regrouper pour l'une toutes les données de portée [Application] et pour l'autre toutes les données de portée [Session].

Voici un exemple d'exécution de l'action [Action17] :



#### 1.4.11 Liaison tardive du modèle de l'action

Nous avons écrit l'action [Action12] suivante :

```
1.     // Action12
2.     public ContentResult Action12(ActionModel05 modèle)
3.     {
4.         string erreurs = getErrorMessagesFor(ModelState);
5.         string texte = string.Format("taux={0}, erreurs={1}", modèle.Taux, erreurs);
6.         return Content(texte, "text/plain", Encoding.UTF8);
7.     }
```

De façon cachée, ASP.NET MVC :

- crée une instance de type [ActionModel05] en utilisant son constructeur sans paramètres ;

- l'initialise avec les informations de la requête qui ont le même nom (indifférent à la casse) que l'une des propriétés de [ActionModel05].

Parfois ce comportement ne nous convient pas. C'est notamment le cas lorsqu'on veut utiliser un constructeur particulier du modèle de l'action. On peut alors procéder de la façon suivante :

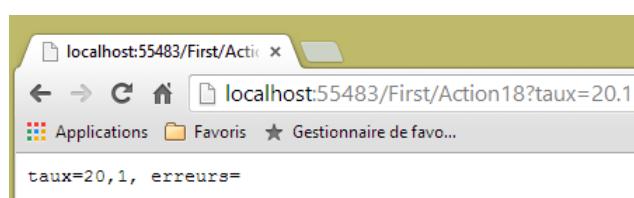
```

1. // Action18
2. public ContentResult Action18()
3. {
4.     ActionModel05 modèle = new ActionModel05();
5.     TryUpdateModel(modèle);
6.     string erreurs = getErrorMessagesFor(ModelState);
7.     string texte = string.Format("taux={0}, erreurs={1}", modèle.Taux, erreurs);
8.     return Content(texte, "text/plain", Encoding.UTF8);
9. }

```

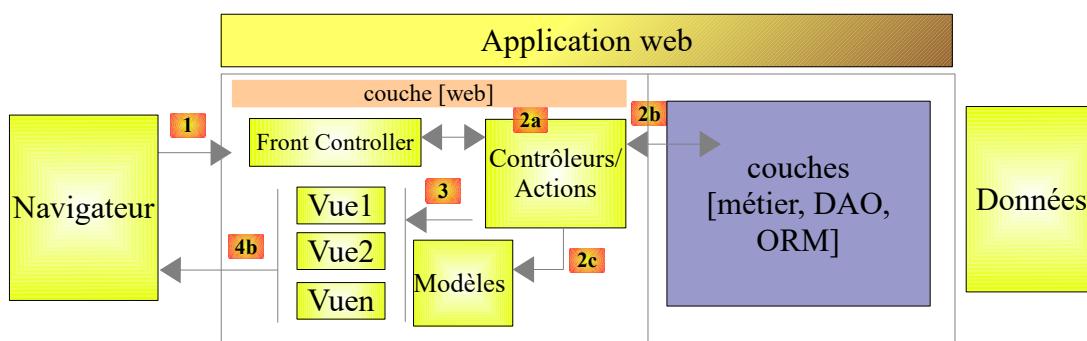
- ligne 2 : l'action ne reçoit plus de paramètres. Il n'y a donc plus de liaison automatique de données ;
- ligne 4 : nous créons nous-mêmes une instance du modèle de l'action. C'est là qu'on pourrait utiliser un constructeur différent ;
- ligne 5 : on initialise le modèle avec les informations de la requête. C'est ASP.NET MVC qui fait ce travail. Il le fait de la même façon qu'il l'aurait fait si le modèle avait été en paramètre ;
- ligne 6 : on est désormais dans la même situation que dans l'action [Action12].

Voici un exemple d'exécution :

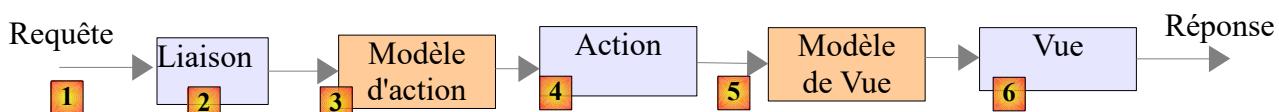


### 1.4.12 Conclusion

Revenons à l'architecture d'une application ASP.NET MVC :



Une requête [1] transporte avec elle diverses informations que ASP.NET MVC présente [2a] à l'action sous forme d'un **modèle** que nous avons appelé **modèle d'action**.



- la requête HTTP du client arrive en [1] ;
- en [2], les informations contenues dans la requête sont transformées en modèle d'action [3] ;
- en [4], l'action, à partir de ce modèle, va générer une réponse. Celle-ci aura deux composantes : une vue V [6] et le modèle M de cette vue [5] ;
- la vue V [6] va utiliser son modèle M [5] pour générer la réponse HTTP destinée au client.

Dans le modèle MVC, l'action [4] fait partie du C (contrôleur), le modèle de la vue [5] est le M et la vue [6] est le V.

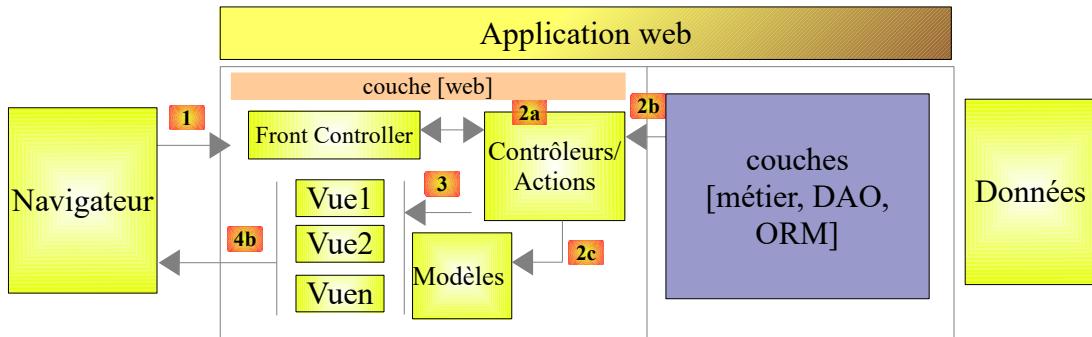
Ce chapitre a étudié les mécanismes de liaison entre les informations transportées par la requête, qui sont par nature des chaînes de caractères et le modèle de l'action qui peut être une classe avec des propriétés de divers types. Nous avons vu également qu'il était possible de vérifier la validité du modèle présenté à l'action. Enfin, nous avons vu comment élargir ce modèle aux données de portée [Session] et [Application].

Nous allons nous intéresser maintenant à la fin de la chaîne de traitement de la requête [1] : la création de la vue [6] et de son modèle [5]. Ces deux éléments sont produits par l'action [4].

## 1.5 La vue et son modèle

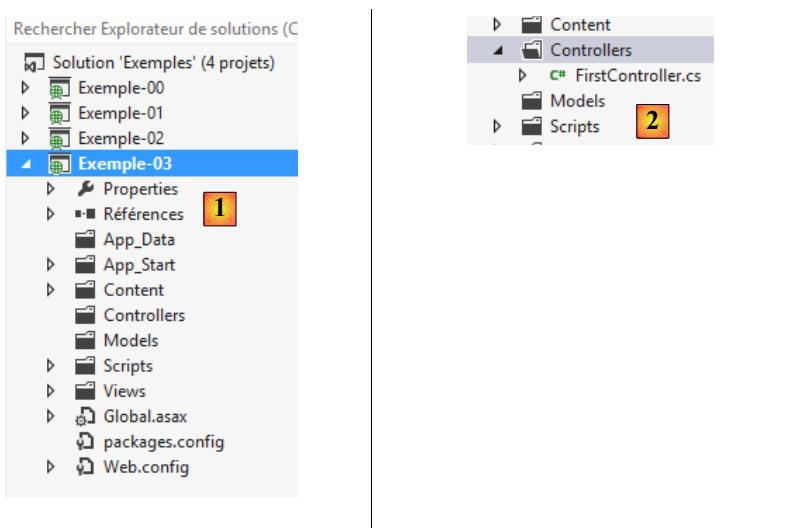
### 1.5.1 Introduction

Revenons à l'architecture d'une application ASP.NET MVC :



Dans le chapitre précédent, nous avons étudié comment ASP.NET MVC présentait les informations de la requête [1] à une action [2a] sous la forme d'un modèle qui pouvait contenir des contraintes de validation. Ce modèle était fourni en entrée à l'action et nous l'avons appelé le modèle de l'action. Nous nous intéressons maintenant au résultat le plus courant d'une action, le type [ViewResult] qui correspond à une vue **V** [3] accompagnée de son modèle **M** [2c]. Ce modèle sera appelé **modèle de la vue V**, à ne pas confondre avec le **modèle de l'action** que nous venons d'étudier. L'un est en entrée de l'action, l'autre est en sortie.

Commençons par créer un nouveau projet [Exemple-03] [1] toujours à l'intérieur de la même solution, de type ASP.NET MVC de base :



Créons un contrôleur nommé [First] [2]. Le code généré pour ce contrôleur est le suivant :

```
1.  using System.Web.Mvc;
2.
3.  namespace Exemple_03.Controllers
4.  {
5.      public class FirstController : Controller
6.      {
7.          public ActionResult Index()
8.          {
9.              return View();
10.         }
11.     }
12. }
```

- lignes 7-10 : une action [Index] a été créée. Le type du résultat de la méthode [Index] est celui de la classe [ActionResult] dont dérivent la plupart des résultats possibles d'une action ;
- ligne 9 : la méthode [View] de la classe [Controller] (ligne 5) rend un type [ViewResult] qui dérive de [ActionResult]. Cette méthode admet de nombreuses surcharges. Nous en verrons quelques unes. La principale est la suivante :

**C#**   **C++**   **F#**   **JScript**   **VB**

```
protected internal ViewResult View(
    string viewName,
    Object model
)
```

**Paramètres**

*viewName*  
Type : System.String  
Nom de la vue restituée dans la réponse.

*model*  
Type : System.Object  
Modèle qui est restitué par la vue.

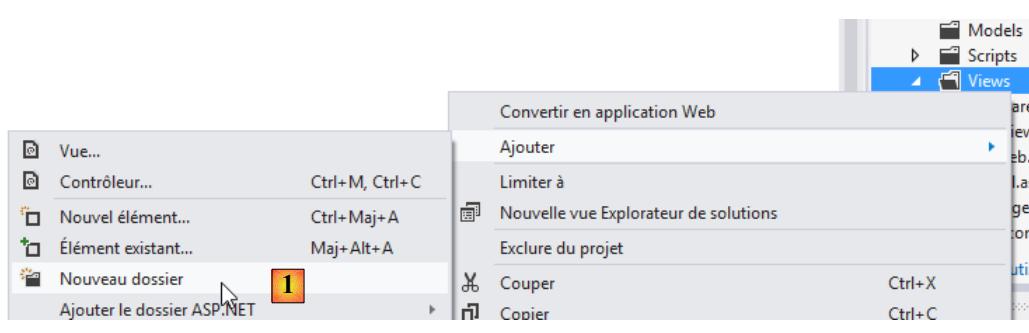
**Valeur de retour**  
Type : System.Web.Mvc.ViewResult  
Résultat de vue.

- le premier paramètre est le nom de la vue. S'il est absent, la vue utilisée est celle qui porte le même nom que l'action qui produit le [ViewResult] et qui sera cherchée dans le dossier [/Views/{controller}] où {controller} est le nom du contrôleur ;
- le second est le modèle de la vue. S'il est absent, la vue n'a pas de modèle.

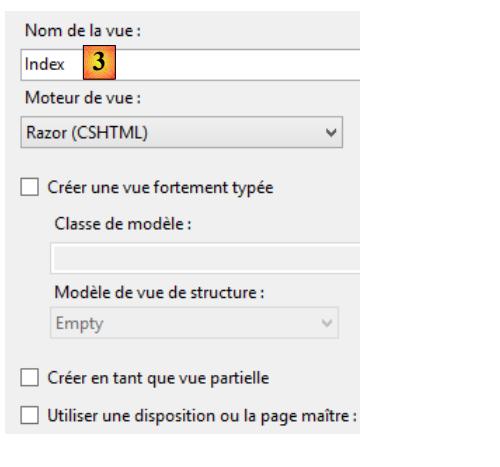
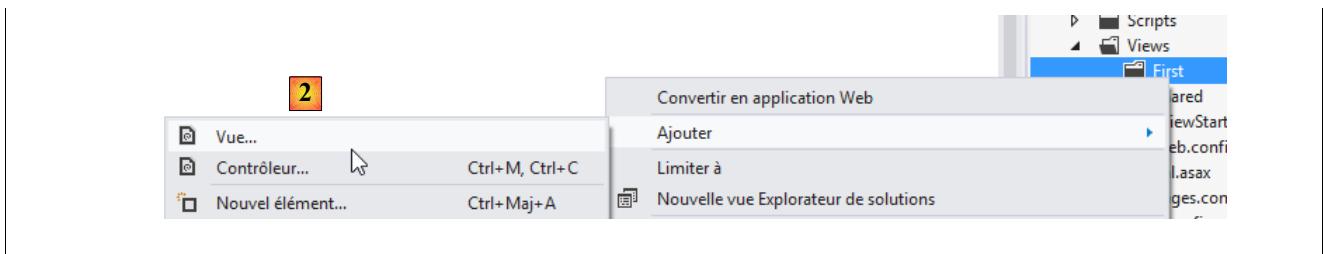
La méthode [Index] ci-dessous :

```
1. public ActionResult Index()
2. {
3.     return View();
4. }
```

demande à la vue [/Views/First/Index.cshtml] de s'afficher. Elle ne lui transmet aucun modèle. Créons [1] le dossier [/Views/First] :



puis créons dedans la vue [Index] [2]



On indique le nom de la vue en [3]. Celle-ci est créée en [4]. Le code généré est le suivant :

```

1. @{
2.     Layout = null;
3. }
4.
5. <!DOCTYPE html>
6.
7. <html>
8. <head>
9.     <meta name="viewport" content="width=device-width" />
10.    <title>Index</title>
11. </head>
12. <body>
13.     <div>
14.
15.     </div>
16. </body>
17. </html>

```

On a là du HTML classique sauf pour les lignes 1-3 qui sont du code C#. Le programme qui gère les vues est appelé un moteur de vues. Il s'occupe de gérer tout ce qui n'est pas du HTML pour en faire du HTML. Au final en effet, c'est ce qui va être envoyé au client. Le moteur de vue s'appelle ici [**Razor**]. Il permet d'inclure du code C# dans une vue. [Razor] va interpréter ce code C# et produire à partir de lui du code HTML. Voici quelques règle de base pour l'inclusion de code C# dans une vue :

- le basculement du HTML vers le C# se fait à la rencontre du caractère @ (ligne 1). Si ce caractère introduit un bloc de code, on mettra les accolades (lignes 1 et 3). S'il introduit une variable dont on veut récupérer la valeur, on écrira simplement **@variable** ;
- le basculement du C# vers le HTML se fait à la rencontre du caractère < (ligne 5). Parfois, on est amené à forcer ce basculement notamment lorsqu'on inclut dans la page du texte brut sans balise HTML. On utilisera alors la balise <text> pour introduire le texte : <text>ici du texte brut</text>.

La ligne 2 ci-dessus indique que la vue [Index] n'a pas de page maître. Nous avons vu ce concept page 47.

Modifions la vue de la façon suivante :

```

1. @{
2.     Layout = null;
3.     string vue = "Index";
4. }
5.
6. <!DOCTYPE html>
7.

```

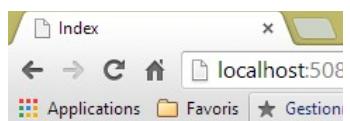
```

8. <html>
9. <head>
10. <meta name="viewport" content="width=device-width" />
11. <title>Index</title>
12. </head>
13. <body>
14. <div>
15.   <h3>Vue @vue</h3>
16. </div>
17. </body>
18. </html>

```

- ligne 3 : définit une variable C# ;
- ligne 15 : affiche la valeur de cette variable.

Demandons maintenant l'URL [/First/Index] :



### Vue Index

Le code HTML reçu est le suivant :

```

1. <!DOCTYPE html>
2.
3. <html>
4. <head>
5.   <meta name="viewport" content="width=device-width" />
6.   <title>Index</title>
7. </head>
8. <body>
9.   <div>
10.    <h3>Vue Index</h3>
11.   </div>
12. </body>
13. </html>

```

C'est un pur document HTML. Tout code C# a disparu.

## 1.5.2 Utiliser le [ViewBag] pour passer des informations à la vue

Nous créons une nouvelle action appelée [Action01] associée à la vue [Action01.cshtml] :



L'action [Action01] est la suivante :

```

1. // Action01
2. public ViewResult Action01()
3. {
4.   ViewBag.info = string.Format("Contrôleur={0}, Action={1}", RouteData.Values["controller"],
5.     RouteData.Values["action"]);
6.   return View();
}

```

- ligne 4 : on utilise la propriété [ViewBag] du contrôleur. C'est un objet dynamique auquel on peut ajouter des propriétés comme il est fait ligne 4. Cet objet a la particularité d'être également accessible à la vue. C'est donc une façon de lui transmettre de l'information ;

- ligne 5 : la vue par défaut de l'action est demandée. C'est la vue [/First/Action01.cshtml]. Aucun modèle ne lui est transmis.

La vue [Action01.cshtml] est la suivante :

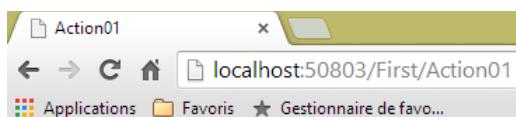
```

1. @{
2.     Layout = null;
3. }
4.
5. <!DOCTYPE html>
6.
7. <html>
8. <head>
9.     <meta name="viewport" content="width=device-width" />
10.    <title>Action01</title>
11. </head>
12. <body>
13.     <div>
14.         <h4>@ViewBag.info</h4>
15.     </div>
16. </body>
17. </html>

```

- ligne 14 : la propriété [ViewBag.info] est affichée.

Testons. Nous demandons l'URL [/First/Action01] :



Contrôleur=First, Action=Action01

### 1.5.3 Utiliser un modèle fortement typé pour passer des informations à la vue

La méthode précédente a l'inconvénient de ne pas permettre la détection d'erreurs avant l'exécution. Ainsi si la vue [Action01.cshtml] utilise le code

```
<h4>@ViewBag.Info</h4>
```

on aura une erreur car la propriété [Info] n'existe pas. Celle créée par l'action [Action01] s'appelle [info]. On peut alors utiliser un modèle fortement typé pour éviter ce désagrément.

Dans un des exemples étudiés précédemment, l'action était la suivante :

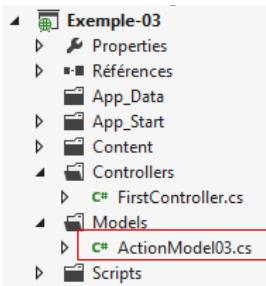
```

1.      // Action10
2.      public ContentResult Action10(ActionModel03 modèle)
3.      {
4.          string erreurs = getErrorMessagesFor ModelState;
5.          string texte = string.Format("email={0}, jour={1}, info1={2}, info2={3}, info3={4}, erreurs={5}",
6.              modèle.Email, modèle.Jour, modèle.Info1, modèle.Info2, modèle.Info3, erreurs);
7.          return Content(texte, "text/plain", Encoding.UTF8);
8.      }

```

L'action [Action10] transmettait à son client six informations (Email, Jour, Info1, Info2, Info3, erreurs) sous la forme d'une chaîne de caractères. Nous allons transmettre ces informations dans un modèle de vue [ViewModel01]. Comme ce modèle reprend des informations de [ActionModel03] nous allons le faire dériver de cette classe.

Nous commençons par recopier [ActionModel03] du projet [Exemple-02] dans le projet [Exemple-03] actuel :



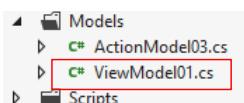
et nous changeons son espace de noms pour reprendre celui du projet [Exemple-03] :

```

1.  using System.ComponentModel.DataAnnotations;
2.  namespace Exemple_03.Models
3.  {
4.      public class ActionModel03
5.      {
6.          [Required(ErrorMessage = "Le paramètre email est requis")]
7.          [EmailAddress(ErrorMessage = "Le paramètre email n'a pas un format valide")]
8.          public string Email { get; set; }
9.
10.         [Required(ErrorMessage = "Le paramètre jour est requis")]
11.         [RegularExpression(@"^\d{1,2}$", ErrorMessage = "Le paramètre jour doit avoir 1 ou 2 chiffres")]
12.         public string Jour { get; set; }
13.
14.         [Required(ErrorMessage = "Le paramètre info1 est requis")]
15.         [MaxLength(4, ErrorMessage = "Le paramètre info1 ne peut avoir plus de 4 caractères")]
16.         public string Info1 { get; set; }
17.
18.         [Required(ErrorMessage = "Le paramètre info2 est requis")]
19.         [MinLength(2, ErrorMessage = "Le paramètre info2 ne peut avoir moins de 2 caractères")]
20.         public string Info2 { get; set; }
21.
22.         [Required(ErrorMessage = "Le paramètre info3 est requis")]
23.         [MinLength(4, ErrorMessage = "Le paramètre info3 doit avoir 4 caractères exactement")]
24.         [MaxLength(4, ErrorMessage = "Le paramètre info3 doit avoir 4 caractères exactement")]
25.         public string Info3 { get; set; }
26.     }
27. }
```

- ligne 2 : le nouvel espace de noms ;

Puis nous créons la classe [ViewModel01] :



Le code de [ViewModel01] est le suivant :

```

1.  namespace Exemple_03.Models
2.  {
3.      public class ViewModel01 : ActionModel03
4.      {
5.          public string Erreurs { get; set; }
6.      }
7. }
```

- ligne 3 : la classe hérite de [ActionModel03] et donc des propriétés [Email, Jour, Info1, Info2, Info3] ;
- ligne 5 : on lui rajoute la propriété [Erreurs].

Nous écrivons maintenant l'action [Action02] qui :

- accepte en entrée le modèle d'action [ActionModel03] ;
- et délivre en sortie le modèle de vue [ViewModel01].

Son code est le suivant :

```
1. // Action02
```

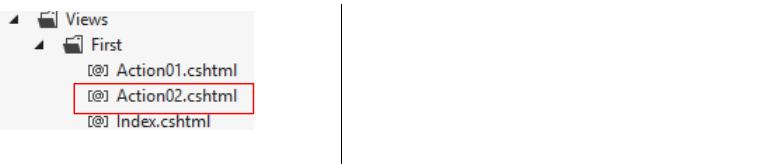
```

2.     public ViewResult Action02(ActionModel03 modèle)
3.     {
4.         string erreurs = getErrorMessagesFor(ModelState);
5.         return View(new ViewModel01(){Email=modèle.Email, Jour=modèle.Jour, Info1=modèle.Info1, Info2=modèle.Info2,
6.           Info3=modèle.Info3, Erreurs=erreurs});
    }

```

- ligne 1 : [Action02] reçoit le modèle d'action [ActionModel03]. Elle rend un résultat de type [ViewResult] ;
- ligne 4 : les erreurs liées au modèle d'action [ActionModel03] sont agrégées dans la chaîne de caractères [erreurs]. La méthode [getErrorMessagesFor] a été décrite page 71 et a été incluse dans le contrôleur [First] du nouveau projet ;
- ligne 5 : la méthode [View] est appelée avec un paramètre. Celui est le modèle de la vue. Cette dernière n'est pas précisée. Ce sera donc la vue par défaut [/Views/First/Action02] qui sera utilisée. Le modèle de vue [ViewModel01] est instancié et initialisé avec les cinq informations du modèle d'action [ActionModel03] et l'information [erreurs] construite ligne 4.

Nous construisons maintenant la vue [/First/Action02.cshtml] :



Son code est le suivant :

```

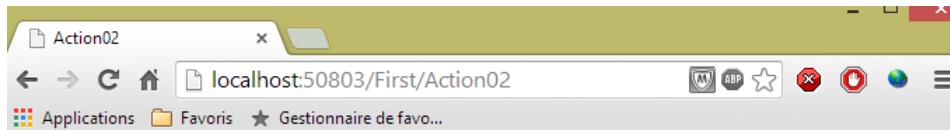
1. @model Exemple_03.Models.ViewModel01
2. @{
3.     Layout = null;
4. }
5.
6. <!DOCTYPE html>
7.
8. <html>
9. <head>
10.   <meta name="viewport" content="width=device-width" />
11.   <title>Action02</title>
12. </head>
13. <body>
14.   <h3>Informations du modèle de vue</h3>
15.   <ul>
16.     <li>Email : @Model.Email</li>
17.     <li>Jour : @Model.Jour</li>
18.     <li>Info1 : @Model.Info1</li>
19.     <li>Info2 : @Model.Info2</li>
20.     <li>Info3 : @Model.Info3</li>
21.     <li>Erreurs : @Model.Eerreurs</li>
22.   </ul>
23. </body>
24. </html>

```

- la nouveauté réside en ligne 1. La notation [@model] fixe le type du modèle de la vue. Ce modèle est ensuite référencé par la notation [@Model] (lignes 16-21) ;
- lignes 15-22 : les informations du modèle sont affichées dans une liste.

Voyons quelques exemples d'exécution de l'action [Action02].

D'abord sans paramètres :



### Informations du modèle de vue

- Email :
- Jour :
- Info1 :
- Info2 :
- Info3 :
- Erreurs : [Le paramètre email est requis][Le paramètre jour est requis][Le paramètre info1 est requis][Le paramètre info2 est requis][Le paramètre info3 est requis]

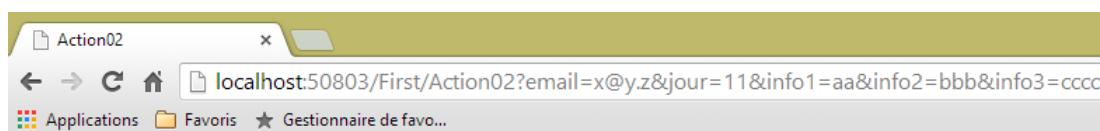
puis avec des paramètres incorrects :



### Informations du modèle de vue

- Email : x
- Jour : 111
- Info1 : aaaaa
- Info2 : b
- Info3 : cc
- Erreurs : [Le paramètre email n'a pas un format valide][Le paramètre jour doit avoir 1 ou 2 chiffres][Le paramètre info1 ne peut avoir plus de 4 caractères][Le paramètre info2 ne peut avoir moins de 2 caractères][Le paramètre info3 doit avoir 4 caractères exactement]

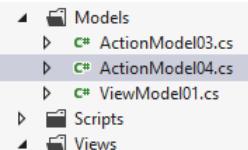
puis avec des paramètres corrects :



### Informations du modèle de vue

- Email : x@y.z
- Jour : 11
- Info1 : aa
- Info2 : bbb
- Info3 : cccc
- Erreurs :

Dans cet exemple, le modèle de la vue [ViewModel01] reprend les informations du modèle d'action [ActionModel03]. C'est souvent le cas. On peut alors utiliser un unique modèle qui servira à la fois de modèle d'action et de vue. Nous créons un nouveau modèle [ActionModel04] :



qui sera le suivant :

```

1.  using System.ComponentModel.DataAnnotations;
2.  using System.Web.Mvc;
3.  namespace Exemple_03.Models
4.  {
5.      [Bind(Exclude="Erreurs")]
6.      public class ActionModel04
7.      {
8.          // ----- Action -----
9.          [Required(ErrorMessage = "Le paramètre email est requis")]
10.         [EmailAddress(ErrorMessage = "Le paramètre email n'a pas un format valide")]
11.         public string Email { get; set; }
12.
13.         [Required(ErrorMessage = "Le paramètre jour est requis")]
14.         [RegularExpression(@"^\d{1,2}$", ErrorMessage = "Le paramètre jour doit avoir 1 ou 2 chiffres")]
15.         public string Jour { get; set; }
16.
17.         [Required(ErrorMessage = "Le paramètre info1 est requis")]
18.         [MaxLength(4, ErrorMessage = "Le paramètre info1 ne peut avoir plus de 4 caractères")]
19.         public string Info1 { get; set; }
20.
21.         [Required(ErrorMessage = "Le paramètre info2 est requis")]
22.         [MinLength(2, ErrorMessage = "Le paramètre info2 ne peut avoir moins de 2 caractères")]
23.         public string Info2 { get; set; }
24.
25.         [Required(ErrorMessage = "Le paramètre info3 est requis")]
26.         [MinLength(4, ErrorMessage = "Le paramètre info3 doit avoir 4 caractères exactement")]
27.         [MaxLength(4, ErrorMessage = "Le paramètre info3 doit avoir 4 caractères exactement")]
28.         public string Info3 { get; set; }
29.
30.         // ----- vue -----
31.         public string Erreurs { get; set; }
32.     }
33. }
```

- lignes 8-28 : le modèle de l'action avec ses contraintes d'intégrité. Ces champs feront également partie de la vue ;
- ligne 31 : une propriété propre au modèle de la vue. Elle a été exclue du modèle de l'action par l'annotation de la ligne 5.

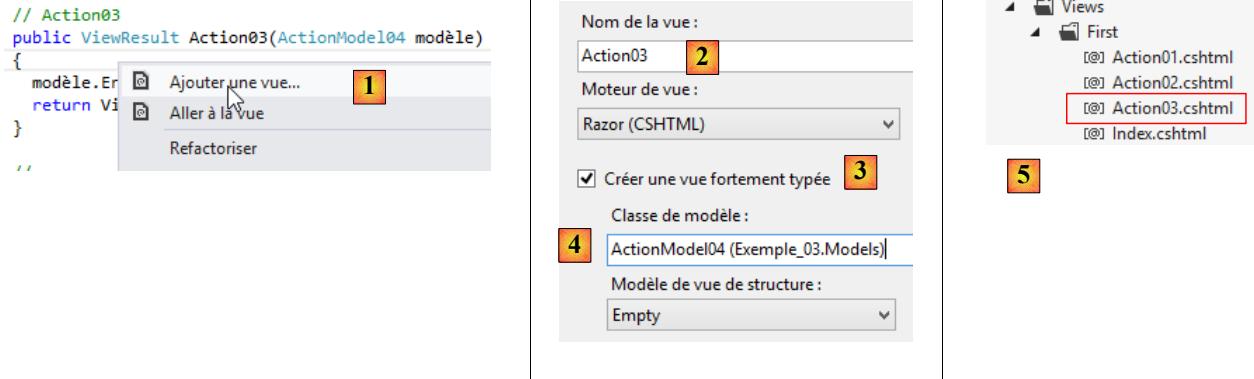
Nous créons la nouvelle action [Action03] suivante :

```

1.      // Action03
2.      public ViewResult Action03(ActionModel04 modèle)
3.      {
4.          modèle.Erreurs = getErrorMessagesFor(ModelState);
5.          return View(modèle);
6.      }
```

- ligne 2 : [Action03] reçoit le modèle d'action de type [ActionModel04] ;
- ligne 5 : et rend comme modèle de vue, ce même modèle ;
- ligne 4 : complété par l'information [Erreurs] ;

Il ne nous reste qu'à créer la vue [/First/Action03.cshtml] :



- en [1] : clic droit dans le code de [Action03] puis [Ajouter une vue] ;
- en [2] : le nom de la vue proposée par défaut ;
- en [3] : indiquer qu'on crée une vue fortement typée ;
- en [4] : prendre dans la liste déroulante la bonne classe, ici la classe [ActionModel04] ;
- en [5] : la vue créée.

Nous donnons à la vue [Action03] le même code qu'à la vue [Action02]. Seul le modèle de vue (ligne 1) et le titre de page (ligne 11) changent :

```

1. @model Exemple_03.Models.ActionModel04
2. @{
3.     Layout = null;
4. }
5.
6. <!DOCTYPE html>
7.
8. <html>
9. <head>
10.    <meta name="viewport" content="width=device-width" />
11.    <title>Action03</title>
12. </head>
13. <body>
14.    <h3>Informations du modèle de vue</h3>
15.    <ul>
16.        <li>Email : @Model.Email</li>
17.        <li>Jour : @Model.Jour</li>
18.        <li>Info1 : @Model.Info1</li>
19.        <li>Info2 : @Model.Info2</li>
20.        <li>Info3 : @Model.Info3</li>
21.        <li>Erreurs : @Model.Erreurs</li>
22.    </ul>
23. </body>
24. </html>

```

Maintenant demandons l'action [Action03] sans paramètres :



### Informations du modèle de vue

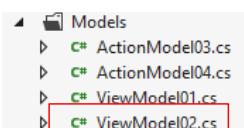
- Email :
- Jour :
- Info1 :
- Info2 :
- Info3 :
- Erreurs : [Le paramètre email est requis][Le paramètre jour est requis][Le paramètre info1 est requis][Le paramètre info2 est requis][Le paramètre info3 est requis]

Les résultats sont les mêmes qu'auparavant. Il est fréquent d'utiliser le même modèle pour l'action et la vue car le modèle de la vue reprend souvent des informations du modèle de l'action. On utilise alors un modèle plus large, utilisable à la fois par l'action et la vue que celle-ci génère. On prendra soin d'exclure de la liaison de données, les informations qui n'appartiennent pas au modèle de l'action. Sinon, un utilisateur bien renseigné pourrait initialiser des parties du modèle de la vue à notre insu.

#### 1.5.4 [Razor] – premiers pas

Nous allons maintenant présenter quelques éléments des vues [Razor], principalement les instructions *foreach* et *if*.

Supposons qu'on veuille présenter une liste de personnes dans un tableau HTML. Le modèle de la vue pourrait être le suivant [ViewModel02] :



```
1. namespace Exemple_03.Models
2. {
3.     public class ViewModel02
4.     {
5.         public Personne[] Personnes { get; set; }
6.         public ViewModel02()
7.         {
8.             Personnes = new Personne[] { new Personne { Nom = "Pierre", Age = 44 }, new Personne { Nom = "Pauline", Age = 12 } };
9.         }
10.    }
11.
12.    public class Personne
13.    {
14.        public string Nom { get; set; }
15.        public int Age { get; set; }
16.    }
17. }
```

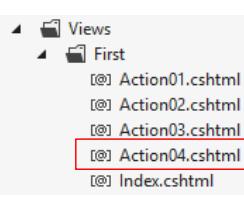
- la vue du modèle est la classe [ViewModel02], lignes 3-10 ;
- ligne 5 : le modèle possède un tableau de personnes de type [Personne] défini lignes 12-16 ;
- lignes 6-10 : le constructeur du modèle initialise la propriété [Personnes] de la ligne 5 avec un tableau de deux personnes.

L'action produisant ce modèle en sortie sera la suivante [Action04] :

```
1. // Action04
2. public ViewResult Action04()
3. {
4.     return View(new ViewModel02());
5. }
```

- ligne 2 : l'action n'a pas de modèle en entrée ;
- ligne 4 : elle passe à sa vue par défaut, une instance du modèle [ViewModel02] que nous venons de définir.

La vue [Action04.cshtml] va afficher le modèle [ViewModel02] :



Le code de la vue [Action04.cshtml] est le suivant :

```
1. @model Exemple_03.Models.ViewModel02
2. @using Exemple_03.Models
3.
4. @{
```

```

5.     Layout = null;
6. }
7.
8. <!DOCTYPE html>
9.
10. <html>
11.   <head>
12.     <meta name="viewport" content="width=device-width" />
13.     <title>Action04</title>
14.   </head>
15.   <body>
16.     <table border="1">
17.       <thead>
18.         <tr>
19.           <th>Nom</th>
20.           <th>Age</th>
21.         </tr>
22.       </thead>
23.       <tbody>
24.         @foreach (Personne p in Model.Personnes)
25.         {
26.           <tr>
27.             <td>@p.Nom</td>
28.             <td>@p.Age</td>
29.           </tr>
30.         }
31.       </tbody>
32.     </table>
33.   </body>
34. </html>

```

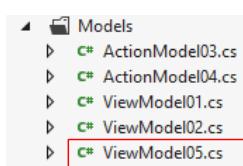
- ligne 1 : le modèle de la vue ;
- ligne 2 : l'import de l'espace de noms de la classe [Personne] utilisée ligne 24 ;
- lignes 16-32 : le tableau HTML qui visualise les personnes du modèle ;
- ligne 24 : le début du code C# est signalé par le caractère @. L'instruction [foreach] va boucler sur toutes les personnes du modèle ;
- lignes 26-27 : le caractère < arrête le C# et commence le HTML. Puis de nouveau, le caractère @ pour basculer en C# et écrire le nom de la personne. Puis de nouveau le caractère < qui fait basculer en mode HTML ;
- ligne 28 : on écrit l'âge de la personne.

L'exécution de l'action [Action04] donne le résultat suivant :



D'autres éléments d'une vue peuvent être alimentés par une collection : les listes, déroulantes ou non, les boutons radio, les cases à cocher. Considérons le nouvel exemple suivant qui affiche une liste déroulante.

Le modèle [ViewModel05] sera le suivant :



```

1. namespace Exemple_03.Models
2. {
3.     public class ViewModel05
4.     {

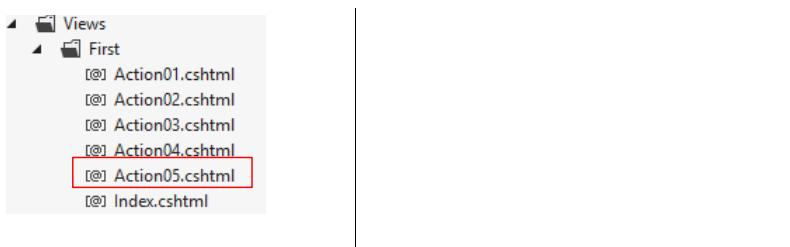
```

```

5.     public Personne2[] Personnes { get; set; }
6.     public int SelectedId { get; set; }
7.
8.     public ViewModel05()
9.     {
10.         Personnes = new Personne2[] {
11.             new Personne2 { Id = 1, Prénom = "Pierre", Nom = "Martino" },
12.             new Personne2 { Id = 2, Prénom = "Pauline", Nom = "Pereiro" },
13.             new Personne2 { Id = 3, Prénom = "Jacques", Nom = "Alfonso" } };
14.         SelectedId = 2;
15.     }
16. }
17.
18. public class Personne2
19. {
20.     public int Id { get; set; }
21.     public string Nom { get; set; }
22.     public string Prénom { get; set; }
23. }
24. }
```

- ligne 18 : une classe [Personne2] avec trois propriétés ;
- ligne 3 : le modèle [ViewModel05] de la vue ;
- ligne 5 : la liste des personnes à afficher dans la liste déroulante sous la forme [Prénom Nom] ;
- ligne 6 : l'[Id] de la personne à sélectionner dans la liste déroulante ;
- lignes 8-16 : le constructeur qui crée un tableau de trois personnes (lignes 10-13) et fixe l'[Id] de la personne qui doit apparaître sélectionnée.

La vue [Action05.cshtml] va afficher ce modèle :



Son code est le suivant :

```

1. @model Exemple_03.Models.ViewModel05
2. @using Exemple_03.Models
3.
4. @{
5.     Layout = null;
6. }
7.
8. <!DOCTYPE html>
9.
10. <html>
11. <head>
12.     <meta name="viewport" content="width=device-width" />
13.     <title>Action05</title>
14. </head>
15. <body>
16.     <select>
17.         @foreach (Personne2 p in Model.Personnes)
18.         {
19.             string selected = "";
20.             if (p.Id == Model.SelectedId)
21.             {
22.                 selected = "selected=\"selected\"";
23.             }
24.             <option value="@p.Id" @selected>@p.Prénom @p.Nom</option>
25.         }
26.     </select>
27. </body>
28. </html>
```

Les caractéristiques de la liste déroulante HTML ont été présentées au paragraphe 1.2.5.2.6, page 37. Rappelons-les :

Combo      <select size="1" name="cmbValeurs">  
               <option value="1">choix1</option>  
               <option selected="selected" value="2">choix2</option>

```

        <option value="3">choix3</option>
    </select>

```



**balise HTML**

```

<select size="" name="">
    <option [selected="selected"] value="v">...</option>
    ...
</select>

```

**attributs**

affiche dans une liste les textes compris entre les balises <option>...</option>  
**name="cmbValeurs"** : nom du contrôle.

**size="1"** : nombre d'éléments de liste visibles. *size="1"* fait de la liste l'équivalent d'un combobox.

**selected="selected"** : si ce mot clé est présent pour un élément de liste, ce dernier apparaît sélectionné dans la liste. Dans notre exemple ci-dessus, l'élément de liste *choix2* apparaît comme l'élément sélectionné du combo lorsque celui-ci est affiché pour la première fois.

**value="v"** : si l'élément est sélectionné par l'utilisateur, c'est cette valeur [v] qui est postée au serveur. En l'absence de cet attribut, c'est le texte affiché et sélectionné qui est posté au serveur.

Le code des lignes 17-25 génère les balises <option> qui prennent place dans la balise <select> de la ligne 16.

- ligne 17 : on parcourt la liste des personnes du modèle ;
- ligne 20 : on regarde si la personne courante est la personne qui doit être sélectionnée. Si oui, on prépare le texte *selected="selected"* qui doit être inséré dans la balise <option> ;
- ligne 24 : la balise <option> est écrite.

Demandons l'action [Action05] :



- en [1,2], les personnes sont affichées sous la forme [Prénom Nom] ;
- en [1,2], la personne sélectionnée est celle qui à l'[Id] égal à 2.

Examinons maintenant le code source HTML de la page ci-dessus :

```

1.  <!DOCTYPE html>
2.
3.  <html>
4.  <head>
5.      <meta name="viewport" content="width=device-width" />
6.      <title>Action05</title>
7.  </head>
8.  <body>
9.      <select>
10.          <option value="1" >Pierre Martino</option>
11.          <option value="2" selected="selected">Pauline Pereiro</option>
12.          <option value="3" >Jacques Alfonso</option>
13.      </select>
14.  </body>
15. </html>

```

- lignes 10-12 : les trois balises <option> générées par le code [Razor] ;
- ligne 11 : c'est bien la personne d'[Id]=2 qui a été sélectionnée.

Les deux exemples ci-dessus nous suffiront. Lorsqu'on écrit une vue [Razor], il faut résister à la tentation de mettre de la logique dedans. Le code C# nous le permettrait. Cependant, dans le modèle MVC, la logique doit être dans l'action ou dans les couches basses [Metier, DAO] mais pas dans la vue. Même en respectant le modèle MVC, on peut se retrouver avec beaucoup de logique dans la vue pour calculer des valeurs intermédiaires. Cela peut vouloir signifier que le modèle utilisé n'est pas assez détaillé. Celui-ci doit contenir les valeurs finales dont la vue a besoin afin qu'elle n'ait pas à les calculer elle-même. Une bonne vue est une vue où il y a un minimum de logique et où la structure HTML de la vue reste claire. Si on insère trop de code C#, la structure HTML peut devenir illisible.

Dans l'exemple ci-dessus, la liste déroulante pourrait être utilisée par un internaute et on voudrait alors savoir quelle personne il a sélectionnée. Il nous faut pour cela un formulaire.

### 1.5.5 Formulaire – premiers pas

Le formulaire présenté à l'internaute sera le suivant :

The screenshot shows a browser window titled "Action06-GET". The address bar displays "localhost:50803/First/Action06". The main content area has a heading "Action06 - GET" and the text "Choisissez une personne". Below this is a dropdown menu containing "Pauline Pereiro" and a "Valider" button.

Le modèle de la vue sera le modèle [ViewModel05] déjà utilisé précédemment. L'action qui affichera cette vue sera la suivante :

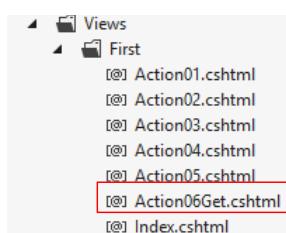
```

1. // Action06-GET
2. [HttpGet]
3. public ViewResult Action06()
4. {
5.     return View("Action06Get", new ViewModel05());
6. }

```

- ligne 2 : l'action ne peut être demandée que par une commande HTTP GET ;
- ligne 5 : la vue [/First/Action06Get.cshtml] sera affichée avec pour modèle une instance de type [ViewModel05].

La vue [/First/Action06Get.cshtml] sera la suivante :



```

1. @model Exemple_03.Models.ViewModel05
2. @using Exemple_03.Models
3.
4. @{
5.     Layout = null;
6. }
7.
8. <!DOCTYPE html>
9.

```

```

10. <html>
11.   <head>
12.     <meta name="viewport" content="width=device-width" />
13.     <title>Action06-GET</title>
14.   </head>
15.   <body>
16.     <h3>Action06 - GET</h3>
17.     <p>Choisissez une personne</p>
18.     <form method="post" action="/First/Action06">
19.       <select name="personneId">
20.         @foreach (Personne2 p in Model.Personnes)
21.         {
22.           string selected = "";
23.           if (p.Id == Model.SelectedId)
24.           {
25.             selected = "selected='selected'";
26.           }
27.           <option value="@p.Id" @selected>@p.Prénom @p.Nom</option>
28.         }
29.       </select>
30.       <input name="valider" type="submit" value="Valider" />
31.     </form>
32.   </body>
33. </html>

```

Les principales nouveautés sont les suivantes :

- ligne 18 : pour que le navigateur puisse transmettre les informations saisies par un utilisateur, il nous faut un formulaire. C'est la balise `<form>` des lignes 18 et 31 qui le délimitent.

La balise HTML `<form>` a été présentée au paragraphe 1.2.5.2.1, page 35. Rappelons ses caractéristiques :

**formulaire**      `<form method="post" action="FormulairePost.aspx">`

**balise HTML**      `<form name="..." method="..." action="...">...</form>`

**attributs**      `name="frmexemple"` : nom du formulaire - facultatif

`method="..."` : méthode utilisée par le navigateur pour envoyer au serveur Web les valeurs récoltées dans le formulaire

`action="..."` : URL à laquelle seront envoyées les valeurs récoltées dans le formulaire.

Un formulaire Web est entouré des balises `<form>...</form>`. Le formulaire peut avoir un nom (`name="xx"`). C'est le cas pour tous les contrôles qu'on peut trouver dans un formulaire. Le but d'un formulaire est de rassembler des informations données par l'utilisateur au clavier/souris et d'envoyer celles-ci à une URL de serveur Web. Laquelle ? Celle référencée dans l'attribut `action="URL"`. Si cet attribut est absent, les informations seront envoyées à l'URL du document dans lequel se trouve le formulaire. Un client Web peut utiliser deux méthodes différentes appelées POST et GET pour envoyer des données à un serveur web. L'attribut `method="métode"`, avec `method` égal à GET ou POST, de la balise `<form>` indique au navigateur la méthode à utiliser pour envoyer les informations recueillies dans le formulaire à l'URL précisée par l'attribut `action="URL"`. Lorsque l'attribut `method` n'est pas précisé, c'est la méthode GET qui est prise par défaut.

- ligne 18 : on voit que les valeurs du formulaire seront envoyées à l'URL [/First/Action06] par une commande HTTP POST ;
- ligne 30 : un formulaire doit avoir un bouton de type [submit]. C'est lui qui déclenche l'envoi des valeurs saisies à l'URL précisée par l'attribut [action] de la balise `<form>`.

Que va au juste transmettre le navigateur lorsque l'internaute va cliquer sur le bouton [Valider] ? Ceci a été expliqué au paragraphe 1.2.5.3.1, page 40. Rappelons ce qui avait été dit :

contrôle HTML	visuel	valeur(s) renvoyée(s)
<code>&lt;input type="radio" value="Oui" name="R1"/&gt;Oui</code>	Etes-vous marié(e) <input checked="" type="radio"/> Oui <input type="radio"/> Non	<b>R1=Oui</b> - la valeur de l'attribut <code>value</code> du bouton radio coché par l'utilisateur.
<code>&lt;input type="radio" name="R1" value="non" checked="checked"/&gt;Non</code>		
<code>&lt;input type="checkbox" name="C1" value="un"/&gt;1</code>	Cases à cocher <input checked="" type="checkbox"/> 1 <input checked="" type="checkbox"/> 2 <input type="checkbox"/> 3	<b>C1=un</b>
<code>&lt;input type="checkbox" name="C2"</code>		<b>C2=deux</b>

```
<input type="checkbox" name="C1" value="deux" checked="checked"/>
```

```
<input type="checkbox" name="C3" value="trois"/>
```

```
<input type="text" name="txtSaisie" size="20" value="qqs mots"/>
```

Champ de saisie

programmation web

```
<input type="password" name="txtMdp" size="20" value="unMotDePasse"/>
```

Mot de passe

\*\*\*\*\*

```
<textarea rows="2" name="areaSaisie" cols="20">
```

ligne1

ligne2

ligne3

```
</textarea>
```

Boîte de saisie

les bases de la programmation web

```
<select size="1" name="cmbValeurs">
```

```
<option value='1'>choix1</option>
```

```
<option selected="selected" value='2'>choix2</option>
```

```
<option value='3'>choix3</option>
```

```
</select>
```

```
<select size="3" name="lst1">
```

```
<option selected="selected" value='1'>liste1</option>
```

```
<option value='2'>liste2</option>
```

```
<option value='3'>liste3</option>
```

```
<option value='4'>liste4</option>
```

```
<option value='5'>liste5</option>
```

```
</select>
```

```
<select size="3" name="lst2" multiple="multiple">
```

```
<option selected="selected" value='1'>liste1</option>
```

```
<option value='2'>liste2</option>
```

```
<option selected="selected" value='3'>liste3</option>
```

```
<option value='4'>liste4</option>
```

```
<option value='5'>liste5</option>
```

```
</select>
```

```
<input type="submit" value="Envoyer" name="cmdRenvoyer"/>
```

combo

choix3 ▾

liste à choix simple

liste1 ▲  
liste2 ▼  
liste3 ▾

liste à choix multiple

liste1 ▲  
liste2 ▼  
liste3 ▾

```
<input type="hidden" name="secret" value="uneValeur"/>
```

- valeurs des attributs *value* des cases cochées par l'utilisateur

**txtSaisie=programmation+Web**

- texte tapé par l'utilisateur dans le champ de saisie. Les espaces ont été remplacés par le signe +

**txtMdp=ceciestsecret**

- texte tapé par l'utilisateur dans le champ de saisie

**areaSaisie=les+bases+de+la%0D%0A**

**programmation+Web**

- texte tapé par l'utilisateur dans le champ de saisie. %OD%OA est la marque de fin de ligne. Les espaces ont été remplacés par le signe +

**cmbValeurs=3**

- attribut [value] de l'élément sélectionné par l'utilisateur

**lst1=3**

- attribut [value] de l'élément sélectionné par l'utilisateur

**lst2=1**

**lst2=3**

- attributs [value] des éléments sélectionnés par l'utilisateur

**cmdRenvoyer=Envoyer**

- nom et attribut *value* du bouton qui a servi à envoyer les données du formulaire au serveur

**secret=uneValeur**

- attribut *value* du champ caché

Dans notre formulaire, nous avons deux balises susceptibles d'envoyer une valeur :

```
1.      <select name="personneId">
2.      ...
3.      </select>
```

et

```
1.      <input name="valider" type="submit" value="Valider" />
```

Si l'internaute sélectionne la personne n° 2, les valeurs postées le seront sous la forme :

```
personneId=2&valider=Valider
```

Les noms des paramètres sont ceux des attributs [name] des balises concernées par le POST. Sans cet attribut, les balises n'émettent pas de valeur. Ainsi ci-dessus, on pourrait omettre l'attribut **name="valider"** du bouton [submit]. La valeur envoyée est l'attribut [value] du bouton. Ici cette information ne nous intéresse pas. Parfois les formulaires ont plusieurs boutons de type [submit]. Il est alors important de savoir quel bouton a été cliqué. On mettra alors l'attribut [name] aux différents boutons.

La balise <select> est composée d'une suite de balises <option> :

```
1.      <select name="personneId">
2.          <option value="1" >Pierre Martino</option>
3.          <option value="2" selected="selected">Pauline Pereiro</option>
4.          <option value="3" >Jacques Alfonso</option>
5.      </select>
```

C'est la valeur de l'attribut [value] de l'option sélectionnée qui est postée. En d'absence de cet attribut, c'est le texte affiché par l'option, par exemple [Pierre Martino], qui est posté.

La chaîne

```
personneId=2&valider=Valider
```

va être postée à l'URL [/First/Action06] suivante :

```
1.      // Action06-POST
2.      [HttpPost]
3.      public ViewResult Action06(ActionModel06 modèle)
4.      {
5.          return View("Action06Post", modèle);
6.      }
```

On se rappelle peut-être que nous avions déjà une action [Action06] :

```
a)      // Action06-GET
b)      [HttpGet]
c)      public ViewResult Action06()
d)      {
e)          return View("Action06Get", new ViewModel05());
f)      }
```

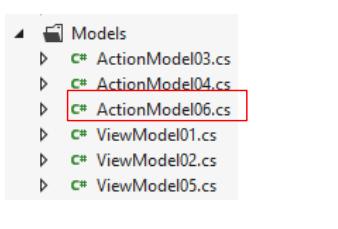
Il est possible d'avoir deux actions de même nom à condition qu'elle ne traite pas les mêmes commandes HTTP :

- [Action06] de la ligne 3 gère un POST (ligne 2) ;
- [Action06] de la ligne c gère un GET (ligne b).

L'action [Action06] qui gère le POST va recevoir la chaîne de paramètres suivant :

```
personneId=2&valider=Valider
```

Il nous faut un modèle d'action pour encapsuler ces valeurs. Ce sera le modèle [ActionModel06] suivant :

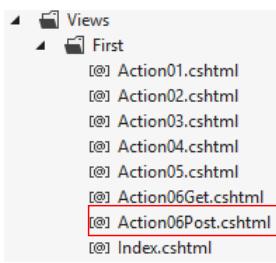


```

1.  using System.ComponentModel.DataAnnotations;
2.  namespace Exemple_03.Models
3.  {
4.      public class ActionModel06
5.      {
6.          [Required(ErrorMessage = "Le paramètre [personneId] est requis")]
7.          public int PersonneId { get; set; }
8.
9.          [Required(ErrorMessage = "Le paramètre [valider] est requis")]
10.         public string Valider { get; set; }
11.     }
12. }

```

L'action [Action06] reçoit ce modèle et le transmet tel quel à la vue [Action06Post] (ligne 5 de l'action) suivante :



```

1. @model Exemple_03.Models.ActionModel06
2.
3. @{
4.     Layout = null;
5. }
6.
7. <!DOCTYPE html>
8.
9. <html>
10. <head>
11.     <meta name="viewport" content="width=device-width" />
12.     <title>Action06Post</title>
13. </head>
14. <body>
15.     <h3>Action06 - POST</h3>
16.     Valeurs postées :
17.     <ul>
18.         <li>ID de la personne sélectionnée : @Model.PersonneId</li>
19.         <li>Commande utilisée : @Model.Valider</li>
20.     </ul>
21. </body>
22. </html>

```

Le modèle est affiché aux lignes 18 et 19.

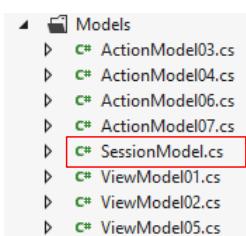
Voyons un exemple :

En [1] on sélectionne la troisième personne d'[Id] égal à 3. En [2] on poste le formulaire. En [3], les valeurs reçues. En [4,5], on voit que la même URL a été appelée, l'une par un GET [4], l'autre par un POST [5]. Ceci ne se voit pas dans l'URL.

Dans la vue affichée à la suite du POST, on pourrait vouloir les *nom* et *prénom* de la personne sélectionnée plutôt que son numéro. Il faut alors faire évoluer la vue du POST et son modèle.

Nous créons une action [Action07] pour traiter ce cas. Cette action va devoir utiliser la session de l'utilisateur pour y stocker la liste des personnes. Nous allons suivre le modèle étudié au paragraphe 1.4.10, page 84 qui permet d'inclure les données de portée [Application] et [Session] dans le modèle de l'action.

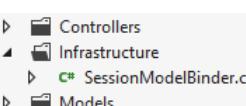
Le modèle de la session sera la classe [SessionModel] suivante :



```
1. namespace Exemple_03.Models
2. {
3.     public class SessionModel
4.     {
5.         public Personne2[] Personnes { get; set; }
6.     }
7. }
```

- ligne 2 : la session mémorisera la liste des personnes affichées dans la liste déroulante ;

Il nous faut lier le type précédent [SessionModel] à un binder que nous appellerons [SessionModelBinder]. Celui-ci sera le même que celui décrit page 88 :



```
1. using System.Web.Mvc;
2.
3. namespace Exemple_03.Infrastructure
4. {
5.     public class SessionModelBinder : IModelBinder
6.     {
7.         public object BindModel(ControllerContext controllerContext, ModelBindingContext bindingContext)
8.         {
9.             // on rend les données de portée [Session]
10.            return controllerContext.HttpContext.Session["data"];
11.        }
12.    }
13. }
```

La liaison entre le modèle [SessionModel] et son *binder* [SessionModelBinder] se fait dans [Global.asax] :

```
1. public class MvcApplication : System.Web.HttpApplication
2. {
3.     protected void Application_Start()
4.     {
5.         ...
6.
7.         // model binders
8.         ModelBinders.Binders.Add(typeof(SessionModel), new SessionModelBinder());
9.     }
10.    // Session
11.    public void Session_Start()
12.    {
13.        Session["data"] = new SessionModel();
14.    }
15. }
```

- ligne 8 : la liaison du modèle à son binder se fait dans [Application\_Start] ;

- ligne 13 : une instance de type [SessionModel] est mise en session associée à la clé [data].

Ceci fait, l'action [Action07] est la suivante :

```

1.    // Action07-GET
2.    [HttpGet]
3.    public ViewResult Action07(SessionModel session)
4.    {
5.        ViewModel05 modèleVue = new ViewModel05();
6.        session.Personnes= modèleVue.Personnes;
7.        return View("Action07Get", modèleVue);
8.    }

```

- ligne 3 : l'action récupère un type [SessionModel], donc la donnée de portée [Session] associée à la clé [data] ;
- ligne 5 : on crée le modèle de la vue ;
- ligne 6 : on met dans la session le tableau des personnes. On en aura besoin dans la requête suivante, celle du POST. Le protocole HTTP est un protocole sans état. Il faut utiliser une **session** pour avoir de la mémoire entre les requêtes. Une session est propre à un utilisateur et est gérée par le serveur web ;
- ligne 7 : la vue [Action07Get.cshtml] est affichée. C'est la suivante :

```

1. @model Exemple_03.Models.ViewModel05
2. @using Exemple_03.Models
3. ...
4. <body>
5.     <h3>Action07 - GET</h3>
6.     <p>Choisissez une personne</p>
7.     <form method="post" action="/First/Action07">
8.     ....
9.     </form>
10.    </body>
11.    </html>

```

Elle est identique à la vue [Action06Get.cshtml] déjà étudiée. La principale différence est ligne 7 : l'URL à laquelle seront postées les valeurs du formulaire. Celles-ci seront traitées par l'action [Action07] suivante :

```

1.    // Action07-POST
2.    [HttpPost]
3.    public ViewResult Action07(SessionModel session, ActionModel06 modèle)
4.    {
5.        Personne2 personne = session.Personnes.Where(p => p.Id == modèle.PersonneId).First<Personne2>();
6.        string strPersonne = string.Format("{0} {1}", personne.Prénom, personne.Nom);
7.        return View("Action07Post", (object)strPersonne);
8.    }

```

- ligne 3 : les valeurs postées sont encapsulées dans le modèle d'action [ActionModel06] déjà utilisé précédemment :

```

1.  using System.ComponentModel.DataAnnotations;
2.  namespace Exemple_03.Models
3.  {
4.      public class ActionModel06
5.      {
6.          [Required(ErrorMessage = "Le paramètre [personneId] est requis")]
7.          public int PersonneId { get; set; }
8.
9.          [Required(ErrorMessage = "Le paramètre [valider] est requis")]
10.         public string Valider { get; set; }
11.     }
12. }

```

- ligne 3 : le premier paramètre est la donnée de portée [Session] associée à la clé [data] ;
- ligne 5 : une requête LINQ récupère la personne ayant l'[Id] qui a été posté ;
- ligne 6 : on construit la chaîne de caractères qui doit être affichée par la vue [Action07Post] (ligne 8) ;
- ligne 7 : pour appeler le bon constructeur [View], il faut transtyper le type [string] en [object].

La vue [Action07Post.cshtml] est la suivante :

```

1.  @model string
2.
3.  @{
4.      Layout = null;
5.  }
6.
7.  <!DOCTYPE html>
8.
9.  <html>
10. <head>
11.     <meta name="viewport" content="width=device-width" />

```

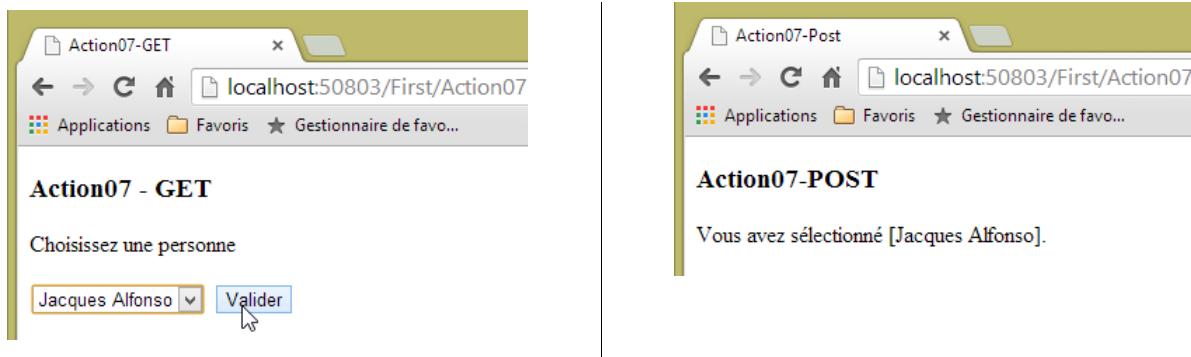
```

12.  <title>Action07-Post</title>
13.  </head>
14.  <body>
15.    <h3>Action07-POST</h3>
16.    Vous avez sélectionné [@Model].
17.  </body>
18. </html>

```

- ligne 1 : le modèle est de type [string] ;
- ligne 16 : la chaîne de caractères est affichée.

Voici un exemple d'exécution :



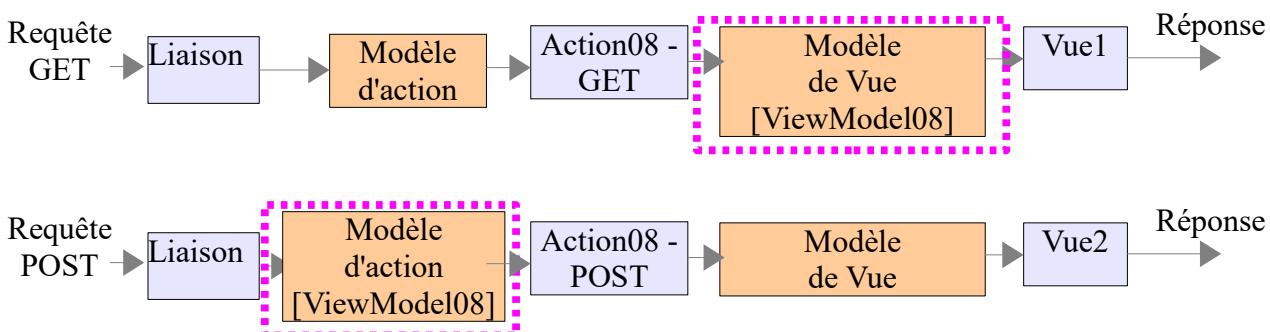
### 1.5.6 Formulaire – un exemple complet

Nous avons étudié au paragraphe 1.2.5.2.1, page 35, le formulaire HTML suivant :

Nous allons étudier une action [Action08Get] qui affiche (GET) ce formulaire et une action [Action08Post] qui traite (POST) les valeurs saisies par l'utilisateur. Un schéma classique.

Le modèle de la vue [1] ci-dessus sera une instance de la classe [ViewModel08]. Cette classe sera à la fois :

- le modèle de la vue produite par un GET sur l'action [Action08Get] ;
- le modèle de l'action [Action08Post] pour une requête POST.



#### 1.5.6.1 Le modèle de portée [Application]

Nous allons supposer que les éléments affichés par les boutons radio, les cases à cocher et les différentes listes sont des données de portée [Application]. Un cas fréquent. Ces informations proviennent d'un fichier de configuration ou d'une base de données exploitées au démarrage de l'application dans la méthode [Application\_Start] de [Global.asax]. Cette méthode évolue de la façon suivante :

```

1.     protected void Application_Start()
2.     {

```

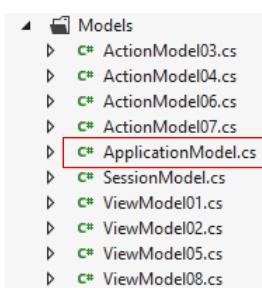
```

3. ....
4.
5.     // model binders
6.     ModelBinders.Binders.Add(typeof(SessionModel), new SessionModelBinder());
7.     ModelBinders.Binders.Add(typeof(ApplicationModel), new ApplicationModelBinder());
8.
9.     // données de portée [Application]
10.    Application["data"] = new ApplicationModel();
11. }

```

- ligne 7 : le type [ApplicationModel] que nous allons décrire bientôt est associé au lieu de données [ApplicationModelBinder] que nous avons déjà présenté page 88 ;
- ligne 10 : une instance de type [ApplicationModel] est enregistrée dans le dictionnaire de l'application, associée à la clé [data].

La classe [ApplicationModel] sert à encapsuler toutes les données de portée [Application]. Ici elle encapsulera les données que doit afficher le formulaire :



```

1. namespace Exemple_03.Models
2. {
3.     public class ApplicationModel
4.     {
5.         // les collections à afficher dans le formulaire
6.         public Item[] RadioButtonFieldItems { get; set; }
7.         public Item[] CheckBoxesFieldItems { get; set; }
8.         public Item[] DropDownListFieldItems { get; set; }
9.         public Item[] SimpleChoiceListFieldItems { get; set; }
10.        public Item[] MultipleChoiceListFieldItems { get; set; }
11.
12.        // initialisation des champs et collections
13.        public ApplicationModel()
14.        {
15.            RadioButtonFieldItems = new Item[]{
16.                new Item {Value="1",Label="oui"},
17.                new Item {Value="2", Label="non"}
18.            };
19.            CheckBoxesFieldItems = new Item[]{
20.                new Item {Value="1",Label="1"},
21.                new Item {Value="2", Label="2"},
22.                new Item {Value="3", Label="3"}
23.            };
24.            DropDownListFieldItems = new Item[]{
25.                new Item {Value="1",Label="choix1"},
26.                new Item {Value="2", Label="choix2"},
27.                new Item {Value="3", Label="choix3"}
28.            };
29.            SimpleChoiceListFieldItems = new Item[]{
30.                new Item {Value="1",Label="liste1"},
31.                new Item {Value="2", Label="liste2"},
32.                new Item {Value="3", Label="liste3"},
33.                new Item {Value="4", Label="liste4"},
34.                new Item {Value="5", Label="liste5"}
35.            };
36.            MultipleChoiceListFieldItems = new Item[]{
37.                new Item {Value="1",Label="liste1"},
38.                new Item {Value="2", Label="liste2"},
39.                new Item {Value="3", Label="liste3"},
40.                new Item {Value="4", Label="liste4"},
41.                new Item {Value="5", Label="liste5"}
42.            };
43.        }
44.        // l'élément des collections
45.        public class Item
46.        {
47.            public string Label { get; set; }

```

```

48.     public string Value { get; set; }
49. }
50.
51. }
52. }
```

- lignes 45-49 : l'élément des différentes collections du formulaire. [Label] est le texte affiché par l'élément du formulaire, [Value] la valeur postée par cet élément lorsqu'il est sélectionné ;
- ligne 6 : la collection affichée par le bouton radio ;
- ligne 7 : la collection affichée par les cases à cocher ;
- ligne 8 : la collection affichée par la liste déroulante ;
- ligne 9 : la collection affichée par la liste à sélection unique ;
- ligne 10 : la collection affichée par la liste à sélection multiple ;
- lignes 13-43 : ces collections sont initialisées par le constructeur sans paramètres de la classe.

Le différentes collections vont alimenter le formulaire suivant :

### Formulaire ASP.NET MVC

**Affiché par : Contrôleur=First, Action=Action08Get**

Etes-vous marié(e)  oui  non

Cases à cocher  1  2  3

Champ de saisie

Mot de passe

Boîte de saisie

Liste déroulante

Liste à choix unique

Liste à choix multiple

**Valider**

#### 1.5.6.2 Le modèle de l'action [Action08Get]

Le formulaire précédent sera affiché par l'action [Action08Get] suivante :

```

1. // Action08-GET
2. [HttpGet]
3. public ViewResult Action08Get(ApplicationModel application)
4. {
5.     ViewBag.info = string.Format("Contrôleur={0}, Action={1}", RouteData.Values["controller"],
6.     RouteData.Values["action"]);
7.     return View("Formulaire", new ViewModel08(application));
}
```

- ligne 2 : [Action08Get] ne répondra qu'à une commande [GET] ;
- ligne 3 : elle reçoit en paramètre, le modèle de l'application que nous venons de décrire ;
- ligne 5 : elle initialise une information dans le conteneur dynamique [ViewBag] ;
- ligne 6 : elle affiche la vue [/First/Formulaire.cshtml] avec le modèle [ViewModel08]. Ce modèle sera celui du formulaire présenté précédemment. Pour cela, on passe au constructeur le modèle de l'application qui définit les éléments à afficher.

#### 1.5.6.3 Le modèle de la vue [Formulaire]

La classe [ViewModel08] sera le modèle du formulaire. Cette classe est la suivante :

```

1. using System.ComponentModel.DataAnnotations;
2. using System.Web.Mvc;
```

```

3.  using Exemple_03.Models;
4.
5.  namespace Exemple_03.Models
6.  {
7.      public class ViewModel08
8.      {
9.          // les champs de saisie
10.         public string RadioButtonField { get; set; }
11.         public string[] CheckBoxesField { get; set; }
12.         public string TextField { get; set; }
13.         public string PasswordField { get; set; }
14.         public string TextAreaField { get; set; }
15.         public string DropDownListField { get; set; }
16.         public string SimpleChoiceListField { get; set; }
17.         public string[] MultipleChoiceListField { get; set; }
18.
19.         // les collections à afficher dans le formulaire
20.         public ApplicationModel.Item[] RadioButtonFieldItems { get; set; }
21.         public ApplicationModel.Item[] CheckBoxesFieldItems { get; set; }
22.         public ApplicationModel.Item[] DropDownListFieldItems { get; set; }
23.         public ApplicationModel.Item[] SimpleChoiceListFieldItems { get; set; }
24.         public ApplicationModel.Item[] MultipleChoiceListFieldItems { get; set; }
25.
26.         // constructeurs
27.         public ViewModel08()
28.         {
29.         }
30.
31.         public ViewModel08(ApplicationModel application)
32.         {
33.             // initialisation collections
34.             RadioButtonFieldItems = application.RadioButtonFieldItems;
35.             CheckBoxesFieldItems = application.CheckBoxesFieldItems;
36.             DropDownListFieldItems = application.DropDownListFieldItems;
37.             SimpleChoiceListFieldItems = application.SimpleChoiceListFieldItems;
38.             MultipleChoiceListFieldItems = application.MultipleChoiceListFieldItems;
39.             // initialisation champs
40.             RadioButtonField = "2";
41.             CheckBoxesField = new string[] { "2" };
42.             TextField = "quelques mots";
43.             PasswordField = "secret";
44.             TextAreaField = "ligne1\nligne2";
45.             DropDownListField = "2";
46.             SimpleChoiceListField = "3";
47.             MultipleChoiceListField = new string[] { "1", "3" };
48.         }
49.     }
50. }

```

- dans un formulaire, il y a deux types d'éléments : ceux qu'on affiche et ceux qui font l'objet d'une saisie ;
- les lignes 20-24 définissent les éléments à afficher. Ce sont les différentes collections du formulaire. Celles-ci sont trouvées dans le modèle de l'application (lignes 34-38) ;
- lignes 10-17 : définissent les champs de saisie du formulaire ;
- ligne 10 : [RadioButtonField] récupèrera **la valeur postée** par les lignes suivantes du formulaire :

```

1.         <!-- les boutons radio -->
2.         <tr>
3.             <td>Etes-vous marié(e)</td>
4.             <td>
5.             <input type="radio" name="RadioButtonField" value="1" />oui
6.             <input type="radio" name="RadioButtonField" value="2" checked="" />non
7.             </td>
8.         </tr>

```

On notera lignes 5 et 6 que l'attribut [name] des deux boutons radio est le nom de la propriété qui va être initialisée. Dans les données postées, on trouvera une chaîne de la forme :

`param1=val1&RadioButtonField=2&param2=val2`

si l'utilisateur a coché l'option libellée [non]. C'est en effet l'attribut [value] de l'option cochée qui est posté.

- ligne 11 : [CheckboxesField] récupèrera **les valeurs postées** par les lignes suivantes du formulaire :

```

1.         <!-- les cases à cocher -->
2.         <tr>
3.             <td>Cases à cocher</td>
4.             <td>
5.             <input type="checkbox" name="CheckboxesField" value="1" />1
6.             <input type="checkbox" name="CheckboxesField" value="2" checked="" />2
7.             <input type="checkbox" name="CheckboxesField" value="3" />3
8.             </td>

```

On notera lignes 5 et 6 que l'attribut [name] des cases à cocher est le nom de la propriété qui va être initialisée. Dans les données postées, on trouvera une chaîne de la forme :

```
param1=val1&CheckBoxesField=2&CheckBoxesField=3&param2=val2
```

si l'utilisateur a coché les cases à cocher libellées [2] et [3]. C'est l'attribut [value] des options cochées qui est posté. C'est parce que plusieurs paramètres de même nom peuvent être postés que [CheckboxesField] est un tableau de valeurs et non une valeur simple. Si aucune case n'est cochée, le paramètre [CheckboxesField] sera absent de la chaîne postée et la propriété de même nom du modèle ne sera pas initialisée. Ce peut être gênant comme nous le verrons.

- ligne 12 : [TextField] récupèrera **la valeur postée** par les lignes suivantes du formulaire :

```
1.      <!-- le champ de saisie texte monoligne -->
2.      <tr>
3.          <td>Champ de saisie</td>
4.          <td>
5.              <input type="text" name="TextField" value="quelques mots" size="30" />
6.          </td>
7.      </tr>
```

Ligne 5, l'attribut [name] du champ de saisie est le nom de la propriété qui va être initialisée. Dans les données postées, on trouvera une chaîne de la forme :

```
param1=val1&TextField=abcdef&param2=val2
```

si l'utilisateur a saisi [abcdef] dans le champ de saisie.

- ligne 13 : [PasswordField] récupèrera **la valeur postée** par les lignes suivantes du formulaire :

```
1.      <!-- le champ de saisie d'un mot de passe -->
2.      <tr>
3.          <td>Mot de passe</td>
4.          <td>
5.              <input type="password" name="PasswordField" value="secret" size="30" />
6.          </td>
7.      </tr>
```

Ligne 5, l'attribut [name] du champ de saisie est le nom de la propriété qui va être initialisée. Dans les données postées, on trouvera une chaîne de la forme :

```
param1=val1&PasswordField=abcdef&param2=val2
```

si l'utilisateur a saisi [abcdef] dans le champ de saisie.

- ligne 14 : [TextAreaField] récupèrera la valeur postée par les lignes suivantes du formulaire :

```
1.      <!-- le champ de saisie texte multilignes -->
2.      <tr>
3.          <td>Boite de saisie</td>
4.          <td>
5.              <textarea name="TextAreaField" cols="40" rows="3">ligne1
6. ligne2</textarea>
7.          </td>
8.      </tr>
```

Ligne 5, l'attribut [name] du champ de saisie est le nom de la propriété qui va être initialisée. Dans les données postées, on trouvera une chaîne de la forme :

```
param1=val1&TextAreaField=abcdef%0D%0Ahijk&param2=val2
```

si l'utilisateur a saisi [abcdef] suivi d'un saut de ligne et de [ijk] dans le champ de saisie.

- ligne 15 : [DropDownListField] récupèrera **la valeur postée** par les lignes suivantes du formulaire :

```
1.      <!-- la liste déroulante -->
2.      <tr>
3.          <td>Liste déroulante</td>
4.          <td>
5.              <select name="DropDownListField">
6. <option value="1" >choix1</option>
7. <option value="2" selected="selected">choix2</option>
8. <option value="3" >choix3</option>
9.             </select>
10.        </td>
11.    </tr>
```

Ligne 5, l'attribut [name] de la balise <select> est le nom de la propriété qui va être initialisée. Dans les données postées, on trouvera une chaîne de la forme :

```
param1=val1&DropDownListField=1&param2=val2
```

si l'utilisateur a sélectionné l'option [choix1]. C'est l'attribut [value] de l'option sélectionnée qui est postée.

- ligne 16 : [SingleChoiceListField] récupèrera **la valeur postée** par les lignes suivantes du formulaire :

```
1.      <!-- la liste à choix unique -->
2.      <tr>
3.          <td>Liste à choix unique</td>
4.          <td>
5.              <select name="SimpleChoiceListField" size="3">
6.      <option value="1" >liste1</option>
7.      <option value="2" >liste2</option>
8.      <option value="3" selected="selected">liste3</option>
9.      <option value="4" >liste4</option>
10.     <option value="5" >liste5</option>
11.
12.         </select>
13.     </tr>
```

Ligne 5, l'attribut [name] de la balise <select> est le nom de la propriété qui va être initialisée. C'est l'attribut [size="3"] qui fait qu'on n'a pas une liste déroulante. Dans les données postées, on trouvera une chaîne de la forme :

```
param1=val1&SimpleChoiceListField=3&param2=val2
```

si l'utilisateur a sélectionné l'option [liste3]. C'est l'attribut [value] de l'option sélectionnée qui est postée. Le paramètre [SingleChoiceListField] peut être absent de la chaîne postée si aucun élément n'a été sélectionné.

- ligne 17 : [MultipleChoiceListField] récupèrera **les valeurs postées** par les lignes suivantes du formulaire :

```
1.      <!-- la liste à choix multiple -->
2.      <tr>
3.          <td>Liste à choix multiple</td>
4.          <td>
5.              <select name="MultipleChoiceListField" size="3" multiple="multiple">
6.      <option value="1" selected="selected">liste1</option>
7.      <option value="2" >liste2</option>
8.      <option value="3" selected="selected">liste3</option>
9.      <option value="4" >liste4</option>
10.     <option value="5" >liste5</option>
11.         </select>
12.     </tr>
```

Ligne 5, l'attribut [name] de la balise <select> est le nom de la propriété qui va être initialisée. C'est l'attribut [size="3"] qui fait qu'on n'a pas une liste déroulante et l'attribut [multiple] qui fait que l'utilisateur peut sélectionner plusieurs éléments en maintenant appuyée la touche [Ctrl]. Dans les données postées, on trouvera une chaîne de la forme :

```
param1=val1&MultipleChoiceListField=1&MultipleChoiceListField=3&param2=val2
```

si l'utilisateur a sélectionné les options [liste1] et [liste3]. C'est l'attribut [value] des options sélectionnées qui est postée. C'est parce que plusieurs paramètres de même nom peuvent être postés que [MultipleChoiceListField] est un tableau de valeurs et non une valeur simple. Si aucune case n'est cochée, le paramètre [MultipleChoiceListField] sera absent de la chaîne postée et la propriété de même nom du modèle ne sera pas initialisée.

Les différents champs de saisie présentés précédemment vont recevoir les valeurs postées par le formulaire. On peut également les initialiser avant d'envoyer le formulaire. C'est ce qui a été fait ici :

```
1.      // initialisation champs
2.      RadioButtonField = "2";
3.      CheckBoxesField = new string[] { "2" };
4.      TextField = "quelques mots";
5.      PasswordField = "secret";
6.      TextAreaField = "ligne1\nligne2";
7.      DropDownListField = "2";
8.      SimpleChoiceListField = "3";
9.      MultipleChoiceListField = new string[] { "1", "3" };
```

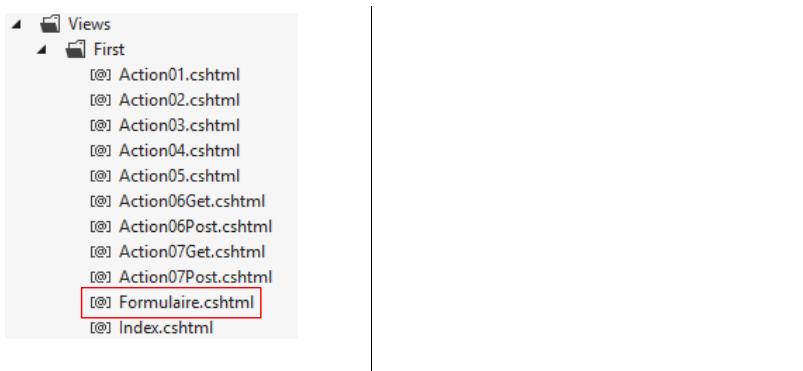
Si ces valeurs avaient été obtenues après un POST du formulaire, cela signifierait que l'utilisateur a :

- ligne 2 : coché l'option [non] du bouton radio ;
- ligne 3 : coché l'option [2] des cases à cocher ;
- ligne 4 : tapé [quelques mots] dans le champ de saisie ;
- ligne 5 : tapé [secret] comme mot de passe ;
- ligne 6 : tapé [ligne1\nligne2] dans le champ de saisie multilignes ;
- ligne 7 : sélectionné l'option [choix2] de la liste déroulante ;
- ligne 8 : sélectionné l'option [liste3] de la liste à choix unique ;
- ligne 9 : sélectionné les options [liste1] et [liste3] de la liste à choix multiples;

Nous allons faire comme si un POST avait eu lieu et que l'on voulait renvoyer le formulaire tel qu'il a été saisi. C'est notamment ce qui est fait lorsqu'on renvoie à l'utilisateur un formulaire erroné. Celui-ci est renvoyé tel qu'il a été saisi.

#### 1.5.6.4 La vue [Formulaire]

La vue [/First/Formulaire.cshtml] affiche le formulaire :



```
1. @model Exemple_03.Models.ViewModel08
2. @using Exemple_03.Models
3. @{
4.     Layout = null;
5. }
6. <html>
7. <head>
8.     <meta name="viewport" content="width=device-width" />
9.     <title>Formulaire</title>
10. </head>
11. <body>
12.     <form method="post" action="Action08Post">
13.         <h2>Formulaire ASP.NET MVC</h2>
14.         <h3>Affiché par : @ViewBag.info</h3>
15.         <table>
16.             <thead></thead>
17.             <tbody>
18.                 <!-- les boutons radio -->
19.                 <tr>
20.                     <td>Etes-vous marié(e)</td>
21.                     <td>
22.                         @foreach (ApplicationModel.Item item in @Model.RadioButtonFieldItems)
23.                         {
24.                             string strChecked = item.Value == @Model.RadioButtonField ? "checked=\"checked\" : "";
25.                             <input type="radio" name="RadioButtonField" value="@item.Value" @strChecked/>@item.Label
26.                         <text/>
27.                         }
28.                     </td>
29.                 </tr>
30. ...
31.             </tbody>
32.         </table>
33.         <input type="submit" value="Valider" />
34.     </form>
35. </body>
36. </html>
```

- ligne 1 : [ViewModel08] est le modèle du formulaire ;
- ligne 12 : la balise <form> du formulaire. Celui-ci sera posté avec la méthode [POST] (attribut *method*) à l'URL [/First/Action08Post] (attribut *action*) ;

- ligne 33 : le bouton de type [submit] qui sert à poster le formulaire ;
- lignes 22-27 : affichent les boutons radio :

Etes-vous marié(e)  oui  non

- ligne 22 : on parcourt la collection affichée par le bouton radio ;
- ligne 24 : le bouton qui a comme attribut [value] la valeur de la propriété [RadioButtonField] doit être coché. Pour cela, il doit avoir l'attribut [checked="checked"] ;
- ligne 25 : génération de la balise <input type="radio"> avec pour valeur [@item.Value] et pour libellé [@item.Label] ;
- ligne 26 : la balise <text/> n'est pas une balise HTML reconnue. Elle est là pour [Razor]. A sa rencontre, [Razor] va générer un saut de ligne. Cela n'a pas d'incidence sur le formulaire affiché mais cela en a une sur le code HTML généré. Les balises <input type="radio"> sont alors sur deux lignes différentes au lieu d'être sur la même ligne. Cela rend plus lisible le code, lorsqu'avec le navigateur on demande à voir le code source de la page affichée ;

Nous passons en revue, les autres éléments de la vue :

```

1.      <!-- les cases à cocher -->
2.      <tr>
3.          <td>Cases à cocher</td>
4.          <td>
5.              @{
6.                  foreach (ApplicationModel.Item item in @Model.CheckBoxesFieldItems)
7.                  {
8.                      string strChecked = @Model.CheckBoxesField.Contains(item.Value) ? "checked=\"checked\"" : "";
9.                      <input type="checkbox" name="CheckBoxesField" value="@item.Value" @strChecked>@item.Label
10.                 <text/>
11.             }
12.         }
13.     </td>

```

- ligne 6 : on parcourt la collection affichée par les cases à cocher ;
- ligne 8 : une case qui a comme attribut [value] l'une des valeurs de la propriété [CheckBoxesField] doit être cochée. Pour cela, il doit avoir l'attribut [checked="checked"]. On utilise une expression LINQ qui permet de savoir si une valeur est contenue dans un tableau ;
- ligne 25 : génération de la balise <input type="checkbox"> avec pour valeur [@item.Value] et pour libellé [@item.Label] ;

```

1.      <!-- le champ de saisie texte monoligne -->
2.          <tr>
3.              <td>Champ de saisie</td>
4.              <td>
5.                  <input type="text" name="TextField" value="@Model.TextField" size="30" />
6.              </td>
7.          </tr>
8.      <!-- le champ de saisie d'un mot de passe -->
9.          <tr>
10.             <td>Mot de passe</td>
11.             <td>
12.                 <input type="password" name="PasswordField" value="@Model.PasswordField" size="30" />
13.             </td>
14.         </tr>
15.      <!-- le champ de saisie texte multilignes -->
16.          <tr>
17.              <td>Boîte de saisie</td>
18.              <td>
19.                  <textarea name="TextAreaField" cols="40" rows="3">@Model.TextAreaField</textarea>
20.              </td>
21.          </tr>

```

- lignes 5, 12 : on donne à l'attribut [value] de la balise, la valeur du modèle ;
- ligne 19 : idem mais avec une syntaxe différente.

```

1.      <!-- la liste déroulante -->
2.      <tr>
3.          <td>Liste déroulante</td>
4.          <td>
5.              <select name="DropDownListField">
6.                  @{
7.                      foreach (ApplicationModel.Item item in @Model.DropDownFieldItems)
8.                      {
9.                          string strChecked = item.Value == @Model.DropDownField ? "selected=\"selected\"" : "";
10.                         <option value="@item.Value" @strChecked>@item.Label</option>
11.                     }

```

```

12.         }
13.     </select>
14. </tr>

```

- ligne 7 : on parcourt la collection affichée par la liste déroulante ;
- ligne 9 : une option qui a comme attribut [value] la valeur de la propriété [DropDownListField] doit alors être sélectionnée. Pour cela, elle doit avoir l'attribut [selected="selected"] ;
- ligne 25 : génération de la balise <option value="valeur">libellé</option> avec pour valeur [@item.Value] et pour libellé [@item.Label] ;

```

1.      <!-- la liste à choix unique -->
2.      <tr>
3.          <td>Liste à choix unique</td>
4.          <td>
5.              <select name="SimpleChoiceListField" size="3">
6.                  @{
7.                      foreach (ApplicationModel.Item item in @Model.SimpleChoiceListFieldItems)
8.                      {
9.                          string strChecked = item.Value == @Model.SimpleChoiceListField ? "selected=\"selected\"" : "";
10.                         <option value="@item.Value" @strChecked>@item.Label</option>
11.                     }
12.                 }
13.             </select>
14. </tr>

```

L'explication est la même que pour la liste déroulante.

```

1.      <!-- la liste à choix multiple -->
2.      <tr>
3.          <td>Liste à choix multiple</td>
4.          <td>
5.              <select name="MultipleChoiceListField" size="3" multiple="multiple">
6.                  @{
7.                      foreach (ApplicationModel.Item item in @Model.MultipleChoiceListFieldItems)
8.                      {
9.                          string strChecked = @Model.MultipleChoiceListField.Contains(item.Value) ? "selected=\"selected\"" :
"";
10.                         <option value="@item.Value" @strChecked>@item.Label</option>
11.                     }
12.                 }
13.             </select>
14. </tr>

```

- ligne 7 : on parcourt la collection affichée par la liste ;
- ligne 9 : une option qui a comme attribut [value] l'une des valeurs de la propriété [MultipleChoiceListField] doit être sélectionnée. Pour cela, elle doit avoir l'attribut [selected="selected"]. On utilise une expression LINQ qui permet de savoir si une valeur est contenue dans un tableau ;
- ligne 10 : génération de la balise <option value="valeur">libellé</option> avec pour valeur [@item.Value] et pour libellé [@item.Label] ;

#### 1.5.6.5 Traitement du POST du formulaire

Nous avons vu que le formulaire allait être posté à l'action [Action08Post] :

```
<form method="post" action="Action08Post">
```

L'action [Action08Post] est la suivante :

```

1.      // Action08-POST
2.      [HttpPost]
3.      public ViewResult Action08Post(ApplicationModel application, FormCollection posted)
4.      {
5.          ViewBag.info = string.Format("Contrôleur={0}, Action={1}", RouteData.Values["controller"],
RouteData.Values["action"]);
6.          ViewModel08 modèle = new ViewModel08(application);
7.          TryUpdateModel(modèle, posted);
8.          return View("Formulaire", modèle);
9.      }

```

- ligne 3 : le modèle de l'application est en paramètre ainsi que les valeurs postées. Celles-ci sont disponibles dans un type [FormCollection]. La valeur du paramètre [RadioButtonField] posté est obtenue par l'expression `posted["RadioButtonField"]`. On obtient là une chaîne de caractères ou le pointeur `null`. Si on écrit `posted["CheckboxesField"]`, on aura un tableau de chaînes de caractères ou le pointeur `null` ;
- pourquoi ne pas écrire :

```
public ViewResult Action08Post(ApplicationModel application, ViewModel08 posted)
```

Il y a deux raisons :

- la première est que le framework va instancier le modèle [ViewModel08] avec le constructeur sans paramètres, ce qui aura pour effet de ne pas initialiser les collections du modèle ;
- le seconde est qu'on veut contrôler ce qui va dans le modèle. On sait qu'il y a quatre sources possibles pour le modèle, les paramètres d'un GET, d'un POST, de la route utilisée et ceux d'un fichier *uploadé*. Ici, on veut initialiser le modèle uniquement avec les valeurs postées.
- ligne 6 : on instancie le modèle en utilisant le bon constructeur ;
- ligne 7 : on l'initialise avec les valeurs postées. Après cette opération, le modèle correspond à la saisie qui a été faite par l'utilisateur ;
- ligne 8 : on affiche de nouveau le formulaire. L'utilisateur va le retrouver tel qu'il a été saisi.

Voyons un exemple :

The image shows two side-by-side browser windows. Both windows have a title bar 'Formulaire' and a URL 'localhost:50803/First/Action08Post'. The left window, labeled [1], is titled 'Formulaire ASP.NET MVC' and contains the following fields:

- Etes-vous marié(e) with radio buttons 'oui' (selected) and 'non'.
- Cases à cocher with checkboxes '1' (checked), '2' (unchecked), and '3' (checked).
- Champ de saisie with input 'd'autres mots'.
- Mot de passe with input '...'.
- Boîte de saisie with input 'ASP.NET MVC par l'exemple'.
- Liste déroulante with dropdown 'choix1' containing items 'liste1', 'liste2', 'liste3', and 'liste4'.
- Liste à choix unique with dropdown 'choix1' containing items 'liste1', 'liste2', 'liste3', and 'liste4'.
- Liste à choix multiple with dropdown 'choix1' containing items 'liste1', 'liste2', 'liste3', and 'liste4'.
- A 'Valider' button at the bottom.

The right window, labeled [2], is also titled 'Formulaire ASP.NET MVC' and shows the same fields, but their values have changed to reflect the POST data:

- Etes-vous marié(e) with radio buttons 'oui' (selected) and 'non'.
- Cases à cocher with checkboxes '1' (checked), '2' (unchecked), and '3' (checked).
- Champ de saisie with input 'd'autres mots'.
- Mot de passe with input '...'.
- Boîte de saisie with input 'ASP.NET MVC par l'exemple'.
- Liste déroulante with dropdown 'choix1' containing items 'liste1', 'liste2', 'liste3', and 'liste4'.
- Liste à choix unique with dropdown 'choix1' containing items 'liste1', 'liste2', 'liste3', and 'liste4'.
- Liste à choix multiple with dropdown 'choix1' containing items 'liste1', 'liste2', 'liste3', and 'liste4'.
- A 'Valider' button at the bottom.

En [2], le résultat du [POST] reflète bien ce qui a été saisi en [1].

#### 1.5.6.6 Traitement des anomalies du POST

Nous avons dit que si aucune valeur n'était cochée ou sélectionnée pour les champs [CheckBoxesField, SimpleChoiceListField, MultipleChoiceListField] alors les paramètres correspondant ne faisaient pas partie de la chaîne postée et que donc les propriétés de mêmes noms du modèle n'étaient pas initialisées.

Voyons l'exemple suivant :

- en [1], aucune case à cocher ne l'a été ;
- en [2], le [POST] ramène une case cochée.

L'explication est la suivante :

- parce qu'il n'y a aucune case cochée, le paramètre [CheckBoxesField] ne fait pas partie des valeurs postées ;
- l'action [Action08Post] procède de la façon suivante :

```

1.   [HttpPost]
2.   public ViewResult Action08Post(ApplicationModel application, FormCollection posted)
3.   {
4.     ViewBag.info = ...
5.     ViewModel08 modèle = new ViewModel08(application);
6.     TryUpdateModel(modèle,posted);
7.     return View("Formulaire", modèle);
8.   }

```

- ligne 5 : le modèle du formulaire est instancié. Or le constructeur utilisé affecte le tableau ["2"] à la propriété [CheckBoxesField] ;
- ligne 6 : les valeurs postées sont enregistrées dans le modèle. Comme le paramètre [CheckBoxesField] ne fait pas partie des valeurs postées, la propriété de même nom n'est pas affectée. Elle reste donc avec sa valeur ["2"], ce qui fait qu'à l'affichage la case n° 2 est cochée alors qu'elle ne devrait pas l'être.

Ce problème peut se régler de diverses façons. Nous choisissons de le régler dans le code de l'action [Action08Post] :

```

1. // Action08-POST
2. [HttpPost]
3. public ViewResult Action08Post(ApplicationModel application, FormCollection posted)
4. {
5.   ViewBag.info = string.Format("Contrôleur={0}, Action={1}", RouteData.Values["controller"],
  RouteData.Values["action"]);
6.   ViewModel08 modèle = new ViewModel08(application);
7.   TryUpdateModel(modèle,posted);
8.   // traitement des valeurs non postées
9.   if (posted["CheckBoxesField"] == null)
10.  {
11.    modèle.CheckBoxesField = new string[] { };
12.  }
13.  if (posted["SimpleChoiceListField"] == null)
14.  {
15.    modèle.SimpleChoiceListField = "";
16.  }
17.  if (posted["MultipleChoiceListField"] == null)
18.  {
19.    modèle.MultipleChoiceListField = new string[] { };
20.  }
21.  // affichage formulaire
22.  return View("Formulaire", modèle);
23. }

```

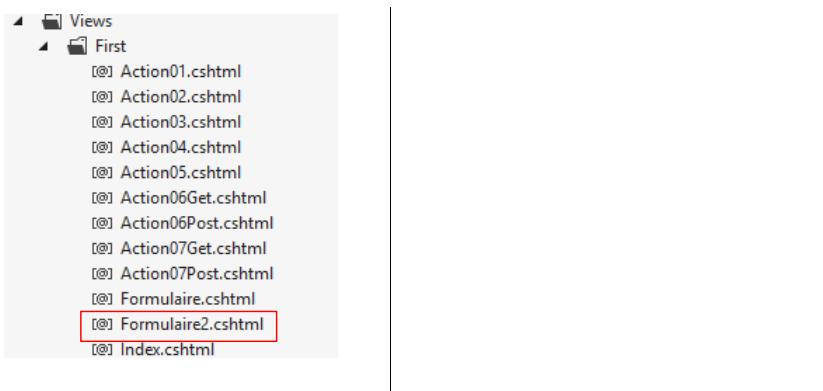
- lignes 9-20 : on vérifie si certains paramètres ont été postés ou non. Si ce n'est pas le cas, on les initialise avec la valeur qui correspond à l'absence de saisie faite par l'utilisateur. Le test n'a pas été fait pour la liste déroulante qui elle, a toujours un élément sélectionné, ce qui n'est pas le cas pour les autres listes.

Le lecteur est invité à tester cette nouvelle version.

## 1.5.7 Utilisation de méthodes spécialisées dans la génération de formulaire

### 1.5.7.1 Le nouveau formulaire

Nous créons un nouveau formulaire [Formulaire2.cshtml] qui délivrera un formulaire identique au précédent :



Revenons sur le code utilisé pour générer la liste déroulante du formulaire :

```
1.      <!-- la liste déroulante -->
2.      <tr>
3.          <td>Liste déroulante</td>
4.          <td>
5.              <select name="DropDownListField">
6.                  @{
7.                      foreach (ApplicationModel.Item item in @Model.DropDownListFieldItems)
8.                      {
9.                          string strChecked = item.Value == @Model.DropDownListField ? "selected=\"selected\"" : "";
10.                         <option value="@item.Value" @strChecked>@item.Label</option>
11.                     }
12.                 }
13.             </select>
14.         </tr>
```

Ce code présente deux inconvénients :

- le plus important est qu'on perd de vue la nature du composant, ici une liste déroulante, à cause de la complexité du code ;
- ligne 5 : si on se trompe dans le nom de la propriété du modèle à utiliser comme attribut [name], on ne s'en apercevra qu'à l'exécution.

ASP.NET MVC offre des méthodes spécialisées appelées [HTML Helpers] qui, comme leur nom l'indique, vise à faciliter la génération du HTML, notamment pour les formulaires. Avec ces classes, la liste déroulante précédente s'écrit comme suit :

```
1.      <!-- la liste déroulante -->
2.      <tr>
3.          <td>Liste déroulante</td>
4.          <td>@Html.DropDownListFor(m => m.DropDownListField,
5.             new SelectList(@Model.DropDownListFieldItems, "Value", "Label"))
6.         </td>
7.     </tr>
```

La liste déroulante est générée par les lignes 4-5. Le code est nettement moins complexe. Le code HTML généré pour la liste déroulante est le suivant :

```
1.      <!-- la liste déroulante -->
2.      <tr>
3.          <td>Liste déroulante</td>
4.          <td><select id="DropDownListField" name="DropDownListField"><option value="1">choix1</option>
5.             <option selected="selected" value="2">choix2</option>
6.             <option value="3">choix3</option>
7.         </select></td>
8.     </tr>
```

- ligne 4 : l'attribut [name] est correct ;
- lignes 4-6 : les options sont correctement générées et la bonne option a été sélectionnée.

Revenons au code qui a généré ces lignes HTML :

```
@Html.DropDownListFor(m => m.DropDownListField, new SelectList(@Model.DropDownListFieldItems, "Value", "Label"))
```

- le premier paramètre est une fonction *lambda* (c'est son nom) où **m** représente le modèle de la vue et **m.DropDownListField** est une propriété de ce modèle. Le générateur de code HTML va utiliser le nom de cette propriété pour générer les attributs [id] et [name] du [select] qui va être généré. Si on utilise une propriété inexistante, on aura une erreur à la compilation et non plus à l'exécution. C'est une amélioration vis à vis de la solution précédente où les erreurs de nommage n'étaient détectées qu'à l'exécution ;
- le second paramètre sert à désigner la collection d'éléments qui va alimenter la liste déroulante. La classe [SelectList] permet de construire cette collection :
  - son premier paramètre est une collection quelconque d'éléments. Ici on a une collection de type [Item] ;
  - son second paramètre est la propriété des éléments qui fournira la valeur de la balise <option>. Ici, c'est la propriété [Value] de la classe [Item] ;
  - son troisième paramètre est la propriété des éléments qui fournira le libellé de la balise <option>. Ici, c'est la propriété [Label] de la classe [Item] ;
- pour savoir quelle option doit être sélectionnée (attribut *selected*), le framework fait comme nous : il compare la valeur de l'option à la valeur actuelle de la propriété [DropDownListField].

Voyons maintenant les autres méthodes que nous pouvons utiliser :

### Boutons radio

Le nouveau code est le suivant :

```
1.      <!-- les boutons radio -->
2.      <tr>
3.          <td>Etes-vous marié(e)</td>
4.          <td>
5.              @{
6.      foreach (ApplicationModel.Item item in @Model.RadioButtonFieldItems)
7.      {
8.          @Html.RadioButtonFor(m => m.RadioButtonField, @item.Value)@item.Label
9.          <text/>
10.     }
11.    }
12.    </td>
13. </tr>
```

Le code HTML généré est le suivant :

```
1.      <!-- les boutons radio -->
2.      <tr>
3.          <td>Etes-vous marié(e)</td>
4.          <td>
5.              <input id="RadioButtonField" name="RadioButtonField" type="radio" value="1" />oui
6.              <input checked="checked" id="RadioButtonField" name="RadioButtonField" type="radio" value="2" />non
7.          </td>
8.      </tr>
```

La méthode utilisée est [Html.RadioButtonFor] :

```
@Html.RadioButtonFor(m => m.RadioButtonField, @item.Value)
```

- le premier paramètre est la propriété du modèle qui va être associée au bouton radio (attribut [name]) ;
- le second paramètre est la valeur à attribuer au bouton radio (attribut [value]).

### Cases à cocher

Le code évolue de la façon suivante :

```
1.      <!-- les cases à cocher -->
2.      <tr>
3.          <td>Cases à cocher</td>
4.          <td>
5.              @{
6.                  @Html.CheckBoxFor(m=>m.CheckBoxField1) @Model.CheckBoxesFieldItems[0].Label
7.                  @Html.CheckBoxFor(m=>m.CheckBoxField2) @Model.CheckBoxesFieldItems[1].Label
8.                  @Html.CheckBoxFor(m=>m.CheckBoxField3) @Model.CheckBoxesFieldItems[2].Label
9.              }
10.         </td>
```

La méthode utilisée pour générer des cases à cocher est [Html.CheckBoxFor] :

```
@Html.CheckBoxFor(m=>m.Propriété)
```

La paramètre est la propriété **booléenne** du modèle qui sera associée à la case à cocher. Si [Propriété=true], la case sera cochée. Si [Propriété=false], la case ne sera pas cochée. Dans tous les cas, l'attribut [value] vaut *true*. Le code HTML généré est le suivant :

```
1. <input id="Propriété" name="Propriété" type="checkbox" value="true" />
2. <input name="Propriété" type="hidden" value="false" />
```

- ligne 1 : la case à cocher avec l'attribut [value="true"] ;
- ligne 2 : un champ caché (type=hidden) de même nom [Propriété] que la case à cocher avec l'attribut [value="false"]. Pourquoi deux balises [input] de même nom ? Il y a deux cas :
- la case de la ligne 1 est cochée. Alors la chaîne de paramètres postée est **Propriété=true&Propriété=false** (lignes 1 et 2). Comme la propriété [Propriété] n'attend qu'une valeur, on peut penser que le framework affecte la valeur [true] à [Propriété]. Il lui suffirait de faire un OU logique entre les valeurs reçues pour y arriver ;
- la case de la ligne 1 n'est pas cochée. Alors la chaîne de paramètres postée est **Propriété=false** (ligne 2 uniquement) et alors la propriété [Propriété] reçoit la valeur [false], ce qui est correct (la case n'a pas été cochée).

### Champ de saisie monoligne

Le nouveau code est le suivant :

```
1.      <!-- le champ de saisie texte monoligne -->
2.      <tr>
3.          <td>Champ de saisie</td>
4.          <td>
5.              @Html.TextBoxFor(m => m.TextField, new { size = "30" })
6.          </td>
7.      </tr>
```

Le code HTML généré est le suivant :

```
1.      <!-- le champ de saisie texte monoligne -->
2.      <tr>
3.          <td>Champ de saisie</td>
4.          <td>
5.              <input id="TextField" name="TextField" size="30" type="text" value="quelques mots" />
6.          </td>
7.      </tr>
```

La méthode utilisée est la suivante :

```
@Html.TextBoxFor(m => m.TextField, new { size = "30" })
```

- le premier paramètre précise la propriété du modèle associé au champ de saisie. Le nom de la propriété sera utilisé dans les attributs [name] et [id] de la balise <input> générée et sa valeur sera affectée à l'attribut [value] ;
- le second paramètre est une classe anonyme précisant certains attributs de la balise HTML générée, ici l'attribut [size].

### Champ de saisie d'un mot de passe

Le nouveau code est le suivant :

```
1.      <!-- le champ de saisie d'un mot de passe -->
2.      <tr>
3.          <td>Mot de passe</td>
4.          <td>
5.              @Html.PasswordFor(m => m.PasswordField, new { size = "15" })
6.          </td>
7.      </tr>
```

Le code HTML généré est le suivant :

```
1.      <!-- le champ de saisie d'un mot de passe -->
2.      <tr>
3.          <td>Mot de passe</td>
4.          <td>
5.              <input id="PasswordField" name="PasswordField" size="15" type="password" />
6.          </td>
7.      </tr>
```

La méthode utilisée est la suivante :

```
@Html.PasswordFor(m => m.PasswordField, new { size = "15" })
```

Le fonctionnement est analogue à celui de la méthode [Html.TextBoxFor].

### Champ de saisie multi-lignes

Le nouveau code est le suivant :

```
1.      <!-- le champ de saisie texte multilignes -->
2.      <tr>
3.          <td>Boîte de saisie</td>
4.          <td>
5.              @Html.TextAreaFor(m => m.TextAreaField, new { cols = "30", rows = "5" })
6.          </td>
7.      </tr>
```

Le code HTML généré est le suivant :

```
1.      <!-- le champ de saisie texte multilignes -->
2.      <tr>
3.          <td>Boîte de saisie</td>
4.          <td>
5.              <textarea cols="30" id="TextAreaField" name="TextAreaField" rows="5">
6.      ligne1
7.      ligne2</textarea>
8.          </td>
9.      </tr>
```

La méthode utilisée est la suivante :

```
@Html.TextAreaFor(m => m.TextAreaField, new { cols = "30", rows = "5" })
```

Le fonctionnement est analogue à celui de la méthode [Html.TextBoxFor].

### Liste à choix unique

Le nouveau code est le suivant :

```
1.      <!-- la liste à choix unique -->
2.      <tr>
3.          <td>Liste à choix unique</td>
4.          <td>
5.              @Html.DropDownListFor(m => m.SimpleChoiceListField, new SelectList(@Model.SimpleChoiceListFieldItems,
"Value", "Label"), new { size = "3" })
6.          </td>
```

et le code HTML généré le suivant :

```
1.      <!-- la liste à choix unique -->
2.      <tr>
3.          <td>Liste à choix unique</td>
4.          <td>
5.              <select id="SimpleChoiceListField" name="SimpleChoiceListField" size="3">
6.                  <option value="1">liste1</option>
7.                  <option value="2">liste2</option>
8.                  <option selected="selected" value="3">liste3</option>
9.                  <option value="4">liste4</option>
10.                 <option value="5">liste5</option>
11.             </select>
12.         </td>
```

Nous avons déjà étudié la méthode [Html.DropDownListFor]. La seule différence est ici le troisième paramètre qui sert à préciser un attribut [size] différent de 1. C'est cette caractéristique qui fait passer d'une liste déroulante [size=1] à une liste simple.

### La liste à choix multiples

Le nouveau code est le suivant :

```
1.      <!-- la liste à choix multiple -->
2.      <tr>
3.          <td>Liste à choix multiple</td>
4.          <td>
5.              @Html.ListBoxFor(m => m.MultipleChoiceListField, new SelectList(@Model.MultipleChoiceListFieldItems, "Value",
"Label"), new { size = "5" })
6.          </td>
```

et le code HTML générée le suivant :

```
1.      <!-- la liste à choix multiple -->
2.      <tr>
3.          <td>Liste à choix multiple</td>
4.          <td>
5.              <select id="MultipleChoiceListField" multiple="multiple" name="MultipleChoiceListField" size="5">
6.                  <option selected="selected" value="1">liste1</option>
7.                  <option value="2">liste2</option>
8.                  <option selected="selected" value="3">liste3</option>
9.                  <option value="4">liste4</option>
10.                 <option value="5">liste5</option>
11.             </select>
12.         </tr>
```

La méthode

```
@Html.ListBoxFor(m => m.MultipleChoiceListField, new SelectList(@Model.MultipleChoiceListFieldItems, "Value", "Label"), new { size = "5" })
```

fonctionne comme la méthode [Html.DropDownListFor] si ce n'est qu'elle génère une liste à sélection multiple. Les options sélectionnées sont celles qui ont leur valeur (attribut *value*) dans le tableau [MultipleChoiceListField].

La balise <form> peut être elle aussi, générée avec une méthode :

```
1.  @using (Html.BeginForm("Action09Post", "First"))
2.  {
3.  ...
4. }
```

Le code HTML générée est le suivant :

```
1.  <form action="/First/Action09Post" method="post">
2.  ...
3.  </form>
```

La méthode

```
Html.BeginForm("Action09Post", "First")
```

a pour premier paramètre le nom d'une **action** et pour second paramètre le nom d'un **contrôleur**.

### 1.5.7.2    *Les actions et le modèle*

Le formulaire sera délivré par l'action [Action09Get] suivante :

```
1.  // Action09-GET
2.  [HttpGet]
3.  public ViewResult Action09Get(ApplicationModel application)
4.  {
5.      ViewBag.info = string.Format("Contrôleur={0}, Action={1}", RouteData.Values["controller"],
RouteData.Values["action"]);
6.      return View("Formulaire2", new ViewModel09(application));
7. }
```

La vue délivrée ligne 6 est [Formulaire2] associée au modèle [ViewModel09] suivant :

```
1.  using System.ComponentModel.DataAnnotations;
2.  using System.Web.Mvc;
3.  using Exemple_03.Models;
4.
5.  namespace Exemple_03.Models
6.  {
7.      public class ViewModel09
8.      {
9.          // les champs de saisie
10.         public string RadioButtonField { get; set; }
11.         public bool CheckBoxField1 { get; set; }
12.         public bool CheckBoxField2 { get; set; }
13.         public bool CheckBoxField3 { get; set; }
14.         public string TextField { get; set; }
15.         public string PasswordField { get; set; }
16.         public string TextAreaField { get; set; }
17.         public string DropDownListField { get; set; }
18.         public string SimpleChoiceListField { get; set; }
19.         public string[] MultipleChoiceListField { get; set; }
```

```

20.    // les collections à afficher dans le formulaire
21.    public ApplicationModel.Item[] RadioButtonFieldItems { get; set; }
22.    public ApplicationModel.Item[] CheckBoxesFieldItems { get; set; }
23.    public ApplicationModel.Item[] DropDownListFieldItems { get; set; }
24.    public ApplicationModel.Item[] SimpleChoiceListFieldItems { get; set; }
25.    public ApplicationModel.Item[] MultipleChoiceListFieldItems { get; set; }
26.
27.    // constructeurs
28.    public ViewModel09()
29.    {
30.    }
31.
32.
33.    public ViewModel09(ApplicationModel application)
34.    {
35.        // initialisation collections
36.        RadioButtonFieldItems = application.RadioButtonFieldItems;
37.        CheckBoxesFieldItems = application.CheckBoxesFieldItems;
38.        DropDownListFieldItems = application.DropDownListFieldItems;
39.        SimpleChoiceListFieldItems = application.SimpleChoiceListFieldItems;
40.        MultipleChoiceListFieldItems = application.MultipleChoiceListFieldItems;
41.        // initialisation champs
42.        RadioButtonField = "2";
43.        CheckBoxField2 = true;
44.        TextField = "quelques mots";
45.        PasswordField = "secret";
46.        TextAreaField = "ligne1\nligne2";
47.        DropDownListField = "2";
48.        SimpleChoiceListField = "3";
49.        MultipleChoiceListField = new string[] { "1", "3" };
50.    }
51. }
52. }
```

[ViewModel09] diffère de [ViewModel08] par sa gestion des cases à cocher. Plutôt que d'avoir un tableau de trois cases à cocher, on a utilisé trois cases à cocher séparées (lignes 11-13).

Le formulaire sera traité par l'action [Action09Post] suivante :

```

1.    // Action09-POST
2.    [HttpPost]
3.    public ViewResult Action09Post(ApplicationModel application, FormCollection posted)
4.    {
5.        ViewBag.info = string.Format("Contrôleur={0}, Action={1}", RouteData.Values["controller"],
   RouteData.Values["action"]);
6.        ViewModel09 modèle = new ViewModel09(application);
7.        TryUpdateModel(modèle, posted);
8.        // traitement des valeurs non postées
9.        if (posted["SimpleChoiceListField"] == null)
10.        {
11.            modèle.SimpleChoiceListField = "";
12.        }
13.        if (posted["MultipleChoiceListField"] == null)
14.        {
15.            modèle.MultipleChoiceListField = new string[] { };
16.        }
17.        // affichage formulaire
18.        return View("Formulaire2", modèle);
19.    }
```

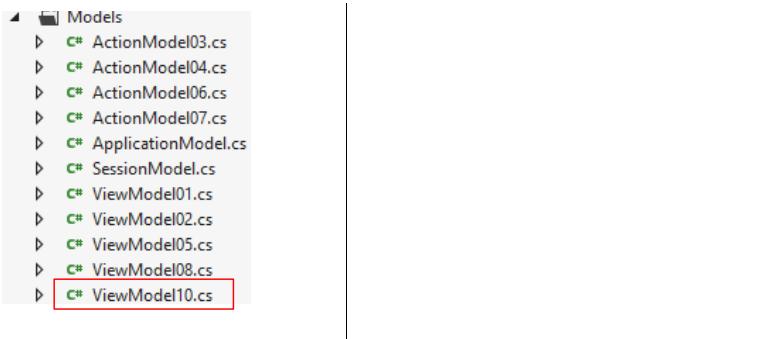
L'action [Action09Post] est identique à l'action [Action08Post] sauf sur deux points :

- ligne 18 : la vue [Formulaire2] est utilisée à la place de la vue [Formulaire] ;
- on n'a plus la gestion des cases à cocher qui ont été non cochées. C'est désormais géré correctement par la méthode [Html.CheckBoxFor].

## 1.5.8 Génération d'un formulaire à partir des métadonnées du modèle

Il existe d'autres méthodes que les précédentes pour générer un formulaire. L'une d'elles consiste à associer des informations à une rubrique du modèle qui vont permettre au framework MVC de savoir quelle balise de saisie il doit générer. On appelle ces informations des métadonnées.

Considérons le modèle de vue [ViewModel10] suivant :



```
1.  using System;
2.  using System.ComponentModel.DataAnnotations;
3.  using System.Drawing;
4.
5.  namespace Exemple_03.Models
6.  {
7.      public class ViewModel10
8.      {
9.          [Display(Name="Text")]
10.         [DataType(DataType.Text)]
11.         public string Text { get; set; }
12.
13.        [Display(Name = "TextArea")]
14.        [DataType(DataType.MultilineText)]
15.        public string MultiLineText { get; set; }
16.
17.        [Display(Name = "Number")]
18.        public int Number { get; set; }
19.
20.        [Display(Name = "Decimal")]
21.        [UIHint("Decimal")]
22.        public double Decimal { get; set; }
23.
24.        [Display(Name = "Tel")]
25.        [DataType(DataType.PhoneNumber)]
26.        public string Tel { get; set; }
27.
28.        [Display(Name = "Date")]
29.        [DataType(DataType.Date)]
30.        public DateTime Date { get; set; }
31.
32.        [Display(Name = "Time")]
33.        [DataType(DataType.Time)]
34.        public DateTime Time { get; set; }
35.
36.        [Display(Name = "HiddenInput")]
37.        [UIHint("HiddenInput")]
38.        public string HiddenInput { get; set; }
39.
40.        [Display(Name = "Boolean")]
41.        [UIHint("Boolean")]
42.        public bool Boolean { get; set; }
43.
44.        [Display(Name = "Email")]
45.        [DataType(DataType.EmailAddress)]
46.        public string Email{ get; set; }
47.
48.        [Display(Name = "Url")]
49.        [DataType(DataType.Url)]
50.        public string Url { get; set; }
51.
52.        [Display(Name = "Password")]
53.        [DataType(DataType.Password)]
54.        public string Password { get; set; }
55.
56.        [Display(Name = "Currency")]
57.        [DataType(DataType.Currency)]
58.        public double Currency { get; set; }
59.
60.        [Display(Name = "CreditCard")]
61.        [DataType(DataType.CreditCard)]
62.        public string CreditCard { get; set; }
63.
64.        // constructeur
65.        public ViewModel10()
66.        {
67.            Text = "tra la la";
68.            MultiLineText = "ligne1\nligne2";
69.            Number = 4;
```

```

70.     Decimal = 10.2;
71.     Tel = "0617181920";
72.     Date = DateTime.Now;
73.     Time = DateTime.Now;
74.     HiddenInput = "caché";
75.     Boolean = true;
76.     Email = "x@y.z";
77.     Url = "http://istia.univ-angers.fr";
78.     Password = "mdp";
79.     Currency = 4.2;
80.     CreditCard = "0123456789012345";
81.   }
82. }
83. }
```

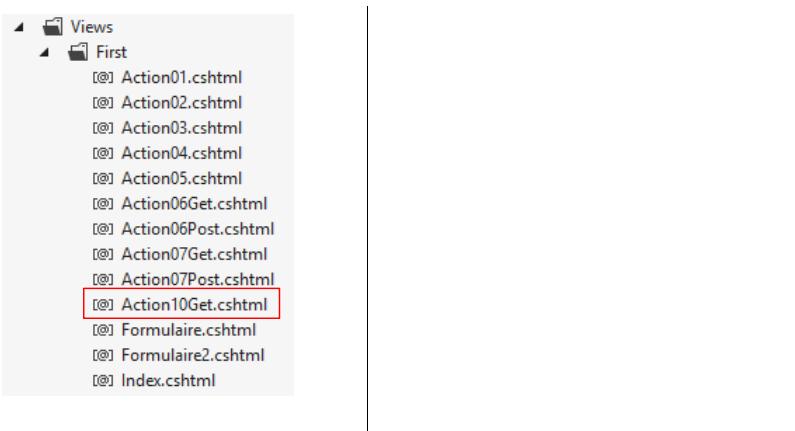
Les métadonnées sont formées des balises [Display, DataType, UIHint].

Ce modèle de vue sera construit par l'action [Action10Get] suivante :

```

1.  // Action10-GET
2.  [HttpGet]
3.  public ViewResult Action10Get()
4.  {
5.    return View(new ViewModel10());
6. }
```

Ci-dessus ligne 5, on demande à la vue par défaut de l'action [/First/Action10Get.cshtml] d'afficher le modèle de vue de type [ViewModel10]. Cette vue est la suivante :



```

1. @model Exemple_03.Models.ViewModel10
2.
3. @{
4.   Layout = null;
5. }
6.
7. <!DOCTYPE html>
8.
9. <html>
10. <head>
11.   <meta name="viewport" content="width=device-width" />
12.   <title>Action10Get</title>
13. </head>
14. <body>
15.   <h3>Formulaire ASP.NET MVC - 2</h3>
16.   @using (Html.BeginForm("Action10Post", "First"))
17.   {
18.     <table>
19.       <thead>
20.         <tr>
21.           <th>LabelFor</th>
22.           <th>EditorFor</th>
23.           <th>DisplayFor</th>
24.         </tr>
25.       </thead>
26.       <tbody>
27.         <tr>
28.           <td>@Html.LabelFor(m => m.Text)</td>
29.           <td>@Html.EditorFor(m => m.Text)</td>
30.           <td>@Html.DisplayFor(m => m.Text)</td>
31.         </tr>
32.         <tr>
```

```

33.      <td>@Html.LabelFor(m => m.MultiLineText)</td>
34.      <td>@Html.EditorFor(m => m.MultiLineText)</td>
35.      <td>@Html.DisplayFor(m => m.MultiLineText)</td>
36.    </tr>
37.    <tr>
38.      <td>@Html.LabelFor(m => m.Number)</td>
39.      <td>@Html.EditorFor(m => m.Number)</td>
40.      <td>@Html.DisplayFor(m => m.Number)</td>
41.    </tr>
42.    <tr>
43.      <td>@Html.LabelFor(m => m.Decimal)</td>
44.      <td>@Html.EditorFor(m => m.Decimal)</td>
45.      <td>@Html.DisplayFor(m => m.Decimal)</td>
46.    </tr>
47.    <tr>
48.      <td>@Html.LabelFor(m => m.Tel)</td>
49.      <td>@Html.EditorFor(m => m.Tel)</td>
50.      <td>@Html.DisplayFor(m => m.Tel)</td>
51.    </tr>
52.    <tr>
53.      <td>@Html.LabelFor(m => m.Date)</td>
54.      <td>@Html.EditorFor(m => m.Date)</td>
55.      <td>@Html.DisplayFor(m => m.Date)</td>
56.    </tr>
57.    <tr>
58.      <td>@Html.LabelFor(m => m.Time)</td>
59.      <td>@Html.EditorFor(m => m.Time)</td>
60.      <td>@Html.DisplayFor(m => m.Time)</td>
61.    </tr>
62.    <tr>
63.      <td>@Html.LabelFor(m => m.HiddenInput)</td>
64.      <td>@Html.EditorFor(m => m.HiddenInput)</td>
65.      <td>@Html.DisplayFor(m => m.HiddenInput)</td>
66.    </tr>
67.    <tr>
68.      <td>@Html.LabelFor(m => m.Boolean)</td>
69.      <td>@Html.EditorFor(m => m.Boolean)</td>
70.      <td>@Html.DisplayFor(m => m.Boolean)</td>
71.    </tr>
72.    <tr>
73.      <td>@Html.LabelFor(m => m.Email)</td>
74.      <td>@Html.EditorFor(m => m.Email)</td>
75.      <td>@Html.DisplayFor(m => m.Email)</td>
76.    </tr>
77.    <tr>
78.      <td>@Html.LabelFor(m => m.Url)</td>
79.      <td>@Html.EditorFor(m => m.Url)</td>
80.      <td>@Html.DisplayFor(m => m.Url)</td>
81.    </tr>
82.    <tr>
83.      <td>@Html.LabelFor(m => m.Password)</td>
84.      <td>@Html.EditorFor(m => m.Password)</td>
85.      <td>@Html.DisplayFor(m => m.Password)</td>
86.    </tr>
87.    <tr>
88.      <td>@Html.LabelFor(m => m.Currency)</td>
89.      <td>@Html.EditorFor(m => m.Currency)</td>
90.      <td>@Html.DisplayFor(m => m.Currency)</td>
91.    </tr>
92.    <tr>
93.      <td>@Html.LabelFor(m => m.CreditCard)</td>
94.      <td>@Html.EditorFor(m => m.CreditCard)</td>
95.      <td>@Html.DisplayFor(m => m.CreditCard)</td>
96.    </tr>
97.  </tbody>
98. </table>
99. <input type="submit" value="Valider" />
100. }
101. </body>
102. </html>
```

Pour chacune des propriétés du modèle, nous utilisons la méthode :

- **Html.LabelFor** pour afficher la valeur de la métadonnée [DisplayName] de la propriété ;
- **Html.EditorFor** pour générer la balise HTML de saisie de la valeur de la propriété. Cette méthode va utiliser les métadonnées [DataType] et [UIHint] de la propriété ;
- **Html.DisplayFor** pour afficher la valeur de la propriété selon le format indiqué par la métadonnée [DataType].

Voici un exemple d'exécution avec le navigateur Chrome :

Selon le navigateur utilisé, on peut avoir des pages différentes. En effet, la vue générée utilise les nouvelles balises amenées par la version 5 de HTML appelée HTML5. Tous les navigateurs ne supportent pas encore cette version. Ci-dessus, le navigateur Chrome la supporte en partie.

### 1.5.8.1 Le [POST] du formulaire

Le [POST] du formulaire est traité par l'action [Action10Post] suivante :

```

1.      // Action10-POST
2.      [HttpPost]
3.      public ContentResult Action10Post(ViewModel10 modèle)
4.      {
5.          string erreurs = getErrorMessagesFor(ModelState);
6.          string texte = string.Format("Contrôleur={0}, Action={1}, valide={2}, erreurs={3}",
7.              RouteData.Values["controller"], RouteData.Values["action"], ModelState.IsValid, erreurs);
8.          return Content(texte, "text/plain", Encoding.UTF8);
}

```

- ligne 3 : l'action [Action10Post] a pour modèle en entrée le formulaire posté ;
- ligne 5 : on récupère les erreurs de validation de ce formulaire ;
- ligne 6 : on prépare la réponse texte au client ;
- ligne 7 : on l'envoie.

Examinons maintenant les propriétés du modèle [ViewModel10] une par une et voyons comment les métadonnées associées influencent le HTML généré et la validation des champs de saisie.

### 1.5.8.2 Propriété [Text]

#### Définition

```

1.      [Display(Name="Text")]
2.      [DataType(DataType.Text)]
3.      public string Text { get; set; }
4.      ...
5.      Text = "tra la la";
}

```

## Vue

```

1.      <tr>
2.          <td>@Html.LabelFor(m => m.Text)</td>
3.          <td>@Html.EditorFor(m => m.Text)</td>
4.          <td>@Html.DisplayFor(m => m.Text)</td>
5.      </tr>

```

## Visuel

LabelFor	EditorFor	DisplayFor
Text	tra la la	tra la la

## HTML généré

```

1.      <tr>
2.          <td><label for="Text">Text</label></td>
3.          <td><input class="text-box single-line" id="Text" name="Text" type="text" value="tra la la" /></td>
4.          <td>tra la la</td>
5.      </tr>

```

## Commentaires

- la méthode [Html.LabelFor] a généré la balise <label> de la ligne 2. La valeur de l'attribut [for] est le nom de la propriété paramètre de la méthode [Html.LabelFor]

```
public string Text { get; set; }
```

Le texte affiché entre le début et la fin de la balise est le texte de la métadonnée

```
[Display(Name="Text")]
```

La méthode [Html.LabelFor] procède toujours ainsi. Nous ne reviendrons pas dessus pour les autres propriétés.

- la méthode [Html.EditorFor] a généré la balise <input> de la ligne 3. On notera qu'elle a un attribut [class] qui associe la classe CSS `text-box single-line` à la balise. Les attributs [id] et [name] ont pour valeur le nom [Text] de la propriété paramètre de la méthode [Html.EditorFor]. L'attribut [type] a eu la valeur [text] à cause de la métadonnée

```
[DataType(DataType.Text)]
```

- la méthode [Html.DisplayFor] a généré le texte de la ligne 4. C'est la valeur de la propriété paramètre de la méthode [Html.DisplayFor]. Cette méthode est influencée par la métadonnée

```
[DataType(DataType.Text)]
```

qui fait que la valeur est affichée comme un texte non formaté.

### 1.5.8.3 Propriété [MultiLineText]

## Définition

```

1.      [Display(Name = "TextArea")]
2.      [DataType(DataType.MultilineText)]
3.      public string MultiLineText { get; set; }

```

## Vue

```

1.      <tr>
2.          <td>@Html.LabelFor(m => m.MultiLineText)</td>
3.          <td>@Html.EditorFor(m => m.MultiLineText)</td>
4.          <td>@Html.DisplayFor(m => m.MultiLineText)</td>
5.      </tr>

```

## Visuel

TextArea    

ligne1	
ligne2	

 ligne1 ligne2

## HTML généré

```
1.      <tr>
2.        <td><label for="MultiLineText">TextArea</label></td>
3.        <td><textarea class="text-box multi-line" id="MultiLineText" name="MultiLineText">
4.          ligne1
5.          ligne2</textarea></td>
6.        <td>ligne1
7.        ligne2</td>
8.      </tr>
```

## Commentaires

- la méthode [Html.EditorFor] a généré la balise <textarea> de la ligne 3. On notera qu'elle a un attribut [class] qui associe la classe CSS [text-box multi-line] à la balise. Les attributs [id] et [name] ont pour valeur le nom [**MultiLineText**] de la propriété paramètre de la méthode [Html.EditorFor]. C'est toujours ainsi. Nous ne le mentionnerons plus. La balise générée est <textarea> à cause de la métadonnée

```
[DataType(DataType.MultilineText)]
```

- qui précisait que la propriété était un texte multi-ligne.
- la méthode [Html.DisplayFor] a généré le texte des lignes 4-5. C'est la valeur de la propriété paramètre de la méthode [Html.DisplayFor].

### 1.5.8.4 Propriété [Number]

## Définition

```
1.      [Display(Name = "Number")]
2.      public int Number { get; set; }
```

## Vue

```
1.      <tr>
2.        <td>@Html.LabelFor(m => m.Number)</td>
3.        <td>@Html.EditorFor(m => m.Number)</td>
4.        <td>@Html.DisplayFor(m => m.Number)</td>
5.      </tr>
```

## Visuel

Number    

4	▲▼	4
---	----	---

## HTML généré

```
1.      <tr>
2.        <td><label for="Number">Number</label></td>
3.        <td><input class="text-box single-line" data-val="true" data-val-number="Le champ Number doit être un
   nombre." data-val-required="Le champ Number est requis." id="Number" name="Number" type="number" value="4" /></td>
4.      </tr>
```

## Commentaires

- la méthode [Html.EditorFor] a généré la balise <input> de la ligne 3 avec un attribut [type] de type [number]. Apparemment simplement parce que la propriété a le type [int]. Les attributs [data-val], [data-val-number] et [data-val-required] sont des attributs non reconnus par HTML5. Ils sont utilisés par un framework Javascript de validation des données côté client ;

- la méthode [Html.DisplayFor] a généré le texte de la ligne 4, la valeur de la propriété.

## Validation

Les attributs [data-x] influencent la validation des données côté client. Voici deux exemples :

On saisit un nombre erroné et on valide :

Number	4x	4
Decimal	10,20	Veuillez saisir un nombre.
Tel	0617181920	0617181920

Ci-dessus, la validation a eu lieu côté client. Le formulaire ne sera pas posté tant que l'erreur n'aura pas été corrigée.

Autre exemple, on ne saisit rien :

Number	4
--------	---

En [1] ci-dessus, [Action10Post] signale une erreur. On se souvient peut-être qu'on avait déjà obtenu ce comportement en utilisant l'attribut [Required] sur la propriété à contrôler (cf page 75), ici la propriété [Number]. Ici, on n'a pas eu à le faire.

### 1.5.8.5 Propriété [Decimal]

#### Définition

```
1.     [Display(Name = "Decimal")]
2.     [UIHint("Decimal")]
3.     public double Decimal { get; set; }
```

#### Vue

```
1.     <tr>
2.         <td>@Html.LabelFor(m => m.Decimal)</td>
3.         <td>@Html.EditorFor(m => m.Decimal)</td>
4.         <td>@Html.DisplayFor(m => m.Decimal)</td>
5.     </tr>
```

#### Visuel

Decimal	10,20	10,20
---------	-------	-------

#### HTML généré

```
1.     <tr>
2.         <td><label for="Decimal">Decimal</label></td>
3.         <td><input class="text-box single-line" data-val="true" data-val-number="Le champ Decimal doit être un
        nombre." data-val-required="Le champ Decimal est requis." id="Decimal" name="Decimal" type="text" value="10,20"
        /></td>
4.         <td>10,20</td>
5.     </tr>
```

## Commentaires

- la méthode [Html.EditorFor] a généré la balise <input> de la ligne 3 avec un attribut [type] de type [text]. Les autres attributs sont identiques à ceux générés pour la propriété [Number] précédente. La métadonnée :

```
[UIHint("Decimal")]
```

fait que la valeur de la propriété est affichée avec deux décimales pour les deux méthodes [Html.EditorFor] et [Html.DisplayFor]

## Validation

Aucune erreur de validation n'est signalée côté client, contrairement au cas précédent. L'erreur n'est signalée que par l'action [Action10Post]. Là encore, le nombre décimal est requis sans qu'on ait besoin de lui mettre l'attribut [Required].

### 1.5.8.6 Propriété [Tel]

#### Définition

```
1.      [Display(Name = "Tel")]
2.      [DataType(DataType.PhoneNumber)]
3.  public string Tel { get; set; }
```

#### Vue

```
1.      <tr>
2.          <td>@Html.LabelFor(m => m.Tel)</td>
3.          <td>@Html.EditorFor(m => m.Tel)</td>
4.          <td>@Html.DisplayFor(m => m.Tel)</td>
5.      </tr>
```

#### Visuel



## HTML généré

```
1.      <tr>
2.          <td><label for="Tel">Tel</label></td>
3.          <td><input class="text-box single-line" id="Tel" name="Tel" type="tel" value="0617181920" /></td>
4.          <td>0617181920</td>
5.      </tr>
```

## Commentaires

- la méthode [Html.EditorFor] a généré la balise <input> de la ligne 3 avec un attribut [type] de type [tel]. Cette valeur a été générée à cause de la métadonnée :

```
[DataType(DataType.PhoneNumber)]
```

Le type [tel] pour une balise <input> est une nouveauté de HTML5. Le navigateur Chrome l'a traitée comme une balise <input> avec le type [text].

## Validation

Aucune erreur de validation n'est signalée côté client ou côté serveur. On peut saisir n'importe quoi.

### 1.5.8.7 Propriété [Date]

#### Définition

```
1.      [Display(Name = "Date")]
2.      [DataType(DataType.Date)]
3.  public DateTime Date { get; set; }
```

## Vue

```

1.      <tr>
2.          <td>@Html.LabelFor(m => m.Date)</td>
3.          <td>@Html.EditorFor(m => m.Date)</td>
4.          <td>@Html.DisplayFor(m => m.Date)</td>
5.      </tr>

```

## Visuel

Date: jj/mm/aaaa  
11/10/2013

Time: octobre 2013

HiddenInput: [HiddenInput]

Boolean: Boolean

Email: Email

Url: Url

Password: Password

Currency: 4.2 4,20 €

## HTML généré

```

1.      <tr>
2.          <td><label for="Date">Date</label></td>
3.          <td><input class="text-box single-line" data-val="true" data-val-date="Le champ Date doit être une date."
   data-val-required="Le champ Date est requis." id="Date" name="Date" type="date" value="11/10/2013" /></td>
4.      <td>11/10/2013</td>
5.  </tr>

```

## Commentaires

- la méthode [Html.EditorFor] a généré la balise <input> de la ligne 3 avec un attribut [type] de type [date]. Cette valeur a été générée à cause de la métadonnée :

[`DataType(DataType.Date)`]

Le type [date] pour une balise <input> est une nouveauté HTML5. Le navigateur Chrome la reconnaît et permet de saisir la date avec un calendrier. Par ailleurs, la date saisie est présentée au format [jj/mm/aaaa], ç-à-d que Chrome adapte le format de date à la [locale] du navigateur.

- la méthode [Html.DisplayFor] a écrit elle aussi la date sous la forme [jj/mm/aaaa] toujours à cause de la présence de la métadonnée [Date].

## Validation

Une date invalide est signalée côté client [1] empêchant le POST du formulaire au serveur.

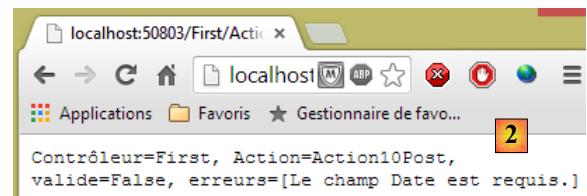
Date: 30/02/2013

Time: 13:39

HiddenInput caché:

Boolean:

**1**



L'absence de date n'est pas signalée côté client mais elle l'est côté serveur [2].

### 1.5.8.8 Propriété [Time]

#### Définition

```
1.      [Display(Name = "Time")]
```

```

2.      [DataType(DataType.Time)]
3.  public DateTime Time { get; set; }

```

## Vue

```

1.      <tr>
2.          <td>@Html.LabelFor(m => m.Time)</td>
3.          <td>@Html.EditorFor(m => m.Time)</td>
4.          <td>@Html.DisplayFor(m => m.Time)</td>
5.      </tr>

```

## Visuel

Time     

11:18

## HTML généré

```

1.      <tr>
2.          <td><label for="Time">Time</label></td>
3.          <td><input class="text-box single-line" data-val="true" data-val-required="Le champ Time est requis." id="Time" name="Time" type="time" value="11:17" /></td>
4.          <td>11:17</td>
5.      </tr>

```

## Commentaires

- la méthode [Html.EditorFor] a généré la balise <input> de la ligne 3 avec un attribut [type] de type [time]. Cette valeur a été générée à cause de la métadonnée :

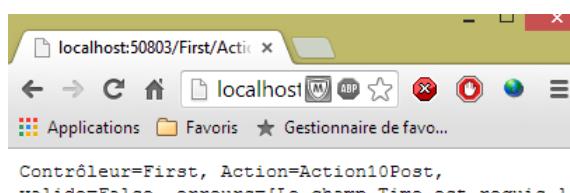
```
[DataType(DataType.Time)]
```

Le type [time] pour une balise <input> est une nouveauté HTML5. Le navigateur Chrome la reconnaît et permet de saisir une heure sous la forme [hh:mm] ;

- la méthode [Html.DisplayFor] a écrit elle aussi l'heure sous la forme [hh:mm] toujours à cause de la présence de la métadonnée [Time].

## Validation

Il n'est techniquement pas possible de saisir une heure invalide. L'absence d'heure est signalée côté serveur :



### 1.5.8.9 Propriété [HiddenInput]

## Définition

```

1.      [Display(Name = "HiddenInput")]
2.      [UIHint("HiddenInput")]
3.  public string HiddenInput { get; set; }

```

## Vue

```

1.      <tr>
2.          <td>@Html.LabelFor(m => m.HiddenInput)</td>
3.          <td>@Html.EditorFor(m => m.HiddenInput)</td>
4.          <td>@Html.DisplayFor(m => m.HiddenInput)</td>

```

5. </tr>

## Visuel

HiddenInput caché      caché

## HTML généré

```
1.      <tr>
2.        <td><label for="HiddenInput">HiddenInput</label></td>
3.        <td>caché;<input id="HiddenInput" name="HiddenInput" type="hidden" value="caché" /></td>
4.        <td>caché;</td>
5.      </tr>
```

## Commentaires

- la méthode [Html.EditorFor] a générée la balise <input> de la ligne 3 avec un attribut [type] de type [**hidden**], c-à-d un champ caché (mais néanmoins posté). Cette valeur a été générée à cause de la métadonnée :

```
[UIHint("HiddenInput")]
```

- la méthode [Html.DisplayFor] a écrit elle la valeur du champ caché.

### 1.5.8.10 Propriété [Boolean]

## Définition

```
1.      [Display(Name = "Boolean")]
2.  public bool Boolean { get; set; }
```

## Vue

```
1.      <tr>
2.        <td>@Html.LabelFor(m => m.Boolean)</td>
3.        <td>@Html.EditorFor(m => m.Boolean)</td>
4.        <td>@Html.DisplayFor(m => m.Boolean)</td>
5.      </tr>
```

## Visuel

Boolean     

## HTML généré

```
1.      <tr>
2.        <td><label for="Boolean">Boolean</label></td>
3.        <td><input checked="checked" class="check-box" data-val="true" data-val-required="Le champ Boolean est
       requis." id="Boolean" name="Boolean" type="checkbox" value="true" /><input name="Boolean" type="hidden"
       value="false" /></td>
4.        <td><input checked="checked" class="check-box" disabled="disabled" type="checkbox" /></td>
5.      </tr>
```

## Commentaires

- la méthode [Html.EditorFor] a générée la balise <input> de la ligne 3 avec un attribut [type] de type [**checkbox**], c-à-d une case à cocher. Cette valeur a été générée parce que la propriété est booléenne :

```
public bool Boolean { get; set; }
```

- la méthode [Html.DisplayFor] a générée la ligne 4, également une case à cocher (attribut *type*) mais désactivée (attribut *disabled*).

### 1.5.8.11 Propriété [Email]

#### Définition

```
1.     [Display(Name = "Email")]
2.     [DataType(DataType.EmailAddress)]
3.     public string Email{ get; set; }
```

#### Vue

```
1.     <tr>
2.         <td>@Html.LabelFor(m => m.Email)</td>
3.         <td>@Html.EditorFor(m => m.Email)</td>
4.         <td>@Html.DisplayFor(m => m.Email)</td>
5.     </tr>
```

#### Visuel

Email	<input type="text" value="x@y.z"/>	x@y.z
-------	------------------------------------	-------

#### HTML généré

```
1.     <tr>
2.         <td><label for="Email">Email</label></td>
3.         <td><input class="text-box single-line" id="Email" name="Email" type="email" value="x@y.z" /></td>
4.         <td><a href="mailto:x@y.z">x@y.z</a></td>
5.     </tr>
```

#### Commentaires

- la méthode [Html.EditorFor] a générée la balise <input> de la ligne 3 avec un attribut [type] de type [email]. Ce type est nouveau dans HTML5. Ce type a été générée à cause de la métadonnée :

```
[DataType(DataType.EmailAddress)]
```

Chrome semble avoir traité ce type comme un type [text].

- la méthode [Html.DisplayFor] a générée la ligne 4 : un lien vers l'adresse mail.

#### Validation

Une adresse invalide est signalée côté client [1] :

Email	<input type="text" value="xx"/>	x@y.z
Url	<input type="text" value="http://istia"/>	Veuillez saisir une adresse e-mail.
Password	<input type="password" value="..."/>	<input type="password" value="mdp"/> 1

L'absence de saisie ne provoque aucune erreur.

### 1.5.8.12 Propriété [Url]

#### Définition

```
1.     [Display(Name = "Url")]
2.     [DataType(DataType.Url)]
3.     public string Url { get; set; }
```

#### Vue

```
1.     <tr>
2.         <td>@Html.LabelFor(m => m.Url)</td>
3.         <td>@Html.EditorFor(m => m.Url)</td>
4.         <td>@Html.DisplayFor(m => m.Url)</td>
```

Visuel

Url  <http://istia.univ-angers.fr>

HTML généré

```

1.      <tr>
2.        <td><label for="Url">Url</label></td>
3.        <td><input class="text-box single-line" id="Url" name="Url" type="url" value="http://istia.univ-angers.fr" /></td>
4.        <td><a href="http://istia.univ-angers.fr">http://istia.univ-angers.fr</a></td>
5.      </tr>

```

Commentaires

- la méthode [Html.EditorFor] a généré la balise <input> de la ligne 3 avec un attribut [type] de type [url]. Ce type est nouveau dans HTML5. Il a été généré à cause de la métadonnée :

`[DataType(DataType.Url)]`

Chrome semble traiter ce type comme un type [text].

- la méthode [Html.DisplayFor] a généré la ligne 4 : un lien vers l'URL.

Validation

Une URL invalide est signalée côté client [1] :

Email	<input type="text" value="x@y.z"/>	x@y.z
Url	<input type="text" value="x"/>	<a href="http://istia.u">http://istia.u</a>
Password	<input type="password" value="..."/>	Veuillez saisir une URL.
Currency	€	1 00 €

L'absence de saisie ne provoque aucune erreur.

1.5.8.13 Propriété [Password]Définition

```

1.      [Display(Name = "Password")]
2.      [DataType(DataType.Password)]
3.      public string Password { get; set; }

```

Vue

```

1.      <tr>
2.        <td>@Html.LabelFor(m => m.Password)</td>
3.        <td>@Html.EditorFor(m => m.Password)</td>
4.        <td>@Html.DisplayFor(m => m.Password)</td>
5.      </tr>

```

Visuel

Password  mdp

HTML généré

```

1.      <tr>
2.          <td><label for="Password">Password</label></td>
3.          <td><input class="text-box single-line password" id="Password" name="Password" type="password" value="mdp" /></td>
4.          <td>mdp</td>
5.      </tr>

```

## Commentaires

- la méthode [Html.EditorFor] a généré la balise <input> de la ligne 3 avec un attribut [type] de type [**password**]. Ce type a été généré à cause de la métadonnée :

[**DataType(DataType.Password)**]

- la méthode [Html.DisplayFor] a généré la ligne 4.

### 1.5.8.14 Propriété [Currency]

#### Définition

```

1.      [Display(Name = "Currency")]
2.      [DataType(DataType.Currency)]
3.      public double Currency { get; set; }

```

#### Vue

```

1.      <tr>
2.          <td>@Html.LabelFor(m => m.Currency)</td>
3.          <td>@Html.EditorFor(m => m.Currency)</td>
4.          <td>@Html.DisplayFor(m => m.Currency)</td>
5.      </tr>

```

#### Visuel

Currency	4,2	4,20 €
----------	-----	--------

#### HTML généré

```

1.      <tr>
2.          <td><label for="Currency">Currency</label></td>
3.          <td><input class="text-box single-line" data-val="true" data-val-number="Le champ Currency doit être un nombre." data-val-required="Le champ Currency est requis." id="Currency" name="Currency" type="text" value="4,2" /></td>
4.          <td>4,20 €</td>
5.      </tr>

```

#### Commentaires

- la méthode [Html.EditorFor] a généré la balise <input> de la ligne 3 avec un attribut [type] de type [**text**] ;
- la méthode [Html.DisplayFor] a généré la ligne 4, un nombre à deux décimales avec un symbole monétaire. Ce format a été utilisé à cause de la métadonnée :

[**DataType(DataType.Currency)**]

#### Validation

Une valeur invalide [1] ou une absence de valeur [2] est signalée côté serveur :



### 1.5.8.15 Propriété [CreditCard]

#### Définition

```
1.     [Display(Name = "CreditCard")]
2.     [DataType(DataType.CreditCard)]
3.     public string CreditCard { get; set; }
```

#### Vue

```
1.     <tr>
2.         <td>@Html.LabelFor(m => m.CreditCard)</td>
3.         <td>@Html.EditorFor(m => m.CreditCard)</td>
4.         <td>@Html.DisplayFor(m => m.CreditCard)</td>
5.     </tr>
```

#### Visuel

CreditCard  0123456789012345

#### HTML généré

```
1.     <tr>
2.         <td><label for="CreditCard">CreditCard</label></td>
3.         <td><input class="text-box single-line" id="CreditCard" name="CreditCard" type="text"
   value="0123456789012345" /></td>
4.         <td>0123456789012345</td>
5.     </tr>
```

#### Commentaires

- la méthode [Html.EditorFor] a généré la balise <input> de la ligne 3 avec un attribut [type] de type [text]. La méthode [Html.DisplayFor] a généré la ligne 4. On ne voit pas ici ce qu'apporte la métadonnée :

```
[DataType(DataType.CreditCard)]
```

#### Validation

Aucune vérification n'est faite, ni côté client, ni côté serveur.

### 1.5.9 Validation d'un formulaire

Nous avons déjà rencontré le problème de la validation du modèle d'une action au paragraphe 1.4.5, page 74, et dans les paragraphes suivants. Nous revenons sur cette problématique dans le cadre d'un formulaire :

- comment signaler les erreurs de saisie à l'utilisateur ;
- faire les validations aussi bien côté client que côté serveur afin de signaler plus rapidement les erreurs à l'utilisateur.

#### 1.5.9.1 Validation côté serveur

Considérons le modèle suivant :

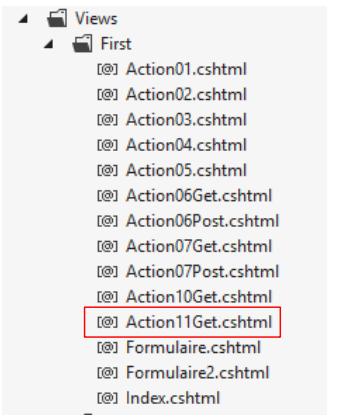
```
1. using System;
2. using System.Collections.Generic;
3. using System.ComponentModel.DataAnnotations;
4. using System.Net.Mail;
```

```

5.   namespace Exemple_03.Models
6.   {
7.     public class ViewModel11 : IValidatableObject
8.     {
9.       [
10.         Required(ErrorMessage = "Information requise")]
11.         [Display(Name = "Chaine d'au moins quatre caractères")]
12.         [RegularExpression(@"^.{4,}", ErrorMessage = "Information incorrecte")]
13.         public string Chaine1 { get; set; }
14.
15.         [Display(Name = "Chaine d'au plus quatre caractères")]
16.         [Required(ErrorMessage = "Information requise")]
17.         [RegularExpression(@"^.{1,4}$", ErrorMessage = "Information incorrecte")]
18.         public string Chaine2 { get; set; }
19.
20.         [Required(ErrorMessage = "Information requise")]
21.         [Display(Name = "Chaine de quatre caractères exactement")]
22.         [RegularExpression(@"^.{4,4}$", ErrorMessage = "Information incorrecte")]
23.         public string Chaine3 { get; set; }
24.
25.         [Required(ErrorMessage = "Information requise")]
26.         [Display(Name = "Nombre entier")]
27.         public int Entier1 { get; set; }
28.
29.         [Display(Name = "Nombre entier dans l'intervalle [1,100]")]
30.         [Required(ErrorMessage = "Information requise")]
31.         [Range(1, 100, ErrorMessage = "Information incorrecte")]
32.         public int Entier2 { get; set; }
33.
34.         [Display(Name = "Nombre réel")]
35.         [Required(ErrorMessage = "Information requise")]
36.         public double Reel1 { get; set; }
37.
38.         [Display(Name = "Nombre réel dans l'intervalle [10.2, 11.3]")]
39.         [Required(ErrorMessage = "Information requise")]
40.         [Range(10.2, 11.3, ErrorMessage = "Information incorrecte")]
41.         public double Reel2 { get; set; }
42.
43.         [Display(Name = "Adresse mail")]
44.         [Required(ErrorMessage = "Information requise")]
45.         public string Email1 { get; set; }
46.
47.         [Display(Name = "Date sous la forme dd/jj/aaaa")]
48.         [RegularExpression(@"^\d{2}/\d{2}/\d{4}\s*", ErrorMessage = "Information incorrecte")]
49.         [Required(ErrorMessage = "Information requise")]
50.         public string Regexp1 { get; set; }
51.
52.         [Display(Name = "Date postérieure à celle d'aujourd'hui")]
53.         [Required(ErrorMessage = "Information requise")]
54.         [DataType(DataType.Date)]
55.         public DateTime Date1 { get; set; }
56.
57.         // validation
58.         public IEnumerable<ValidationResult> Validate(ValidationContext validationContext)
59.         {
60.           List<ValidationResult> résultats = new List<ValidationResult>();
61.           // Date 1
62.           if (Date1.Date <= DateTime.Now.Date)
63.           {
64.             résultats.Add(new ValidationResult("Information incorrecte", new string[] { "Date1" }));
65.           }
66.           // Email1
67.           try
68.           {
69.             new MailAddress(Email1);
70.           }
71.           catch
72.           {
73.             résultats.Add(new ValidationResult("Information incorrecte", new string[] { "Email1" }));
74.           }
75.           // on rend la liste des erreurs
76.           return résultats;
77.         }
78.       }
79.     }
80.   }

```

Ce modèle sera affiché par la vue suivante [Action11Get.cshtml] :



```

1. @model Exemple_03.Models.ViewModel11
2. @{
3.     Layout = null;
4. }
5.
6. <!DOCTYPE html>
7.
8. <html>
9. <head>
10.    <meta name="viewport" content="width=device-width" />
11.    <title>Action11Get</title>
12.    <link rel="stylesheet" href="~/Content/Site.css" />
13. </head>
14. <body>
15.    <h3>Formulaire ASP.NET MVC - Validation 1</h3>
16.    @using (Html.BeginForm("Action11Post", "First"))
17.    {
18.        <table>
19.            <thead>
20.                <tr>
21.                    <th>Type attendu</th>
22.                    <th>Valeur saisie</th>
23.                    <th>Message d'erreur</th>
24.                </tr>
25.            </thead>
26.            <tbody>
27.                <tr>
28.                    <td>@Html.LabelFor(m => m.Chaine1)</td>
29.                    <td>@Html.EditorFor(m => m.Chaine1)</td>
30.                    <td>@Html.ValidationMessageFor(m => m.Chaine1)</td>
31.                </tr>
32.                <tr>
33.                    <td>@Html.LabelFor(m => m.Chaine2)</td>
34.                    <td>@Html.EditorFor(m => m.Chaine2)</td>
35.                    <td>@Html.ValidationMessageFor(m => m.Chaine2)</td>
36.                </tr>
37.                <tr>
38.                    <td>@Html.LabelFor(m => m.Chaine3)</td>
39.                    <td>@Html.EditorFor(m => m.Chaine3)</td>
40.                    <td>@Html.ValidationMessageFor(m => m.Chaine3)</td>
41.                </tr>
42.                <tr>
43.                    <td>@Html.LabelFor(m => m.Entier1)</td>
44.                    <td>@Html.EditorFor(m => m.Entier1)</td>
45.                    <td>@Html.ValidationMessageFor(m => m.Entier1)</td>
46.                </tr>
47.                <tr>
48.                    <td>@Html.LabelFor(m => m.Entier2)</td>
49.                    <td>@Html.EditorFor(m => m.Entier2)</td>
50.                    <td>@Html.ValidationMessageFor(m => m.Entier2)</td>
51.                </tr>
52.                <tr>
53.                    <td>@Html.LabelFor(m => m.Reel1)</td>
54.                    <td>@Html.EditorFor(m => m.Reel1)</td>
55.                    <td>@Html.ValidationMessageFor(m => m.Reel1)</td>
56.                </tr>
57.                <tr>
58.                    <td>@Html.LabelFor(m => m.Reel2)</td>
59.                    <td>@Html.EditorFor(m => m.Reel2)</td>
60.                    <td>@Html.ValidationMessageFor(m => m.Reel2)</td>
61.                </tr>
62.                <tr>
63.                    <td>@Html.LabelFor(m => m.Email1)</td>
64.                    <td>@Html.EditorFor(m => m.Email1)</td>

```

```

55.      <td>@Html.ValidationMessageFor(m => m.Email1)</td>
56.    </tr>
57.    <tr>
58.      <td>@Html.LabelFor(m => m.Regexp1)</td>
59.      <td>@Html.EditorFor(m => m.Regexp1)</td>
60.      <td>@Html.ValidationMessageFor(m => m.Regexp1)</td>
61.    </tr>
62.    <tr>
63.      <td>@Html.LabelFor(m => m.Date1)</td>
64.      <td>@Html.EditorFor(m => m.Date1)</td>
65.      <td>@Html.ValidationMessageFor(m => m.Date1)</td>
66.    </tr>
67.  </tbody>
68. </table>
69. <p>
70.   <input type="submit" value="Valider" />
71. </p>
72. }
73. </body>
74. </html>

```

- ligne 12 : on référence la feuille de style [Site.css]. Elle contient par défaut des classes utilisées pour mettre en relief les erreurs de saisie du formulaire ;
- lignes 18-25 : un tableau à trois colonnes :
  - la colonne 1 affiche du texte avec la méthode [Html.LabelFor],
  - la colonne 2 affiche la saisie avec la méthode [Html.EditorFor],
  - la colonne 3 affiche l'éventuelle erreur de saisie avec la méthode [Html.ValidationMessageFor] ;

L'action [Action11Get] sert à afficher le formulaire :

```

1.  // Action11-GET
2.  [HttpGet]
3.  public ViewResult Action11Get()
4.  {
5.    return View("Action11Get", new ViewModel11());
6.  }

```

L'action [Action11Post] sert à réafficher le formulaire avec les éventuelles erreurs de saisie :

```

1.  // Action11-POST
2.  [HttpPost]
3.  public ViewResult Action11Post(ViewModel11 modèle)
4.  {
5.    return View("Action11Get", modèle);
6.  }

```

- ligne 3 : le modèle [ViewModel11] est créé puis initialisé avec les valeurs postées. Il peut alors se produire des erreurs. A chaque propriété erronée P du modèle est associé un message d'erreur. C'est ce message que permet d'obtenir la méthode [Html.ValidationMessageFor] du formulaire.

Voici un exemple d'exécution :

Type attendu	Valeur saisie	Message d'erreur
Chaîne d'au moins quatre caractères		
Chaîne d'au plus quatre caractères		
Chaîne de quatre caractères exactement		
Nombre entier	0	
Nombre entier dans l'intervalle [1,100]	0	
Nombre réel	0	
Nombre réel dans l'intervalle [10.2, 11.3]	0	
Adresse mail		
Date sous la forme dd/jj/aaaa		
Date postérieure à celle d'aujourd'hui	jj/mm/aaaa	

Type attendu	Valeur saisie	Message d'erreur
Chaîne d'au moins quatre caractères		Information requise
Chaîne d'au plus quatre caractères		Information requise
Chaîne de quatre caractères exactement		Information requise
Nombre entier	0	
Nombre entier dans l'intervalle [1,100]	0	Information incorrecte
Nombre réel	0	
Nombre réel dans l'intervalle [10.2, 11.3]	0	Information incorrecte
Adresse mail		Information requise
Date sous la forme dd/jj/aaaa		Information requise
Date postérieure à celle d'aujourd'hui	jj/mm/aaaa	Information requise

**Valider**

Voici un autre exemple :

Type attendu	Valeur saisie	Message d'erreur
Chaîne d'au moins quatre caractères	aaaaaaaa	
Chaîne d'au plus quatre caractères	a	
Chaîne de quatre caractères exactement	aaaa	
Nombre entier	0	
Nombre entier dans l'intervalle [1,100]	2	
Nombre réel	0x	La valeur « 0x » n'est pas valide pour Nombre réel.
Nombre réel dans l'intervalle [10.2, 11.3]	12	Information incorrecte
Adresse mail	x@y.z	
Date sous la forme dd/jj/aaaa	40/11/2011	
Date postérieure à celle d'aujourd'hui	09/10/2013	Information incorrecte

**Valider**

On notera que les deux dates sont erronées (on est le 11/10/2013) mais que les erreurs ne sont pas signalées. Ces erreurs sont détectées par la méthode [Validate] du modèle :

```

1.    // validation
2.    public IEnumerable<ValidationResult> Validate(ValidationContext validationContext)
3.    {
4.        List<ValidationResult> résultats = new List<ValidationResult>();
5.        // Date 1
6.        if (Date1.Date <= DateTime.Now.Date)
7.        {
8.            résultats.Add(new ValidationResult("Information incorrecte", new string[] { "Date1" }));
9.        }
10.       // Email1
11.       try

```

```

12.      {
13.          new MailAddress(Email1);
14.      }
15.      catch
16.      {
17.          résultats.Add(new ValidationResult("Information incorrecte", new string[] { "Email1" }));
18.      }
19.      // Regexp1
20.      try
21.      {
22.          DateTime.ParseExact(Regexp1, "dd/MM/yyyy", CultureInfo.CreateSpecificCulture("fr-FR"));
23.      }
24.      catch
25.      {
26.          résultats.Add(new ValidationResult("Information incorrecte", new string[] { "Regexp1" }));
27.      }
28.
29.      // on rend la liste des erreurs
30.      return résultats;
31.  }

```

La méthode [Validate] n'est exécutée que lorsque toutes les validations par attributs sont passées. C'est ce que montre un dernier exemple :

The screenshot shows a browser window with the title 'Action11Get' and the URL 'localhost:50803/First/Action11Post'. The page displays a table titled 'Formulaire ASP.NET MVC - Validation 1' with three columns: 'Type attendu', 'Valeur saisie', and 'Message d'erreur'. The table contains the following data:

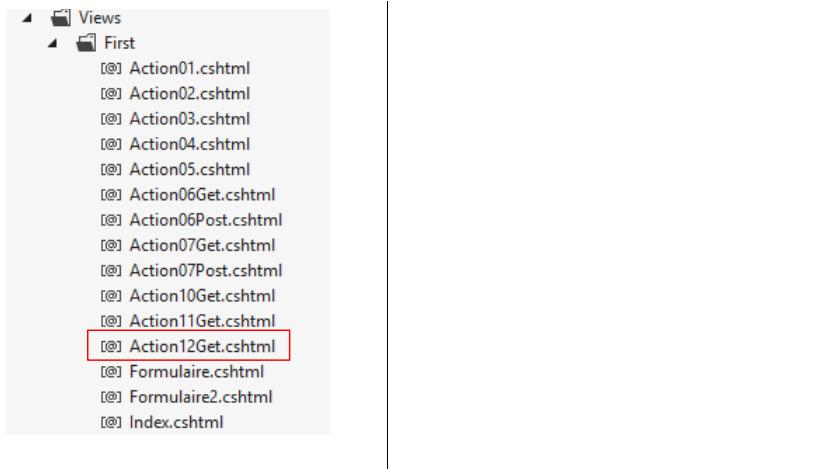
Type attendu	Valeur saisie	Message d'erreur
Chaîne d'au moins quatre caractères	aaaaaaaa	
Chaîne d'au plus quatre caractères	a	
Chaîne de quatre caractères exactement	aaaa	
Nombre entier	0	
Nombre entier dans l'intervalle [1,100]	2	
Nombre réel	10,7	
Nombre réel dans l'intervalle [10.2, 11.3]	11	
Adresse mail	x@y.z	
Date sous la forme dd/jj/aaaa	40/11/2011	Information incorrecte
Date postérieure à celle d'aujourd'hui	09/10/2013	Information incorrecte

A 'Valider' button is located at the bottom left of the form.

### 1.5.9.2 Validation côté client

Toutes les validations précédentes se sont faites côté serveur. Il faut donc un aller-retour entre le client et le serveur pour que l'utilisateur s'aperçoive de ses erreurs. La validation côté client utilise du code Javascript pour signaler à l'utilisateur ses erreurs le plus tôt possible et en tout cas avant le POST. Celui-ci ne peut avoir lieu que lorsque toutes les erreurs détectées ont été corrigées.

Nous reprenons le modèle [ViewModel11] précédent mais nous l'affichons maintenant avec la vue [Action12Get.cshtml] suivante :



```

1. @model Exemple_03.Models.ViewModel11
2. @{
3.     Layout = null;
4. }
5.
6. <!DOCTYPE html>
7.
8. <html>
9. <head>
10.    <meta name="viewport" content="width=device-width" />
11.    <title>Action12Get</title>
12.    <link rel="stylesheet" href("~/Content/Site.css" />
13.    <script type="text/javascript" src("~/Scripts/jquery-1.8.2.min.js" ></script>
14.    <script type="text/javascript" src "~/Scripts/jquery.validate.min.js" ></script>
15.    <script type="text/javascript" src "~/Scripts/jquery.validate.unobtrusive.min.js" ></script>
16. </head>
17. <body>
18.    <h3>Formulaire ASP.NET MVC - Validation 1</h3>
19.    @using (Html.BeginForm("Action11Post", "First"))
20.    {
21.        <table>
22.            <thead>
23.                <tr>
24.                    <th>Type attendu</th>
25.                    <th>Valeur saisie</th>
26.                    <th>Message d'erreur</th>
27.                </tr>
28.            </thead>
29.            <tbody>
30. ...
31.        </tbody>
32.    </table>
33.    <p>
34.        <input type="submit" value="Valider" />
35.    </p>
36.    }
37. </body>
38. </html>

```

**Note :** ligne 13, adaptez la version de jQuery à celle que vous avez avec votre version de Visual Studio (voir ci-après).

La validation côté client nécessite la présence de la ligne 3 ci-dessous dans le fichier [Web.config] de l'application.

```

1.    <appSettings>
2.        ...
3.            <add key="ClientValidationEnabled" value="true" />
4.    </appSettings>

```

- lignes 1-4 : la section [appSettings] doit être un enfant direct de la section [configuration] du fichier [Web.config] ;

La vue [Action12Get] est identique à la vue [Action11Get] précédente aux lignes 13-15 près. Celles-ci incluent dans la vue, les scripts Javascript nécessaires pour la validation côté client. Ces scripts sont trouvés dans le dossier [Scripts] du projet :



Chaque script a une version normale [.js] et une version minifiée [min.js]. Cette dernière version est plus légère mais illisible. On l'utilise en production. La version lisible est utilisée en développement.

La vue [Action12Get.cshtml] sera affichée par l'action [Action12Get] suivante :

```

1. // Action12-GET
2. [HttpGet]
3. public ViewResult Action12Get()
4. {
5.     return View("Action12Get", new ViewModel11());
6. }

```

Le formulaire saisi sera traité par l'action [Action12Post] suivante :

```

1. // Action12-POST
2. [HttpPost]
3. public ViewResult Action12Post(ViewModel11 modèle)
4. {
5.     return View("Action12Get", modèle);
6. }

```

Voyons ce que ça change sur un exemple :

Type attendu	Valeur saisie	Message d'erreur
Chaîne d'au moins quatre caractères	a	Information incorrecte 2
Chaîne d'au plus quatre caractères	aaaaaa	
Chaîne de quatre caractères exactement	a	
Nombre entier	aaaa	
Nombre entier dans l'intervalle [1,100]	0	

Dès qu'on tape un caractère en [1], le message en [2] s'affiche parce que la valeur attendue doit avoir au moins quatre caractères. Ainsi la validation est-elle faite à chaque nouveau caractère tapé. Le message d'erreur disparaît au quatrième caractère tapé. Ceci fait, validons le formulaire :

Action12Get

localhost:50803/First/Action12Get 3

Applications Favoris Gestionnaire de favo...

### Formulaire ASP.NET MVC - Validation 2

Type attendu	Valeur saisie	Message d'erreur
Chaîne d'au moins quatre caractères	<input type="text" value="aaaa"/>	
Chaîne d'au plus quatre caractères	<input type="text" value="aaaaaaa"/>	Information requise
Chaîne de quatre caractères exactement	<input type="text" value="aaaa"/>	Information requise
Nombre entier	<input type="text" value="0"/>	
Nombre entier dans l'intervalle [1,100]	<input type="text" value="0"/>	Information incorrecte
Nombre réel	<input type="text" value="0"/>	
Nombre réel dans l'intervalle [10.2, 11.3]	<input type="text" value="0"/>	Information incorrecte
Adresse mail	<input type="text" value=""/>	Information requise
Date sous la forme dd/jj/aaaa	<input type="text" value=""/>	Information requise
Date postérieure à celle d'aujourd'hui	<input type="text" value="jj/mm/aaaa"/>	Information requise

L'URL [3] nous montre que le [POST] n'a pas eu lieu. Mais le clic sur le bouton [Valider] a déclenché toutes les validations côté client et de nouveaux messages d'erreur sont apparus.

Regardons le HTML généré pour la première saisie par exemple :

```

1.      <tr>
2.          <td><label for="Chaine1">Cha&#238;ne d'au moins quatre caract&#232;res</label></td>
3.          <td><input class="text-box single-line" data-val="true" data-val-regex="Information incorrecte" data-val-regex-pattern="^.{4,}$" data-val-required="Information requise" id="Chaine1" name="Chaine1" type="text" value="" /></td>
4.          <td><span class="field-validation-valid" data-valmsg-for="Chaine1" data-valmsg-replace="true"></span></td>
5.      </tr>
```

- ligne 3, on retrouve :
  - le message d'erreur pour le cas où la saisie est manquante [data-val-required],
  - le message d'erreur pour le cas où la saisie est erronée [data-val-regex],
  - l'expression régulière pour la chaîne saisie [data-val-regex-pattern] ;
- ligne 4, d'autres attributs [data-x] utilisés pour afficher l'éventuel message d'erreur ;

Les attributs [data-x] des balises générées sont exploités par le Javascript que nous avons embarqué dans la vue. Si celui-ci est absent, ces attributs sont tout simplement ignorés et on n'a alors pas de validation côté client. On fonctionne comme dans l'exemple précédent. D'où le terme [unobtrusive] pour cette technique.

### 1.5.10 Gestion des liens de navigation et d'action

Nous allons créer les deux vues suivantes pour illustrer la gestion de liens dans une vue :

- en [1] et [2], on a deux liens de navigation ;
- en [3], on a un lien d'action qui poste le formulaire. Il ne sert pas à naviguer.

La page 1 est générée par la vue [Action16Get.cshtml] suivante :

```

1. @{
2.     Layout = null;
3. }
4.
5. <!DOCTYPE html>
6.
7. <html>
8. <head>
9.     <meta name="viewport" content="width=device-width" />
10.    <title>Action16Get</title>
11.    <script>
12.        function postForm() {
13.            // on récupère le formulaire du document
14.            var form = document.forms[0];
15.            // soumission
16.            form.submit();
17.        }
18.    </script>
19. </head>
20. <body>
21.     <h3>Navigation - page 1</h3>
22.     <h4>@ViewBag.info</h4>
23.     @using (Html.BeginForm("Action16Post", "Second"))
24.     {
25.         @Html.Label("data", "Tapez un texte")
26.         @Html.TextBox("data")
27.         <a href="javascript:postForm()">Valider</a>
28.     }
29.     <p>
30.         @Html.ActionLink("Page 2", "Action17Get", "Second")
31.     </p>
32. </body>
33. </html>
```

- ligne 22 : une information initialisée par l'action qui va délivrer la vue ;
- lignes 23-28 : un formulaire ;
- ligne 25 : un libellé pour le champ [data] ;
- ligne 26 : une boîte de saisie nommée [data] ;
- ligne 27 : un lien de type [submit]. Lors du clic dessus, la fonction Javascript [postForm] est exécutée (attribut href). Celle-ci est définie aux lignes 12-17 ;
- ligne 14 : on récupère une référence sur le 1<sup>er</sup> formulaire du document, celui de la ligne 23 ;
- ligne 16 : ce formulaire est posté. Au final, tout se passe comme si on avait cliqué sur un bouton de type [submit]. Le formulaire est posté au contrôleur et à l'action spécifiées ligne 23 ;
- ligne 30 : un lien de navigation. Le code HTML généré est le suivant :

```
<a href="/Second/Action17Get">Page 2</a>
```

La méthode utilisée est **ActionLink(Texte, Action, Contrôleur)**.

La page 2 est générée par la vue [Action17Get.cshtml] suivante :

```

1.  @{
2.    Layout = null;
3.  }
4.
5.  <!DOCTYPE html>
6.
7.  <html>
8.  <head>
9.    <meta name="viewport" content="width=device-width" />
10.   <title>Action17Get</title>
11.  </head>
12.  <body>
13.    <h3>Navigation - Page 2</h3>
14.    <h4>@ViewBag.info</h4>
15.    <p>
16.      @Html.ActionLink("Page 1", "Action16Get", "Second")
17.    </p>
18.  </body>
19. </html>

```

Les actions qui génèrent ces vues sont les suivantes :

```

1.      // Action16-GET
2.      [HttpGet]
3.      public ViewResult Action16Get()
4.      {
5.        ViewBag.info = string.Format("Contrôleur={0}, Action={1}", RouteData.Values["controller"],
       RouteData.Values["action"]);
6.        return View("Action16Get");
7.      }
8.
9.      // Action16-POST
10.     [HttpPost]
11.     public ViewResult Action16Post(string data)
12.     {
13.       ViewBag.info = string.Format("Contrôleur={0}, Action={1}, Data={2}", RouteData.Values["controller"],
       RouteData.Values["action"], data);
14.       return View("Action16Get");
15.     }
16.
17.      // Action17-GET
18.      [HttpGet]
19.      public ViewResult Action17Get()
20.      {
21.        ViewBag.info = string.Format("Contrôleur={0}, Action={1}", RouteData.Values["controller"],
       RouteData.Values["action"]);
22.        return View();
23.      }

```

- ligne 6, l'action [Action16Get] génère la vue [Action16Get.cshtml], c-à-d la page 1 de l'exemple. Cette vue a pour modèle le [ViewBag] (ligne 5) ;
- ligne 19, l'action [Action17Get] génère la vue [Action17Get.cshtml], c-à-d la page 2 de l'exemple. Cette vue a pour modèle le [ViewBag] (ligne 21) ;
- ligne 11 : l'action [Action16Post] traite le POST du formulaire de la vue [Action16Get.cshtml]. Elle reçoit le paramètre nommé [data]. On se rappelle que c'est le nom du champ de saisie dans le formulaire ;
- ligne 13 : une information est mise dans le [ViewBag] ;
- ligne 14 : la vue [Action16Get.cshtml] est affichée.

Le lecteur est invité à tester cet exemple.

## 1.6 Internationalisation des vues

Nous allons aborder ici le problème de l'internationalisation des vues. C'est un problème complexe dont on trouvera une bonne description dans l'article suivant de Scott Hanselman :

[<http://www.hanselman.com/blog/GlobalizationInternationalizationAndLocalizationInASPNETMVC3JavaScriptAndjQueryPart1.aspx>]

Reprenons tout d'abord sa définition des différents termes liés à l'internationalisation des vues :

Internationalisation (i18n)	faire que l'application supporte différentes langues et locales
Localisation (l10n)	faire que l'application supporte un couple langue / locale spécifique
Globalisation	la combinaison de <i>Internationalisation</i> et <i>Localisation</i>
Langue	langue parlée – désignée par un code ISO (fr : français, es : espagnol, en : anglais, ...)
Locale	une variante de la langue – désignée également par un code ISO (en_GB : anglais de Grande-Bretagne, en_US : anglais des Etats-Unis, ...)

Abordons le problème par un premier exemple.

### 1.6.1 Localisation des nombres réels

On peut remarquer une anomalie dans le formulaire de saisie précédent :

Nombre entier dans l'intervalle [1,100]

Nombre réel  Le champ Nombre réel doit être un nombre.

Nombre réel dans l'intervalle [10.2, 11.3]

Pour le nombre réel, nous avons tapé [0,3] et cela n'a pas été accepté. Il faut taper [0.3] :

Nombre entier dans l'intervalle [1,100]

Nombre réel

Nombre réel dans l'intervalle [10.2, 11.3]

Le format attendu est donc le format anglo-saxon et non le format français. En recherchant sur internet, on trouve des solutions. En voici une.

Les actions [GET] et [POST] deviennent les suivantes :

```
1. // Action13-GET
2. [HttpGet]
3. public ViewResult Action13Get()
4. {
5.     return View("Action13Get", new ViewModel11());
6. }

1. // Action13-POST
2. [HttpPost]
3. public ViewResult Action13Post(ViewModel11 modèle)
4. {
5.     return View("Action13Get", modèle);
6. }
```

La vue [Action13Get.cshtml] est identique à la vue [Action12Get.cshtml] aux scripts Javascript près :

```
1. <head>
2.   <meta name="viewport" content="width=device-width" />
3.   <title>Action13Get</title>
4.   <link rel="stylesheet" href="~/Content/Site.css" />
```

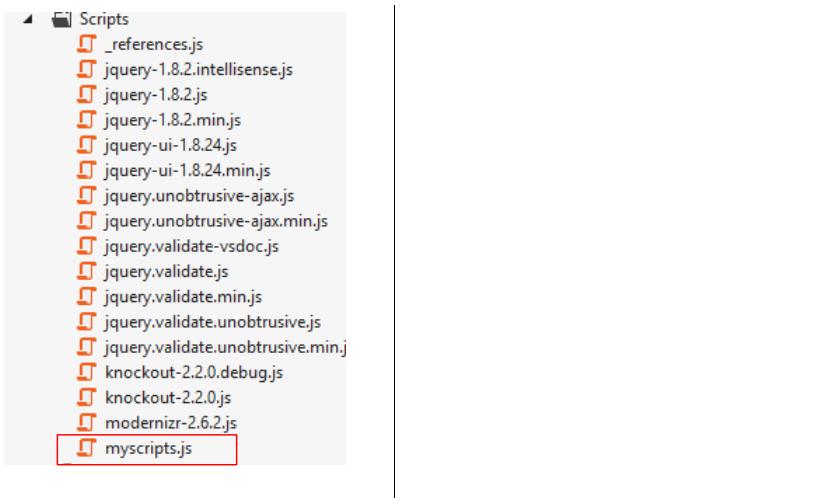
```

5.   <script type="text/javascript" src("~/Scripts/jquery-1.8.2.min.js")></script>
6.   <script type="text/javascript" src "~/Scripts/jquery.validate.min.js"></script>
7.   <script type="text/javascript" src "~/Scripts/jquery.validate.unobtrusive.min.js"></script>
8. ...
9.   <script type="text/javascript" src "~/Scripts/myscripts.js"></script>
10.
11. </head>

```

Note : ligne 5, adaptez la version de jQuery à celle de votre version de Visual Studio.

- ligne 9, nous avons ajouté un script [myscripts.js]. Celui-ci est le suivant :

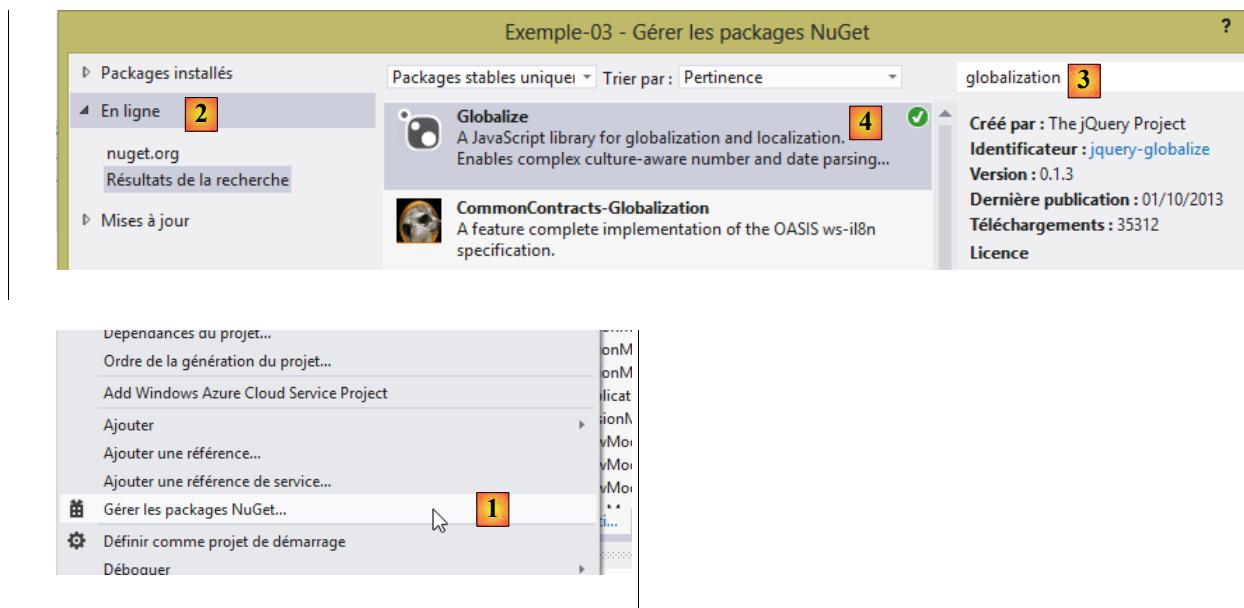


```

1. // http://blog.instance-factory.com/?p=268
2. $.validator.methods.number = function (value, element) {
3.     return this.optional(element) ||
4.         !isNaN(Globalize.parseFloat(value));
5. }
6.
7. $.validator.methods.date = function (value, element) {
8.     return this.optional(element) ||
9.         !isNaN(Globalize.parseDate(value));
10. }
11.
12. jQuery.extend(jQuery.validator.methods, {
13.     range: function (value, element, param) {
14.         //Use the Globalization plugin to parse the value
15.         var val = Globalize.parseFloat(value);
16.         return this.optional(element) || (
17.             val >= param[0] && val <= param[1]);
18.     }
19. });
20.
21. // au chargement du document
22. $(document).ready(function () {
23.     var culture = 'fr-FR';
24.     Globalize.culture(culture);
25. });

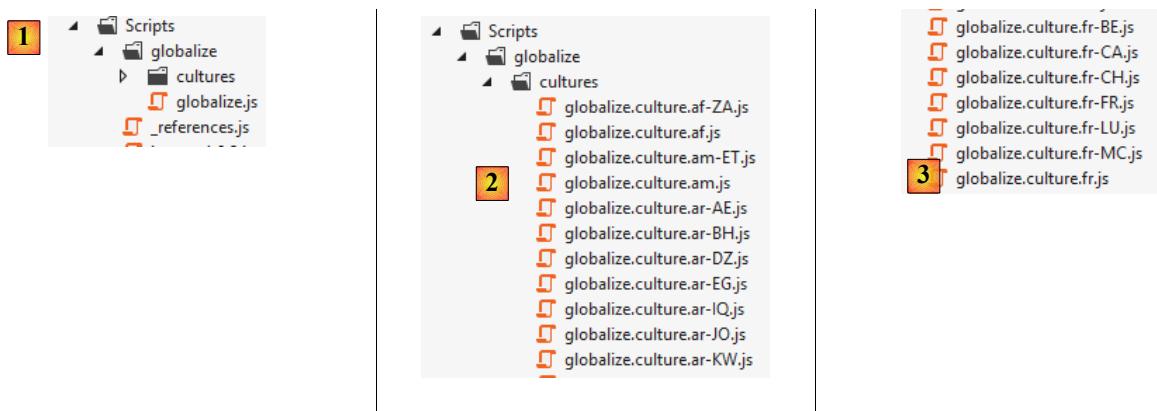
```

J'ai indiqué en ligne 1, où ce script avait été trouvé. Je n'essaierai pas de l'expliquer car je ne le comprends pas. Le Javascript a parfois des côtés hermétiques. Lignes 4, 9, 15, un objet [Globalize] est utilisé. Celui-ci est fourni par la bibliothèque JQuery Globalization qu'on peut obtenir avec [NuGet] :



- en [1], gérez les paquetages [NuGet] du projet [Exemple-03] ;
- en [2], consultez les paquetages en ligne ;
- en [3], tapez le terme [globalization] ;
- en [4], installez le paquetage [Globalize] du projet JQuery.

Une fois le paquetage [Globalize] installé, une nouvelle branche apparaît dans le dossier [Scripts] :



- en [1], un dossier [globalize] a été créé avec le script principal [globalize.js] ;
- en [2], le script principal [globalize.js] est complété par des scripts spécifiques à une langue et une locale ;
- en [3], les scripts spécifiques à la langue française avec les locales (variantes) belges (BE), canadiennes (CA), françaises (FR), suisses (CH), luxembourgeoises (LU), Monégasque (MC).

Le script [globalize.js] et le script de notre culture [globalize.culture.fr-FR.js] doivent faire partie de la liste des scripts inclus dans notre page [Action13Get.cshtml] :

```

1. <head>
2.   <meta name="viewport" content="width=device-width" />
3.   <title>Action13Get</title>
4. ...
5.   <script type="text/javascript" src="~/Scripts/globalize/globalize.js"></script>
6.   <script type="text/javascript" src="~/Scripts/globalize/cultures/globalize.culture.fr-FR.js"></script>
7.   <script type="text/javascript" src="~/Scripts/myscripts.js"></script>
8. </head>

```

- ligne 5 : le script [globalize] ;
- ligne 6 : le script [globalize.culture.fr-FR.js] ;
- ligne 7 : le script [myscripts.js] ;

Revenons sur ce dernier script :

```
1. // http://blog.instance-factory.com/?p=268
2. $validator.methods.number = function (value, element) {
3.   return this.optional(element) ||
4.     !isNaN(Globalize.parseFloat(value));
5. }
6.
7. ...
8.
9. // au chargement du document
10. $(document).ready(function () {
11.   var culture = 'fr-FR';
12.   Globalize.culture(culture);
13. });
```

Les lignes 10-13 fixent la culture côté client à [fr-FR] :

- ligne 10 : la fonction JQuery [ready] est exécutée lorsque le document dans lequel se trouve le script a été entièrement chargé par le navigateur ;
- lignes 11-12 : on fixe la culture côté client à [fr-FR]. Pour cela, il faut que le fichier [globalize.culture.fr-FR.js] soit inclus dans la liste des scripts Javascript associés au document.

Maintenant, nous pouvons tester la nouvelle application :

Nombre entier dans l'intervalle [1,100]	<input type="text" value="0"/>
Nombre réel	<input type="text" value="0,3"/>
Nombre réel dans l'intervalle [10.2, 11.3]	<input type="text" value="0"/>

On peut maintenant taper [0,3] pour le nombre réel, ce qu'on ne pouvait faire auparavant. On rencontre cependant une autre anomalie :

Nombre réel dans l'intervalle [10.2, 11.3]

Ci-dessus, la validation côté client nous laisse taper [11.2] avec la notation anglo-saxonne. Cette valeur n'est pas acceptée côté serveur lorsqu'on valide le formulaire :

Nombre réel dans l'intervalle [10.2, 11.3]  La valeur « 11.2 » n'est pas valide pour Nombre réel dans l'intervalle [10.2, 11.3].

Il faut taper [11,2] et là ça fonctionne côtés client et serveur. Il faudrait que côté client, la notation anglo-saxonne ne soit pas acceptée. Ca doit être possible...

Abordons maintenant l'internationalisation des vues. Nous allons continuer avec l'exemple du formulaire précédent en l'offrant en deux langues : français et anglais.

## 1.6.2 Gérer une culture

La langue des vues est contrôlée par l'objet [Thread.CurrentCulture.CurrentUICulture]. Pour afficher les pages dans la culture [fr-FR], on écrit :

```
Thread.CurrentCulture.CurrentUICulture=new CultureInfo("fr-FR");
```

La localisation (dates, nombres, monnaies, heures, ...) est contrôlée par l'objet [Thread.CurrentCulture.CurrentCulture]. De façon similaire à ce qui a été écrit précédemment, on écrira :

```
Thread.CurrentCulture.CurrentCulture=new CultureInfo("fr-FR");
```

Ces deux instructions pourraient être dans le constructeur de chaque contrôleur de l'application. Mais on pourrait vouloir également factoriser ce code commun à tous les contrôleurs. Nous suivons cette voie.

Nous créons deux nouveaux contrôleurs :



- [I18NController] sera la classe mère de tous les contrôleurs utilisant l'internationalisation ;
- [SecondController] est un contrôleur exemple dérivé de [I18NController].

Le code du contrôleur [I18NController] est le suivant :

```
1.  using System.Threading;
2.  using System.Web;
3.  using System.Web.Mvc;
4.
5.  namespace Exemples.Controllers
6.  {
7.      public abstract class I18NController : Controller
8.      {
9.          public I18NController()
10.         {
11.             // on récupère le contexte de la requête courante
12.             HttpContext httpContext = HttpContext.Current;
13.             // on examine la requête à la recherche du paramètre [lang]
14.             // on le cherche dans les paramètres de l'URL
15.             string langue = httpContext.Request.QueryString["lang"];
16.             if (langue == null)
17.             {
18.                 // on le cherche dans les paramètres postés
19.                 langue = httpContext.Request.Form["lang"];
20.             }
21.             if (langue == null)
22.             {
23.                 // on le cherche dans la session de l'utilisateur
24.                 langue = httpContext.Session["lang"] as string;
25.             }
26.             if (langue == null)
27.             {
28.                 // 1er paramètre de l'en-tête HTTP AcceptLanguages
29.                 langue = httpContext.Request.UserLanguages[0];
30.             }
31.             if (langue == null)
32.             {
33.                 // culture fr-FR
34.                 langue = "fr-FR";
35.             }
36.             // on met la langue en session
37.             httpContext.Session["lang"] = langue;
38.             // on modifie les cultures du thread
39.             Thread.CurrentThread.CurrentCulture = new System.Globalization.CultureInfo(langue);
40.             Thread.CurrentThread.CurrentUICulture = Thread.CurrentThread.CurrentCulture;
41.         }
42.     }
43. }
```

- ligne 7 : [I18NController] dérive de la classe [Controller] ;
- ligne 7 : la classe est déclarée [abstract] pour empêcher son intanciation directe : elle ne peut qu'être dérivée pour être utilisée ;
- ligne 9 : le constructeur de la classe – sera exécuté à chaque instanciation d'un contrôleur dérivé de [I18NController] ;
- ligne 12 : on récupère le contexte de la requête HTTP en cours de traitement par le contrôleur ;
- ligne 15 : on fait l'hypothèse que la langue est fixée par un paramètre [lang] qu'on peut trouver à différents endroits. On cherche dans l'ordre :
  - ligne 15 : dans les paramètres de l'URL [?lang=en-US],
  - ligne 19 : dans les paramètres postés [lang=de],
  - ligne 24 : dans la session de l'utilisateur,
  - ligne 29 : dans les préférences de langue envoyées par le client HTTP,
  - ligne 26 : si on n'a rien trouvé, on fixe la culture à [fr-FR] ;
- ligne 37 : on mémorise la culture dans la session. C'est là qu'elle sera retrouvée lors des requêtes suivantes. L'utilisateur pourra la modifier en la mettant dans les paramètres d'une commande GET ou POST ;

- lignes 39-40 : on fixe la culture de la vue qui sera affichée à l'issue du traitement de la requête courante.

Le contrôleur [SecondController] sera le suivant :

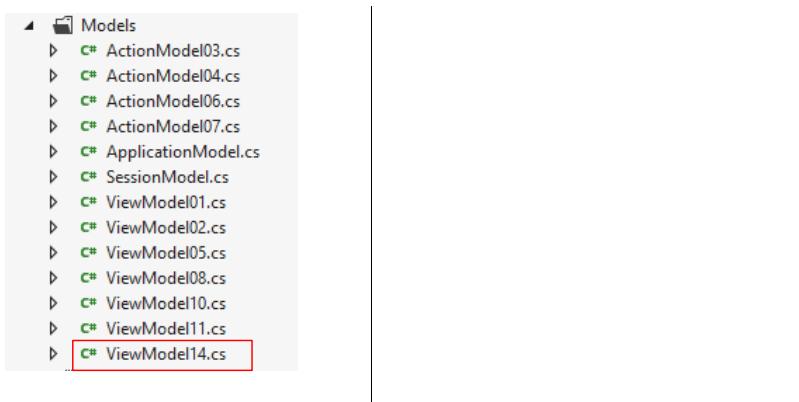
```

1.  using Exemple_03.Models;
2.  using Exemples.Controllers;
3.  using System.Web.Mvc;
4.
5.  namespace Exemple_03.Controllers
6.  {
7.      public class SecondController : I18NController
8.      {
9.          // Action14-GET
10.         [HttpGet]
11.         public ViewResult Action14Get()
12.         {
13.             return View("Action14Get", new ViewModel14());
14.         }
15.
16.         // Action14-POST
17.         [HttpPost]
18.         public ViewResult Action14Post(ViewModel14 modèle)
19.         {
20.             return View("Action14Get", modèle);
21.         }
22.     }
23. }
```

- ligne 7 : [SecondController] dérive de [I18NController]. Ainsi est-on assuré que la culture de la vue à afficher aura été initialisée ;
- ligne 13 : on utilise le modèle de vue [ViewModel14] que nous allons présenter ;
- lignes 13 et 20 : la vue [Action14Get.cshtml] assure l'affichage du formulaire.

### 1.6.3 Internationaliser le modèle de vue [ViewModel14]

Le modèle de vue [ViewModel14] est le suivant :



```

1.  using Exemple_03.Resources;
2.  using System;
3.  using System.Collections.Generic;
4.  using System.ComponentModel.DataAnnotations;
5.  using System.Globalization;
6.  using System.Net.Mail;
7.
8.  namespace Exemple_03.Models
9.  {
10.     public class ViewModel14 : IValidatableObject
11.     {
12.
13.         [Required(ErrorMessageResourceType = typeof(MyResources), ErrorMessageResourceName = "infoRequise")]
14.         [Display(ResourceType = typeof(MyResources), Name = "chaineaumoins4")]
15.         [RegularExpression(@"^.{4}?$", ErrorMessageResourceType = typeof(MyResources), ErrorMessageResourceName =
16.             "infoIncorrecte")]
17.         public string Chaine1 { get; set; }
18.
19.         [Display(ResourceType = typeof(MyResources), Name = "chaineauplus4")]
20.         [Required(ErrorMessageResourceType = typeof(MyResources), ErrorMessageResourceName = "infoRequise")]
21.         [RegularExpression(@"^{1,4}$", ErrorMessageResourceType = typeof(MyResources), ErrorMessageResourceName =
161 / 518
"infoIncorrecte")]
22.     }
23. }
```

```

21.     public string Chaine2 { get; set; }
22.
23.     [Required(ErrorMessageResourceType = typeof(MyResources), ErrorMessageResourceName = "infoRequise")]
24.     [Display(ResourceType = typeof(MyResources), Name = "chaine4exactement")]
25.     [RegularExpression(@"^.{4,4}$", ErrorMessageResourceType = typeof(MyResources), ErrorMessageResourceName =
26.         "infoIncorrecte")]
26.     public string Chaine3 { get; set; }
27.
28.     [Required(ErrorMessageResourceType = typeof(MyResources), ErrorMessageResourceName = "infoRequise")]
29.     [Display(ResourceType = typeof(MyResources), Name = "entier")]
30.     public int Entier1 { get; set; }
31.
32.     [Display(ResourceType = typeof(MyResources), Name = "entierentrebornes")]
33.     [Required(ErrorMessageResourceType = typeof(MyResources), ErrorMessageResourceName = "infoRequise")]
34.     [Range(1, 100, ErrorMessageResourceType = typeof(MyResources), ErrorMessageResourceName = "infoIncorrecte")]
35.     public int Entier2 { get; set; }
36.
37.     [Display(ResourceType = typeof(MyResources), Name = "reel")]
38.     [Required(ErrorMessageResourceType = typeof(MyResources), ErrorMessageResourceName = "infoRequise")]
39.     public double Reel1 { get; set; }
40.
41.     [Display(ResourceType = typeof(MyResources), Name = "reelentrebornes")]
42.     [Required(ErrorMessageResourceType = typeof(MyResources), ErrorMessageResourceName = "infoRequise")]
43.     [Range(10.2, 11.3, ErrorMessageResourceType = typeof(MyResources), ErrorMessageResourceName = "infoIncorrecte")]
44.     public double Reel2 { get; set; }
45.
46.     [Display(ResourceType = typeof(MyResources), Name = "email")]
47.     [Required(ErrorMessageResourceType = typeof(MyResources), ErrorMessageResourceName = "infoRequise")]
48.     [EmailAddress(ErrorMessageResourceType = typeof(MyResources), ErrorMessageResourceName = "infoIncorrecte",
49.         ErrorMessage = "")]
50.     public string Email1 { get; set; }
51.
52.     [Display(ResourceType = typeof(MyResources), Name = "date1")]
53.     [RegularExpression(@"^\s*\d{2}/\d{2}/\d{4}\s*", ErrorMessageResourceType = typeof(MyResources), ErrorMessageResourceName =
54.         "infoIncorrecte")]
55.     [Required(ErrorMessageResourceType = typeof(MyResources), ErrorMessageResourceName = "infoRequise")]
56.     public string Regexp1 { get; set; }
57.
58.     [Display(ResourceType = typeof(MyResources), Name = "date2")]
59.     [Required(ErrorMessageResourceType = typeof(MyResources), ErrorMessageResourceName = "infoRequise")]
60.     [DataType(DataType.Date)]
61.     public DateTime Date1 { get; set; }
62.
63.     // validation
64.     public IEnumerable<ValidationResult> Validate(ValidationContext validationContext)
65.     {
66.         // liste des erreurs
67.         List<ValidationResult> résultats = new List<ValidationResult>();
68.         // le même msg d'erreur pour tous
69.         string errorMessage=MyResources.ResourceManager.GetObject("infoIncorrecte", new
CultureInfo(System.Web.HttpContext.Current.Session["lang"] as string)).ToString();
70.
71.         // Date 1
72.         if (Date1.Date <= DateTime.Now.Date)
73.         {
74.             résultats.Add(new ValidationResult(errorMessage, new string[] { "Date1" }));
75.         }
76.         // Email1
77.         try
78.         {
79.             new MailAddress(Email1);
80.         }
81.         catch
82.         {
83.             résultats.Add(new ValidationResult(errorMessage, new string[] { "Email1" }));
84.         }
85.         // Regexp1
86.         try
87.         {
88.             DateTime.ParseExact(Regexp1, "dd/MM/yyyy", CultureInfo.CreateSpecificCulture("fr-FR"));
89.         }
90.         catch
91.         {
92.             résultats.Add(new ValidationResult(errorMessage, new string[] { "Regexp1" }));
93.         }
94.
95.         // on rend la liste des erreurs
96.         return résultats;
97.     }
98. }

```

Ce modèle est le modèle précédent [ViewModel1] internationalisé. Nous allons décrire le mécanisme d'internationalisation pour le premier attribut de la première propriété. Les autres attributs suivent le même mécanisme.

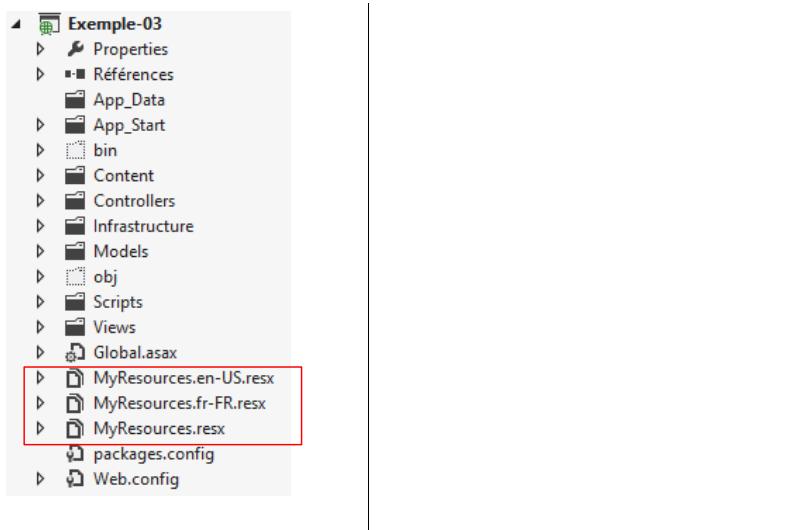
1. [Required(ErrorMessageResourceType = typeof(MyResources), ErrorMessageResourceName = "infoRequise")]

```
2. public string Chaine1 { get; set; }
```

Dans le modèle précédent [ViewModel1], ces lignes étaient les suivantes :

```
a) [Required(ErrorMessage = "Information requise")]
b) public string Chaine1 { get; set; }
```

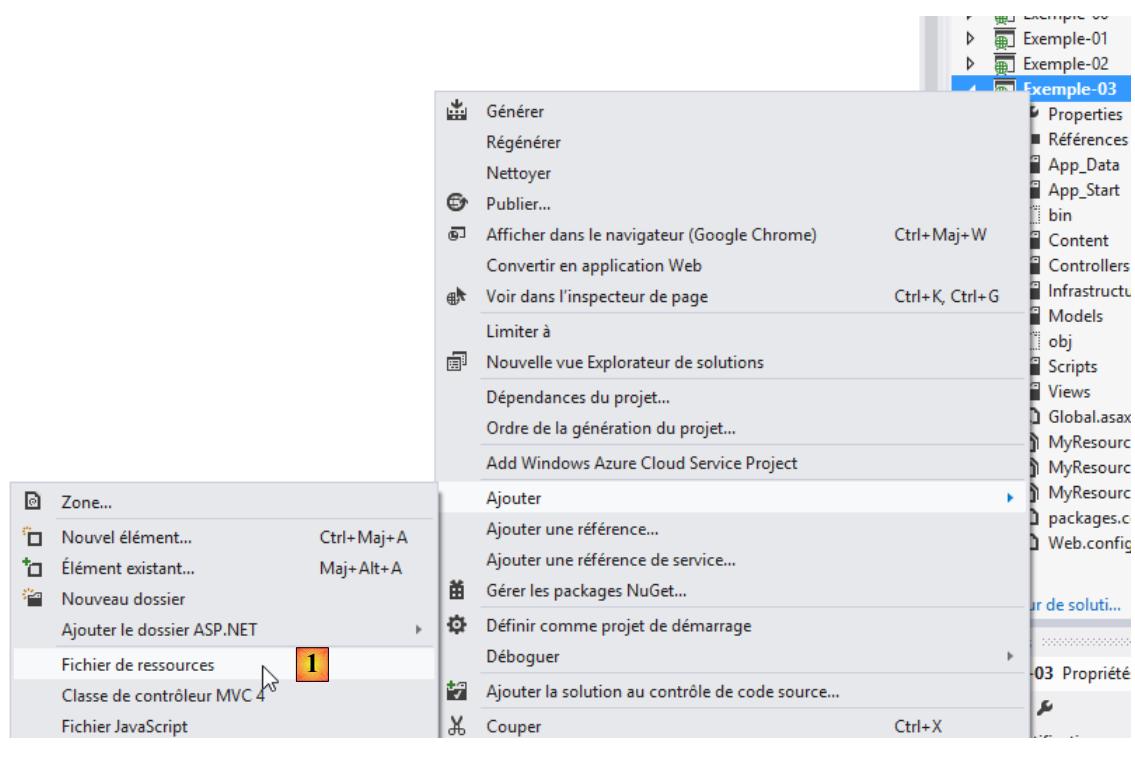
Dans la version internationalisée, ligne 1, les textes à afficher sont placés dans un fichier de ressources. Ici ce fichier s'appelle [MyResources.resx] (*typo*) et a été placé à la racine du projet. On l'appelle un fichier de ressources.

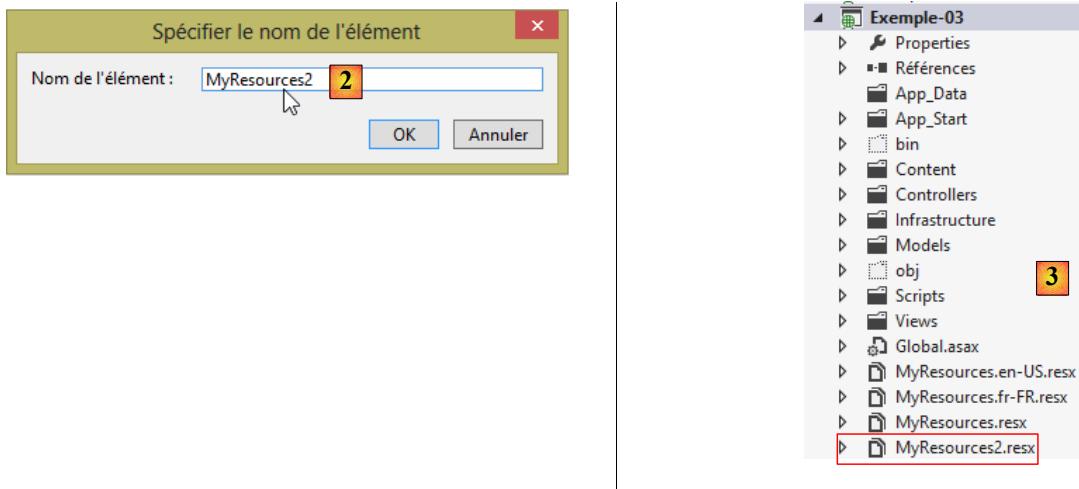


Nous avons créé ici trois fichiers de ressources :

- [MyResources] : ressource par défaut lorsqu'il n'y a pas de ressource pour la locale courante ;
- [MyResources.fr-FR] : ressource pour la locale [fr-FR] ;
- [MyResources.en-US] : ressource pour la locale [en-US] ;

Pour créer un fichier de ressources on procède ainsi [1, 2, 3] :





Cela crée le fichier de ressources [MyResources2.resx]. Lorsqu'on double-clique dessus, on a la page suivante :

	Nom	Valeur	Commentaire
*	String1	2	

Un fichier de ressources est un dictionnaire avec des clés et des valeurs associées à ces clés. On entre la clé en [1], la valeur en [2], la portée de la ressource en [3]. Pour que ces ressources soient lisibles, il faut qu'elles aient la portée [Public]. Revenons à la ligne :

```
[Required(ErrorMessageResourceType = typeof(MyResources), ErrorMessageResourceName = "infoRequise")]
```

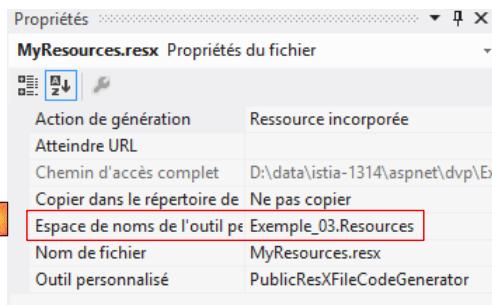
- [ErrorMessageResourceType] : désigne le fichier des ressources. Le paramètre du [typeof] est le nom du fichier. Celui-ci est transformé en classe par le processus de compilation et son binaire inclus dans l'assembly du projet. Donc au final [MyResources] est le nom de la classe des ressources ;
- [ErrorMessageResourceName = "infoRequise"] : désigne une clé dans le fichier des ressources. Au final, la ligne signifie que le message d'erreur à afficher est la valeur du fichier [MyResources] associée à la clé [infoRequise].

Pour créer la clé [infoRequise] et la valeur associée dans le fichier [MyResources] on procède comme suit :

	Nom	Valeur	Commentaire
▶	infoRequise	Information requise	
*			

On entre la clé en [1], la valeur en [2], la portée de la ressource en [3].

Il reste un dernier point à clarifier : l'espace de noms de la classe [MyResources]. Celui-ci est défini dans les propriétés du fichier [MyResources.resx] :



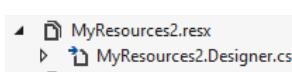
En [1], nous définissons l'espace de noms de la classe [MyResources] qui va être créée à partir du fichier de ressources [MyResources.resx]. Revenons à la ligne internationalisée étudiée :

```
[Required(ErrorMessageResourceType = typeof(MyResources), ErrorMessageResourceName = "infoRequise")]
```

L'opérateur `typeof` attend une classe, ici la classe [MyResources]. Pour que celle-ci soit trouvée, il faut importer son espace de noms dans la classe [ViewModel14] :

```
using Exemple_03.Resources;
```

Pour que la classe [MyResources] soit visible, il faut qu'auparavant le projet ait été généré au moins une fois depuis la création du fichier de ressources [MyResources]. Le code de cette classe est visible dans le fichier [MyResources.Designer.cs] :

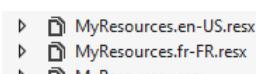


Lorsqu'on double-clique sur ce fichier, on accède au code de la classe [MyResources] :

```
1. namespace Exemple_03.Resources {
2.     using System;
3.
4.
5.     [global::System.CodeDom.Compiler.GeneratedCodeAttribute("System.Resources.Tools.StronglyTypedResourceBuilder", "4.0.0.0")]
6.     [global::System.Diagnostics.DebuggerNonUserCodeAttribute()]
7.     [global::System.Runtime.CompilerServices.CompilerGeneratedAttribute()]
8.     public class MyResources2 {
9.
10.    ...
11.    public static string infoRequise {
12.        get {
13.            return ResourceManager.GetString("infoRequise", resourceCulture);
14.        }
15.    }
16. }
17. }
```

- ligne 1 : l'espace de noms de la classe ;
- ligne 11 : la clé [infoRequise] est devenue une propriété statique de la classe [MyResources]. Elle est accessible via la notation [MyResources.infoRequise]. Par ailleurs, on notera que cette propriété est de portée [public]. Sans cela, elle ne serait pas accessible. Il est bon de se le rappeler car malheureusement la portée par défaut est [internal] et cela est cause d'erreurs difficiles à comprendre lorsqu'on oublie de changer cette portée.

Pourquoi maintenant, trois fichiers de ressources ?



Nous avons créé [MyResources.resx]. C'est la ressource racine. Ensuite nous créons autant de fichiers de ressources [MyResources.locale.resx] qu'il y a de locales (langues) à gérer. Ici nous gérons le français [fr-FR] et l'anglais américain [en-US]. Lorsque la locale courante n'est ni [fr-FR], ni [en-US], c'est la ressource racine [MyResources.resx] qui est utilisée.

Le contenu final de [MyResources.resx] est le suivant :

Nom	Valeur
chaine4exactement	Chaîne de quatre caractères exactement
chaineaumoins4	Chaîne d'au moins quatre caractères
chaineauplus4	Chaîne d'au plus quatre caractères
date1	Date sous la forme dd/jj/aaaa
date2	Date postérieure à celle d'aujourd'hui
email	Adresse mail
entier	Nombre entier
entierentrebornes	Nombre entier dans l'intervalle [1,100]
error	Message d'erreur
infolncorrecte	Information incorrecte
infoRequise	Information requise
reel	Nombre réel
reelentrebornes	Nombre réel dans l'intervalle [10,2,-11,3]
type	Type attendu
value	Valeur saisie

Les messages seront en français lorsque la locale ne sera pas reconnue. Le contenu final de [MyResources.fr-FR.resx] est identique et obtenu par simple copie de fichier.

Le contenu final de [MyResources.en-US.resx] est obtenu lui aussi par copie de fichier puis modifié comme suit :

Nom	Valeur
chaine4exactement	String with exactly four characters
chaineaumoins4	String with at least four characters
chaineauplus4	String with at most four characters
date1	Date with format dd/mm/yyyy
date2	Date after today's date
email	Mail address
entier	Integer number
entierentrebornes	Integer number in range [1,100]
error	Error message
infolncorrecte	Invalid data
infoRequise	Required data
reel	Real number
reelentrebornes	Real number in range [10.2-11.3]
type	Expected Type
value	Input data

Revenons sur la vue [ViewModel14] et sa méthode [Validate] :

```

1.      // validation
2.      public IEnumerable<ValidationResult> Validate(ValidationContext validationContext)
3.      {
4.          // liste des erreurs
5.          List<ValidationResult> résultats = new List<ValidationResult>();
6.          // le même msg d'erreur pour tous
7.          string errorMessage=MyResources.ResourceManager.GetObject("infoInCorrecte", new
    CultureInfo(System.Web.HttpContext.Current.Session["lang"] as string)).ToString();
8.
9.          // Date 1
10.         if (Date1.Date <= DateTime.Now.Date)
11.         {
12.             résultats.Add(new ValidationResult(errorMessage, new string[] { "Date1" }));
13.         }
14.     ...
15.     // on rend la liste des erreurs

```

```

16.         return résultats;
17.     }

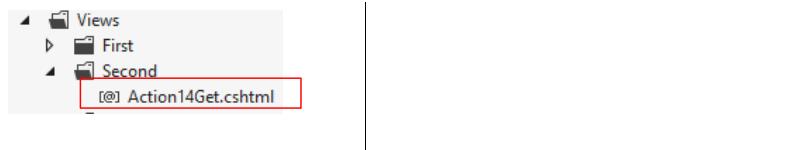
```

La ligne 7 montre comment récupérer un message du fichier de ressources [MyResources]. Ici on veut récupérer le message associé à la clé [infoIncorrecte] et ceci dans la culture du moment :

- **MyResources.ResourceManager.GetObject("infoIncorrecte", new CultureInfo("en-US"))** : obtient l'objet associé à la clé [infoIncorrecte] dans le fichier de ressources [MyResources.en-US.resx] ;
- nous avons vu que le contrôleur [I18NController] mettait la culture courante en session associée à la clé [lang]. La culture courante peut donc être récupérée par **System.Web.HttpContext.Current.Session["lang"] as string** ;
- la ressource est récupérée avec le type [object]. Pour avoir le message d'erreur, on lui applique la méthode [ToString].

#### 1.6.4 Internationaliser la vue [Action14Get.cshtml]

Nous faisons évoluer la vue d'affichage du formulaire de la façon suivante :



```

1. @model Exemple_03.Models.ViewModel14
2. @using Exemple_03.Resources
3. @{
4.     Layout = null;
5. }
6.
7. <!DOCTYPE html>
8.
9. <html>
10. <head>
11.     <meta name="viewport" content="width=device-width" />
12.     <title>Action14Get</title>
13.     <link rel="stylesheet" href="~/Content/Site.css" />
14.     <script type="text/javascript" src="~/Scripts/jquery-1.8.2.min.js"></script>
15.     <script type="text/javascript" src="~/Scripts/jquery.validate.min.js"></script>
16.     <script type="text/javascript" src="~/Scripts/jquery.validate.unobtrusive.min.js"></script>
17.     <script type="text/javascript" src="~/Scripts/globalize/globalize.js"></script>
18.     <script type="text/javascript" src="~/Scripts/globalize/cultures/globalize.culture.fr-FR.js"></script>
19.     <script type="text/javascript" src="~/Scripts/globalize/cultures/globalize.culture.en-US.js"></script>
20.     <script type="text/javascript" src="~/Scripts/myscripts2.js"></script>
21. <script>
22.     $(document).ready(function () {
23.         var culture = '@System.Threading.Thread.CurrentThread.CurrentCulture';
24.         Globalize.culture(culture);
25.     });
26. </script>
27.
28. </head>
29. <body>
30.     <h3>Formulaire ASP.NET MVC - Internationalisation</h3>
31.     @using (Html.BeginForm("Action14Post", "Second"))
32.     {
33.         <table>
34.             <thead>
35.                 <tr>
36.                     <th>@MyResources.type</th>
37.                     <th>@MyResources.value</th>
38.                     <th>@MyResources.error</th>
39.                 </tr>
40.             </thead>
41.             <tbody>
42.                 <tr>
43.                     <td>@Html.LabelFor(m => m.Chaine1)</td>
44.                     <td>@Html.EditorFor(m => m.Chaine1)</td>
45.                     <td>@Html.ValidationMessageFor(m => m.Chaine1)</td>
46.                 </tr>
47. ...
48.             </tbody>
49.         </table>
50.         <p>
51.             <input type="submit" value="Valider" />
52.         </p>
53.     }
54. </body>
55. </html>
56. <!-- choix d'une langue -->
57. @using (Html.BeginForm("Lang", "Second"))

```

```

58. {
59.     <table>
60.         <tr>
61.             <td><a href="javascript:postForm('fr-FR','/Second/Action14Get')">Français</a></td>
62.             <td><a href="javascript:postForm('en-US','/Second/Action14Get')">English</a></td>
63.         </tr>
64.     </table>
65. }

```

**Note :** ligne 14, adaptez la version de jQuery à celle de votre version de Visual Studio.

Commençons par le plus simple, les lignes 36-38. Elles utilisent les propriétés statiques de la classe [MyResources] que nous venons de décrire. Pour avoir accès à la classe [MyResources], il faut importer son espace de noms (ligne 2).

Dans les messages internationalisés, il faut prévoir également ceux qui sont affichés par le framework de validation côté client. Pour cela, il faut utiliser les bibliothèques JQuery des lignes 17-19. Nous utilisons les fichiers JQuery pour les deux cultures que nous gérons [fr-FR] et [en-US]. Par ailleurs, on se souvient peut être que la vue [Action13Get] utilisait le script Javascript [myscripts.js] suivant :

```

1. // au chargement du document
2. $(document).ready(function () {
3.     var culture = 'fr-FR';
4.     Globalize.culture(culture);
5. });

```

Maintenant, la culture n'est plus seulement [fr-FR], elle varie. Aussi ces lignes sont-elles désormais générées par la vue [Action14Get] elle-même aux lignes 21-26. Ces six lignes seront incluses dans la page HTML envoyée au client.

- ligne 23 : la variable Javascript [culture] est initialisée avec la culture courante du thread de la requête en cours de traitement. On se souvient peut être que celle-ci a été initialisée par le constructeur de la classe [I18NController] (page 160) :

```

1.     // on met la langue en session
2.     httpContext.Session["lang"] = langue;
3.     // on modifie les cultures du thread
4.     Thread.CurrentCulture = new System.Globalization.CultureInfo(langue);
5.     Thread.CurrentThread.CurrentUICulture = Thread.CurrentThread.CurrentCulture;

```

Si la culture courante est [en-US], le script Javascript incorporé dans la page HTML devient :

```

1.     <script>
2.         $(document).ready(function () {
3.             var culture = 'en-US';
4.             Globalize.culture(culture);
5.         });
6.     </script>

```

On a déjà dit que la fonction [\$(document).ready] était exécutée à la fin du chargement de la page par le navigateur. Son exécution va avoir pour effet de fixer la culture du framework de validation côté client. Avec la culture [en-US] les messages d'erreur du framework seront en anglais et proviendront du fichier de ressources [MyResources.en-US.resx]. Nous verrons comment.

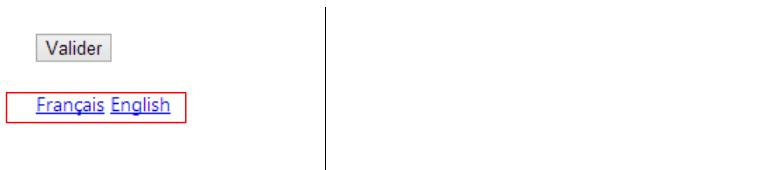
Maintenant examinons les lignes 57-65 :

```

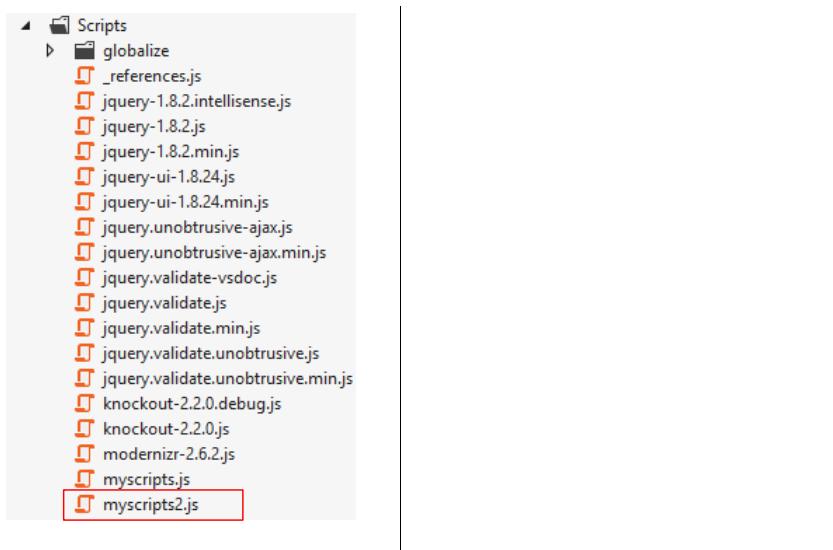
1. <!-- choix d'une langue -->
2. @using (Html.BeginForm("Lang", "Second"))
3. {
4.     <table>
5.         <tr>
6.             <td><a href="javascript:postForm('fr-FR','/Second/Action14Get')">Français</a></td>
7.             <td><a href="javascript:postForm('en-US','/Second/Action14Get')">English</a></td>
8.         </tr>
9.     </table>
10. }

```

On a là un second formulaire, le premier étant aux lignes 31-53. Ce formulaire affiche en bas de page les liens suivants :



- ligne 2 : le formulaire est posté à l'action [Lang] du contrôleur [Second]. Pour l'instant on ne voit aucune valeur qui pourrait être postée ;
- lignes 6 et 7 : un clic sur les liens provoque l'exécution de la fonction Javascript [postForm]. Où se trouve cette fonction ? Dans le script [myscripts2.js] référencé ligne 20 de la vue :



Son contenu est le suivant :

```

1. function postForm(lang, url) {
2.   // on récupère le deuxième formulaire du document
3.   var form = document.forms[1];
4.   // on lui ajoute l'attribut caché lang
5.   var hiddenField = document.createElement("input");
6.   hiddenField.setAttribute("type", "hidden");
7.   hiddenField.setAttribute("name", "lang");
8.   hiddenField.setAttribute("value", lang);
9.   // ajout du champ caché dans le formulaire
10.  form.appendChild(hiddenField);
11.  // on lui ajoute l'attribut caché url
12.  var hiddenField = document.createElement("input");
13.  hiddenField.setAttribute("type", "hidden");
14.  hiddenField.setAttribute("name", "url");
15.  hiddenField.setAttribute("value", url);
16.  // ajout du champ caché dans le formulaire
17.  form.appendChild(hiddenField);
18.  // soumission
19.  form.submit();
20. }
21.
22. // http://blog.instance-factory.com/?p=268
23. $.validator.methods.number = function (value, element) {
24.   return this.optional(element) ||
25.     isNaN(Globalize.parseFloat(value));
26. }
27.
28. $.validator.methods.date = function (value, element) {
29.   return this.optional(element) ||
30.     isNaN(Globalize.parseDate(value));
31. }
32.
33. jQuery.extend(jQuery.validator.methods, {
34.   range: function (value, element, param) {
35.     //Use the Globalization plugin to parse the value
36.     var val = Globalize.parseFloat(value);
37.     return this.optional(element) || (
38.       val >= param[0] && val <= param[1]);
39.   }
40. });
  
```

Les lignes 22-40 sont celles déjà présentes dans le script [myscripts.js] utilisé dans l'exemple précédent (page 157). Nous ne revenons pas dessus. La fonction [postForm] exécutée lors du clic sur les liens des langues est aux lignes 1-20 :

- ligne 1 : la fonction admet deux paramètres, [`lang`] qui est la culture choisie par l'utilisateur et [`url`] qui est l'URL vers laquelle doit être redirigé le navigateur client une fois le changement de culture opéré. Ces deux paramètres sont précisés à l'appel :

```
<td><a href="javascript:postForm('fr-FR', '/Second/Action14Get')">Français</a></td>
<td><a href="javascript:postForm('en-US', '/Second/Action14Get')">English</a></td>
```

- ligne 3 : on récupère une référence sur le deuxième formulaire du document ;
- lignes 5-8 : on crée par programmation la balise

```
<input type="hidden" value="xx-XX"/>
```

où [xx-XX] est la valeur du paramètre [lang] de la fonction ;

- ligne 10 : toujours par programmation, on ajoute cette balise au second formulaire. Au final, tout se passe comme si cette balise était présente depuis le début dans le second formulaire. Sa valeur sera donc postée. C'est ce qu'on voulait ;
- lignes 11-17 : on répète le même mécanisme pour une balise

```
<input type="hidden" value="url"/>
```

où [url] est la valeur du paramètre [url] de la fonction ;

- ligne 19 : le second formulaire est maintenant posté. A quelle URL ?

Il faut revenir au code du second formulaire dans la page [Action14Get.cshtml] :

```
1. @using (Html.BeginForm("Lang", "Second"))
2. {
3. ...
4. }
```

Le formulaire est donc posté à l'URL [/Second/Lang]. Il nous faut alors définir une action [Lang] dans le contrôleur [SecondController]. Ce sera la suivante :

```
1. public class SecondController : I18NController
2. {
3.     // Action14-GET
4.     [HttpGet]
5.     public ViewResult Action14Get()
6.     {
7.         return View("Action14Get", new ViewModel14());
8.     }
9.
10.    // Action14-POST
11.    [HttpPost]
12.    public ViewResult Action14Post(ViewModel14 modèle)
13.    {
14.        return View("Action14Get", modèle);
15.    }
16.
17.    // langue
18.    [HttpPost]
19.    public RedirectResult Lang(string url)
20.    {
21.        // on redirige le client vers url
22.        return new RedirectResult(url);
23.    }
24.
25. }
```

- ligne 18 : l'action ne répond qu'à un [POST] ;
- ligne 19 : elle ne récupère que le paramètre nommé [url] ;
- ligne 22 : elle répond au client de se rediriger vers cette URL.

Mais qu'est devenu le paramètre nommé [lang] ? Il faut maintenant se rappeler que le contrôleur [SecondController] dérive de la classe [I18NController] (ligne 1 ci-dessous). C'est ce contrôleur (cf page 160) qui gère le paramètre [lang] :

```
1. public abstract class I18NController : Controller
2. {
3.     public I18NController()
4.     {
5.         // on récupère le contexte de la requête courante
6.         HttpContext httpContext = System.Web.HttpContext.Current;
7.         // on examine la requête à la recherche du paramètre [lang]
8.         // on le cherche dans les paramètres de l'URL
9.         string langue = httpContext.Request.QueryString["lang"];
10.        if (langue == null)
11.        {
12.            // on le cherche dans les paramètres postés
13.            langue = httpContext.Request.Form["lang"];
14.        }
15.    }
16.
17.    // ...
18. }
```

```

15.     if (langue == null)
16.     {
17.         // on le cherche dans la session de l'utilisateur
18.         langue = httpContext.Session["lang"] as string;
19.     }
20.     if (langue == null)
21.     {
22.         // 1er paramètre de l'entête HTTP AcceptLanguages
23.         langue = httpContext.Request.UserLanguages[0];
24.     }
25.     if (langue == null)
26.     {
27.         // culture fr-FR
28.         langue = "fr-FR";
29.     }
30.     // on met la langue en session
31.     httpContext.Session["lang"] = langue;
32.     // on modifie les cultures du thread
33.     Thread.CurrentThread.CurrentCulture = new CultureInfo(langue);
34.     Thread.CurrentThread.CurrentUICulture = Thread.CurrentThread.CurrentCulture;
35. }
```

Dans notre exemple étudié, le paramètre [lang] est posté. Il sera donc trouvé à la ligne 13, mis en session à la ligne 31 et utilisé pour mettre à jour la culture du thread courant lignes 33-34.

Que va-t-il se passer ensuite ? Revenons sur les liens :

```
<td><a href="javascript:postForm('fr-FR','/Second/Action14Get')">Français</a></td>
<td><a href="javascript:postForm('en-US','/Second/Action14Get')">English</a></td>
```

L'URL de redirection est [/Second/Action14Get]. L'action [Action14Get] est donc exécutée :

```

1.  public class SecondController : I18NController
2.  {
3.      // Action14-GET
4.      [HttpGet]
5.      public ViewResult Action14Get()
6.      {
7.          return View("Action14Get", new ViewModel14());
8.      }
9.  ...
10. }
```

Auparavant, le constructeur de la classe [I18NController] est exécuté :

```

1.  public abstract class I18NController : Controller
2.  {
3.      public I18NController()
4.      {
5.          // on récupère le contexte de la requête courante
6.          HttpContext httpContext = System.Web.HttpContext.Current;
7.          // on examine la requête à la recherche du paramètre [lang]
8.          // on le cherche dans les paramètres de l'URL
9.          string langue = httpContext.Request.QueryString["lang"];
10.         if (langue == null)
11.         {
12.             // on le cherche dans les paramètres postés
13.             langue = httpContext.Request.Form["lang"];
14.         }
15.         if (langue == null)
16.         {
17.             // on le cherche dans la session de l'utilisateur
18.             langue = httpContext.Session["lang"] as string;
19.         }
20.         if (langue == null)
21.         {
22.             // 1er paramètre de l'entête HTTP AcceptLanguages
23.             langue = httpContext.Request.UserLanguages[0];
24.         }
25.         if (langue == null)
26.         {
27.             // culture fr-FR
28.             langue = "fr-FR";
29.         }
30.         // on met la langue en session
31.         httpContext.Session["lang"] = langue;
32.         // on modifie les cultures du thread
33.         Thread.CurrentThread.CurrentCulture = new CultureInfo(langue);
34.         Thread.CurrentThread.CurrentUICulture = Thread.CurrentThread.CurrentCulture;
35.     }
```

Cette fois-ci, le paramètre [lang] sera trouvé dans la session par la ligne 18. Supposons que sa valeur soit [en-US]. Cette culture devient donc la culture du thread d'exécution de la requête (lignes 33-34). Revenons à l'action [Action14Get] :

```
1.      // Action14-GET
2.      [HttpGet]
3.      public ViewResult Action14Get()
4.      {
5.          return View("Action14Get", new ViewModel14());
6.      }
```

Ligne 5, une instance du modèle de vue [ViewModel14] va être créée :

```
1.  public class ViewModel14 : IValidatableObject
2.  {
3.
4.      [Required(ErrorMessageResourceType = typeof(MyResources), ErrorMessageResourceName = "infoRequise")]
5.      [Display(ResourceType = typeof(MyResources), Name = "chaineau moins 4")]
6.      [RegularExpression(@"^.{4,}$", ErrorMessageResourceType = typeof(MyResources), ErrorMessageResourceName =
7.          "infoIncorrecte")]
7.      public string Chaine1 { get; set; }
8.  ....
```

Parce que la culture du thread courant est [en-US], c'est le fichier [MyResources.en-US.resx] qui va être exploité. Les messages d'erreur seront donc en anglais.

Le modèle [ViewModel14] instancié, la vue [Action14Get.cshtml] est affichée :

```
1.  @model Exemple_03.Models.ViewModel14
2.  @using Exemple_03.Resources
3.  @using System.Threading
4.  @{
5.      Layout = null;
6.  }
7.
8.  <!DOCTYPE html>
9.
10. <html>
11. <head>
12.     <meta name="viewport" content="width=device-width" />
13.     <title>Action14Get</title>
14.     ...
15.     <script>
16.         $(document).ready(function () {
17.             var culture = '@Thread.CurrentCulture';
18.             Globalize.culture(culture);
19.         });
20.     </script>
21.
22. </head>
23. <body>
24.     <h3>Formulaire ASP.NET MVC - Internationalisation</h3>
25.     @using (Html.BeginForm("Action14Post", "Second"))
26.     {
27.         <table>
28.             <thead>
29.                 <tr>
30.                     <th>@MyResources.type</th>
31.                     <th>@MyResources.value</th>
32.                     <th>@MyResources.error</th>
33.                 </tr>
34.             </thead>
35.             <tbody>
36.                 <tr>
37.                     ...
38.                 </tr>
39.             </tbody>
```

Parce que la culture du thread courant est [en-US], le script embarqué dans la page aux lignes 15-20 est :

```
<script>
$(document).ready(function () {
    var culture = 'en-US';
    Globalize.culture(culture);
});
```

Cela assure que le framework de validation va travailler avec les formats américains (date, monnaie, nombres, ...). Toujours pour la même raison, les messages des lignes 30-32 seront tirés du fichier de ressources [MyResources.en-US.resx] et seront donc en anglais.

## 1.6.5 Exemples d'exécution

Voici quelques exemple d'exécution :

The screenshot shows a browser window titled "Action14Get" with the URL "localhost:50803/Second/Action14Get". The page title is "Formulaire ASP.NET MVC - Internationalisation". It contains a table with columns "Type attendu", "Valeur saisie", and "Message d'erreur". The rows represent various validation rules:

Type attendu	Valeur saisie	Message d'erreur
Chaine d'au moins quatre caractères		
Chaine d'au plus quatre caractères		
Chaine de quatre caractères exactement		
Nombre entier	0	
Nombre entier dans l'intervalle [1,100]	0	
Nombre réel	0	
Nombre réel dans l'intervalle [10.2,-11.3]	0	
Adresse mail		
Date sous la forme dd/jj/aaaa		
Date postérieure à celle d'aujourd'hui	jj/mm/aaaa	jj/mm/aaaa

A "Valider" button is at the bottom left, and language links "Français English" are at the bottom right.

The screenshot shows a browser window titled "Action14Get" with the URL "localhost:50803/Second/Action14Get". The page title is "ASP.NET MVC Form - Internationalization". It contains a table with columns "Expected Type", "Input data", and "Error message". The rows represent various validation rules:

Expected Type	Input data	Error message
String with at least four characters		
String with at most four characters		
String with exactly four characters		
Integer number	0	
Integer number in range [1,100]	0	
Real number	0	
Real number in range [10.2-11.3]	0	
Mail address		
Date with format dd/mm/yyyy		
Date after today's date	jj/mm/aaaa	jj/mm/aaaa

A "Valider" button is at the bottom left, and language links "Français English" are at the bottom right.

- en [1], le formulaire en français, en [2], le formulaire en anglais.



- en [3], côté client, les messages d'erreur sont désormais en anglais.

Si on regarde le code source de la page, on voit que ces messages d'erreur ont été embarqués dans la page, donc générés par la vue ASP.NET [Action14Get] et son modèle [ViewModel14] :

```
1.      <tr>
2.          <td><label for="Reel1">Real number</label></td>
3.          <td><input class="text-box single-line" data-val="true" data-val-number="The field Real number must be
   a number." data-val-required="Required data" id="Reel1" name="Reel1" type="text" value="0" /></td>
4.          <td><span class="field-validation-valid" data-valmsg-for="Reel1" data-valmsg-
   replace="true"></span></td>
5.      </tr>
6.      <tr>
7.          <td><label for="Reel2">Real number in range [10.2-11.3]</label></td>
8.          <td><input class="text-box single-line" data-val="true" data-val-number="The field Real number in
   range [10.2-11.3] must be a number." data-val-range="Invalid data" data-val-val-range-max="11.3" data-val-range-
   min="10.2" data-val-required="Required data" id="Reel2" name="Reel2" type="text" value="0" /></td>
9.          <td><span class="field-validation-valid" data-valmsg-for="Reel2" data-valmsg-
   replace="true"></span></td>
10.     </tr>
```

## 1.6.6 Internationalisation des dates

L'internationalisation est un problème complexe. Ainsi regardons la propriété [Date1] et son calendrier :

Date with format dd/mm/yyyy

Date after today's date

[Français](#) [English](#)

On constate que le calendrier est un calendrier français alors que la culture de la page est [en-US]. En HTML5 existe un attribut [lang] permettant de fixer la langue de la page ou d'un composant de la page. On peut alors écrire dans la vue [Action14Get.cshtml] le code suivant :

```

1. @model Exemple_03.Models.ViewModel14
2. @using Exemple_03.Resources
3. @using System.Threading
4. @{
5.     Layout = null;
6.     var lang = Session["lang"] as string;
7. }
8.
9. <!DOCTYPE html>
10.
11. <html lang="@lang">
12. <head>
13. ...

```

- ligne 6 : on récupère la culture dans la session ;
- ligne 11 : on fixe l'attribut [lang] de la page avec cette valeur.

Les tests montrent que le calendrier reste en français même lorsque la page est par ailleurs affichée en anglais. Il y a également un problème avec l'autre date du formulaire :

Date with format dd/mm/yyyy

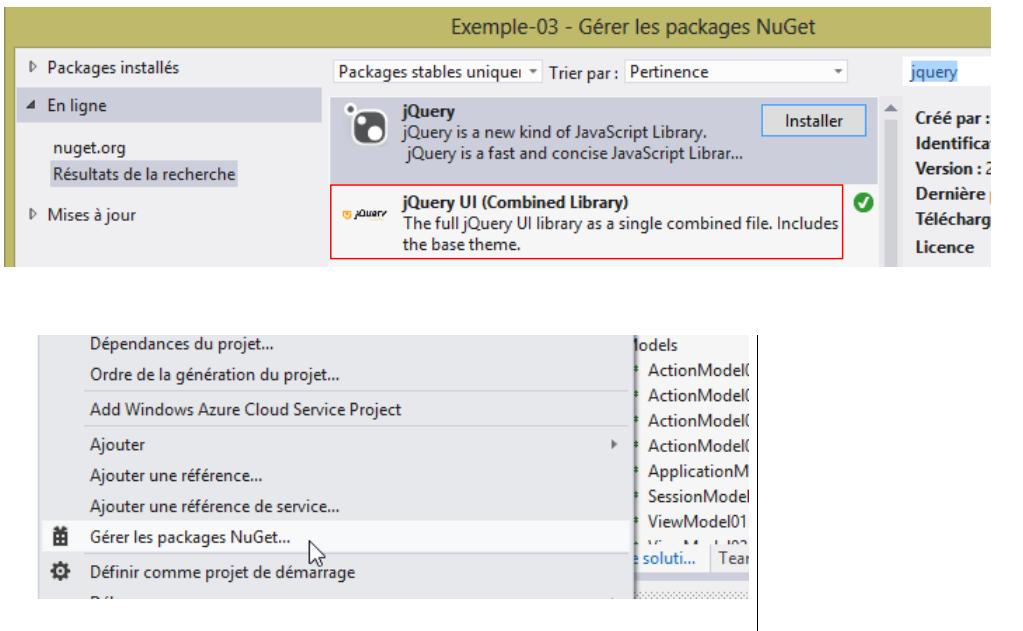
Date after today's date

[Français](#) [English](#)

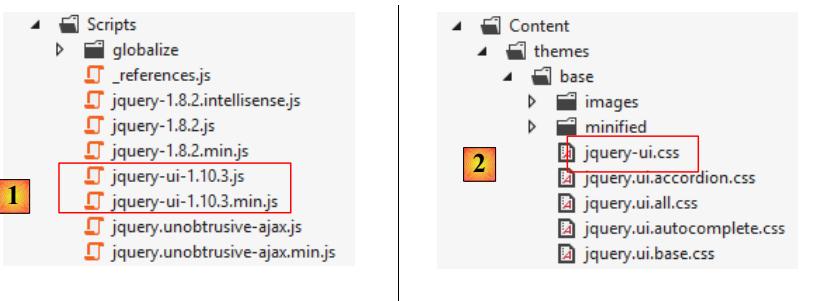
En [1], la date continue à être demandée au format français **jj/mm/aaaa** (20/11/2013) alors que le format américain est **mm/dd/yyyy** (10/21/2013). Nous allons essayer de résoudre ces deux problèmes avec une nouvelle vue et un nouveau modèle de vue.

JQuery UI est un projet dérivé du projet JQuery et offre des composants pour les formulaires dont un calendrier. Ce calendrier peut être internationalisé. C'est ce que nous allons montrer.

Pour commencer, ajoutons [JQuery UI] à notre projet.



Une fois JQuery UI installé, de nouveaux éléments apparaissent dans le projet :

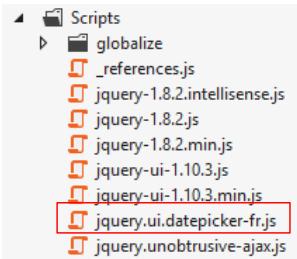


- en [1], la bibliothèque [JQuery UI] en versions normale et minifiée ;
- en [2], la feuille de style de [JQuery UI] ;

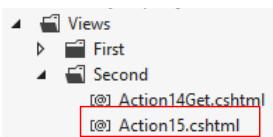
Le calendrier JQuery UI est par défaut en anglais. Pour être internationalisé, il faut ajouter des scripts qu'on peut trouver à l'URL [\[https://github.com/jquery/jquery-ui/tree/master/ui/i18n\]](https://github.com/jquery/jquery-ui/tree/master/ui/i18n) :

<a href="#">jquery.ui.datepicker-fr-CA.js</a>	Datepicker i18n: Fix line endings.	9 months ago
<a href="#">jquery.ui.datepicker-fr-CH.js</a>	Datepicker i18n: Updated Spanish and French locales. Fixes #9289 - Da...	5 months ago
<a href="#">jquery.ui.datepicker-fr.js</a>	Datepicker i18n: Updated Spanish and French locales. Fixes #9289 - Da...	5 months ago

Pour avoir le calendrier JQuery UI en français, on copiera le contenu du fichier [jquery.ui.datepicker-fr.js] ci-dessus dans le dossier [Scripts] du projet.



Le code de la nouvelle vue [Action15.cshtml] est obtenu par recopie de la vue précédente [Action14.cshtml] puis modifié. Nous ne présentons que les modifications :



```

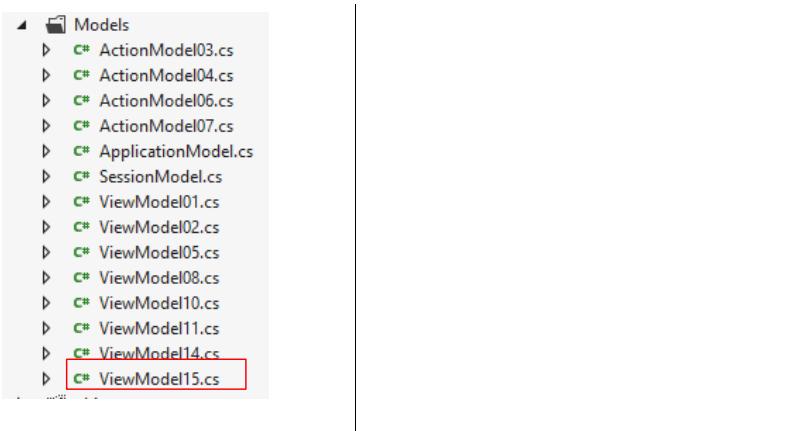
1. @model Exemple_03.Models.ViewModel15
2. @using Exemple_03.Resources
3. @using System.Threading
4. @{
5.     Layout = null;
6. }
7.
8. <!DOCTYPE html>
9.
10. <html lang="@Model.Culture">
11. <head>
12.     <meta name="viewport" content="width=device-width" />
13.     <title>Action15</title>
14. ...
15.     <link rel="stylesheet" href("~/Content/themes/base/jquery-ui.css" />
16.     <script type="text/javascript" src("~/Scripts/jquery-ui-1.10.3.js")></script>
17.     <script type="text/javascript" src("~/Scripts/jquery.ui.datepicker-fr.js")></script>
18.     <script>
19.         $(document).ready(function () {
20.             var culture = '@Thread.CurrentThread.CurrentCulture';
21.             Globalize.culture(culture);
22.             $('#Date1').datepicker($.datepicker.regional['@Model.Regionale']);
23.         });
24.     </script>
25. </head>
26. <body>
27.     <h3>@MyResources.titre</h3>
28.     @using (Html.BeginForm("Action15", "Second"))
29.     {
30.         <table>
31. ...
32.             <tr>
33.                 <td>@Html.LabelFor(m => m.Date1)</td>
34.                 <td>@Html.TextBox("Date1", Model.StrDate1)</td>
35.                 <td>@Html.ValidationMessageFor(m => m.Date1)</td>
36.             </tr>
37.         </tbody>
38.     </table>
39.     <p>
40.         <input type="submit" value="Valider" />
41.     </p>
42.     }
43.     <!-- choix d'une langue --&gt;
44.     @using (Html.BeginForm("Lang", "Second"))
45.     {
46.         &lt;table&gt;
47.             &lt;tr&gt;
48.                 &lt;td&gt;&lt;a href="javascript:postForm('fr-FR','/Second/Action15')"&gt;Français&lt;/a&gt;&lt;/td&gt;
49.                 &lt;td&gt;&lt;a href="javascript:postForm('en-US','/Second/Action15')"&gt;English&lt;/a&gt;&lt;/td&gt;
50.             &lt;/tr&gt;
51.         &lt;/table&gt;
52.     }
53. &lt;/body&gt;
54. &lt;/html&gt;</pre>

```

Note : ligne 16, adaptez la version de jQuery-ui à celle que vous avez téléchargée.

- ligne 15 : on référence la feuille de style de JQuery UI ;
- ligne 16 : on référence la version de JQuery UI téléchargée ;
- ligne 17 : on référence le script du calendrier français que nous venons de télécharger ;
- ligne 34 : la méthode [Html.TextBox] va générer ici une balise [input] de type [text], d'id [Date1] et de name [Date1] ;
- ligne 19 : lorsque le chargement de la page sera terminé, la fonction JQuery UI [datepicker] sera appliquée à l'élément d'**id** [Date1], donc l'élément de la ligne 34. Cette fonction fait que lorsque l'utilisateur va mettre le focus sur le champ de saisie de [Date1], un calendrier va apparaître lui permettant de saisir une date. La fonction [datepicker] admet un paramètre qui lui indique la langue du calendrier. La variable [**@Model.Regionale**] doit valoir :
  - 'fr' pour un calendrier français,
  - " pour un calendrier anglais ;

Le modèle de la vue précédente [Action15.cshtml] sera le modèle [ViewModel15] suivant :



Son code est celui du modèle [ViewModel14] légèrement modifié. Nous ne présentons que les modifications :

```

1.  using Exemple_03.Resources;
2. ...
3.  using System.Web;
4.
5.  namespace Exemple_03.Models
6.  {
7.    [Bind(Exclude = "Culture,Regionale,StrDate1,FormatDate")]
8.    public class ViewModel15 : IValidatableObject
9.    {
10.
11.    ...
12.    [Display(ResourceType = typeof(MyResources), Name = "date1")]
13.    [RegularExpression(@"\s*\d{2}/\d{2}/\d{4}\s*", ErrorMessageResourceType = typeof(MyResources), ErrorMessageResourceName
= "infoIncorrecte")]
14.    [Required(ErrorMessageResourceType = typeof(MyResources), ErrorMessageResourceName = "infoRequise")]
15.    public string Regexp1 { get; set; }
16.
17.    [Display(ResourceType = typeof(MyResources), Name = "date2")]
18.    [Required(ErrorMessageResourceType = typeof(MyResources), ErrorMessageResourceName = "infoRequise")]
19.    [DataType(DataType.Date)]
20.    public DateTime Date1 { get; set; }
21.
22.    // constructeur
23.    public ViewModel15()
24.    {
25.      // Culture du moment
26.      Culture = HttpContext.Current.Session["lang"] as string;
27.      cultureInfo=new CultureInfo(Culture);
28.      // Régionale du calendrier JQuery
29.      Regionale = MyResources.ResourceManager.GetObject("regionale", cultureInfo).ToString();
30.      // format de date
31.      FormatDate = MyResources.ResourceManager.GetObject("formatDate", cultureInfo).ToString();
32.    }
33.
34.
35.
36.    // validation
37.    public IEnumerable<ValidationResult> Validate(ValidationContext validationContext)
38.    {
39.      // liste des erreurs
40.      résultats = new List<ValidationResult>();

```

```

41.     // le même msg d'erreur pour tous
42.     string errorMessage = MyResources.ResourceManager.GetObject("infoIncorrecte", cultureInfo).ToString();
43. ...
44.     // Regexp1
45.     try
46.     {
47.         DateTime.ParseExact(Regexp1, FormatDate, cultureInfo);
48.     }
49.     catch
50.     {
51.         résultats.Add(new ValidationResult(errorMessage, new string[] { "Regexp1" }));
52.     }
53.
54.     // on rend la liste des erreurs
55.     return résultats;
56.
57. }
58.
59. // champs en-dehors du modèle de l'action
60. public string Culture { get; set; }
61. public string Regionale { get; set; }
62. public string StrDate1 { get; set; }
63. public string FormatDate { get; set; }
64.
65. // données locales
66. private CultureInfo cultureInfo;
67. }
68. 
```

Par rapport au modèle précédent [ViewModel14], nous avons quatre propriétés supplémentaires :

- ligne 60 : la culture de la vue, 'fr-FR' ou 'en-US'. Cette culture est initialisée dans le constructeur ligne 26 ;
- ligne 61 : la culture régionale du calendrier JQuery, 'fr' pour un calendrier français, '' pour un calendrier anglais. Ce champ est initialisé par la ligne 29 du constructeur ;
- ligne 63 : le format de la date de la ligne 15 : 'dd/MM/yyyy' pour une date française, 'MM/dd/yyyy' pour une date anglaise. Ce champ est initialisé ligne 31 du constructeur ;
- ligne 62 : la chaîne de caractères à afficher dans le champ de saisie de [Date1]. Ce champ sera initialisé par l'action ;
- ligne 47 : la date [Regexp1] est maintenant vérifiée selon le format de la culture courante.

Les valeurs des propriétés [Regionale] et [FormatDate] sont trouvées dans les fichiers de ressources [MyResources]. Les fichiers de ressources français [MyResources] [MyResources.fr-FR] [1] et le fichier de ressources anglais [2] évoluent comme suit :

formatDate	dd/MM/yyyy	formatDate	MM/dd/yyyy
regionale	fr	regionale	

Nous sommes presque prêts. Nous ajoutons une action [Action15] au contrôleur [SecondController] :

```

1.      // Action15
2.      public ViewResult Action15(FormCollection formData)
3.      {
4.          // méthode HTTP
5.          string method = Request.HttpMethod.ToLower();
6.          // modèle
7.          ViewModel15 modèle = new ViewModel15();
8.          if (method == "get")
9.          {
10.              modèle.StrDate1 = "";
11.          }
12.          else
13.          {
14.              TryUpdateModel(modèle, formData);
15.              modèle.StrDate1 = modèle.Date1.ToString(modèle.FormatDate);
16.          }
17.          // affichage vue
18.          return View("Action15", modèle);
19.      }
```

- ligne 2 : la méthode [Action15] traite aussi bien les [GET] que les [POST]. Dans ce dernier cas, les valeurs postées sont récupérées dans le paramètre [formData] ;
- ligne 5 : on récupère la méthode HTTP de la requête ;
- ligne 7 : on crée le modèle de la vue qui va être affichée (le formulaire) ;
- lignes 8-11 : dans le cas d'une commande [GET], la zone de saisie de [Date1] est initialisée avec une chaîne vide ;
- lignes 12-16 : dans le cas d'une commande [POST] :

- ligne 14 : le modèle est initialisé avec les valeurs postées,
- ligne 15 : la zone de saisie de [Date1] est initialisée avec une chaîne de caractères qui est la valeur de [Date1] formatée selon la culture courante [dd/MM/yyyy] pour une date française, [MM/dd/yyyy] pour une date anglaise ;
- ligne 18 : la vue [Action15.cshtml] est affichée avec son modèle.

Faisons des tests :

Date sous la forme dd/jj/aaaa  **3**

Date postérieure à celle d'aujourd'hui

[Français](#) [English](#)

octobre 2013						
S	M	M	J	V	S	D
			1	2	3	4
5	6	7	8	9	10	11
12	13	14	15	16	17	18
19	20	21	22	23	24	25
26	27	28	29	30	31	

Date formatted as mm/dd/yyyy  **4**

Date after today's date

[Français](#) [English](#)

October 2013						
Su	Mo	Tu	We	Th	Fr	Sa
			1	2	3	4
5	6	7	8	9	10	11
12	13	14	15	16	17	18
19	20	21	22	23	24	25
26	27	28	29	30	31	

- en [1], un calendrier français lorsque la page est en français ;
- en [2], un calendrier anglais lorsque la page est en anglais ;
- en [3], une date au format français lorsque la page est en français ;
- en [4], la même date au format anglais lorsque la page est en anglais ;

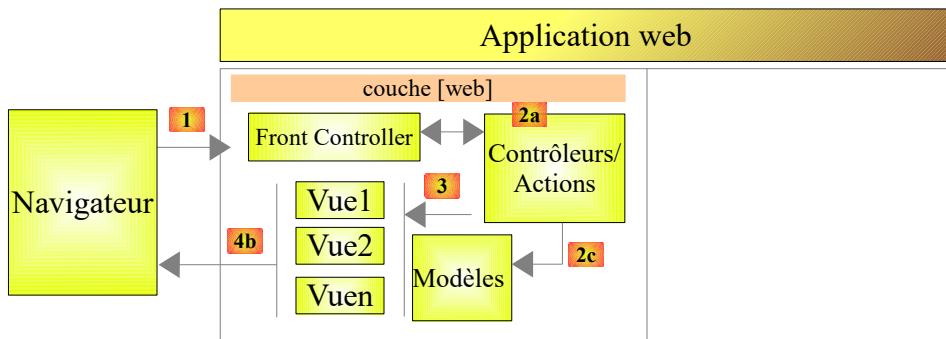
### 1.6.7 Conclusion

On l'aura compris, le thème de l'internationalisation d'une application est un thème complexe...

## 1.7 Ajaxification d'une application ASP.NET MVC

### 1.7.1 La place d'AJAX dans une application web

Pour l'instant, les exemples d'apprentissage étudiés ont l'architecture suivante :



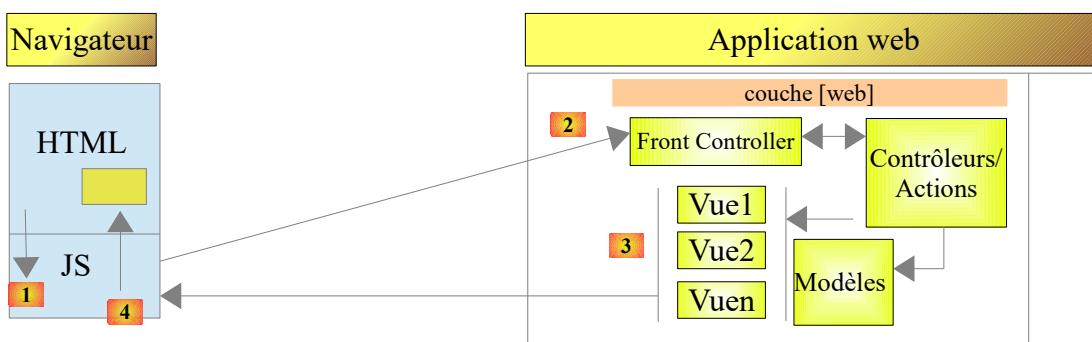
Pour passer d'une vue [Vue1] à une vue [Vue2], le navigateur :

- émet une requête vers l'application web ;
- reçoit la vue [Vue2] et l'affiche à la place de la vue [Vue1].

C'est le schéma classique :

- demande du navigateur ;
- élaboration d'une vue en réponse au client par le serveur web ;
- affichage de cette nouvelle vue par le navigateur.

Il existe un autre mode d'interaction entre le navigateur et le serveur web : **AJAX** (Asynchronous Javascript And XML). Il s'agit en fait d'interactions entre la vue affichée par le navigateur et le serveur web. Le navigateur continue à faire ce qu'il sait faire, afficher une vue HTML mais il est désormais manipulé par du Javascript embarqué dans la vue HTML affichée. Le schéma est le suivant :



- en [1], un événement se produit dans la page affichée dans le navigateur (clic sur un bouton, changement d'un texte, ...). Cet événement est intercepté par du Javascript (JS) embarqué dans la page ;
- en [2], le code Javascript fait une requête HTTP comme l'aurait fait le navigateur. La requête est **asynchrone** : l'utilisateur peut continuer à interagir avec la page sans être bloqué par l'attente de la réponse à la requête HTTP. La requête suit le processus classique de traitement. Rien (ou peu) ne la distingue d'une requête classique ;
- en [3], une réponse est envoyée au client JS. Plutôt qu'une vue HTML complète, c'est plutôt une vue HTML partielle, un flux XML ou JSON (JavaScript Object Notation) qui est envoyé ;
- en [4], le Javascript récupère cette réponse et l'utilise pour mettre à jour une région de la page HTML affichée.

Pour l'utilisateur, il y a changement de vue car ce qu'il voit a changé. Il n'y a cependant pas recharge total d'une page mais simplement modification partielle de la page affichée. Cela contribue à donner de la fluidité et de l'interactivité à la page : parce qu'il n'y a pas de recharge total de la page, on peut se permettre de gérer des événements qu'auparavant on ne gérait pas. Par exemple, proposer à l'utilisateur une liste d'options au fur et à mesure qu'il saisit des caractères dans une boîte de saisie. A chaque nouveau caractère tapé, une requête AJAX est faite vers le serveur qui renvoie alors d'autres propositions. Sans Ajax, ce genre d'aide à la saisie était auparavant impossible. On ne pouvait pas recharger une nouvelle page à chaque caractère tapé.

## 1.7.2 Rudiments de JQuery et de Javascript

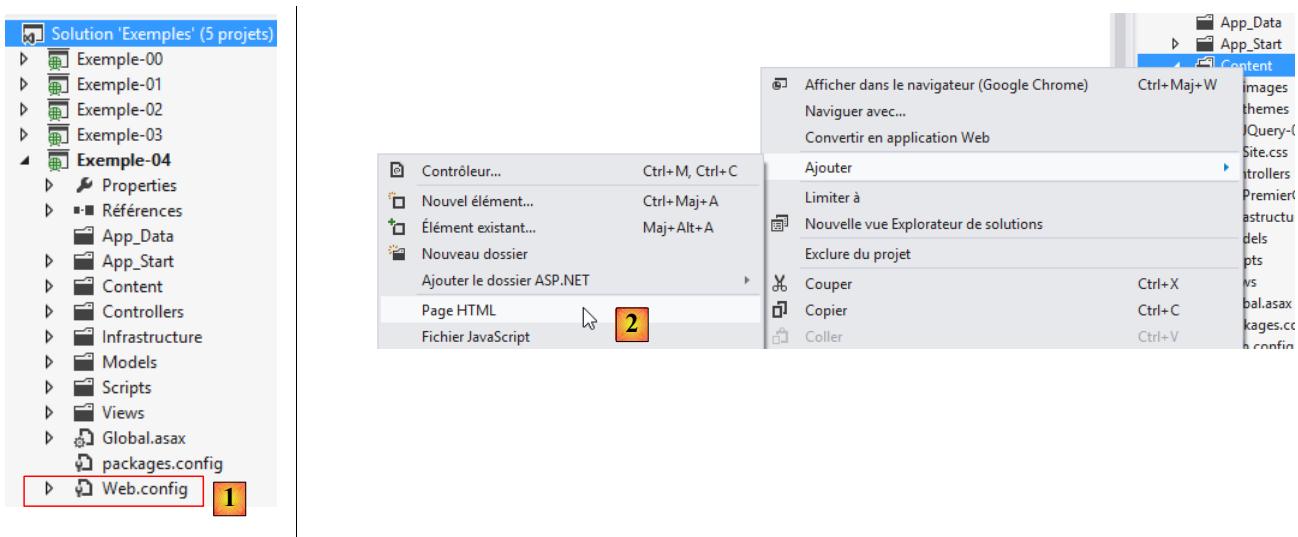
Nous avons souvent associé la bibliothèque Javascript JQuery à nos pages. On y trouve ainsi la ligne :

```
<script type="text/javascript" src="~/Scripts/jquery-1.8.2.min.js"></script>
```

Note : adaptez la version de jQuery à celle de votre version de Visual Studio.

La technologie Ajax d'ASP.NET MVC utilise JQuery. Nous allons nous-mêmes écrire quelques scripts JQuery. Aussi présentons-nous maintenant les rudiments de JQuery à connaître pour comprendre les scripts de ce chapitre.

Nous créons un nouveau projet [Exemple-04] à l'intérieur de notre solution [Exemples] :

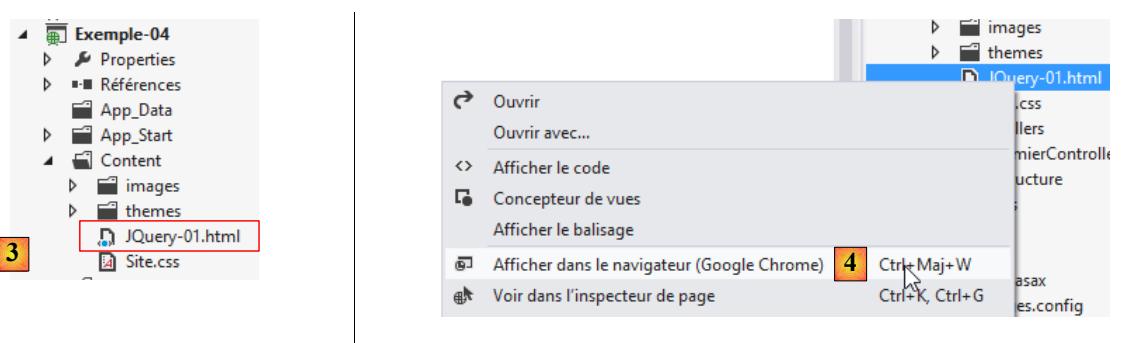


Pour utiliser Ajax avec ASP.NET MVC, une ligne doit être présente dans le fichier de configuration [Web.config] [1] :

```
1. <appSettings>
2. ...
3.   <add key="UnobtrusiveJavaScriptEnabled" value="true" />
4. </appSettings>
```

La ligne 3 autorise l'utilisation d'Ajax dans les vues ASPNET. Elle est présente par défaut.

Nous créons un fichier HTML [JQuery-01.html] dans le dossier [Content] du nouveau projet [2] :



Ce fichier aura le contenu suivant :

```
1. <!DOCTYPE html>
2. <html xmlns="http://www.w3.org/1999/xhtml">
3. <head>
4.   <meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
5.   <title>JQuery-01</title>
6.   <script type="text/javascript" src="/Scripts/jquery-1.8.2.min.js"></script>
7. </head>
```

```

8. <body>
9.   <h3>Rudiments de JQuery</h3>
10.  <div id="element1">
11.    Elément 1
12.  </div>
13. </body>
14. </html>

```

- ligne 6 : importation de JQuery (adaptez la version à celle de votre version de Visual Studio) ;
- lignes 10-12 : un élément de la page d'**id** [element1]. Nous allons jouer avec cet élément.

On visualise ce fichier dans le navigateur Google Chrome [4] et [5] :



Avec Google Chrome faire [Ctrl-Maj-I] pour faire apparaître les outils de développement [6]. L'onglet [Console] [7] permet d'exécuter du code Javascript. Nous donnons dans ce qui suit des commandes Javascript à taper et nous en donnons une explication.

<pre>JS</pre> <pre>\$("#element1") : rend la collection de tous les éléments d'id [element1], donc normalement une collection de 0 ou 1 élément parce qu'on ne peut avoir deux id identiques dans une page HTML.</pre>	<pre>résultat</pre> <pre>Elements Resources Network &gt; \$("#element1") [&lt;div id="element1"&gt;]   Elément 1 &lt;/div&gt;</pre>
--	---

<pre>\$("#element1").text("blabla") : affecte le texte [blabla] à tous les éléments de la collection. Ceci a pour effet de changer le contenu affiché par la page</pre>	<pre>&gt; \$("#element1").text("blabla") [&lt;div id="element1"&gt;blabla&lt;/div&gt;]</pre>
---	--



<pre>\$("#element1").hide() cache les éléments de la collection. Le texte [blabla] n'est plus affiché.</pre>	<pre>&gt; \$("#element1").hide() [&lt;div id="element1"&gt;blabla&lt;/div&gt;]</pre>
--	--



<pre>\$("#element1") : affiche de nouveau la collection. Cela nous permet de voir que l'élément d'id [element1] a</pre>	<pre>&gt; \$("#element1") [&lt;div id="element1" style="display: none;"&gt;blabla&lt;/div&gt;]</pre>
---	--

l'attribut CSS **style='display : none;'** qui fait que l'élément est caché.

```
$("#element1").show()
```

: affiche les éléments de la collection. Le texte [blabla] apparaît de nouveau. C'est l'attribut CSS **style='display : block;'** qui assure cet affichage.

```
> $("#element1").show()  
[<div id="element1" style="display: block;">blabla</div>]
```



```
$("#element1").attr('style','color: red')
```

: fixe un attribut à tous les éléments de la collection. L'attribut est ici [style] et sa valeur [color: red]. Le texte [blabla] passe en rouge.

```
> $("#element1").attr('style','color: red')  
[<div id="element1" style="color: red">blabla</div>]
```



## Tableau

```
> var data=["zéro",1,"deux"]  
undefined  
> data[0]  
"zéro"  
> data[2]  
"deux"  
> data.length  
3  
> data[3]="trois"  
"trois"  
> data  
["zéro", 1, "deux", "trois"]  
> data[1]="un"  
"un"  
> data  
["zéro", "un", "deux", "trois"]
```

## Dictionnaire

```
> data={"zéro":0,"un":1,"erreur":"msg"}  
Object {zéro: 0, un: 1, erreur: "msg"}  
> data["zéro"]  
0  
> data.zéro  
0  
> data.msg="msg2"  
"msg2"  
> data  
Object {zéro: 0, un: 1, erreur: "msg", msg: "msg2"}  
> data.erreur="autre msg"  
"autre msg"  
> data  
Object {zéro: 0, un: 1, erreur: "autre msg", msg: "msg2"}  
,
```

On notera que l'URL du navigateur n'a pas changé pendant toutes ces manipulations. Il n'y a pas eu d'échanges avec le serveur web. Tout se passe à l'intérieur du navigateur. Maintenant, visualisons le code source de la page :



```

1.  <!DOCTYPE html>
2.  <html xmlns="http://www.w3.org/1999/xhtml">
3.  <head>
4.    <meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
5.    <title>jQuery-01</title>
6.    <script type="text/javascript" src="/Scripts/jquery-1.8.2.min.js"></script>
7.  </head>
8.  <body>
9.    <h3>Rudiments de JQuery</h3>
10.   <div id="element1">
11.     Elément 1
12.   </div>
13. </body>
14. </html>

```

C'est le texte initial. Il ne reflète en rien les manipulations que l'on a faites sur l'élément des lignes 10-12. Il est important de s'en souvenir lorsqu'on fait du débogage Javascript. Il est alors souvent inutile de visualiser le code source de la page affichée. Pour connaître le code source de la page actuellement affichée, on procèdera de la façon suivante :



Nous en savons assez pour comprendre les scripts JS qui vont suivre.

### 1.7.3 Mise à jour d'une page avec un flux HTML

#### 1.7.3.1 Les vues

On se propose d'étudier l'application suivante :

The figure consists of two side-by-side screenshots of a web browser window titled "Ajax - 01". Both screenshots show the URL `localhost:59038/Premier/Action01Get`.  
 Left Screenshot (Initial State):  
 - Title: "Ajax - 01"  
 - Subtitle: "Heure de chargement : 09:43:41" (highlighted with orange box 1)  
 - Subtitle: "Opérations arithmétiques sur deux nombres réels A et B positifs ou nuls"  
 - Input fields:  
 - "Valeur de A": 12,78  
 - "Valeur de B": 4,57  
 - Buttons: "Calculer" (highlighted with orange box 2)  
 - Another "Calculer" button  
 Right Screenshot (Result State):  
 - Title: "Ajax - 01"  
 - Subtitle: "Heure de chargement : 09:43:41" (highlighted with orange box 5)  
 - Subtitle: "Opérations arithmétiques sur deux nombres réels A et B positifs ou nuls"  
 - Input fields:  
 - "Valeur de A": 12,78  
 - "Valeur de B": 4,57  
 - Buttons: "Calculer" (highlighted with orange box 6)  
 - Another "Calculer" button  
 - Results box (highlighted with orange box 3):  
 - "Résultats"  
 - "Heure de calcul : 09:44:38" (highlighted with orange box 4)  
 - Calculated values:  
 - A+B=17,35  
 - A-B=8,21  
 - A\*B=58,4046  
 - A/B=2,7964989059081

- en [1], l'heure de chargement de la page ;
- en [2], on fait les quatre opérations arithmétiques sur deux nombres réels A et B ;
- en [3], la réponse du serveur vient s'inscrire dans une région de la page ;
- en [4], l'heure du calcul. Celle-ci est différente de l'heure de chargement de la page [5]. Cette dernière est égale à [1] montrant que la région [6] n'a pas été rechargée. Par ailleurs l'URL [7] de la page n'a pas changé.

#### 1.7.3.2 Le contrôleur, les actions, le modèle, la vue

Nous créons un contrôleur nommé [Premier] :



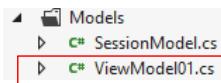
Pour afficher la vue initiale, nous créons l'action [Action01Get] suivante :

```

1.     [HttpGet]
2.     public ViewResult Action01Get()
3.     {
4.         ViewModel01 modèle = new ViewModel01();
5.         modèle.HeureChargement = DateTime.Now.ToString("hh:mm:ss");
6.         return View(modèle);
7.     }
  
```

- ligne 4 : instanciation du modèle de la vue ;
- ligne 5 : initialisation de l'heure de chargement de la vue ;
- ligne 6 : affichage de la vue [Action10Get.cshtml] et de son modèle.

Le modèle [ViewModel01] est le suivant :



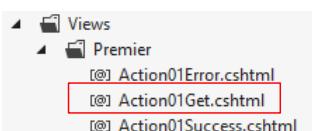
```

1.  using System;
2.  using System.ComponentModel.DataAnnotations;
3.  using System.Web.Mvc;
4.
5.  namespace Exemple_04.Models
6.  {
7.      [Bind(Exclude = "AplusB, AmoinsB, AmultipliéparB, AdiviséparB, Erreur, HeureChargement, HeureCalcul")]
8.      public class ViewModel01
9.      {
10.          // formulaire
11.          [Required(ErrorMessage="Donnée requise")]
12.          [Display(Name="Valeur de A")]
13.          [Range(0, Double.MaxValue, ErrorMessage = "Tapez un nombre positif ou nul")]
14.          public double A { get; set; }
15.          [Required(ErrorMessage = "Donnée requise")]
16.          [Display(Name = "Valeur de B")]
17.          [Range(0, Double.MaxValue, ErrorMessage="Tapez un nombre positif ou nul")]
18.          public double B { get; set; }
19.
20.          // résultats
21.          public string AplusB { get; set; }
22.          public string AmoinsB { get; set; }
23.          public string AmultipliéparB { get; set; }
24.          public string AdiviséparB { get; set; }
25.          public string Erreur { get; set; }
26.          public string HeureChargement { get; set; }
27.          public string HeureCalcul { get; set; }
28.      }
29.  }

```

- lignes 11-14 : la valeur A du formulaire ;
- lignes 15-18 : la valeur B du formulaire ;
- lignes 21-24 : les résultats des quatre opérations arithmétiques sur A et B ;
- ligne 25 : le texte d'une éventuelle erreur ;
- ligne 26 : l'heure de chargement de la vue dans le navigateur ;
- ligne 27 : l'heure de calcul des champs des lignes 21-24 ;
- ligne 7 : ce modèle de vue est également un modèle d'action. On exclut de ce dernier les champs qui ne sont pas postés par le navigateur.

La vue [Action01Get.cshtml] est la suivante :



```

1.  @model Exemple_04.Models.ViewModel01
2.  @{
3.      Layout = null;
4.      AjaxOptions ajaxOpts = new AjaxOptions
5.      {
6.          UpdateTargetId = "résultats",
7.          HttpMethod = "post",
8.          Url = Url.Action("Action01Post"),
9.          LoadingElementId = "loading",
10.         LoadingElementDuration = 1000
11.     };
12. }
13.
14. <!DOCTYPE html>
15.
16. <html lang="fr-FR">
17. <head>
18.     <meta name="viewport" content="width=device-width" />
19.     <title>Ajax-01</title>
20.     <link rel="stylesheet" href="~/Content/Site.css" />
21.     <script type="text/javascript" src="~/Scripts/jquery-1.8.2.min.js"></script>
22.     <script type="text/javascript" src="~/Scripts/jquery.validate.min.js"></script>
23.     <script type="text/javascript" src="~/Scripts/jquery.validate.unobtrusive.min.js"></script>
24.     <script type="text/javascript" src="~/Scripts/globalize.js"></script>

```

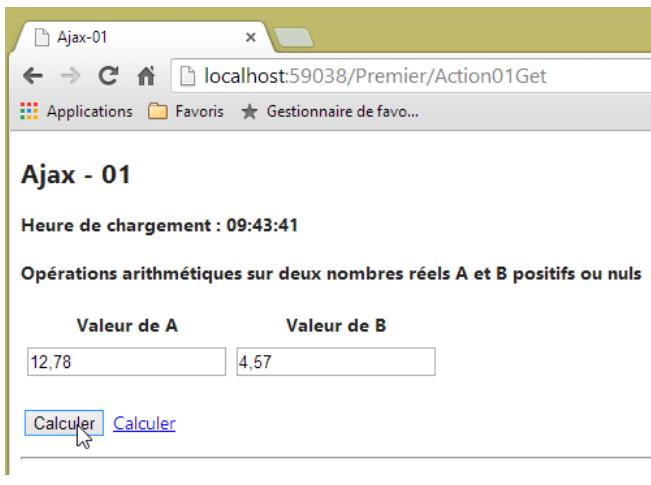
```

25.  <script type="text/javascript" src("~/Scripts/globalize/cultures/globalize.culture.fr-FR.js")></script>
26.  <script type="text/javascript" src "~/Scripts/globalize/cultures/globalize.culture.en-US.js"></script>
27.  <script type="text/javascript" src "~/Scripts/jquery.unobtrusive-ajax.js"></script>
28.  <script type="text/javascript" src "~/Scripts/myScripts-01.js"></script>
29. </head>
30. <body>
31.
32.  <h2>Ajax - 01</h2>
33.  <p><strong>Heure de chargement : @Model.HeureChangement</strong></p>
34.  <h4>Opérations arithmétiques sur deux nombres réels A et B positifs ou nuls</h4>
35.  @using (Ajax.BeginForm("Action01Post", null, ajaxOpts, new { id = "formulaire" }))
36.  {
37.    <table>
38.      <thead>
39.        <tr>
40.          <th>@Html.LabelFor(m => m.A)</th>
41.          <th>@Html.LabelFor(m => m.B)</th>
42.        </tr>
43.      </thead>
44.      <tbody>
45.        <tr>
46.          <td>@Html.TextBoxFor(m => m.A)</td>
47.          <td>@Html.TextBoxFor(m => m.B)</td>
48.        </tr>
49.        <tr>
50.          <td>@Html.ValidationMessageFor(m => m.A)</td>
51.          <td>@Html.ValidationMessageFor(m => m.B)</td>
52.        </tr>
53.      </tbody>
54.    </table>
55.    <p>
56.      <input type="submit" value="Calculer" />
57.      <img id="loading" style="display: none" src "~/Content/images/indicator.gif" />
58.      <a href="javascript:postForm()">Calculer</a>
59.    </p>
60.  }
61.  <hr />
62.  <div id="résultats" />
63. </body>
64. </html>

```

- ligne 1 : la vue a pour modèle un type [ViewModel01] ;
- ligne 21 : on a besoin de JQuery à la fois pour les validations et Ajax ;
- lignes 22-23 : les bibliothèques de validation ;
- lignes 24-26 : les bibliothèques de l'internationalisation ;
- ligne 27 : la bibliothèque Ajax ;
- ligne 28 : une bibliothèque Javascript locale ;
- ligne 33 : affichage de l'heure de chargement de la vue ;
- ligne 35 : un formulaire Ajax - nous reviendrons dessus ;
- lignes 40-41 : libellés des saisies des nombres A et B ;
- lignes 46-47 : saisies des nombres A et B ;
- lignes 50-51 : messages d'erreurs pour les saisies des nombres A et B ;
- ligne 56 : le bouton qui poste le formulaire. Celui-ci sera posté avec une requête Ajax ;
- ligne 57 : une image d'attente affichée lors de la requête Ajax ;
- ligne 58 : un lien pour poster le formulaire avec une requête Ajax ;
- ligne 62 : une balise <div> avec l'id [résultats]. C'est là que nous placerons le flux HTML renvoyé par le serveur web.

Cette vue affiche la page suivante :



Examinons maintenant le code qui ajaxifie le formulaire :

```

1. ...
2. @{
3.     Layout = null;
4.     AjaxOptions ajaxOpts = new AjaxOptions
5.     {
6.         UpdateTargetId = "résultats",
7.         HttpMethod = "post",
8.         Url = Url.Action("Action01Post"),
9.         LoadingElementId = "loading",
10.        LoadingElementDuration = 1000
11.    };
12. }
13.
14. ...
15. @using (Ajax.BeginForm("Action01Post", null, ajaxOpts, new { id = "formulaire" }))
16. {
17. ....
18.     <p>
19.         <input type="submit" value="Calculer" />
20.         
21.         <a href="javascript:postForm()">Calculer</a>
22.     </p>
23. }
24. ...
25. <div id="résultats" />

```

- ligne 15 : au lieu d'utiliser `[@Html.BeginForm]`, on utilise `[@Ajax.BeginForm]`. Cette méthode admet de nombreuses surcharges. Celle utilisée a la signature suivante :

```
Ajax.BeginForm(string ActionName, RouteValueDictionary routeValues, AjaxOptions ajaxOptions, IDictionary<string,object> htmlAttributes)
```

Nous utilisons ici les paramètres effectifs suivants :

**Action01Post** : le nom de l'action qui va traiter le POST du formulaire,  
**null** : il n'y a pas d'informations de route à donner,  
**ajaxOpts** : les options de l'appel Ajax. Elles ont été définies lignes 6-10,  
**new { id = "formulaire" }** : pour affecter l'attribut `[id='formulaire']` à la balise `<form>` générée ;

Les options Ajax utilisées sont les suivantes :

- ligne 8 : l'URL cible de la requête HTTP Ajax ;
- ligne 7 : méthode de la requête HTTP Ajax ;
- ligne 6 : **id** de la région de la page qui sera mise à jour par la réponse à la requête Ajax ;
- ligne 9 : **id** de la région de la page qui sera affichée pendant la requête Ajax – en général une image d'attente. Ici c'est la ligne 20 qui sera affichée. Elle contient une image animée symbolisant une attente. Au départ, cette image est cachée par le style `[display : none]` ;
- ligne 10 : temps d'attente en millisecondes avant que l'image animée ne soit affichée, ici 1 seconde.

Le code HTML généré par le formulaire Ajax est le suivant :

```
1.  <form action="/Premier/Action01Post" data-ajax="true" data-ajax-loading="#Loading" data-ajax-loading-duration="1000"
      data-ajax-method="post" data-ajax-mode="replace" data-ajax-update="#résultats" data-ajax-url="/Premier/Action01Post"
      id="formulaire" method="post">      <table>
2.  ...
3.    <p>
4.      <input type="submit" value="Calculer" />
5.      
6.      <a href="javascript:postForm()">Calculer</a>
7.    </p>
8.  </form>
9.  <hr />
10. <div id="résultats" />
```

- ligne 1 : la balise <form> générée. On notera les attributs [data-ajax-attr] qui reflètent les valeurs des champs de l'objet de type [AjaxOptions] qui a été associé à la requête Ajax. Ces attributs sont gérés par la bibliothèque Ajax. Sans eux, la balise <form> devient :

```
1.  <form action="/Premier/Action01Post" id="formulaire" method="post">
2.  ...
3.    <p>
4.      <input type="submit" value="Calculer" />
5.      
6.      <a href="javascript:postForm()">Calculer</a>
7.    </p>
8.  </form>
```

On a alors affaire à un formulaire HTML classique. C'est ce code qui sera exécuté si l'utilisateur désactive le Javascript sur son navigateur. Les lignes 5-6 sont alors inutilisées.

#### 1.7.3.3 L'action [Action01Post]

L'action [Action01Post] qui traite la requête HTTP Ajax est la suivante :

```
1.  [HttpPost]
2.  public PartialViewResult Action01Post(FormCollection postData, SessionModel session)
3.  {
4.      // simulation attente
5.      Thread.Sleep(2000);
6.      // instanciation modèle de l'action
7.      ViewModel01 modèle = new ViewModel01();
8.      // l'heure de calcul
9.      modèle.HeureCalcul = DateTime.Now.ToString("hh:mm:ss");
10.     // mise à jour du modèle
11.     TryUpdateModel(modèle, postData);
12.     if (!ModelState.IsValid)
13.     {
14.         // on retourne une erreur
15.         modèle.Erreur = getErrorMessagesFor(ModelState);
16.         return PartialView("Action01Error", modèle);
17.     }
18.     // une fois sur deux, on simule une erreur
19.     int val = session.Randomizer.Next(2);
20.     if (val == 0)
21.     {
22.         modèle.Erreur = "[erreur aléatoire]";
23.         return PartialView("Action01Error", modèle);
24.     }
25.     // calculs
26.     modèle.AplusB = string.Format("{0}", modèle.A + modèle.B);
27.     modèle.AmoinsB = string.Format("{0}", modèle.A - modèle.B);
28.     modèle.AmultipliéparB = string.Format("{0}", modèle.A * modèle.B);
29.     modèle.AdiviséparB = string.Format("{0}", modèle.A / modèle.B);
30.     // vue
31.     return PartialView("Action01Success", modèle);
32. }
```

- ligne 1 : l'action ne traite qu'un [POST] ;
- ligne 2 : elle admet pour modèle d'action :
  - [FormCollection postData] : l'ensemble des valeurs postées par la requête Ajax POST,
  - [SessionModel session] : les éléments de la session. On utilise ici une technique décrite au paragraphe 3, page 87 ;
- ligne 2 : l'action va rendre un fragment HTML et non une page HTML complète ;
- ligne 5 : artificiellement, on s'arrête deux secondes pour simuler une action Ajax longue ;
- ligne 7 : un modèle de type [ViewModel01] est instancié ;
- ligne 9 : l'heure de calcul est initialisée ;

- ligne 11 : on essaie de mettre à jour le modèle de type [ViewModel01] avec les valeurs postées. On rappelle qu'il y en a deux : les valeurs des nombres A et B ;
- ligne 12 : on vérifie le succès ou non de cette mise à jour ;
- ligne 15 : en cas d'erreur on renseigne le champ [Erreur] du modèle ;
- ligne 16 : on rend une vue partielle [Action01Error.cshtml] exploitant le modèle [ViewModel01] ;
- lignes 19-24 : une fois sur deux, on simule une erreur ;
- ligne 19 : on génère un nombre entier aléatoire dans l'intervalle [0,1]. Le générateur de nombres est pris dans la session ;
- ligne 20 : si la valeur générée est 0, on simule une erreur ;
- ligne 22 : le message d'erreur est mis dans le modèle ;
- ligne 23 : on rend une vue partielle [Action01Error.cshtml] exploitant le modèle [ViewModel01] ;
- lignes 26-29 : les calculs arithmétiques sur les nombres A et B sont faits et les résultats placés dans le modèle sous forme de chaînes de caractères ;
- ligne 31 : on rend une vue partielle [Action01Success.cshtml] exploitant le modèle [ViewModel01] ;

#### 1.7.3.4 La vue [Action01Error]

La vue [Action01Error.cshtml] est la suivante :

```
1. @model Exemple_04.Models.ViewModel01
2. <h4>Résultats</h4>
3. <p><strong>Heure de calcul : @Model.HeureCalcul</strong></p>
4. <p style="color: red;">Une erreur s'est produite : @Model.Erreur</p>
```

On rappelle que ce flux HTML partiel va être envoyé en réponse à la requête HTTP Ajax de type POST et être placé dans la page dans la région d'id [résultats]. Tous ces renseignements viennent de la configuration Ajax utilisée dans la page principale [Action01Get.cshtml] :

```
1. @model Exemple_04.Models.ViewModel01
2. @{
3.     Layout = null;
4.     AjaxOptions ajaxOpts = new AjaxOptions
5.     {
6.         UpdateTargetId = "résultats",
7.         HttpMethod = "post",
8.         Url = Url.Action("Action01Post"),
9.         LoadingElementId = "loading",
10.        LoadingElementDuration = 1000
11.    };
12. }
```

Voici un exemple de réponse avec erreur :

#### 1.7.3.5 La vue [Action01Success]

La vue [Action01Success.cshtml] est la suivante :

```
1. @model Exemple_04.Models.ViewModel01
2. <h4>Résultats</h4>
3. <p><strong>Heure de calcul : @Model.HeureCalcul</strong></p>
4. <p>A+B=@Model.AplusB</p>
5. <p>A-B=@Model.AmoinsB</p>
6. <p>A*B=@Model.AmultipliéparB</p>
7. <p>A/B=@Model.AdiviséparB</p>
```

Là encore, ce flux HTML partiel va être envoyé en réponse à la requête HTTP Ajax de type POST et être placé dans la page dans la région d'id [résultats] :

Ajax - 01

Heure de chargement : 01:41:39

Opérations arithmétiques sur deux nombres réels A et B positifs ou nuls

Valeur de A	Valeur de B
12,56	4,32

Calculer    [Calculer](#)

Résultats

Heure de calcul : 04:09:24

A+B=16,88  
A-B=8,24  
A\*B=54,2592  
A/B=2,90740740740741

#### 1.7.3.6 Gestion de la session

Nous avons vu que [Action01Post] utilisait la session. Le modèle de la session est le type [SessionModel] suivant :

```
1. using System;
2. namespace Exemple_03.Models
3. {
4.     public class SessionModel
5.     {
6.         public Random Randomizer { get; set; }
7.     }
8. }
```

La session est initialisée dans [Global.asax] :

```
1. // Session
2. protected void Session_Start()
3. {
4.     SessionModel sessionModel=new SessionModel();
5.     sessionModel.Randomizer=new Random(DateTime.Now.Millisecond);
6.     Session["data"] = sessionModel;
7. }
```

L'association de la session à un modèle est faite dans [Application\_Start] :

```

1.     protected void Application_Start()
2.     {
3.     ...
4.         // model binders
5.         ModelBinders.Binders.Add(typeof(SessionModel), new SessionModelBinder());
6.     }

```

La classe [SessionModelBinder] a été décrite page 88.

#### 1.7.3.7 Gestion de l'image d'attente

```

1. @model Exemple_04.Models.ViewModel01
2. @{
3.     Layout = null;
4.     AjaxOptions ajaxOpts = new AjaxOptions
5.     {
6.     ...
7.         LoadingElementId = "loading",
8.         LoadingElementDuration = 1000
9.     };
10. }
11.
12. ...
13. <body>
14.
15. ...
16. @using (Ajax.BeginForm("Action01Post", null, ajaxOpts, new { id = "formulaire" }))
17. {
18. ...
19.     <p>
20.         <input type="submit" value="Calculer" />
21.         <img id="loading" style="display: none" src "~/Content/images/indicator.gif" />
22.         <a href="javascript:postForm()">Calculer</a>
23.     </p>
24. }
25. ...

```

Lorsque la requête Ajax démarre, la région d'id [loading] ligne 7 est affichée au bout d'une seconde [ligne 8]. Cette région est l'image de la ligne 21 initialement cachée. Cela donne l'interface suivante :



#### 1.7.3.8 Gestion du lien [Calculer]

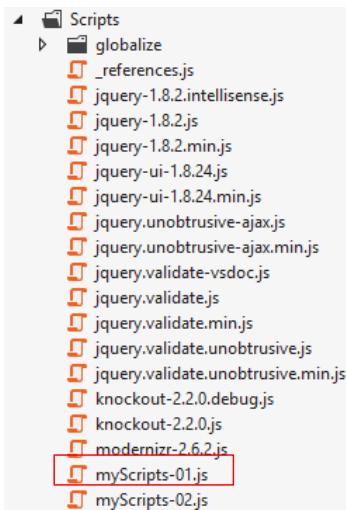
Examinons le lien [Calculer] de la page principale [Action01Get.cshtml] :

```

1. <head>
2.     <meta name="viewport" content="width=device-width" />
3.     <title>Ajax-01</title>
4.     ...
5.     <script type="text/javascript" src="~/Scripts/myScripts-01.js"></script>
6. </head>
7. <body>
8.
9.     <h2>Ajax - 01</h2>
10.    <p><strong>Heure de chargement : @Model.HeureChargement</strong></p>
11.    <h4>Opérations arithmétiques sur deux nombres réels A et B positifs ou nuls</h4>
12.    @using (Ajax.BeginForm("Action01Post", null, ajaxOpts, new { id = "formulaire" }))
13.    {
14.    ...
15.        <p>
16.            <input type="submit" value="Calculer" />
17.            <img id="loading" style="display: none" src "~/Content/images/indicator.gif" />
18.            <a href="javascript:postForm()">Calculer</a>
19.        </p>
20.    }
21.    <hr />
22. <div id="résultats" />

```

- ligne 18 : un clic sur le lien [Calculer] provoque l'exécution de la fonction JS [postForm]. Celle-ci est définie dans le fichier [myScripts-01.js] de la ligne 5. Ce script est le suivant :



```

1. function postForm() {
2.     // on fait un appel Ajax à la main avec JQuery
3.     var loading = $("#loading");
4.     var formulaire = $("#formulaire");
5.     var résultats = $("#résultats");
6.     $.ajax({
7.         url: '/Premier/Action01Post',
8.         type: 'POST',
9.         data: formulaire.serialize(),
10.        dataType: 'html',
11.        begin: loading.show(),
12.        success: function (data) {
13.            loading.hide()
14.            résultats.html(data);
15.        }
16.    })
17. }
18.
19. // http://blog.instance-factory.com/?p=268
20. $.validator.methods.number = function (value, element) {
21.     return this.optional(element) ||
22.           !isNaN(Globalize.parseFloat(value));
23. }
24.
25. $.validator.methods.date = function (value, element) {
26.     return this.optional(element) ||
27.           !isNaN(Globalize.parseDate(value));
28. }
29.
30. jQuery.extend(jQuery.validator.methods, {
31.     range: function (value, element, param) {
32.         //Use the Globalization plugin to parse the value
33.         var val = Globalize.parseFloat(value);
34.         return this.optional(element) ||
35.             val >= param[0] && val <= param[1];
36.     }
37. });

```

Les fonctions des lignes 19-37 ont été déjà rencontrées au paragraphe 1.6.1, page 156. Elles gèrent l'internationalisation des pages. Nous ne revenons pas dessus. Lignes 1-17, nous faisons à la main l'appel Ajax qui dans le cas du bouton [Calculer] était fait par la bibliothèque Ajax associée au projet. Pour cela, nous utilisons la bibliothèque JQuery associée au projet.

- ligne 3 : une référence sur le composant d'**id** [loading]. `[$("#loading")]` ramène la collection des éléments d'**id** [loading]. Il n'y en a qu'un ;
- ligne 4 : une référence sur le composant d'**id** [formulaire] ;
- ligne 5 : une référence sur le composant d'**id** [résultats] ;
- ligne 6 : l'appel Ajax avec ses options ;
- ligne 7 : l'URL cible de l'appel Ajax ;
- ligne 8 : la méthode HTTP utilisée ;
- ligne 9 : les données postées. `[formulaire.serialize]` crée la chaîne `[A=val1&B=val2]` du POST du formulaire d'**id** [formulaire] ;
- ligne 10 : le type de données attendu en retour. On sait que le serveur va renvoyer un flux HTML ;

- ligne 11 : la méthode à exécuter lorsque la requête démarre. Ici, on indique qu'il faut afficher le composant d'**id** [loading]. C'est l'image animée d'attente ;
- ligne 12 : la méthode à exécuter en cas de succès de la requête Ajax. Le paramètre [data] est la réponse complète du serveur. On sait que c'est un flux HTML ;
- ligne 13 : on cache le signal d'attente ;
- ligne 14 : on met à jour le composant d'**id** [résultats] avec le HTML du paramètre [data].

Le lecteur est invité à tester le lien [Calculer]. Il fonctionne comme le bouton [Calculer] à une anomalie près. Une fois qu'on a utilisé ce lien, on peut alors poster des valeurs de A et B invalides :

The screenshot shows a browser window titled "Ajax-01" with the URL "localhost:59038/Premier/Action01Get". The page content is as follows:

**Ajax - 01**

Heure de chargement : 12:52:25

Opérations arithmétiques sur deux nombres réels A et B positifs ou nuls

Valeur de A	Valeur de B
0x <b>1</b>	0x <b>2</b>

Le champ Valeur de A doit être un nombre. Le champ Valeur de B doit être un nombre.

**Calculer** **Calculer** **3**

---

**Résultats**

Heure de calcul : 12:52:33 **4**

Une erreur s'est produite : [La valeur « 0x » n'est pas valide pour Valeur de A.][La valeur « 0x » n'est pas valide pour Valeur de B.]

- en [1] et [2], on a entré des valeurs invalides. Elles sont signalées par les validateurs côté client ;
- en [3], on a cliqué sur le lien [Calculer] ;
- en [4], il y a eu un [POST] puisqu'on obtient la réponse [4].

Lorsque les valeurs sont invalides et qu'on clique sur le bouton [Calculer], le [POST] vers le serveur n'a pas lieu. Dans le même cas, avec le lien [Calculer] le [POST] vers le serveur a lieu. Il y a donc un comportement du bouton [Calculer] qu'on n'a pas su reproduire avec le lien [Calculer]. Plutôt que d'essayer de résoudre ce problème maintenant, nous le laissons pour un exemple ultérieur qui illustrera également un autre problème de validation côté client.

## 1.7.4 Mise à jour d'une page HTML avec un flux JSON

Dans l'exemple précédent, le serveur web répondait à la requête HTTP Ajax par un flux HTML. Dans ce flux, il y avait des données accompagnées par du formatage HTML. On se propose de reprendre l'exemple précédent avec cette fois-ci des réponses JSON (JavaScript Object Notation) ne contenant que les données. L'intérêt est qu'on transmet ainsi moins d'octets.

### 1.7.4.1 L'action [Action02Get]

L'action [Action02Get] sera le point d'entrée de la nouvelle application. Son code est le suivant :

```

1. @model Exemple_04.Models.ViewModel02
2. @{
3.     Layout = null;
4.     AjaxOptions ajaxOpts = new AjaxOptions
5.     {
6.         HttpMethod = "post",
7.         Url = Url.Action("Action02Post"),
8.         LoadingElementId = "loading",
9.         LoadingElementDuration = 1000,
10.        OnBegin = "OnBegin",
11.        OnFailure = "OnFailure",
12.        OnSuccess = "OnSuccess",
13.        OnComplete = "OnComplete"

```

```

14.    };
15. }
16.
17. <!DOCTYPE html>
18.
19. <html lang="fr-FR">
20. <head>
21.   <meta name="viewport" content="width=device-width" />
22.   <title>Ajax-02</title>
23. ....
24.   <script type="text/javascript" src="/~/Scripts/myScripts-02.js"></script>
25. </head>
26. <body>
27.   <h2>Ajax - 02</h2>
28.   <p><strong>Heure de chargement : @Model.HeureChargement</strong></p>
29.   <h4>Opérations arithmétiques sur deux nombres réels A et B positifs ou nuls</h4>
30.   @using (Ajax.BeginForm("Action02Post", null, ajaxOpts, new { id = "formulaire" }))
31.   {
32. ...
33.     <p>
34.       <input type="submit" value="Calculer" />
35.       
36.       <a href="javascript:postForm()">Calculer</a>
37.     </p>
38.   }
39.   <hr />
40.   <div id="entete">
41.     <h4>Résultats</h4>
42.     <p><strong>Heure de calcul : <span id="heureCalcul"/></strong></p>
43.   </div>
44.   <div id="résultats">
45.     <p>A+B=<span id="AplusB"/></p>
46.     <p>A-B=<span id="AmoinsB"/></p>
47.     <p>A*B=<span id="AmultipliéparB"/></p>
48.     <p>A/B=<span id="AdiviséparB"/></p>
49.   </div>
50.   <div id="erreur">
51.     <p style="color: red;">Une erreur s'est produite : <span id="msg"/></p>
52.   </div>
53. </body>
54. </html>

```

- lignes 4-14 : les options de l'appel Ajax ;
- ligne 10 : la fonction JS à exécuter au démarrage de la requête. Cette fonction est définie dans le fichier JS référencé à la ligne 24 ;
- ligne 11 : la fonction JS à exécuter en cas d'échec de la requête ;
- ligne 12 : la fonction JS à exécuter en cas de réussite de la requête ;
- ligne 13 : la fonction JS à exécuter après que la requête Ajax a obtenu son résultat (échec ou réussite) ;
- lignes 40-43 : une région d'id [entete] ;
- lignes 44-49 : une région d'id [résultats]. Elle affichera les résultats des quatre opérations arithmétiques ;
- lignes 50-52 : une région d'id [erreur]. Elle affichera un éventuel message d'erreur.

#### 1.7.4.2 L'action [Action02Post]

La requête Ajax est traitée par l'action [Action02Post] suivante :

```

1.  [HttpPost]
2.  public JsonResult Action02Post(FormCollection postData, SessionModel session)
3.  {
4.      // simulation attente
5.      Thread.Sleep(2000);
6.      // validation modèle
7.      ViewModel02 modèle = new ViewModel02();
8.      // les heures de chargement et de calcul
9.      string HeureChargement = DateTime.Now.ToString("hh:mm:ss");
10.     string HeureCalcul = DateTime.Now.ToString("hh:mm:ss");
11.     // mise à jour du modèle
12.     TryUpdateModel(modèle, postData);
13.     if (!ModelState.IsValid)
14.     {
15.         // on retourne une erreur
16.         return Json(new { Erreur = getErrorMessagesFor(ModelState), HeureCalcul = HeureCalcul });
17.     }
18.     // une fois sur deux, on simule une erreur
19.     int val = session.Randomizer.Next(2);
20.     if (val == 0)
21.     {
22.         // on retourne une erreur
23.         return Json(new { Erreur = "[erreur aléatoire]", HeureCalcul = HeureCalcul });
24.     }
25.     // calculs
26.     string AplusB = string.Format("{0}", modèle.A + modèle.B);

```

```

27.     string AmoinsB = string.Format("{0}", modèle.A - modèle.B);
28.     string AmultipliéparB = string.Format("{0}", modèle.A * modèle.B);
29.     string AdiviséparB = string.Format("{0}", modèle.A / modèle.B);
30.     // on retourne les résultats
31.     return Json(new { Erreur = "", AplusB = AplusB, AmoinsB = AmoinsB, AmultipliéparB = AmultipliéparB, AdiviséparB = AdiviséparB, HeureCalcul = HeureCalcul });
32.   }

```

- ligne 2 : la méthode rend un type [JsonResult] ç-à-d un texte au format JSON ;
- ligne 16 : les informations sont rendues sous la forme d'une instance de classe anonyme sérialisée en JSON. La méthode [getErrorMessagesFor] a déjà été présentée page 71. La chaîne JSON envoyée au navigateur aura la forme suivante :

```
{"Erreur": "[erreur aléatoire]", "HeureCalcul": "05:31:37"}
```

- ligne 31 : même démarche pour les résultats arithmétiques. Cette fois, la chaîne JSON envoyée au navigateur aura la forme suivante :

```
{"Erreur": "", "AplusB": "4", "AmoinsB": "-2", "AmultipliéparB": "3", "AdiviséparB": "0,33333333333333", "HeureCalcul": "05:52:17"}
```

#### 1.7.4.3 Le code Javascript côté client

Rappelons la configuration de l'appel Ajax dans la page HTML envoyée au navigateur client :

```

1.   AjaxOptions ajaxOpts = new AjaxOptions
2.   {
3.     HttpMethod = "post",
4.     Url = Url.Action("Action02Post"),
5.     LoadingElementId = "loading",
6.     LoadingElementDuration = 1000,
7.     OnBegin = "OnBegin",
8.     OnFailure = "OnFailure",
9.     OnSuccess = "OnSuccess",
10.    OnComplete = "OnComplete"
11.  };

```

Les fonctions JS référencées aux lignes 7-10 (à droite du signe =) sont définies dans le fichier [myScripts-02.js] suivant :

```

1. // données globales
2. var entete;
3. var loading;
4. var résultats;
5. var erreur;
6. var heureCalcul;
7. var msg;
8. var AplusB;
9. var AmoinsB;
10. var AmultipliéparB;
11. var AdiviséparB;
12. var formulaire;
13. ...
14. function postForm() {
15. ...
16. }
17.
18. // au chargement du document
19. $(document).ready(function () {
20.   formulaire = $("#formulaire");
21.   entete = $("#entete");
22.   loading = $("#loading");
23.   erreur = $("#erreur");
24.   résultats = $("#résultats");
25.   heureCalcul = $("#heureCalcul");
26.   msg = $("#msg");
27.   AplusB = $("#AplusB");
28.   AmoinsB = $("#AmoinsB");
29.   AmultipliéparB = $("#AmultipliéparB");
30.   AdiviséparB = $("#AdiviséparB");
31.
32.   // on cache certains éléments de la page
33.   entete.hide();
34.   résultats.hide();
35.   erreur.hide();
36. });
37.
38. // démarrage
39. function OnBegin() {
40. ...
41. }
42.
43. // fin de la requête

```

```

44. function OnComplete() {
45. ...
46. }
47.
48. // réussite
49. function OnSuccess(data) {
50. ....
51. }
52.
53. // erreur
54. function onFailure(request, error) {
55. ...
56. }

```

- ligne 19 : la fonction JS exécutée à la fin du chargement de la page dans le navigateur ;
- lignes 20-30 : on récupère les références de tous les composants de la page qui nous intéressent. La recherche d'un composant dans une page a un coût et il est préférable de ne la faire qu'une fois ;
- lignes 33-35 : les composants [entete], [résultats] et [loading] sont cachés ;

Au démarrage de la requête Ajax, la fonction suivante est exécutée :

```

1. // démarrage
2. function OnBegin() {
3.   // signal d'attente allumé
4.   loading.show();
5.   // on cache certains éléments de la page
6.   entete.hide();
7.   résultats.hide();
8.   erreur.hide();
9. }

```

- ligne 4 : le composant [loading] est affiché. C'est l'image animée ;
- lignes 6-8 : les composants [entete], [résultats] et [erreur] sont cachés ;

Si la requête Ajax réussit, on exécute le code JS suivant :

```

1. // réussite
2. function OnSuccess(data) {
3.   // affichage résultats
4.   heureCalcul.text(data.HeureCalcul);
5.   entete.show();
6.   if (data.Erreur != '') {
7.     msg.text(data.Erreur);
8.     erreur.show();
9.     return;
10.  }
11.  // pas d'erreur
12.  AplusB.text(data.AplusB);
13.  AmoinsB.text(data.AmoinsB);
14.  AmultipliéparB.text(data.AmultipliéparB);
15.  AdiviséparB.text(data.AdiviséparB);
16.  résultats.show();
17. }

```

Pour comprendre ce code il faut se rappeler les deux textes JSON susceptibles d'être envoyés en réponse au navigateur :

```
{"Erreur":"[erreur aléatoire]","HeureCalcul":"05:31:37"}
```

en cas d'erreur sinon la chaîne :

```
{"Erreur":"","AplusB":"4","AmoinsB":"-2","AmultipliéparB":"3","AdiviséparB":"0,3333333333333333","HeureCalcul":"05:52:17"}
```

Si on appelle [data] cette chaîne, la valeur du champ [Erreur] est obtenue par la notation [**data.Erreur**] ou [**data["Erreur"]**], au choix. Idem pour les autres champs de la chaîne JSON. Par ailleurs, pour affecter un texte non formaté à un composant d'**id** X, on écrit [**X.text(chaine)**]. Revenons au code de la fonction [OnSuccess] :

- ligne 2 : [data] est la chaîne JSON reçue ;
- ligne 4 : le composant [heureCalcul] reçoit sa valeur ;
- ligne 5 : le composant [entete] est affiché ;
- ligne 6 : test du champ [Erreur] de la chaîne JSON ;
- ligne 7 : le composant [msg] reçoit sa valeur ;
- ligne 8 : le composant [erreur] est affiché ;
- ligne 9 : c'est terminé pour le cas d'erreur ;
- ligne 12 : le composant [AplusB] reçoit sa valeur ;

- ligne 13 : le composant [AmoinsB] reçoit sa valeur ;
- ligne 14 : le composant [AmultipliéparB] reçoit sa valeur ;
- ligne 15 : le composant [AdiviséparB] reçoit sa valeur ;
- ligne 16 : le composant [résultats] est affiché.

La fonction [OnFailure] sera exécutée en cas d'échec de la requête HTTP Ajax. Cet échec est mesuré avec le code HTTP renvoyé par le serveur. Le code 500 [Internal Server Error] par exemple indique que le serveur n'a pu exécuter la requête. La fonction [OnFailure] est la suivante :

```
1. // erreur
2. function OnFailure(request, error) {
3.   alert("L'erreur suivante s'est produite :" + error);
4. }
```

On se contente d'afficher une boîte de dialogue avec l'erreur qui s'est produite. Dans la pratique, il faudrait être plus précis. Nous allons bientôt proposer une autre solution.

Enfin la fonction [OnComplete] est exécutée lorsque la requête est terminée que ce soit sur une réussite ou un échec.

```
1. // fin de la requête
2. function OnComplete() {
3.   // signal d'attente éteint
4.   loading.hide();
5. }
```

On rappelle ici que c'est la configuration de l'appel Ajax dans la vue [Action02Get.cshtml] qui fait que ces différentes fonctions sont appelées :

```
1. AjaxOptions ajaxOpts = new AjaxOptions
2. {
3. ...
4.   OnBegin = "OnBegin",
5.   OnFailure = "OnFailure",
6.   OnSuccess = "OnSuccess",
7.   OnComplete = "OnComplete"
8. };
```

#### 1.7.4.4 Le lien [Calculer]

Le code HTML du lien [Calculer] dans la vue [Action02Get.cshtml] est le suivant :

```
<a href="javascript:postForm()">Calculer</a>
```

La fonction JS [postForm] est trouvé dans le fichier importé [myScripts-02.js] :

```
<script type="text/javascript" src "~/Scripts/myScripts-02.js"></script>
```

Son code est le suivant :

```
1. function postForm() {
2.   // on fait un appel Ajax à la main avec JQuery
3.   $.ajax({
4.     url: '/Premier/Action02Post',
5.     type: 'POST',
6.     data: formulaire.serialize(),
7.     dataType: 'json',
8.     beforeSend: OnBegin,
9.     success: OnSuccess,
10.    error: OnFailure,
11.    complete: OnComplete
12.  })
13. }
```

Nous avons déjà rencontré un code similaire page 193.

- ligne 4 : URL cible de l'appel Ajax ;
- ligne 5 : commande HTTP utilisée par l'appel Ajax ;
- ligne 6 : valeurs postées. Elles sont le résultat de la sérialisation des valeurs du formulaire. Celui-ci désigné par l'**id** [formulaire] est référencé par la variable [formulaire]. [data] sera une chaîne de caractères de la forme [A=val1&B=val2] ;
- ligne 7 : type de formatage de la réponse attendue. C'est une chaîne JSON ;
- ligne 8 : fonction JS à exécuter au démarrage de l'appel Ajax ;
- ligne 9 : fonction JS à exécuter si l'appel Ajax réussit ;

- ligne 10 : fonction JS à exécuter si l'appel Ajax échoue ;
- ligne 11 : fonction JS à exécuter une fois reçue la réponse du serveur, quelle que soit celle-ci, réussite ou erreur.

Revenons sur la fonction Javascript qui gère le cas où l'appel Ajax échoue (ligne 10). L'appel Ajax échoue dans diverses situations, par exemple lorsque le serveur renvoie un code d'erreur tel que [403 Forbidden], [404 Not Found], [500 Internal Server Error] [301 Moved Permanently], ...

Dans l'exemple de la page 198, la fonction [OnFailure] est la suivante :

```
1. // erreur
2. function OnFailure(request, error) {
3.   alert("L'erreur suivante s'est produite :" + error);
4. }
```

Généralement, l'affichage de l'objet [error] n'apporte aucune information intéressante. Si on utilise un appel Ajax fait avec JQuery, on peut utiliser la méthode [OnFailure] suivante :

```
1. // erreur
2. function OnFailure(jqXHR) {
3.   alert("Erreur : " + jqXHR.status + " " + jqXHR.statusText);
4.   msg.html(jqXHR.responseText);
5.   erreur.show();
6. }
```

L'objet JQuery [**jqXHR**] a parmi ses propriétés les suivantes :

- **responseText** : le texte de la réponse du serveur ;
- **status** : le code d'erreur retourné par le serveur ;
- **statusText** : le texte associé à ce code d'erreur.
- ligne 3 : on affiche le code d'erreur et le libellé qui va avec ;
- ligne 4 : on met la réponse HTML du serveur dans le composant d'**id** [msg] ;
- ligne 5 : on affiche la région d'**id** [erreur].

Pour tester cette fonction d'erreur, nous allons créer artificiellement une exception dans l'action [Action02Post] :

```
1.      [HttpPost]
2.      public JsonResult Action02Post(FormCollection postData, SessionModel session)
3.      {
4.        // une exception artificielle pour tester la fonction d'erreur de l'appel Ajax
5.        throw new Exception();
6.        // simulation attente
7.        Thread.Sleep(2000);
8.        // validation modèle
9.      ...
```

La ligne 5 lance une exception. Maintenant testons l'application :

On obtient la réponse suivante [1] et [2] :

The screenshot shows a web browser window titled "Ajax-02". The address bar indicates the URL is "localhost:59038/Premier/Action02Get". The main content area displays the following text:

**Ajax - 02**  
Heure de chargement : 08:08:52  
Opérations arithmétiques sur deux nombres réels A et B positifs ou nuls

Valeur de A      Valeur de B  
      
Calculer    [Calculer](#)

Une erreur s'est produite :

**Erreur du serveur dans l'application '/'.** 2

*Une exception de type 'System.Exception' a été levée.*

Description : Une exception non gérée s'est produite au moment de l'exécution de la requête Web actuelle. Contrôlez la trace de la pile po

Détails de l'exception: System.Exception: Une exception de type 'System.Exception' a été levée.

Erreur source:

```
Ligne 63 :     public JsonResult Action02Post/FormCollection postData, SessionMode
Ligne 64 : {
Ligne 65 :     throw new Exception();
Ligne 66 :     // simulation attente
Ligne 67 :     Thread.Sleep(2000);
```

Fichier source : d:\data\istia-1314\aspnet\dvp\Exemples\Exemple-04\Controllers\PremierController.cs   Ligne : 65

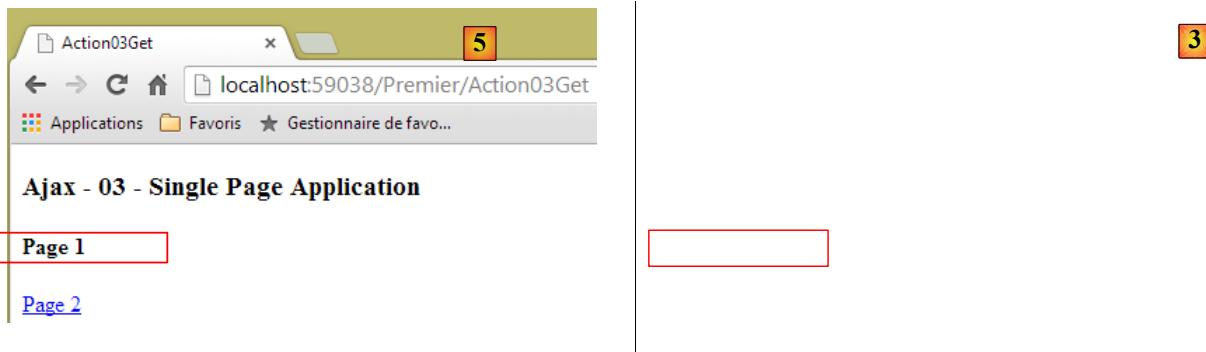
La réponse du serveur nous permet de voir à quelle ligne du code serveur s'est produite l'erreur. C'est souvent une information utile à connaître. Dorénavant nous utiliserons cette technique pour gérer les erreurs des appels Ajax.

### 1.7.5 Application web à page unique

La technologie Ajax permet de construire des applications à **page unique** :

- la première page est issue d'une requête classique d'un navigateur ;
- les pages suivantes sont obtenues avec des appels Ajax. Aussi, au final le navigateur ne change jamais d'URL et ne charge jamais de nouvelle page. On appelle ce type d'application, **Application à Page Unique** (APU) ou en anglais **Single Page Application** (SPA).

Voici un exemple basique d'une telle application. La nouvelle application aura deux vues :



- en [1], l'action [Action03Get] nous permet d'avoir la première page, la **page 1** ;
- en [2], un lien nous permet de passer à la **page 2** grâce à un appel Ajax ;
- en [3], l'URL n'a pas changé. La page présentée est la **page 2** ;
- en [4], un lien nous permet de revenir à la **page 1** grâce à un appel Ajax ;
- en [5], l'URL n'a pas changé. La page présentée est la **page 1**.

Le code de l'action [Action03Get] est le suivant :

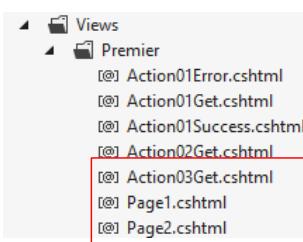
```

1.     [HttpGet]
2.     public ViewResult Action03Get()
3.     {
4.         return View();
5.     }

```

- ligne 4 : la vue [Action03Get.cshtml] est affichée.

La vue [Action03Get.cshtml] est la suivante :



```

1. @{
2.     Layout = null;
3. }
4.
5. <!DOCTYPE html>
6.
7. <html>
8. <head>
9.     <meta name="viewport" content="width=device-width" />
10.    <title>Action03Get</title>
11.    <script type="text/javascript" src="~/Scripts/jquery-1.8.2.min.js"></script>
12.    <script type="text/javascript" src="~/Scripts/jquery.unobtrusive-ajax.min.js"></script>
13. </head>
14. <body>
15.     <h3>Ajax - 03 - Single Page Application</h3>
16.     <div id="content">
17.         @Html.Partial("Page1")
18.     </div>
19. </body>
20. </html>

```

- lignes 16-18 : un élément d'**id** [content]. C'est dans cet élément que les différentes pages vont s'afficher ;
- ligne 17 : par défaut c'est la page [Page1.cshtml] qui va d'abord s'afficher.

La page [Page1.cshtml] est la suivante :

```

1.  <h4>Page 1</h4>
2.  <p>
3.      @Ajax.ActionLink("Page 2", "Action04", new { Page = 2 }, new AjaxOptions() { UpdateTargetId = "content" })
4.  </p>

```

- ligne 1 : le titre de la page pour la différencier de la page 2 ;
- ligne 3 : un lien Ajax avec les paramètres suivants :
  - le libellé du lien [Page 2] ;
  - l'action cible du lien [Action04] ;
  - les paramètres de l'URL demandée. Celle-ci sera [/Premier/Action04?Page=2] ;
- les options de l'appel Ajax. Ici, seulement l'**id** de la région à mettre à jour avec la réponse du serveur. Pour les autres options, des valeurs par défaut sont alors utilisées lorsqu'elles existent. La méthode HTTP par défaut est GET.

Voyons ce qui se passe lorsque le lien est cliqué. L'URL [/Premier/Action04?Page=2] est demandée avec un GET. L'action [Action04] est alors exécutée :

```

1.  [HttpGet]
2.  public PartialViewResult Action04(string page = "1")
3.  {
4.      string vue = "Page1";
5.      if (page == "2")
6.      {
7.          vue = "Page2";
8.      }
9.      return PartialView(vue);
10. }

```

- ligne 2 : l'action rend un flux HTML partiel ;
- ligne 2 : l'action a pour modèle la chaîne [page]. Or on sait que l'URL a cette information : [/Premier/Action04?Page=2]. On rappelle que le modèle est insensible à la casse ;
- lignes 4-8 : [vue] va recevoir la valeur [Page2] ;
- ligne 9 : la vue partielle [Page2.cshtml] est rendue.

La vue partielle [Page2.cshtml] est la suivante :

```

1.  <h4>Page 2</h4>
2.  <p>
3.      @Ajax.ActionLink("Page 1", "Action04", new { Page = 1 }, new AjaxOptions() { UpdateTargetId = "content" })
4.  </p>

```

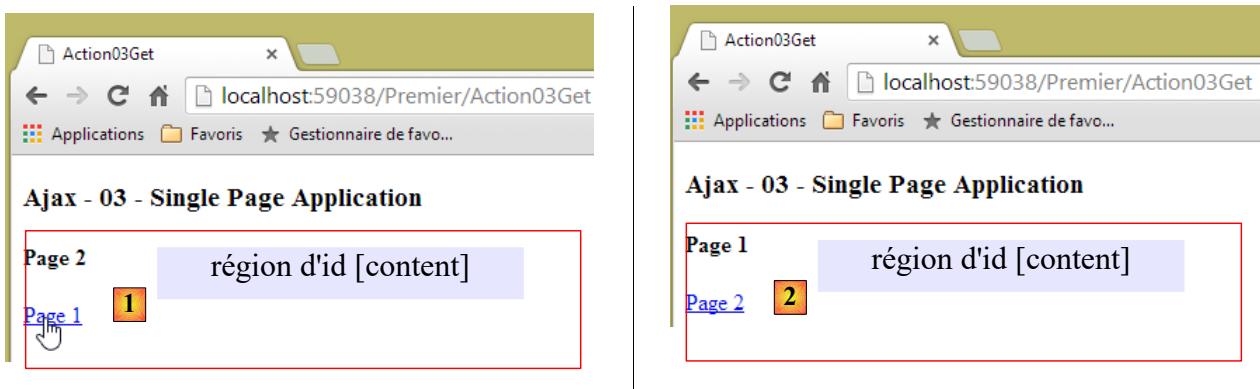
Le serveur renvoie donc le flux HTML ci-dessus comme réponse à l'appel Ajax GET [/Premier/Action04?Page=2]. On se rappelle que cet appel Ajax utilise cette réponse pour mettre à jour la région d'id [content] (ligne 3 ci-dessous) :

```

1.  <h4>Page 1</h4>
2.  <p>
3.      @Ajax.ActionLink("Page 2", "Action04", new { Page = 2 }, new AjaxOptions() { UpdateTargetId = "content" })
4.  </p>

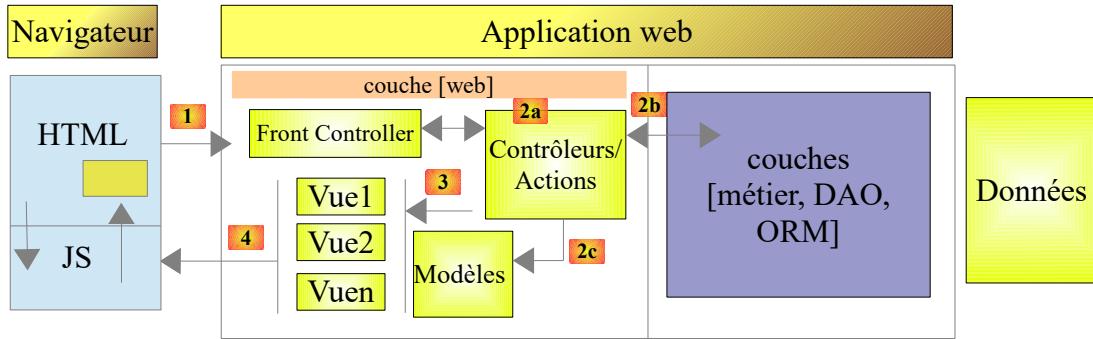
```

Ceci provoque le nouvel affichage suivant [1] :

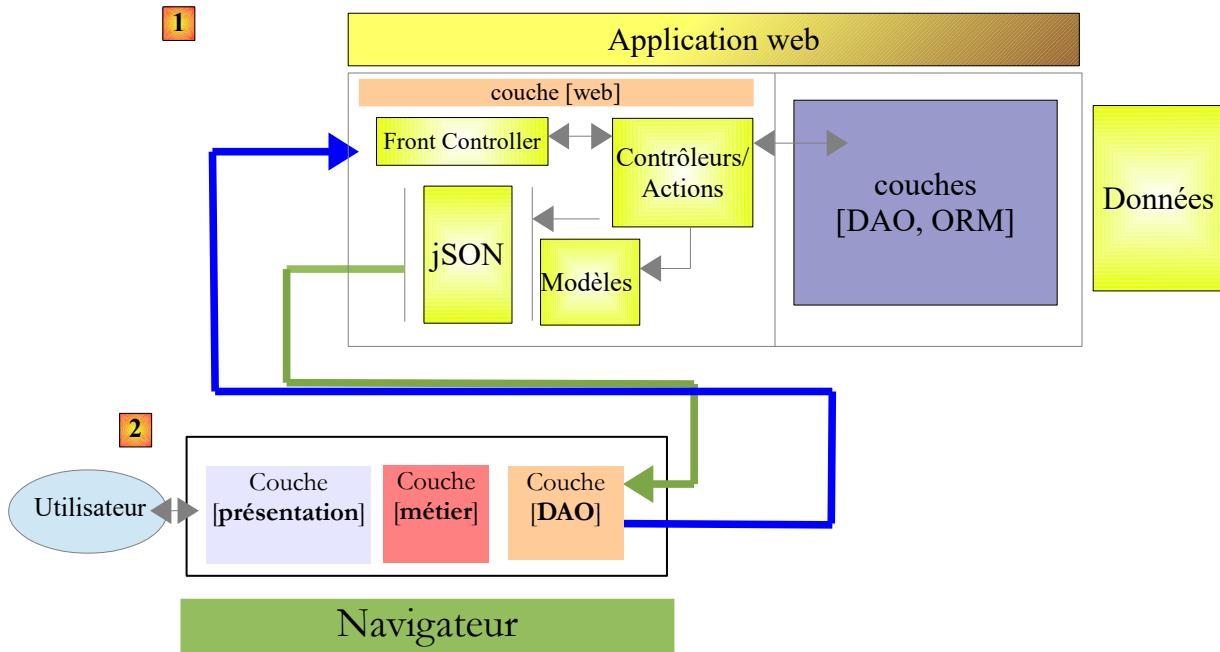


En suivant le même raisonnement, on voit que le clic sur le lien [Page 1] de [1] va provoquer l'affichage [2].

Revenons au schéma général d'une application ASP.NET MVC :



Grâce au Javascript embarqué dans les pages HTML et exécuté dans le navigateur, on peut déporter du code sur le navigateur et aboutir à l'architecture suivante :



- en [1], la couche Web ASP.NET MVC est devenue une **interface web** d'accès aux données, généralement logées dans une base de données. Les vues délivrées ne contiennent que des données et aucun habillage HTML, par exemple des flux XML ou JSON ;
- en [2] : le navigateur affiche des vues statiques (çà-d non générées dynamiquement) délivrées par un serveur web qui peut être ou non sur la même machine que le serveur [1]. Ces vues statiques sont ensuite enrichies par les données obtenues par le Javascript auprès de l'interface web [1] ;
- le code Javascript embarqué dans les pages HTML peut être structuré en couches :
  - la couche [présentation] s'occupe des interactions avec l'utilisateur,
  - la couche [DAO] s'occupe de l'accès aux données via le serveur web [1] ,
  - la couche [métier] correspond à la couche [métier] qui auparavant était sur le serveur [1] et qui a été déportée sur le navigateur [2] ;

L'intérêt de cette architecture est qu'elle met en jeu des compétences différentes :

- le code du serveur web [1] nécessite des compétences .NET mais pas de compétences Javascript, HTML, CSS ;
- le code embarqué dans le navigateur [2] nécessite des compétences Javascript, HTML, CSS mais est indifférent à la technologie du serveur web [1].

Ainsi, cette architecture facilite-t-elle le travail en parallèle d'équipes aux compétences différentes. Elle s'applique particulièrement bien aux **Applications à Page Unique**.

## 1.7.6 Application web à page unique et validation côté client

Nous avons évoqué précédemment une anomalie sur l'exemple Ajax-01 (page 194). Nous en rappelons le contexte :

Ajax - 01

Heure de chargement : 12:52:25

Opérations arithmétiques sur deux nombres réels A et B positifs ou nuls

Valeur de A      Valeur de B

0x **1**      0x **2**

Le champ Valeur de A doit être un nombre. Le champ Valeur de B doit être un nombre.

Calculer    Calculer **3**

Résultats

Heure de calcul : 12:52:33 **4**

Une erreur s'est produite : [La valeur « 0x » n'est pas valide pour Valeur de A.][La valeur « 0x » n'est pas valide pour Valeur de B.]

- en [1] et [2], on a entré des valeurs invalides. Elles sont signalées par les validateurs côté client ;
- en [3], on a cliqué sur le lien [Calculer] ;
- en [4], il y a eu un [POST] puisqu'on obtient la réponse [4].

Lorsque les valeurs sont invalides et qu'on clique sur le bouton [Calculer], le [POST] vers le serveur n'a pas lieu. Dans le même cas, avec le lien [Calculer] le [POST] vers le serveur a lieu. Il y a donc un comportement du bouton [Calculer] qu'on n'a pas su reproduire avec le lien [Calculer].

Nous allons reprendre cet exemple dans un nouveau contexte : l'application aura plusieurs vues et sera du type [Application à Page Unique] que nous venons de décrire.

### 1.7.6.1 Les vues de l'exemple

L'exemple a plusieurs vues :

Ajax - 05, Page unique - Validation formulaire côté client

Heure de chargement : 01:15:12

Opérations arithmétiques sur deux nombres réels A et B positifs ou nuls

Valeur de A	Valeur de B
4,5	6,7

[Calculer](#) [Effacer](#)

**1**

Heure de chargement : 01:15:12

Opérations arithmétiques sur deux nombres réels A et B positifs ou nuls

Heure de calcul : 01:15:40

A=4,5  
B=6,7

**3**

**Les erreurs suivantes se sont produites**

- [erreur aléatoire]

[Retour aux saisies](#)

- en [1], la vue [Action05Get] ;
- en [2], la vue partielle [Formulaire05] ;
- en [3], la vue partielle [Failure05] ;

Ajax - 05, Page unique - Validation formulaire côté client

Heure de chargement : 01:15:12

Opérations arithmétiques sur deux nombres réels A et B positifs ou nuls

Valeur de A	Valeur de B
4,5	6,7

[Calculer](#) [Effacer](#)

**4**

Heure de calcul : 01:21:29

A=4,5  
B=6,7

**Résultats**

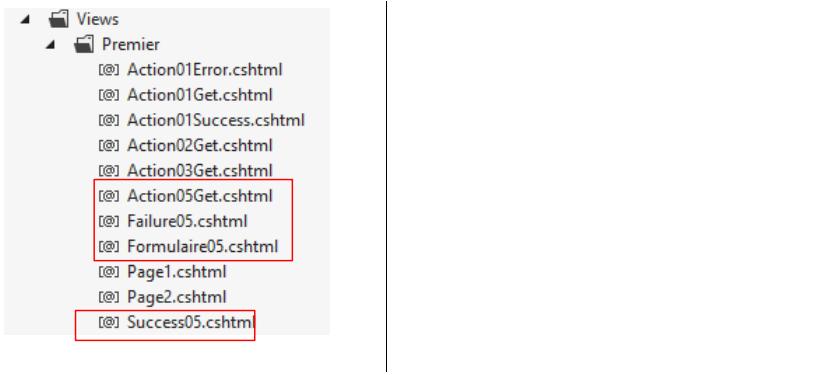
A+B=11,2  
A-B=-2,2  
A\*B=30,15  
A/B=0,671641791044776

[Retour aux saisies](#)

- en [4], la vue partielle [Success05].

L'application est à page unique : celle-ci est chargée par le navigateur lors de la première requête. Elle est ensuite mise à jour par des appels Ajax.

Les pages précédentes sont générées par les vues [cshtml] suivantes :



La vue chargée initialement est la vue [Action05Get.cshtml] suivante :

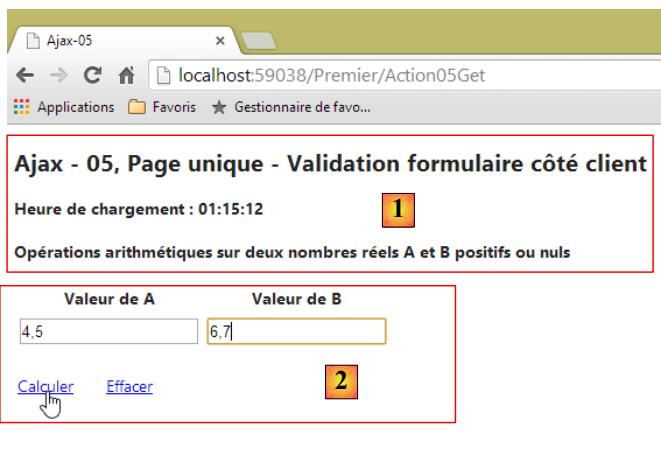
```

1. @model Exemple_04.Models.ViewModel05
2. @{
3.     Layout = null;
4. }
5.
6. <!DOCTYPE html>
7.
8. <html lang="fr-FR">
9. <head>
10.    <meta name="viewport" content="width=device-width" />
11.    <title>Ajax-05</title>
12.    <link rel="stylesheet" href("~/Content/Site.css" />
13.    <script type="text/javascript" src("~/Scripts/jquery-1.8.2.min.js")></script>
14.    <script type="text/javascript" src "~/Scripts/jquery.validate.min.js"></script>
15.    <script type="text/javascript" src "~/Scripts/jquery.validate.unobtrusive.min.js"></script>
16.    <script type="text/javascript" src "~/Scripts/globalize/globalize.js"></script>
17.    <script type="text/javascript" src "~/Scripts/globalize/cultures/globalize.culture.fr-FR.js"></script>
18.    <script type="text/javascript" src "~/Scripts/globalize/cultures/globalize.culture.en-US.js"></script>
19.    <script type="text/javascript" src "~/Scripts/jquery.unobtrusive-ajax.js"></script>
20.    <script type="text/javascript" src "~/Scripts/myScripts-05.js"></script>
21. </head>
22. <body>
23.
24.    <h2>Ajax - 05, Page unique - Validation formulaire côté client</h2>
25.    <p><strong>Heure de chargement : @Model.HeureChangement</strong></p>
26.    <h4>Opérations arithmétiques sur deux nombres réels A et B positifs ou nuls</h4>
27.    <img id="loading" style="display: none" src "~/Content/images/indicator.gif" />
28.    <div id="content">
29.        @Html.Partial("Formulaire05", Model)
30.    </div>
31. </body>
32. </html>
```

On notera les points suivants :

- ligne 1 : le modèle de la vue est le type [ViewModel05] que nous allons présenter prochainement ;
- lignes 13-19 : on retrouve les scripts Javascript nécessaires pour faire de l'Ajax et de la validation côté client ;
- ligne 20 : nous allons ajouter nos propres fonctions Javascript dans [myScripts-05.js] ;
- ligne 27 : l'image animée d'attente ;
- lignes 28-30 : une balise d'**id** [content]. C'est dans cette balise que les vues partielles [Formulaire05, Success05, Failure05] vont venir s'insérer ;
- ligne 29 : insertion de la vue partielle [Formulaire05].

La vue [Action05Get] est responsable de l'affichage de la partie [1] de la page initiale :



La vue partielle [Formulaire05] va générer la partie [2] ci-dessus. Son code est le suivant :

```

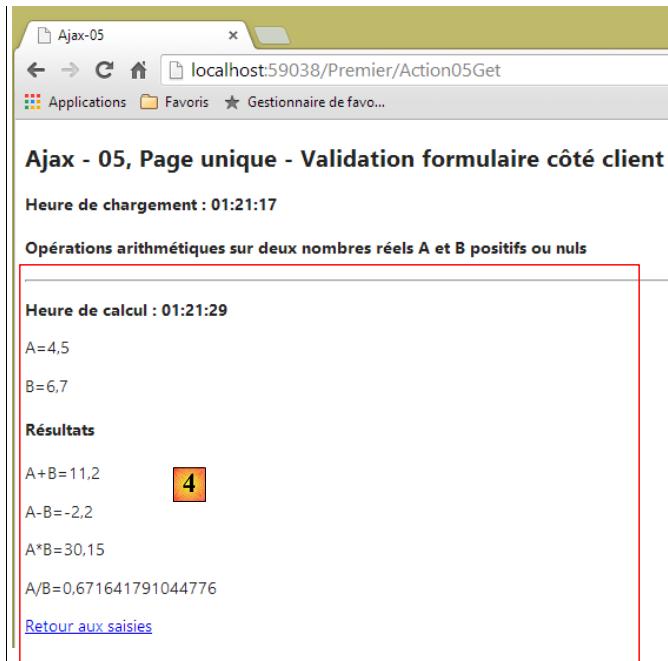
1. @model Exemple_04.Models.ViewModel05
2.
3. @using (Html.BeginForm("Action05Post", "Premier", FormMethod.Post, new { id = "formulaire" }))
4. {
5.     <table>
6.         <thead>
7.             <tr>
8.                 <th>@Html.LabelFor(m => m.A)</th>
9.                 <th>@Html.LabelFor(m => m.B)</th>
10.            </tr>
11.        </thead>
12.        <tbody>
13.            <tr>
14.                <td>@Html.TextBoxFor(m => m.A)</td>
15.                <td>@Html.TextBoxFor(m => m.B)</td>
16.            </tr>
17.            <tr>
18.                <td>@Html.ValidationMessageFor(m => m.A)</td>
19.                <td>@Html.ValidationMessageFor(m => m.B)</td>
20.            </tr>
21.        </tbody>
22.    </table>
23.    <p>
24.        <table>
25.            <tbody>
26.                <tr>
27.                    <td><a href="javascript:calculer()">Calculer</a>
28.                </td>
29.                <td style="width: 20px" />
30.                <td><a href="javascript:effacer()">Effacer</a>
31.            </td>
32.        </tr>
33.    </tbody>
34. </table>
35. </p>
36. }

```

- ligne 1 : la vue partielle admet pour modèle un type [ViewModel05] ;
- ligne 3 : le formulaire généré par la méthode [Html.BeginForm]. Parce que ce formulaire sera posté par un appel Ajax, les trois premiers paramètres de la méthode seront ignorés. Sauf si l'utilisateur a inhibé le Javascript sur son navigateur. Nous ignorons ici cette possibilité. Le quatrième paramètre est important. Le formulaire aura l'**id** [formulaire] ;
- lignes 5-22 : le formulaire de saisie des nombres A et B ;
- ligne 27 : un lien Javascript qui lance l'exécution des quatre opérations arithmétiques sur A et B ;
- ligne 30 : un lien Javascript qui efface les saisies et les éventuels messages d'erreur qui leur sont liés.

On notera que le formulaire n'a pas de bouton de type [submit]. Nous serons amenés à faire à la main le [Post] des valeurs A et B saisies.

S'il n'y a pas d'erreurs, les résultats sont affichés :



La partie [4] ci-dessus est générée par la vue partielle [Success05.cshtml] suivante :

```

1. @model Exemple_04.Models.ViewModel05
2. <hr />
3. <p><strong>Heure de calcul : @Model.HeureCalcul</strong></p>
4. <p>A=@Model.A</p>
5. <p>B=@Model.B</p>
6. <h4>Résultats</h4>
7. <p>A+B=@Model.AplusB</p>
8. <p>A-B=@Model.AmoinsB</p>
9. <p>A*B=@Model.AmultipliéparB</p>
10. <p>A/B=@Model.AdiviséparB</p>
11. <p>
12. <a href="javascript:retourSaisies()">Retour aux saisies</a>
13. </p>
```

- ligne 1 : la vue partielle [Success05.cshtml] reçoit un modèle de type [ViewModel05] ;
- ligne 12 : un lien Javascript pour revenir aux saisies.

En cas d'erreur, une autre vue partielle [3] est affichée :



Cette vue est générée par le code [Failure05.cshtml] suivant :

```

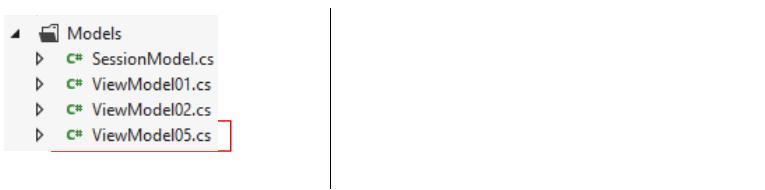
1. @model Exemple_04.Models.ViewModel05
2. <hr />
3. <p><strong>Heure de calcul : @Model.HeureCalcul</strong></p>
4. <p>A=@Model.A</p>
5. <p>B=@Model.B</p>
6. <h2>Les erreurs suivantes se sont produites</h2>
7. <ul>
8.   @foreach (string msg in Model.Erreurs)
9.   {
10.     <li>@msg</li>
11.   }
12. </ul>
13. <p>
14.   <a href="javascript:retourSaisies()">Retour aux saisies</a>
15. </p>

```

- ligne 1 : la vue partielle [Failure05.cshtml] reçoit un modèle de type [ViewModel05] ;
- ligne 14 : un lien Javascript pour revenir aux saisies.

### 1.7.6.2 Le modèle des vues

Toutes les vues précédentes partagent le même modèle [ViewModel05] :



```

1. using System;
2. using System.Collections.Generic;
3. using System.ComponentModel.DataAnnotations;
4. using System.Web.Mvc;
5.
6. namespace Exemple_04.Models
7. {
8.   [Bind(Exclude = "AplusB, AmoinsB, AmultipliéparB, AdiviséparB, Erreurs, HeureChargement, HeureCalcul")]
9.   public class ViewModel05
10.  {
11.    // formulaire
12.    [Required(ErrorMessage="Donnée A requise")]
13.    [Display(Name="Valeur de A")]
14.    [Range(0, Double.MaxValue, ErrorMessage = "Tapez un nombre A positif ou nul")]
15.    public string A { get; set; }
16.    [Required(ErrorMessage = "Donnée B requise")]
17.    [Display(Name = "Valeur de B")]
18.    [Range(0, Double.MaxValue, ErrorMessage="Tapez un nombre B positif ou nul")]
19.    public string B { get; set; }
20.
21.    // résultats
22.    public string AplusB { get; set; }
23.    public string AmoinsB { get; set; }
24.    public string AmultipliéparB { get; set; }
25.    public string AdiviséparB { get; set; }
26.    public List<string> Erreurs { get; set; }
27.    public string HeureChargement { get; set; }
28.    public string HeureCalcul { get; set; }
29.  }
30. }

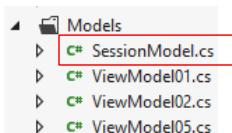
```

C'est le modèle [ViewModel01] déjà présenté page 185 à quelques détails près :

- lignes 15 et 19 : les champs A et B sont désormais de type [string] afin d'afficher des champs de saisie vides plutôt que des champs avec la valeur 0, lors de l'affichage initial du formulaire de saisie ;
- lignes 14 et 18 : cela n'empêche pas de vérifier la valeur saisie avec un validateur [Range] ;
- ligne 26 : une liste de messages d'erreurs affichée par la vue [Failure05].

### 1.7.6.3 Les données de portée [Session]

Page 191, paragraphe 1.7.3.6, nous avons vu que les données de la session étaient encapsulées dans le modèle [SessionModel] suivant :



```

1. using System;
2. namespace Exemple_03.Models
3. {
4.     public class SessionModel
5.     {
6.         // le générateur de nombres aléatoires
7.         public Random Randomizer { get; set; }
8.     }
9. }
```

Ce modèle de session est élargi pour intégrer les valeurs de A et B :

```

1. using System;
2. namespace Exemple_03.Models
3. {
4.     public class SessionModel
5.     {
6.         // le générateur de nombres aléatoires
7.         public Random Randomizer { get; set; }
8.         // les valeurs de A et B
9.         public string A { get; set; }
10.        public string B { get; set; }
11.    }
12. }
```

Il est en effet nécessaire de mémoriser les valeurs de A et B dans la session comme le montre la séquence suivante :

## Requête 1

## Requête 2

En [4], on retrouve les saisies faites en [1]. Or il y a deux requêtes HTTP distinctes. On sait que la mémoire entre deux requêtes HTTP est la session. Pour que la seconde puisse retrouver les valeurs postées par la première, il faut que ces dernières soient mises en session.

#### 1.7.6.4 L'action serveur [Action05Get]

L'action [Action05Get] est l'action qui fait afficher la page unique initiale. Son code est le suivant :

```

1.     [HttpGet]
2.     public ViewResult Action05Get()
3.     {
4.         ViewModel05 modèle = new ViewModel05();
5.         modèle.HeureChangement = DateTime.Now.ToString("hh:mm:ss");
6.         return View(modèle);
7.     }

```

- ligne 6 : la vue [Action05Get.cshtml] déjà étudiée est affichée avec un modèle de type [ViewModel05] ;

#### 1.7.6.5 L'action client [Calculer]

Examinons les interactions de l'utilisateur avec les vues :

Le lien [1] est un lien Javascript :

```
<a href="javascript:calculer()">Calculer</a>
```

La fonction Javascript [calculer] est trouvée dans le fichier [myScripts-05.js] :

```
<script type="text/javascript" src="~/Scripts/myScripts-05.js"></script>
```

Le code de la fonction Javascript [calculer] est le suivant :

```
1. // données globales
2. var content;
3. var loading;
4.
5. function calculer() {
6.     // d'abord les références sur le DOM
7.     var formulaire = $("#formulaire");
8.     // ensuite validation du formulaire
9.     if (!formulaire.validate().form()) {
10.         // formulaire invalide - terminé
11.         return;
12.     }
13.     // on fait un appel Ajax à la main
14.     $.ajax({
15.         url: '/Premier/Action05FaireCalcul',
16.         type: 'POST',
17.         data: formulaire.serialize(),
18.         dataType: 'html',
19.         beforeSend: function () {
20.             loading.show();
21.         },
22.         success: function (data) {
23.             content.html(data);
24.         },
25.         complete: function () {
26.             loading.hide();
27.         },
28.         error: function (jqXHR) {
29.             // affichage réponse serveur
30.             content.html(jqXHR.responseText);
31.         }
32.     })
33. }
34.
35. function retourSaisies() {
36. ...
37. }
38.
39. function effacer() {
40. ...
41. }
42.
43. // au chargement du document
44. $(document).ready(function () {
45.     // on récupère les références des différents composants de la page
46.     loading = $("#loading");
47.     content = $("#content");
48.     // on cache l'image animée
49.     loading.hide();
50. });


```

- on rappelle que le code Javascript **est toujours exécuté côté client**, dans le navigateur ;
- ligne 44 : la fonction JS exécutée lorsque le chargement initial de la page unique est terminé ;
- ligne 46 : référence sur l'image animée d'**id** [loading] ;
- ligne 47 : référence sur la région d'**id** [content]. C'est cette région qui reçoit les vues partielles [Formulaire05, Success05, Failure05] ;
- lignes 2-3 : les variables des lignes 46-47 sont déclarées **globales** afin que les autres fonctions y aient accès. Il y a un coût à rechercher des éléments dans une page (lignes 46-47). Il n'y a pas lieu de répéter cette recherche si on peut l'éviter ;
- ligne 5 : la fonction [calculer] ;
- ligne 7 : on récupère une référence sur le formulaire. La vue partielle [Formulaire05] lui a donné l'**id** [formulaire] ;
- ligne 9 : cette instruction exécute les validateurs du formulaire côté client. C'est ce qui manquait dans l'anomalie constatée page 194. Cette méthode est fournie par la bibliothèque [jquery.unobtrusive-ajax] utilisée par la page unique :

```
<script type="text/javascript" src="~/Scripts/jquery.unobtrusive-ajax.js"></script>
```

L'instruction rend [false] si le formulaire est déclaré invalide ;

- ligne 11 : l'appel Ajax au serveur n'est pas fait si le formulaire est invalide ;
- lignes 14-32 : l'appel Ajax fait au serveur ;
- ligne 15 : l'URL cible est l'action serveur [Action05FaireCalcul] ;

- ligne 16 : elle est demandée via un [POST] ;
- ligne 17 : les valeurs postées. Ce sont les saisies du formulaire, ici les valeurs de A et B ;
- lignes 22-24 : en cas de succès de l'appel Ajax, la fonction [calculer] met à jour la région d'**id** [content] avec le flux HTML envoyé par le serveur.

Ce flux HTML est celui envoyé par l'action [Action05FaireCalcul] ciblée par l'appel Ajax. Le code de cette action côté serveur est le suivant :

```

1.   [HttpPost]
2.   public PartialViewResult Action05FaireCalcul(FormCollection postData, SessionModel session)
3.   {
4.       // modèle
5.       ViewModel05 modèle = new ViewModel05();
6.       // l'heure de calcul
7.       modèle.HeureCalcul = DateTime.Now.ToString("hh:mm:ss");
8.       // mise à jour du modèle
9.       TryUpdateModel(modèle, postData);
10.      if (!ModelState.IsValid)
11.      {
12.          // on retourne une erreur
13.          modèle.Erreurs = getListOfMessagesFor(ModelState);
14.          return PartialView("Failure05", modèle);
15.      }
16.  ...
17. }
```

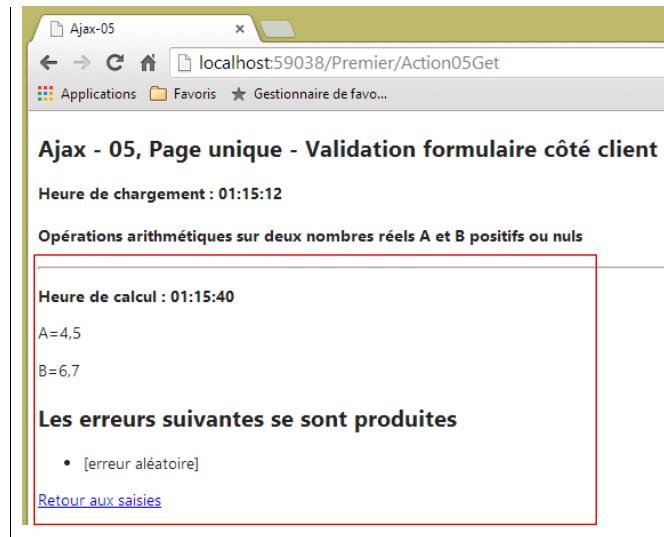
- ligne 1 : l'action n'accepte qu'un [post] ;
- ligne 2 : elle rend une vue partielle ;
- ligne 2 : elle reçoit en paramètres, les valeurs postées (postData) et le modèle de la session (session) ;
- ligne 5 : le modèle de la vue partielle est créé ;
- ligne 7 : il est mis à jour avec l'heure de calcul ;
- ligne 9 : on essaie d'appliquer les valeurs postées au modèle. Les validateurs de celui-ci vont alors être exécutés. On peut se demander pourquoi on prend cette peine alors que les validateurs côté client empêchent le POST si les données saisies sont invalides. En fait, on n'est pas sûr de la provenance du POST. Il a peut être été fait par un code qui n'est pas le nôtre. Aussi doit-on toujours faire les vérifications côté serveur ;
- ligne 10 : on teste si les validateurs ont réussi ;
- ligne 13 : si le modèle est invalide, on le met à jour avec une liste d'erreurs. On ne détaillera pas la méthode interne [getListOfMessagesFor] analogue à la méthode [GetErrorMessageFor] décrite page 71 ;
- ligne 14 : la vue partielle [Failure05] est affichée avec son modèle. On rappelle le code de cette vue ;

```

1. @model Exemple_04.Models.ViewModel05
2. <hr />
3. <p><strong>Heure de calcul : @Model.HeureCalcul</strong></p>
4. <p>A=@Model.A</p>
5. <p>B=@Model.B</p>
6. <h2>Les erreurs suivantes se sont produites</h2>
7. <ul>
8.     @foreach (string msg in Model.Erreurs)
9.     {
10.         <li>@msg</li>
11.     }
12. </ul>
13. <p>
14.     <a href="javascript:retourSaisies()">Retour aux saisies</a>
15. </p>
```

- lignes 7-12 : la liste des erreurs du modèle est affichée à l'aide d'une balise <ul>.

On se rappelle que la fonction JS [calculer] à l'origine du [Post] à l'action serveur [Action05FaireCalcul] va mettre ce flux HTML dans la région d'**id** [content]. Cela donne quelque chose comme ceci :



Continuons l'étude du code de l'action [Action05FaireCalcul] :

```

1.      [HttpPost]
2.      public PartialViewResult Action05FaireCalcul(FormCollection postData, SessionModel session)
3.      {
4.          // modèle
5.          ViewModel05 modèle = new ViewModel05();
6.          ...
7.          // on met les valeurs de A et B en session
8.          session.A = modèle.A;
9.          session.B = modèle.B;
10.         // pas d'erreurs pour l'instant
11.         List<string> erreurs = new List<string>();
12.         // une fois sur deux, on simule une erreur
13.         int val = session.Randomizer.Next(2);
14.         if (val == 0)
15.         {
16.             erreurs.Add("[erreur aléatoire]");
17.         }
18.         if (erreurs.Count != 0)
19.         {
20.             modèle.Erreurs = erreurs;
21.             return PartialView("Failure05", modèle);
22.         }
23.         // calculs
24.         double A = double.Parse(modèle.A);
25.         double B = double.Parse(modèle.B);
26.         modèle.AplusB = string.Format("{0}", A + B);
27.         modèle.AmoinsB = string.Format("{0}", A - B);
28.         modèle.AmultipliéparB = string.Format("{0}", A * B);
29.         modèle.AdiviséparB = string.Format("{0}", A / B);
30.         // vue
31.         return PartialView("Success05", modèle);
32.     }

```

- ligne 7 : le modèle a été déclaré valide ;
- lignes 8-9 : on met en session les valeurs saisies A et B. On veut pouvoir les retrouver dans la requête qui va suivre ;
- lignes 11-22 : on génère de façon aléatoire une erreur une fois sur deux ;
- lignes 24-29 : on fait les quatre opérations arithmétiques sur les nombres réels saisis ;
- ligne 31 : on retourne la vue partielle [Success05] avec son modèle. Cette vue partielle est la suivante :

```

1. @model Exemple_04.Models.ViewModel05
2. <hr />
3. <p><strong>Heure de calcul : @Model.HeureCalcul</strong></p>
4. <p>A=@Model.A</p>
5. <p>B=@Model.B</p>
6. <h4>Résultats</h4>
7. <p>A+B=@Model.AplusB</p>
8. <p>A-B=@Model.AmoinsB</p>
9. <p>A*B=@Model.AmultipliéparB</p>
10. <p>A/B=@Model.AdiviséparB</p>
11. <p>
12.     <a href="javascript:retourSaisies()">Retour aux saisies</a>
13. </p>

```

On se rappelle que la fonction JS [calculer] à l'origine du [Post] à l'action serveur [Action05FaireCalcul] va mettre ce flux HTML dans la région d'**id** [content]. Cela donne quelque chose comme ceci :

Ajax - 05, Page unique - Validation formulaire côté client

Heure de chargement : 01:21:17

Opérations arithmétiques sur deux nombres réels A et B positifs ou nuls

Heure de calcul : 01:21:29

A=4,5  
B=6,7

Résultats

A+B=11,2  
A-B=-2,2  
A\*B=30,15  
A/B=0,671641791044776

[Retour aux saisies](#)

#### 1.7.6.6 L'action client [Effacer]

Le lien Javascript [Effacer] permet de remettre le formulaire dans son état initial :

Ajax - 05, Page unique - Validation formulaire côté client

Heure de chargement : 03:31:27

Opérations arithmétiques sur deux nombres réels A et B positifs ou nuls

Valeur de A      Valeur de B

1,4x      2,8x

Tapez un nombre A positif ou nul Tapez un nombre B positif ou nul

[Calculer](#)    [Effacer](#)

Ajax - 05, Page unique - Validation formulaire côté client

Heure de chargement : 03:31:27

Opérations arithmétiques sur deux nombres réels A et B positifs ou nuls

Valeur de A      Valeur de B

[Calculer](#)    [Effacer](#)

Dans le formulaire, le lien JS [Effacer] est défini comme suit :

```
<a href="javascript:effacer()">Effacer</a>
```

La fonction JS [effacer] est définie dans le fichier [myScripts-05.js] de la façon suivante :

```
1. // données globales
2. var content;
3. var loading;
4.
5. function calculer() {
6. ...
7. }
8.
9. function retourSaisies() {
```

```

10. ...
11. }
12.
13. function effacer() {
14.   // d'abord les références sur le DOM
15.   var formulaire = $("#formulaire");
16.   var A = $("#A");
17.   var B = $("#B");
18.   // on affecte des valeurs valides aux saisies
19.   A.val("0");
20.   B.val("0");
21.   // puis on valide le formulaire pour faire disparaître
22.   // les éventuels msg d'erreur
23.   formulaire.validate().form();
24.   // puis on affecte des chaînes vides aux champs de saisie
25.   A.val("");
26.   B.val("");
27. }
28.
29. // au chargement du document
30. $(document).ready(function () {
31.   // on récupère les références des différents composants de la page
32.   loading = $("#loading");
33.   content = $("#content");
34.   // on cache l'image animée
35.   loading.hide();
36. });

```

- lignes 15-17 : on récupère des références sur divers éléments du DOM (**Document Object Model**) ;
- lignes 19-20 : on met des valeurs valides dans les champs de saisie des nombres A et B ;
- ligne 23 : on exécute les validateurs côté client. Comme les valeurs de A et B sont valides, cela va avoir pour effet de faire disparaître d'éventuels messages d'erreur qui pourraient être affichés ;
- lignes 25-26 : on met des chaînes vides dans les champs de saisie des nombres A et B ;

#### 1.7.6.7 L'action client [Retour aux Saisies]

Le lien Javascript [Retour aux Saisies] permet de revenir au formulaire après avoir obtenu des résultats :

The image consists of two side-by-side screenshots of a web browser window titled "Ajax - 05". Both screenshots show the URL `localhost:59038/Premier/Action05Get`.

**Left Screenshot (Results Page):**

- The title is "Ajax - 05, Page unique - Validation formulaire côté client".
- The subtitle is "Opérations arithmétiques sur deux nombres réels A et B positifs ou nuls".
- Below the subtitle, there is a horizontal line.
- The text "Heure de calcul : 03:48:01" is displayed.
- The text "A=2,5" is displayed.
- The text "B=4,9" is displayed.
- A section titled "Résultats" contains the following calculations:
  - $A+B=7,4$
  - $A-B=-2,4$
  - $A*B=12,25$
  - $A/B=0,510204081632653$
- At the bottom, there is a blue link labeled "Retour aux saisies" with a hand cursor icon pointing at it.

**Right Screenshot (Form Page):**

- The title is "Ajax - 05, Page unique - Validation formulaire côté client".
- The subtitle is "Opérations arithmétiques sur deux nombres réels A et B positifs ou nuls".
- Below the subtitle, there is a horizontal line.
- Two input fields are present: "Valeur de A" containing "2,5" and "Valeur de B" containing "4,9".
- Below the input fields are two buttons: "Calculer" and "Effacer".

Dans le formulaire, le lien JS [Retour aux Saisies] est défini comme suit :

```
<a href="javascript:retourSaisies()">Retour aux saisies</a>
```

La fonction JS [retourSaisies] est définie dans le fichier [myScripts-05.js] de la façon suivante :

```
1. // données globales
```

```

2. var content;
3. var loading;
4.
5. function calculer() {
6. ...
7. }
8.
9. function retourSaisies() {
10. // on fait un appel Ajax à la main
11. $.ajax({
12.   url: '/Premier/Action05RetourSaisies',
13.   type: 'POST',
14.   dataType: 'html',
15.   beforeSend: function () {
16.     loading.show();
17.   },
18.   success: function (data) {
19.     content.html(data);
20.   },
21.   complete: function () {
22.     loading.hide();
23.     // IMPORTANT !! validation
24.     $.validator.unobtrusive.parse($("#formulaire"));
25.   },
26.   error: function (jqXHR) {
27.     content.html(jqXHR.responseText);
28.   }
29. })
30. }
31.
32. function effacer() {
33. ...
34. }
35.
36. // au chargement du document
37. $(document).ready(function () {
38.   // on récupère les références des différents composants de la page
39.   loading = $("#loading");
40.   content = $("#content");
41.   // on cache l'image animée
42.   loading.hide();
43. }));

```

- lignes 11-29 : un appel Ajax ;
- ligne 12 : l'URL cible ;
- ligne 13 : elle sera demandée par une commande HTTP POST. C'est un POST sans paramètres postés. C'est pourquoi, on ne trouve pas une ligne du type :

```
  data: formulaire.serialize(),
```

- dans l'appel Ajax ;
- ligne 14 : le flux attendu du serveur est un flux HTML ;
  - lignes 18-20 : ce flux HTML servira à mettre à jour la région d'**id** [content] ;

L'action serveur [Action05RetourSaisies] est la suivante :

```

1. [HttpPost]
2. public PartialViewResult Action05RetourSaisies(SessionModel session)
3. {
4.   // vue
5.   return PartialView("Formulaire05", new ViewModel05() { A = session.A, B = session.B });
6. }
```

- ligne 2 : l'action reçoit pour paramètre le modèle de la session dans lequel nous avons mémorisé précédemment les valeurs de A et B saisies ;
- ligne 5 : on retourne la vue partielle [Formulaire05] avec un modèle de type [ViewModel05] dans lequel on prend soin d'initialiser les champs A et B avec les valeurs de A et B prises dans la session ;

Maintenant revenons au code de la fonction Javascript [retourSaisies] :

```

1. function retourSaisies() {
2.   // on fait un appel Ajax à la main
3.   $.ajax({
4.     url: '/Premier/Action05RetourSaisies',
5.     type: 'POST',
6.     dataType: 'html',
7.     beforeSend: function () {
8.       loading.show();
9.     },
```

```

10.     success: function (data) {
11.         content.html(data);
12.     },
13.     complete: function () {
14.         loading.hide();
15.         // IMPORTANT !! validation
16.         $.validator.unobtrusive.parse($("#formulaire"));
17.     },
18.     error: function (jqXHR) {
19.         content.html(jqXHR.responseText);
20.     }
21. })
22. }

```

- ligne 13 : la méthode exécutée lorsque l'appel Ajax est terminé ;
- ligne 14 : l'image animée d'attente est cachée ;
- ligne 16 : une instruction un peu hermétique pour moi trouvée sur le net pour résoudre le problème suivant : dans le formulaire affiché par le lien [Retour aux saisies], les validateurs côté client ne fonctionnaient plus. En cherchant des informations sur la bibliothèque JS [jquery.unobtrusive-ajax], j'ai trouvé la solution de la ligne 16. Elle parse le formulaire, peut-être pour activer les validateurs côté client.

### **1.7.7 Rendre accessible sur Internet une application ASP.NET**

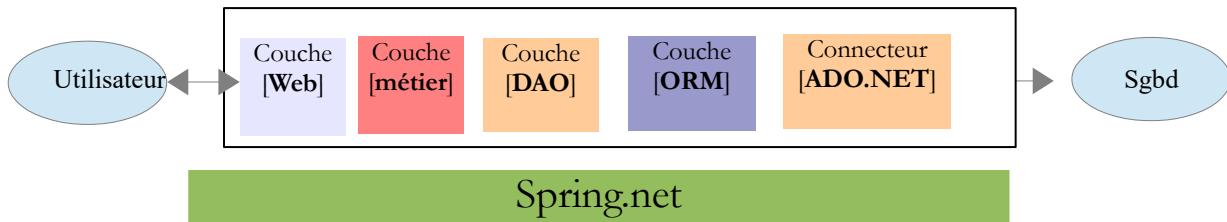
Voir le paragraphe 2.26, page 321.

### **1.7.8 Génération d'une application native pour Android à partir d'une application à page unique APU**

Voir le paragraphe 2.27, page 332.

## 1.8 Conclusion intermédiaire

Ici se termine la présentation du framework ASP.NET MVC. Nous allons poursuivre avec une étude de cas illustrant l'utilisation de ce framework dans une architecture en couches :



- la couche **[Web]** est la couche en contact avec l'utilisateur de l'application Web. Celui-ci interagit avec l'application Web au travers de pages Web visualisées par un navigateur. **C'est dans cette couche que se situe ASP.NET MVC et uniquement dans cette couche.**
- la couche **[métier]** implémente les règles de gestion de l'application, tels que le calcul d'un salaire ou d'une facture. Cette couche utilise des données provenant de l'utilisateur via la couche [Web] et du SGBD via la couche [DAO].
- la couche **[DAO]** (Data Access Objects), la couche **[ORM]** (Object Relational Mapper) et le connecteur ADO.NET gèrent l'accès aux données du SGBD. La couche **[ORM]** fait un pont entre les objets manipulés par la couche [DAO] et les lignes et les colonnes des tables d'une base de données relationnelle. Nous utiliserons l'ORM **Entity Framework** (<http://msdn.microsoft.com/en-us/data/ef.aspx>).
- l'intégration des couches peut être réalisée par un conteneur d'injection de dépendances (Dependency Injection container). Nous utiliserons **Spring.net** (<http://www.springframework.net/>).

Bien que ce document soit déjà fort volumineux, il est incomplet. Le lecteur pourra compléter sa formation avec le livre "**Pro ASP.NET MVC 4**" écrit par **Adam Freeman** aux éditions **Apress**. C'est un très bon livre. Ses 800 pages satisferont les lecteurs les plus exigeants.

## 2 Etude de cas n° 1 : gestion basique de salaires

### 2.1 Introduction

Nous allons présenter une étude de cas déjà publiée dans un article disponible à l'URL [<http://tahe.developpez.com/dotnet/pam-aspnet/>]. Dans cet article, l'étude de cas est réalisée avec **ASP.NET classique** et l'ORM **NHibernate**. Nous allons ici, la réaliser avec **ASP.NET MVC** et l'ORM **Entity Framework**. Comme dans l'article existant, l'étude de cas est présentée comme un TD d'université. Elle est donc destinée à des étudiants. Pour toutes les questions, des renvois aux chapitres que nous venons de détailler sont faits pour indiquer les lectures utiles.

Les compétences mises en oeuvre par cet exercice sont les suivantes :

- savoir écrire une page ASP.NET MVC ;
- savoir gérer le modèle et les événements d'une page ASP.NET MVC ;
- savoir faire des appels AJAX avec JQuery ;
- connaître et savoir utiliser les différentes portées des modèles d'une application web : application, session, request ;
- connaître le rôle des [ModelBinder] ;
- maîtriser Visual Studio et son environnement de développement et notamment savoir utiliser le débogueur ;
- savoir utiliser WampServer, PhpMyAdmin, MySQL ;

### 2.2 Le problème à résoudre

Nous souhaitons écrire une application web 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 sera de type APU [Application à Page Unique] et utilisera exclusivement des appels Ajax pour communiquer avec le serveur. Elle présentera à l'utilisateur les vues suivantes :

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



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

Pam

localhost:65013

[Applications](#) [Favoris](#) [Gestionnaire de favo...](#)

[Faire la simulation](#)  
[Effacer la simulation](#)  
[Enregistrer la simulation](#)  
[Terminer la session](#)

Employé	Heures travaillées	Jours travaillés
Marie Jouveinal	150	20

**Informations Employé**

Nom	Prénom	Adresse
Jouveinal	Marie	5 rue des oiseaux

Ville	Code Postal	Indice
St Corentin	49203	2

**Informations Cotisations**

CGSRDS	CSGD	Retraite	Sécurité sociale
3,49 %	6,15 %	7,88 %	9,39 %

**Informations Indemnités**

Salaire horaire	Entretien / Jour	Repas / Jour	Congés payés
2,10 €	2,10 €	3,10 €	15 %

**Informations Salaire**

Salaire de base	Cotisations sociales	Indemnités d'entretien	Indemnités de repas
362,25 €	97,48 €	42,00 €	62,00 €

Salaire net à payer : 368,77 €

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

N°	Nom	Prénom	Heures travaillées	Jours travaillés	Salaire de base	Indemnités	Cotisations sociales	Salaire net
1	Jouveinal	Marie	150	20	362,25 €	104,00 €	97,48 €	368,77 €
2	Laverti	Justine	50	20	108,08 €	100,00 €	29,08 €	179,00 €

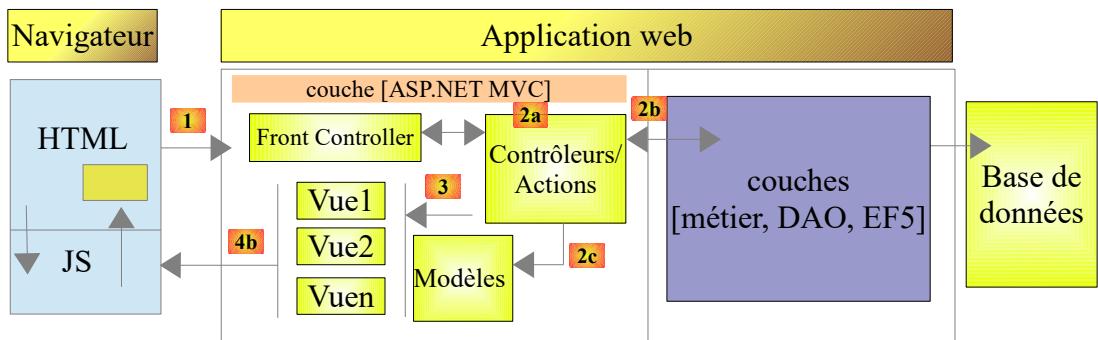
- la vue [VueSimulationsVides] qui indique que le client n'a pas ou plus de simulations :

- la vue [VueErreurs] qui indique une ou plusieurs erreurs (ici le SGBD MySQL a été arrêté) :

- GetEmploye
- Échec du fournisseur sous-jacent sur Open.
- Unable to connect to any of the specified MySQL hosts.

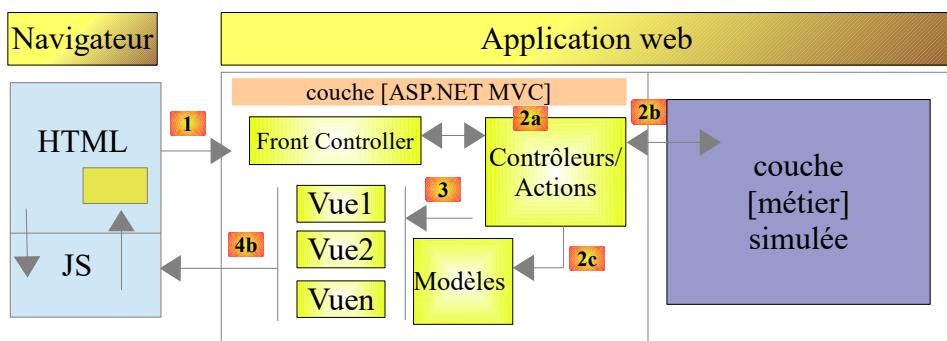
## 2.3 Architecture de l'application

L'architecture de l'application sera la suivante :



La couche [EF5] désigne l'ORM Entity Framework 5. Le SGBD utilisé sera MySQL.

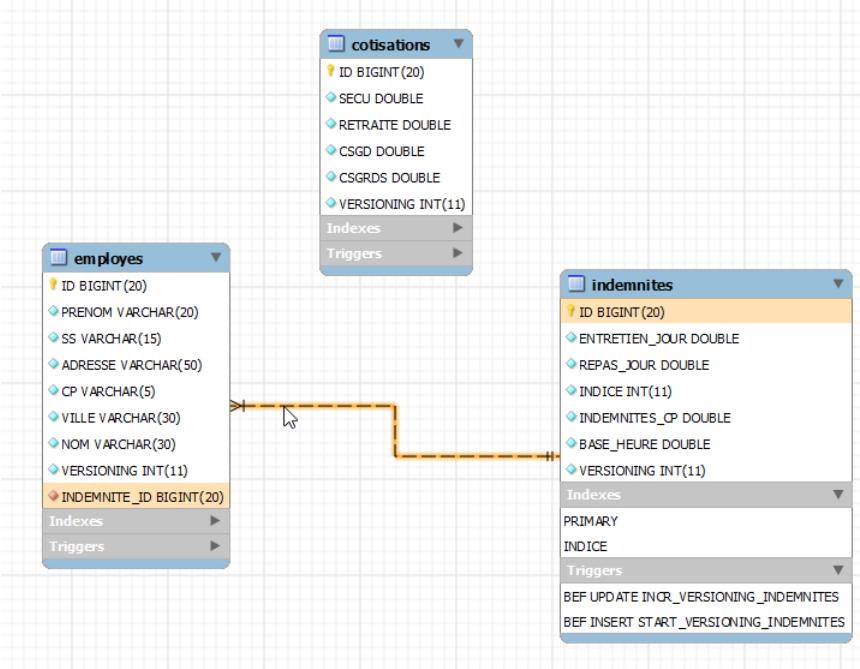
Nous construirons cette application d'abord avec une couche [métier] simulée :



Cela nous permettra de nous concentrer uniquement sur la couche [web]. La couche [métier] simulée respectera l'interface de la couche [métier] réelle. Lorsque la couche [web] sera opérationnelle, on construira alors les couches [métier], [DAO] et [EF5].

## 2.4 La base de données

Les données statiques utiles pour construire la fiche de paie sont placées dans une base de données MySQL nommée [**dbpam\_ef5**] (**pam**=Paie Assistante Maternelle). Cette base a un administrateur appelé **root** sans mot de passe. Elle a trois tables :



Il y a une relation de clé étrangère entre la colonne EMPLOYES(INDEMNITE\_ID) et la colonne INDEMNITES(ID). La structure de cette base est dictée par son utilisation avec EF5. Nous reviendrons dessus lorsque nous construirons les couches basses de l'application.

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

Structure :

<table border="1"> <thead> <tr> <th colspan="2">employees</th> </tr> </thead> <tbody> <tr><td>ID</td><td>BIGINT(20)</td></tr> <tr><td>PRENOM</td><td>VARCHAR(20)</td></tr> <tr><td>SS</td><td>VARCHAR(15)</td></tr> <tr><td>ADRESSE</td><td>VARCHAR(50)</td></tr> <tr><td>CP</td><td>VARCHAR(5)</td></tr> <tr><td>VILLE</td><td>VARCHAR(30)</td></tr> <tr><td>NOM</td><td>VARCHAR(30)</td></tr> <tr><td>VERSIONING</td><td>INT(11)</td></tr> <tr><td>INDEMNITE_ID</td><td>BIGINT(20)</td></tr> <tr> <td colspan="2">Indexes</td></tr> <tr> <td colspan="2">Triggers</td></tr> </tbody> </table>	employees		ID	BIGINT(20)	PRENOM	VARCHAR(20)	SS	VARCHAR(15)	ADRESSE	VARCHAR(50)	CP	VARCHAR(5)	VILLE	VARCHAR(30)	NOM	VARCHAR(30)	VERSIONING	INT(11)	INDEMNITE_ID	BIGINT(20)	Indexes		Triggers		ID	clé primaire incrémentée automatiquement par le SGBD
employees																										
ID	BIGINT(20)																									
PRENOM	VARCHAR(20)																									
SS	VARCHAR(15)																									
ADRESSE	VARCHAR(50)																									
CP	VARCHAR(5)																									
VILLE	VARCHAR(30)																									
NOM	VARCHAR(30)																									
VERSIONING	INT(11)																									
INDEMNITE_ID	BIGINT(20)																									
Indexes																										
Triggers																										
SS	numéro de sécurité sociale de l'employé - unique																									
NOM	nom de l'employé																									
PRENOM	son prénom																									
ADRESSE	son adresse																									
VILLE	sa ville																									
CP	son code postal																									
VERSIONING	un entier autoincrémenté à chaque fois que l'enregistrement est modifié																									
INDEMNITE_ID	clé étrangère sur le champ [ID] de la table [INDEMNITES]																									

Son contenu pourrait être le suivant :

ID	PRENOM	SS	ADRESSE	CP	VILLE	NOM	VERSIONING	INDEMNITE_ID
24	Marie	254104940426058	5 rue des oiseaux	49203	St Corentin	Jouvenal	1	93
25	Justine	260124402111742	La Brûlerie	49014	St Marcel	Laverti	1	94

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

Structure :

	ID	clé primaire incrémentée automatiquement par le SGBD
	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
	VERSIONING	un entier autoincrémenté à chaque fois que l'enregistrement est modifié

Son contenu pourrait être le suivant :

ID	SECU	RETRAITE	CSGD	CSGRDS	VERSIONING
11	9.39	7.88	6.15	3.49	1

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

	ID	clé primaire incrémentée automatiquement par le SGBD
	INDICE	indice de traitement - unique
	BASE_HEURE	prix net en euro d'une heure de garde
	ENTRETIEN_JOUR	indemnité d'entretien en euro par jour de garde
	REPAS_JOUR	indemnité de repas en euro par jour de garde
	INDEMNITES_CP	indemnité de congés payés. C'est un pourcentage à appliquer au salaire de base.
	VERSIONING	un entier autoincrémenté à chaque fois que l'enregistrement est modifié

Son contenu pourrait être le suivant :

ID	ENTRETIEN_JOUR	REPAS_JOUR	INDICE	INDEMNITES_CP	BASE_HEURE	VERSIONING
93	2.1	3.1	2	15	2.1	1
94	2	3	1	12	1.93	1

## 2.5 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.

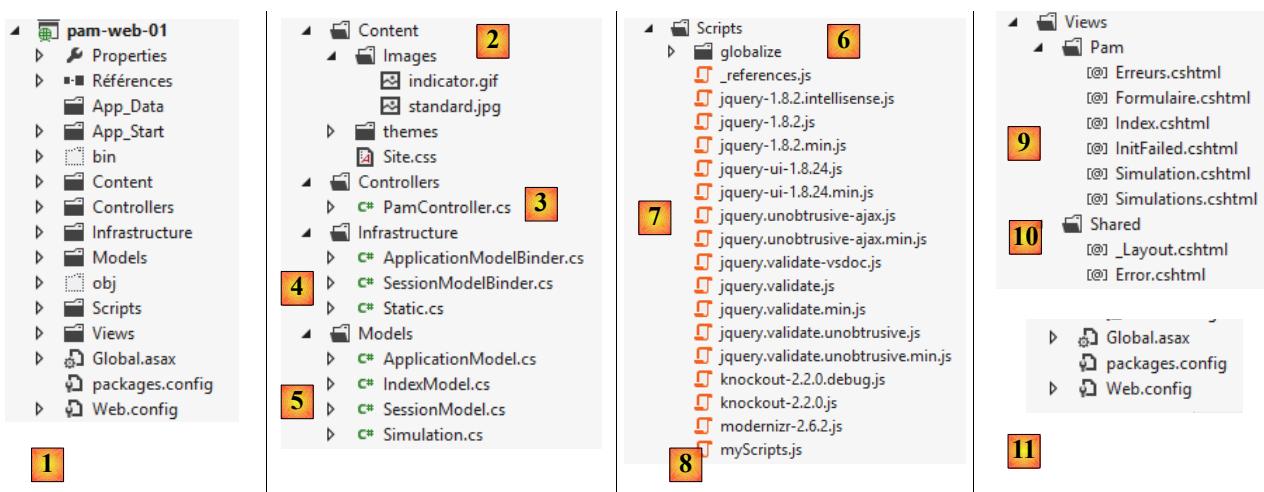
Les éléments suivants sont pris en compte :	$[\text{TOTALHEURES}]$ : total des heures travaillées dans le mois $[\text{TOTALJOURS}]$ : total des jours travaillés dans le mois	$[\text{TOTALHEURES}] = 150$ $[\text{TOTALJOURS}] = 20$
Le salaire de base de l'assistante maternelle est donné par la formule suivante :	$[\text{SALAIREBASE}] = ([\text{TOTALHEURES}] * [\text{BAS EHEURE}]) * (1 + [\text{INDEMNITESCP}] / 100)$	$[\text{SALAIREBASE}] = (150 * [2.1]) * (1 + 0.15) = 362,25$
Un certain nombre de cotisations sociales	Contribution sociale généralisée	$\text{CSGRDS} : 12,64$

Les éléments suivants sont pris en compte :	[TOTALHEURES]: total des heures travaillées dans le mois  [TOTALJOURS]: total des jours travaillés dans le mois  et contribution au remboursement de la dette sociale : [SALAIREBASE]* [CSGRDS/100]	[TOTALHEURES]=150 [TOTALJOURS] = 20  CSGD : 22,28  Contribution sociale généralisée déductible : [SALAIREBASE]* [CSGD/100]  Sécurité sociale, veuvage, vieillesse : [SALAIREBASE]* [SECU/100]  Retraite Complémentaire + AGPF + Assurance Chômage : [SALAIREBASE]* [RETRAITE/100]
Total des cotisations sociales :	[COTISATIONSSOCIALES]=[SALAIREBASE]* (CSGRDS+CSGD+SECU+RETRAITE)/100	[COTISATIONSSOCIALES]=97,48
Par ailleurs, l'assistante maternelle a droit, chaque jour travaillé, à une indemnité d'entretien ainsi qu'à une indemnité de repas. A ce titre elle reçoit les indemnités suivantes :	[INDEMNITES]=[TOTALJOURS]* (ENTRETIEENJOUR+REPASJOUR)	[INDEMNITES]=104
Au final, le salaire net à payer à l'assistante maternelle est le suivant :	[SALAIREBASE]-[COTISATIONSSOCIALES]+[INDEMNITES]	[salaire NET]=368,77

## 2.6 Le projet Visual Studio de la couche [web]

Note : ce document utilise Visual Studio Express 2012. Adaptez les copies d'écran qui sont présentées à la version de Visual Studio que vous utilisez.

Le projet Visual Web Developer de l'application sera le suivant :

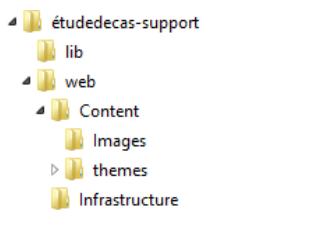


- en [1], la structure générale du projet [pam-web-01] ;
- en [2], le dossier [Content] est le dossier où l'on met les ressources statiques du projet :
  - [indicator.gif] : l'image animée de l'attente de fin d'une requête Ajax,
  - [standard.jpg] : l'image de fond des différentes vues,
  - [Site.css] : la feuille de style de l'application ;
- en [3], l'unique contrôleur de l'application [PamController] ;

- en [4], des classes nécessaires à l'application mais ne pouvant être cataloguées comme éléments du MVC :
  - [ApplicationModelBinder] : la classe qui permet d'inclure les données de portée [Application] dans le modèle des actions,
  - [SessionModelBinder] : la classe qui permet d'inclure les données de portée [Session] dans le modèle des actions,
  - [Static] une classe d'aide avec des méthodes statiques ;
- en [5], les modèles de l'application, que ce soit des modèles d'actions ou de vues :
  - [ApplicationModel] : modèle contenant les données de portée [Application],
  - [SessionModel] : modèle contenant les données de portée [Session],
  - [Simulation] : classe encapsulant les éléments d'une simulation de calcul de salaire,
  - [IndexModel] : modèle de la première vue [Index] affichée par l'application ;
- en [6], les scripts JS nécessaires à la globalisation de l'application ;
- en [7], les scripts JS de la famille JQuery nécessaires à l'internationalisation, la validation côté client et l'ajaxification de l'application ;
- en [8], [myScripts.js] est le fichier contenant nos propres scripts JS ;
- en [9], les vues de l'application :
  - [Index] : la page d'accueil,
  - [Formulaire] : formulaire de saisie de l'employé et de ses heures et jours travaillés,
  - [Simulation] : la vue présentant une simulation,
  - [Simulations] : la vue présentant la liste des simulations faites,
  - [Erreurs] : la vue présentant la liste des éventuelles erreurs,
  - [InitFailed] : la vue affichant des messages d'erreurs si l'initialisation de l'application échoue ;
- en [10], la page maître de l'application [Layout] ;
- en [11], les fichiers [Web.config], [Global.asax] utilisés pour configurer l'application.

## 2.7 Étape 1 – mise en place de couche [métier] simulée

A partir de maintenant, nous décrivons les étapes à suivre pour réaliser l'étude de cas. Lorsque c'est utile, nous rappelons le n° du chapitre à relire éventuellement pour réaliser le travail demandé. Certains éléments du projet vous sont donnés dans un dossier [**aspnetmvc-v2.rar**] qu'on trouvera sur le site de ce document. On y trouvera le dossier [étude de cas n° 1] avec le contenu suivant :

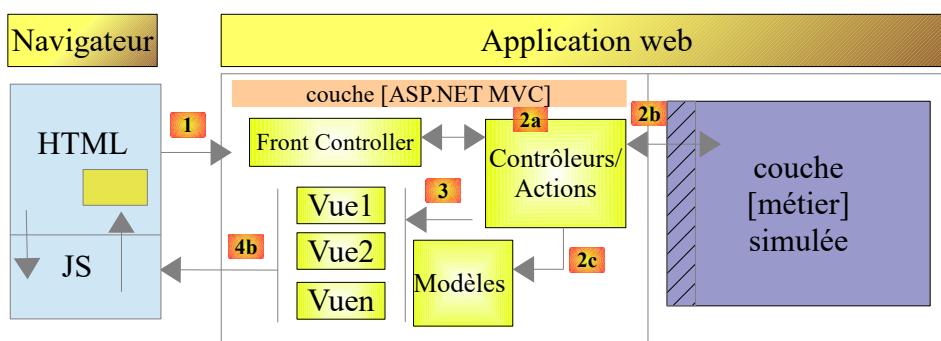


Le projet reprend par ailleurs des éléments présentés dans les chapitres précédents. Il suffit alors de récupérer ceux-ci par copier / coller entre ce PDF et Visual Studio.

### 2.7.1 La solution Visual Studio de l'application complète

Nous allons tout d'abord créer une solution Visual Studio dans laquelle nous allons créer deux projets :

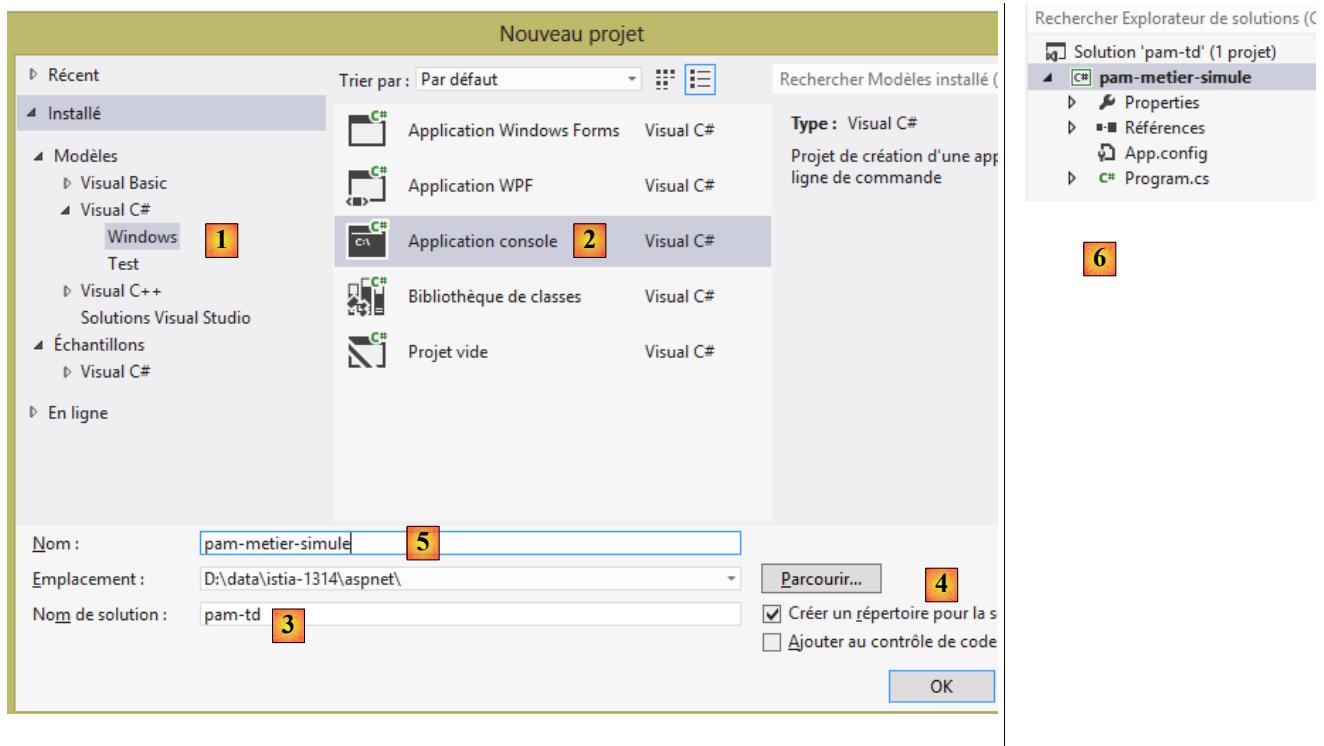
- un projet pour la couche [métier] simulée ;
- un projet pour la couche web MVC.



Nous utiliserons deux outils :

- Visual Studio Express 2012 pour le bureau qui servira à construire la couche [métier] ;
- Visual Studio Express 2012 pour le Web qui servira à construire la couche [web].

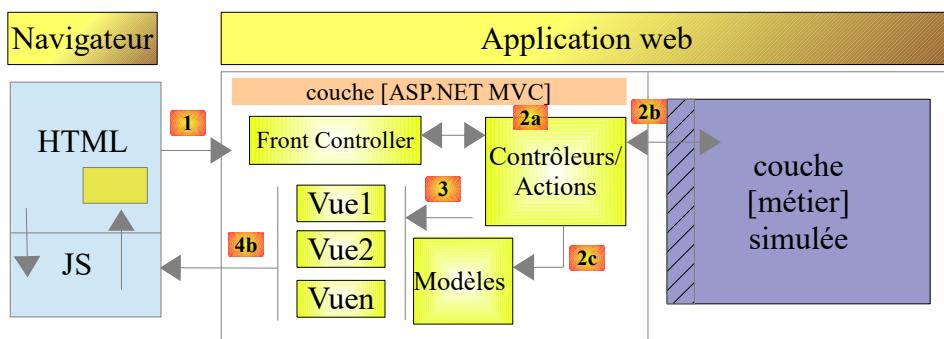
Avec Visual Studio Express pour le bureau, nous créons une solution [pam-td] :



- en [1], choisir une application C# ;
- en [2], choisir [Application console] ;
- en [3], donner un nom à la solution ;
- en [4], générer un répertoire pour cette solution ;
- en [5], donner un nom à la couche [métier] ;
- en [6], la solution générée.

## 2.7.2 L'interface de la couche [métier]

Dans une architecture en couches, il est de bonne pratique que la communication entre couches se fasse via des interfaces :



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

**Simulateur de calcul de paie**

Faire la simulation  
Effacer la simulation  
Enregistrer la simulation  
Terminer la session

Employé	Heures travaillées	Jours travaillés
Marie Jouveinal	150	20 3

**Informations Employé** 5

Nom	Prénom	Adresse
Jouveinal	Marie	5 rue des oiseaux

Ville	Code Postal	Indice
St Corentin	49203	10 2 11

**Informations Cotisations**

CGSRDS	CSGD	Retraite	Sécurité sociale
3,49 %	6,15 %	7,88 %	9,39 %
12	13	14	15

**Informations Indemnités**

Salaire horaire	Entretien / Jour	Repas / Jour	Congés payés
2,10 €	2,10 €	3,10 €	15 %
16	17	18	19

**Informations Salaire**

Salaire de base	Cotisations sociales	Indemnités d'entretien	Indemnités de repas
362,25 €	97,48 €	42,00 €	62,00 €
20	21	22	23

Salaire net à payer : 368,77 € 24

- 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).
- les informations 12 à 15 sont les différents taux de cotisations.
- les informations 16 à 19 sont les indemnités de l'employé
- 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 [web] par la couche [métier] doit répondre aux exigences ci-dessus. Il existe de nombreuses interfaces possibles. Nous proposons la suivante :

```

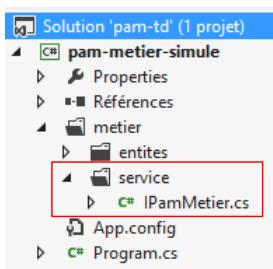
1. using Pam.Metier.Entites;
2. namespace Pam.Metier.Service
3. {
4.     public interface IPamMetier
5.     {
6.         // liste de toutes les identités des employés
7.         Employe[] GetAllIdentitesEmployes();
8.
9.         // ----- le calcul du salaire
10.        FeuilleSalaire GetSalaire(string 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] que nous allons décrire prochainement.

Nous mettrons cette interface dans un dossier [metier/service] :

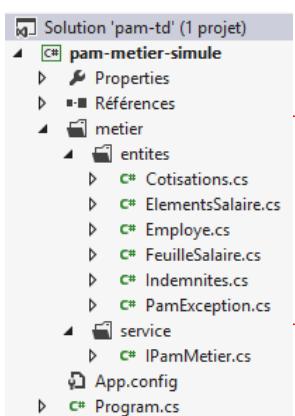


### 2.7.3 Les entités de la couche [métier]

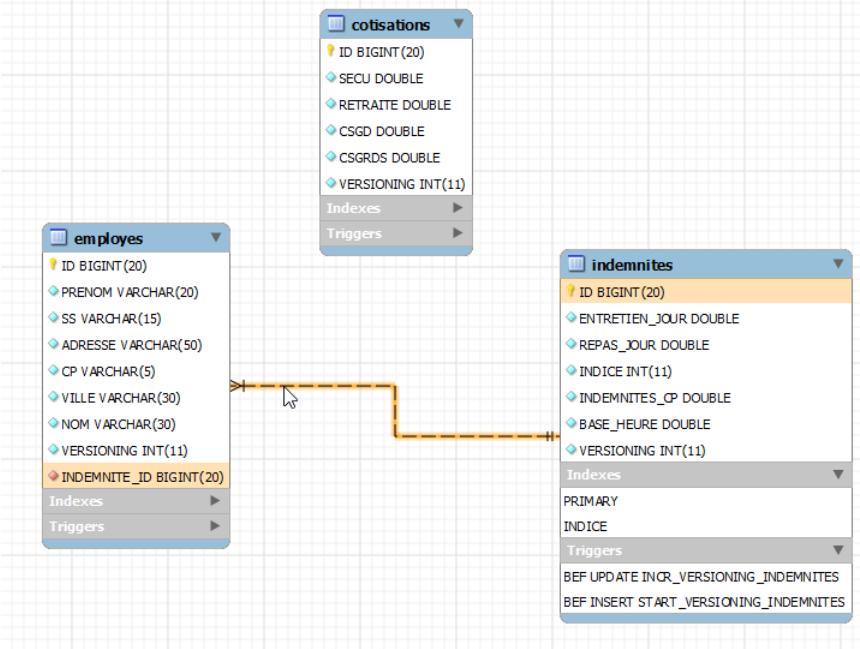
L'interface précédente utilise deux classes [Employe] et [FeuilleSalaire] qu'il nous faut définir :

- [Employe] est l'image d'une ligne de la table [employes] de la base de données ;
- [FeuilleSalaire] est la feuille de salaire d'un employé.

Les entités seront placées dans un dossier [metier / entites] du projet :



Dans l'architecture finale, la couche [métier] va manipuler des entités images de la base de données :



Nous utiliserons les classes suivantes pour représenter les lignes des trois tables de la base de données. On se reportera au paragraphe 2.4, page 223, pour connaître la signification des différents champs.

### Classe [Employe]

Elle représente une ligne de la table [employees]. Son code est le suivant :

```

1.  using System;
2.
3.  namespace Pam.Metier.Entites
4.  {
5.
6.      public class Employe
7.      {
8.          public string SS { get; set; }
9.          public string Nom { get; set; }
10.         public string Prenom { get; set; }
11.         public string Adresse { get; set; }
12.         public string Ville { get; set; }
13.         public string CodePostal { get; set; }
14.         public Indemnites Indemnites { get; set; }
15.
16.         // signature
17.         public override string ToString()
18.         {
19.             return string.Format("Employé[{0},{1},{2},{3},{4},{5}]", SS, Nom, Prenom, Adresse, Ville, CodePostal);
20.         }
21.     }
22. }
```

### Classe [Indemnites]

Elle représente une ligne de la table [indemnites]. Son code est le suivant :

```

1.  using System;
2.
3.  namespace Pam.Metier.Entites
4.  {
5.      public class Indemnites
6.      {
7.          public int Indice { get; set; }
8.          public double BaseHeure { get; set; }
9.          public double EntretienJour { get; set; }
10.         public double RepasJour { get; set; }
11.         public double IndemnitesCp { get; set; }
12.         // signature
13.         public override string ToString()
14.     }
```

```

15.         return string.Format("Indemnités[{0},{1},{2},{3},{4}]", Indice, BaseHeure, EntretienJour, RepasJour, IndemnitesCp);
16.     }
17. }
18. }
```

## Classe [Cotisations]

Elle représente une ligne de la table [cotisations]. Son code est le suivant :

```

1.  using System;
2.
3.  namespace Pam.Metier.Entites
4.  {
5.
6.      public class Cotisations
7.      {
8.          public double CsgRds { get; set; }
9.          public double Csgd { get; set; }
10.         public double Secu { get; set; }
11.         public double Retraite { get; set; }
12.         // signature
13.         public override string ToString()
14.         {
15.             return string.Format("Cotisations[{0},{1},{2},{3}]", CsgRds, Csgd, Secu, Retraite);
16.         }
17.     }
18. }
```

On notera que les classes ne reprennent pas les colonnes [ID] et [VERSIONING] des tables. Ces colonnes, utiles lorsqu'on utilisera l'ORM EF5, ne le sont pas dans le contexte de la couche [métier] simulée.

La classe [FeuilleSalaire] encapsule les informations 6 à 24 du formulaire présenté page 228 :

```

1.  namespace Pam.Metier.Entites
2.  {
3.      public class FeuilleSalaire
4.      {
5.
6.          // propriétés automatiques
7.          public Employe Employe { get; set; }
8.          public Cotisations Cotisations { get; set; }
9.          public ElementsSalaire ElementsSalaire { get; set; }
10.
11.         // ToString
12.         public override string ToString()
13.         {
14.             return string.Format("[{0},{1},{2}]", Employe, Cotisations, ElementsSalaire);
15.         }
16.     }
17. }
```

- ligne 7 : les informations 6 à 11 sur l'employé dont on calcule le salaire et les informations 16 à 19 sur ses indemnités. Il ne faut pas oublier ici qu'un objet [Employe] encapsule un objet [Indemnites] représentant ses indemnités ;
- ligne 8 : les informations 12 à 15 ;
- ligne 9 : les informations 20 à 24 ;
- lignes 12-14 : la méthode [ToString].

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

```

1.  namespace Pam.Metier.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.         // ToString
14.         public override string ToString()
15.         {
16.             return string.Format("[{0} : {1} : {2} : {3} : {4} ]", SalaireBase, CotisationsSociales, IndemnitesEntretien,
17.             IndemnitesRepas, SalaireNet);
18.         }
19.     }
19. }
```

- lignes 6-10 : les éléments du salaire tels qu'expliqués dans les règles métier décrites page 225 ;
- 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 14-17 : la méthode [ToString] de la classe.

## 2.7.4 La classe [PamException]

Nous créons un type d'exceptions spécifique pour notre application. C'est le type [PamException] suivant :

```

1.  using System;
2.
3.  namespace Pam.Metier.Entites
4.  {
5.      // classe d'exception
6.      public class PamException : Exception
7.      {
8.
9.          // le code de l'erreur
10.         public int Code { get; set; }
11.
12.         // constructeurs
13.         public PamException()
14.         {
15.         }
16.
17.         public PamException(int Code)
18.             : base()
19.         {
20.             this.Code = Code;
21.         }
22.
23.         public PamException(string message, int Code)
24.             : base(message)
25.         {
26.             this.Code = Code;
27.         }
28.
29.         public PamException(string message, Exception ex, int Code)
30.             : base(message, ex)
31.         {
32.             this.Code = Code;
33.         }
34.     }
35. }
```

- ligne 6 : la classe dérive de la classe [Exception] ;
- ligne 10 : elle a une propriété publique [Code] qui est un code d'erreur ;
- nous utiliserons dans notre application deux sortes de constructeur :
  - celui des lignes 23-27 qu'on peut utiliser comme montré ci-dessous :

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

- ou celui des lignes 29-33 destiné à faire remonter une exception survenue en l'encapsulant dans une exception de type [PamException] :

```

try{
...
}catch (IOException ex){
    // on encapsule l'exception ex
    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.

## 2.7.5 Implémentation de la couche [métier]

L'interface [IPamMetier] sera implémentée par la classe [PamMetier] suivante :

```

1.  using System;
2.  using Pam.Metier.Entites;
3.  using System.Collections.Generic;
4.
5.  namespace Pam.Metier.Service
6.  {
```

```

7.  public class PamMetier : IPamMetier
8.  {
9.      // liste des employés en cache
10.     public Employe[] Employes { get; set; }
11.     // employés indexés par leur n° SS
12.     private IDictionary<string, Employe> dicEmployes = new Dictionary<string, Employe>();
13.
14.     // liste des employés
15.     public Employe[] GetAllIdentitesEmployes()
16.     {
17.         ...
18.         // on rend la liste des employés
19.         return Employes;
20.     }
21.
22.     // calcul salaire
23.     public FeuilleSalaire GetSalaire(string ss, double heuresTravaillées, int joursTravaillés)
24.     {
25.         ...
26.     }
27. }
```

- ligne 7 : la classe [PamMetier] implémente l'interface [IPamMetier] ;
- ligne 10 : la classe [PamMetier] maintient la liste des employés en cache ;
- ligne 12 : un dictionnaire qui associe un employé à son n° de sécurité sociale ;
- lignes 15-20 : la méthode qui rend la liste des employés ;
- lignes 23-26 : la méthode qui calcule le salaire d'un employé.

La méthode [GetAllIdentitesEmploye] est la suivante :

```

1.  // liste des employés
2.  public Employe[] GetAllIdentitesEmployes()
3.  {
4.      if (Employes == null)
5.      {
6.          // on crée un tableau de trois employés
7.          Employes = new Employe[3];
8.          Employes[0] = new Employe()
9.          {
10.              SS = "254104940426058",
11.              Nom = "Jouveinal",
12.              Prenom = "Marie",
13.              Adresse = "5 rue des oiseaux",
14.              Ville = "St Corentin",
15.              CodePostal = "49203",
16.              Indemnites = new Indemnites() { Indice = 2, BaseHeure = 2.1, EntretienJour = 2.1, RepasJour = 3.1,
17.              IndemnitesCp = 15 }
18.          };
19.          dicEmployes.Add(Employes[0].SS, Employes[0]);
20.          Employes[1] = new Employe()
21.          {
22.              SS = "260124402111742",
23.              Nom = "Laverti",
24.              Prenom = "Justine",
25.              Adresse = "La brûlerie",
26.              Ville = "St Marcel",
27.              CodePostal = "49014",
28.              Indemnites = new Indemnites() { Indice = 1, BaseHeure = 1.93, EntretienJour = 2, RepasJour = 3, IndemnitesCp =
29.              = 12 }
30.          };
31.          dicEmployes.Add(Employes[1].SS, Employes[1]);
32.          // un employé fictif qui ne sera pas mis dans le dictionnaire
33.          // afin de simuler un employé inexistant
34.          Employes[2] = new Employe()
35.          {
36.              SS = "XX",
37.              Nom = "X",
38.              Prenom = "X",
39.              Adresse = "X",
40.              Ville = "X",
41.              CodePostal = "X",
42.              Indemnites = new Indemnites() { Indice = 0, BaseHeure = 0, EntretienJour = 0, RepasJour = 0, IndemnitesCp =
43.              0 }
44.          };
45.      }
46.      // on rend la liste des employés
47.      return Employes;
48.  }
```

- ligne 4 : on regarde si la liste des employés n'a pas déjà été construite ;
- ligne 7 : si ce n'est pas le cas, on crée un tableau de trois employés ;
- lignes 8-17 : le premier employé ;

- ligne 18 : il est mis dans le dictionnaire ;
- lignes 19-28 : le second employé ;
- ligne 29 : il est mis dans le dictionnaire ;
- lignes 32-42 : le troisième employé. Celui-ci n'est pas mis dans le dictionnaire pour une raison qu'on va expliquer.

La méthode [GetSalaire] sera la suivante :

```

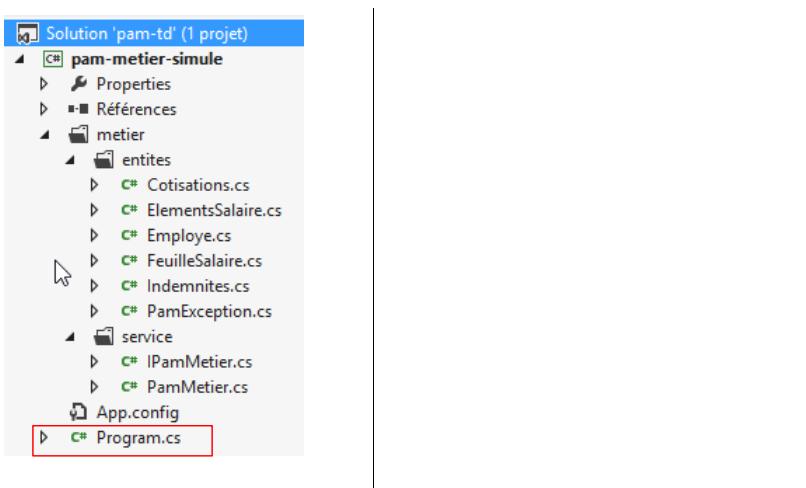
1.    // calcul salaire
2.    public FeuilleSalaire GetSalaire(string ss, double heuresTravaillées, int joursTravaillés)
3.    {
4.        // on récupère l'employé de n° SS
5.        Employe e = dicEmployes.ContainsKey(ss) ? dicEmployes[ss] : null;
6.        // existe ?
7.        if (e == null)
8.        {
9.            throw new PamException(string.Format("L'employé de n° SS [{0}] n'existe pas", ss), 10);
10.       }
11.       // on rend une feuille de salaire fictive
12.       return new FeuilleSalaire()
13.       {
14.           Employe = e,
15.           Cotisations = new Cotisations() { CsgRds = 3.49, Csgd = 6.15, Secu = 9.38, Retraite = 7.88 },
16.           ElementsSalaire = new ElementsSalaire() { CotisationsSociales = 100, IndemnitesEntretien = 100,
17.               IndemnitesRepas = 100, SalaireBase = 100, SalaireNet = 100 }
18.       };

```

- ligne 2 : la méthode reçoit le n° SS de l'employé dont on veut calculer le salaire, son nombre d'heures travaillées et son nombre de jours travaillés ;
- ligne 5 : on va chercher l'employé dans le dictionnaire. On se rappelle que l'un d'eux n'y est pas ;
- lignes 7-10 : si l'employé n'est pas trouvé, une exception [PamException] est lancée ;
- lignes 12-17 : on rend une feuille de salaire fictive.

## 2.7.6 Le test console de la couche [métier]

Le projet de la couche [métier] est actuellement le suivant :



La classe [Program] ci-dessus va tester les méthodes de l'interface [IPamMetier]. Un exemple basique pourrait être le suivant :

```

1.  using Pam.Metier.Entites;
2.  using Pam.Metier.Service;
3.  using System;
4.
5.  namespace Pam.Metier.Tests
6.  {
7.      class Program
8.      {
9.          public static void Main()
10.         {
11.             // instanciation couche [métier]
12.             IPamMetier pamMetier = new PamMetier();
13.             // liste des employés
14.             Employe[] employes = pamMetier.GetAllIdentitesEmployes();
15.             Console.WriteLine("Liste des employés-----");
16.             foreach (Employe e in employes)

```

```

17.     {
18.         Console.WriteLine(e);
19.     }
20. // calculs de feuilles de salaire
21. Console.WriteLine("Calculs de feuilles de salaire-----");
22. Console.WriteLine(pamMetier.GetSalaire(employes[0].SS, 30, 5));
23. Console.WriteLine(pamMetier.GetSalaire(employes[1].SS, 150, 20));
24. try
25. {
26.     Console.WriteLine(pamMetier.GetSalaire(employes[2].SS, 150, 20));
27. }
28. catch (PamException ex)
29. {
30.     Console.WriteLine(string.Format("PamException : {0}", ex.Message));
31. }
32. }
33. }
34. }

```

- ligne 12 : instanciation de la couche [métier] ;
- lignes 14-19 : test de la méthode [GetAllIdentitesEmploye] de l'interface [IPamMetier] ;
- lignes 21-31 : test de la méthode [GetSalaire] de l'interface [IPamMetier].

L'exécution de ce programme console donne les résultats suivants :

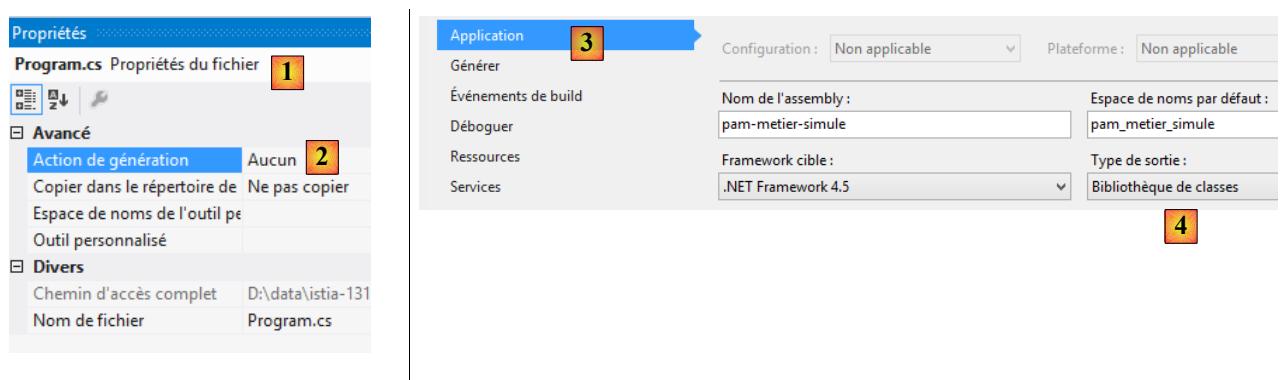
```

1. Liste des employés-----
2. Employé[254104940426058,Jouveinal,Marie,5 rue des oiseaux,St Corentin,49203]
3. Employé[260124402111742,Laverti,Justine,La brûlerie,St Marcel,49014]
4. Employé[XX,X,X,X,X]
5. Calculs de feuilles de salaire-----
6. [Employé[254104940426058,Jouveinal,Marie,5 rue des oiseaux,St Corentin,49203],Co
7. tisations[3,49,6,15,9,38,7,88],[100 : 100 : 100 : 100]]
8. [Employé[260124402111742,Laverti,Justine,La brûlerie,St Marcel,49014],Cotisation
9. s[3,49,6,15,9,38,7,88],[100 : 100 : 100 : 100]]
10. PamException : L'employé de n° SS [XX] n'existe pas

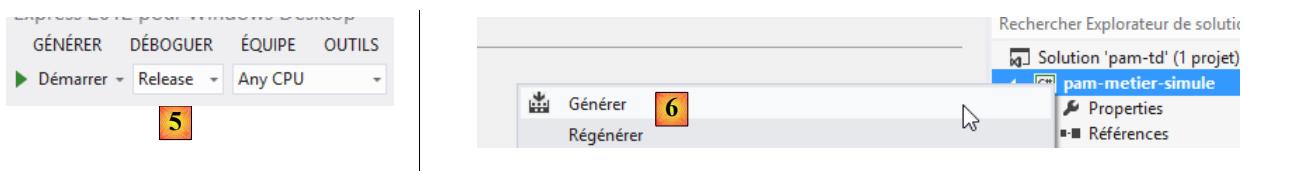
```

Le lecteur est invité à faire le lien entre ces résultats et le code exécuté.

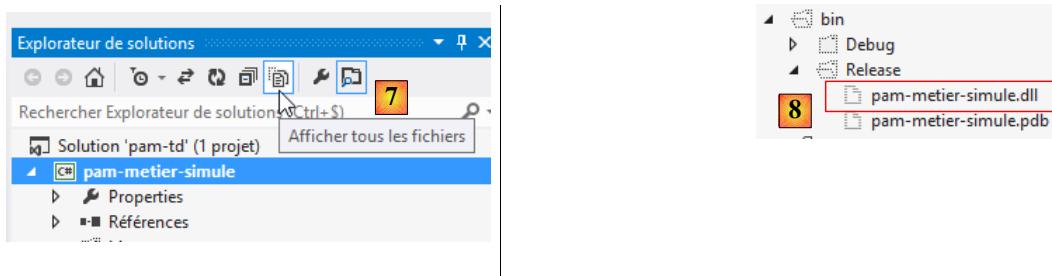
Afin de pouvoir utiliser ce projet dans le projet web que nous allons construire, nous en faisons une bibliothèque de classes :



- en [1], dans les propriétés du fichier [Program.cs] ;
- en [2], on indique que le fichier ne fera pas partie de l'assembly généré ;
- en [3, 4], dans les **propriétés** du projet [pam-metier-simule], dans l'option [Application] [3], on indique [4] que la génération doit fournir une bibliothèque de classes (sous la forme d'une DLL).



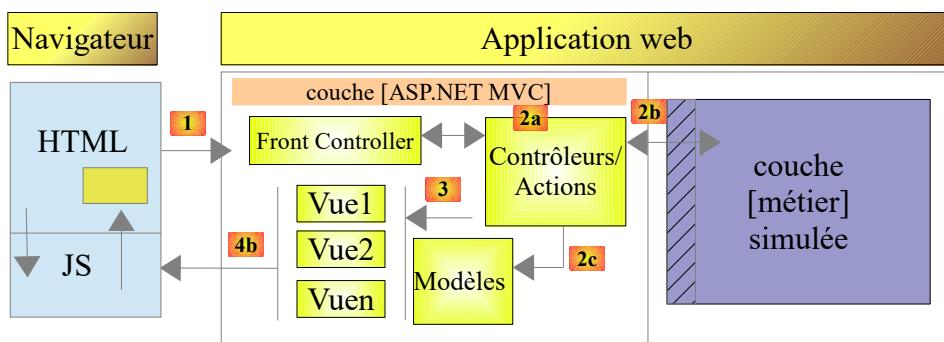
- en [5], on demande un assembly de type [Release]. L'autre type est [Debug]. L'assembly contient alors des informations facilitant le débogage ;
- en [6], on génère le projet [pam-metier-simule] ;



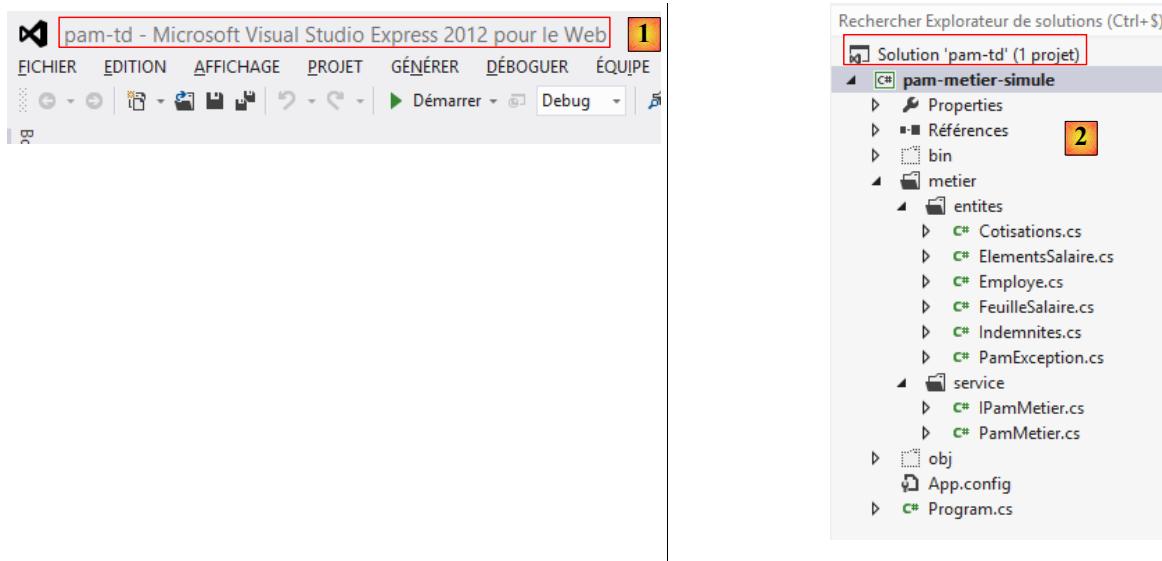
- en [7], on fait afficher tous les fichiers de la solution ;
- en [8], dans le dossier [bin / Release], la DLL de notre projet.

## 2.8 Étape 2 : mise en place de l'application web

Dans la solution Visual Studio précédente nous allons créer le projet pour la couche web MVC.

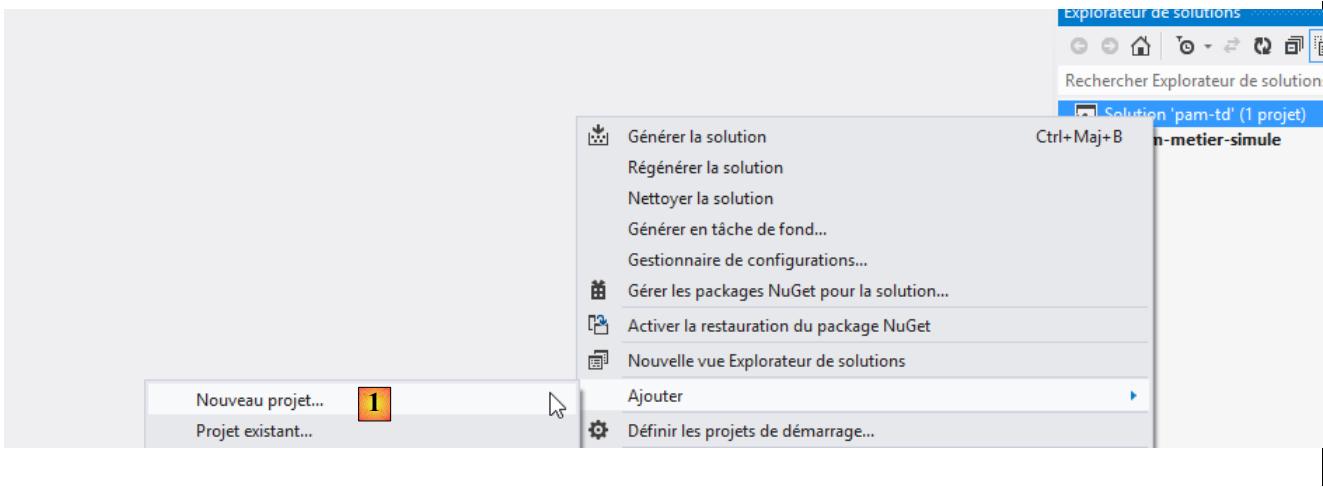


Avec Visual Studio Express **pour le web**, nous ouvrons la solution [pam-td] créée précédemment avec Visual Studio Express **pour le bureau**.

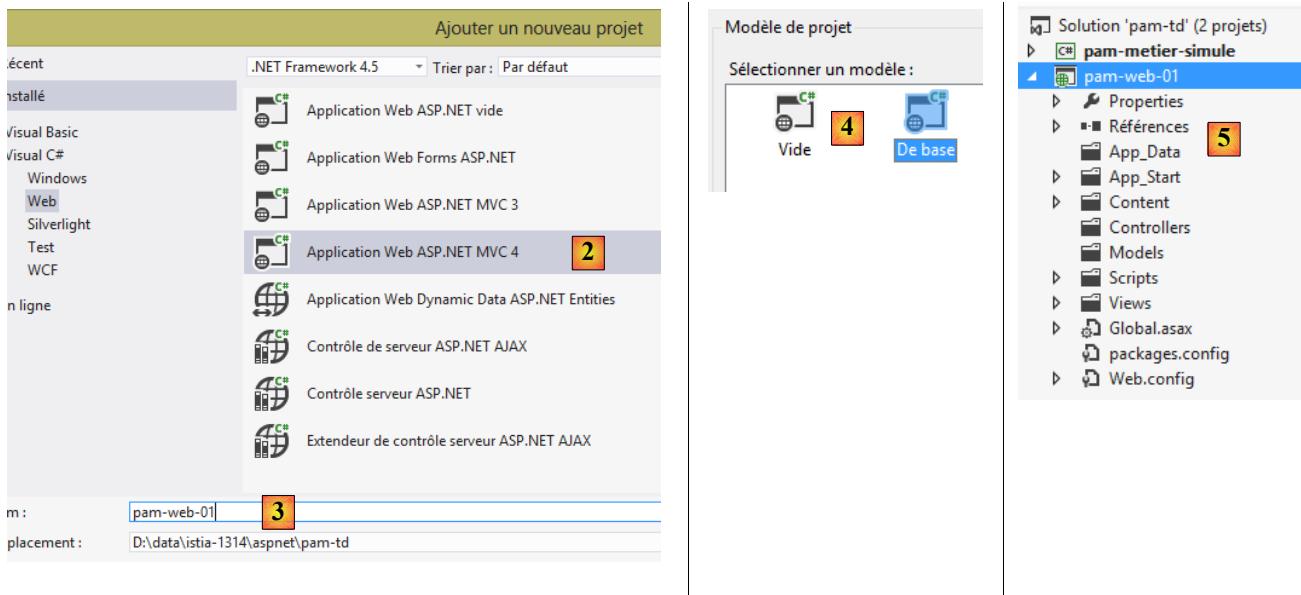


- en [1], la solution [pam-td] a été chargée dans Visual Studio Express pour le Web ;
- en [2], la solution et le projet pour la couche [métier] simulée que nous venons de créer.

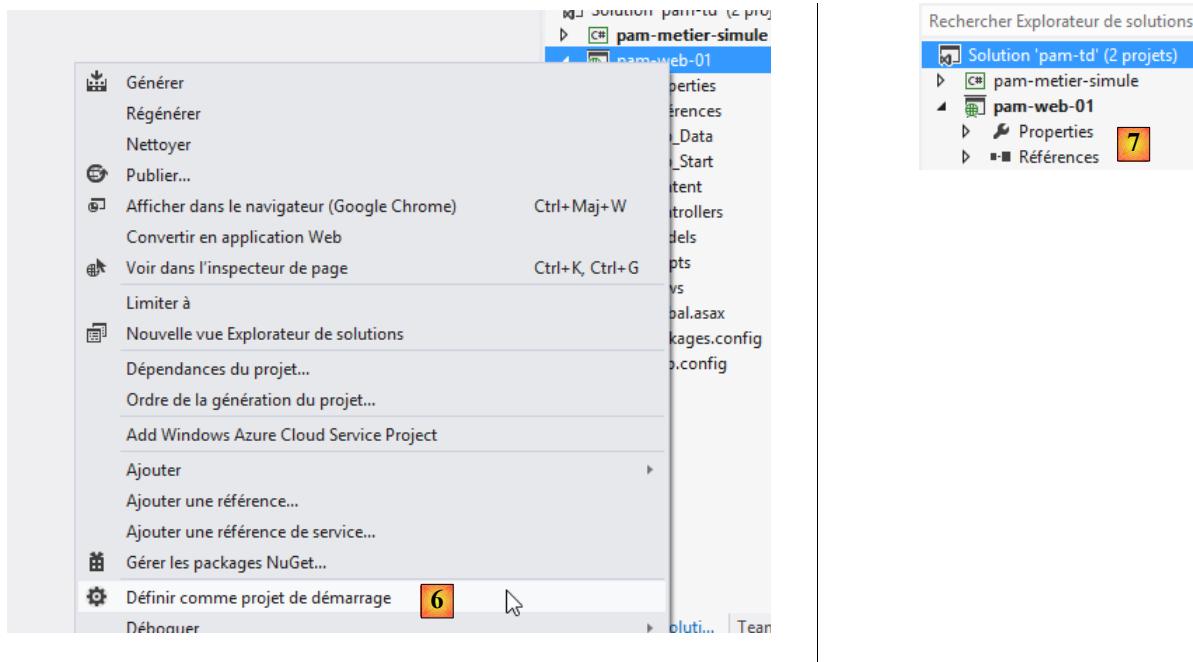
Dans cette nouvelle étape, nous allons créer le squelette de l'application web.



- en [1], nous ajoutons un nouveau projet à la solution [pam-td] ;

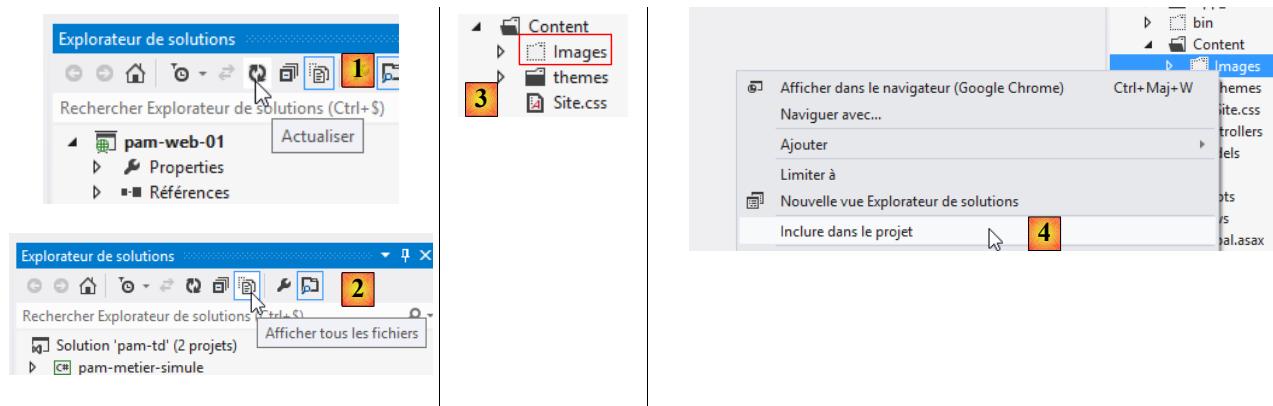


- en [2], on choisit un projet ASP.NET MVC 4 ;
- nommé [pam-web-01] [3] ;
- en [4], on choisit le modèle ASP.NET MVC de base ;
- en [5], le projet créé ;



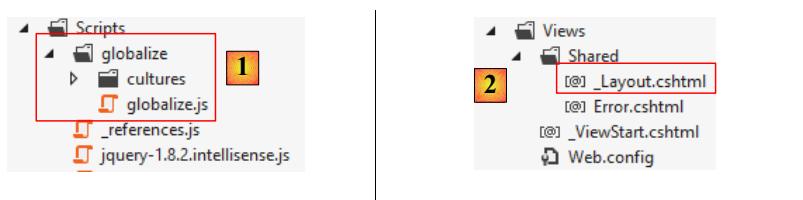
- en [6], on fait du nouveau projet, le projet de démarrage de la solution, celui qui sera exécuté lorsqu'on fera [Ctrl-F5] ;
- en [7], le nom du nouveau projet est passé en gras, indiquant qu'il est le projet de démarrage de la solution.

Maintenant, on remplace avec l'explorateur Windows le dossier [Content] du projet par le dossier [étudiedecas-support / web / Content]. Ceci fait, il faut inclure les nouveaux fichiers dans le projet [pam-web-01]. On procèdera ainsi :



- en [1], on rafraîchit la solution ;
- en [2], on fait afficher tous les fichiers de la solution ;
- en [3], apparaît un dossier [Images] ;
- qu'on inclut dans le projet en [4].

Dans le dossier [Scripts], ajoutez les scripts JQuery Globalization [1] nécessaires à la validation côté client (cf page 157).



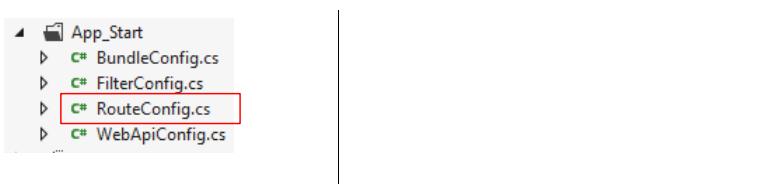
La page maître [`_Layout.cshtml`] [2] aura le contenu suivant :

```
1.  <!DOCTYPE html>
2.  <html>
3.  <head>
4.    <title>@ViewBag.Title</title>
5.    <meta charset="utf-8" />
6.    <meta name="viewport" content="width=device-width" />
7.    <link rel="stylesheet" href("~/Content/Site.css" />
8.    <script type="text/javascript" src("~/Scripts/jquery-1.8.2.min.js")></script>
9.    <script type="text/javascript" src "~/Scripts/jquery.validate.min.js"></script>
10.   <script type="text/javascript" src "~/Scripts/jquery.validate.unobtrusive.min.js"></script>
11.   <script type="text/javascript" src "~/Scripts/globalize/globalize.js"></script>
12.   <script type="text/javascript" src "~/Scripts/globalize/cultures/globalize.culture.fr-FR.js"></script>
13.   <script type="text/javascript" src "~/Scripts/jquery.unobtrusive-ajax.js"></script>
14.   <script type="text/javascript" src "~/Scripts/myScripts.js"></script>
15. </head>
16. <body>
17.   <table>
18.     <tbody>
19.       <tr>
20.         <td>
21.           <h2>Simulateur de calcul de paie</h2>
22.         </td>
23.         <td style="width: 20px">
24.           <img id="loading" style="display: none" src "~/Content/images/indicator.gif" />
25.         </td>
26.         <td>
27.           <a id="lnkFaireSimulation" href="javascript:faireSimulation()">| Faire la simulation<br />
28.           </a>
29.           <a id="lnkEffacerSimulation" href="javascript:effacerSimulation()">| Effacer la simulation<br />
30.           </a>
31.           <a id="lnkVoirSimulations" href="javascript:voirSimulations()">| Voir les simulations<br />
32.           </a>
33.           <a id="lnkRetourFormulaire" href="javascript:retourFormulaire()">| Retour au formulaire de simulation<br />
34.           </a>
35.           <a id="lnkEnregistrerSimulation" href="javascript:enregistrerSimulation()">| Enregistrer la simulation<br />
36.           </a>
37.           <a id="lnkTerminerSession" href="javascript:terminerSession()">| Terminer la session<br />
38.           </a>
39.         </td>
40.       </tbody>
41.     </table>
42.     <hr />
43.     <div id="content">
44.       @RenderBody()
45.     </div>
46.   </body>
47. </html>
```

Note : ligne 8, adaptez la version de jQuery à celle de votre version de Visual Studio.

- ligne 7 : référence sur la feuille de style de l'application ;
- lignes 8-10 : références sur les scripts nécessaires à la validation côté client ;
- lignes 11-12 : références sur les scripts nécessaires à la saisie des nombres réels français avec virgule ;
- ligne 13 : référence sur les scripts nécessaires au mode Ajax ;
- ligne 14 : les scripts propres à l'application ;
- ligne 24 : l'image d'attente de fin des appels Ajax ;
- lignes 26-39 : six liens Javascript ;
- ligne 43 : la section où viendront s'afficher les différentes vues de l'application ;
- ligne 44 : le corps des différentes vues de l'application.

Ensuite, on modifiera la route par défaut de l'application :



Le fichier [RouteConfig] aura le contenu suivant :

```
1.  using System.Web.Mvc;
2.  using System.Web.Routing;
3.
```

```

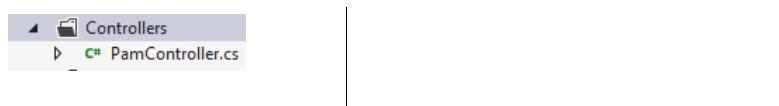
4.  namespace Pam.Web_01
5.  {
6.    public class RouteConfig
7.    {
8.      public static void RegisterRoutes(RouteCollection routes)
9.      {
10.        routes.IgnoreRoute("{resource}.axd/{*pathInfo}");
11.
12.        routes.MapRoute(
13.          name: "Default",
14.          url: "{controller}/{action}",
15.          defaults: new { controller = "Pam", action = "Index" }
16.        );
17.      }
18.    }
19.  }

```

- ligne 14 : les URL auront la forme [{controller}/{action}] ;
- ligne 15 : en l'absence d'action, c'est l'action [Index] qui sera utilisée. En l'absence de contrôleur, c'est le contrôleur [Pam] qui sera utilisé.

De cette configuration, il résulte que l'URL [/] est équivalente à l'URL [/Pam/Index]. Comme notre application est de type APU, l'URL [/] sera l'unique URL de celle-ci.

Créez le contrôleur [Pam] (cf page 50) :



Modifiez le contrôleur [PamController] de la façon suivante :

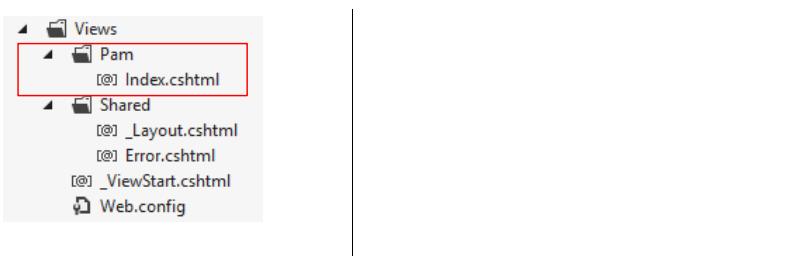
```

1.  using System.Web.Mvc;
2.
3.  namespace Pam.Web.Controllers
4.  {
5.    public class PamController : Controller
6.    {
7.      [HttpGet]
8.      public ViewResult Index()
9.      {
10.        return View();
11.      }
12.
13.    }
14.  }

```

- ligne 3 : nous mettons le contrôleur dans l'espace de noms [Pam.Web.Controllers] ;
- ligne 7 : l'action [Index] traitera uniquement la commande HTTP GET ;
- ligne 8 : on rend un type [ViewResult] plutôt qu'un type [ActionResult].

Créez maintenant la vue [Index.cshtml] affichée par l'action [Index] ci-dessus (cf page 93) :



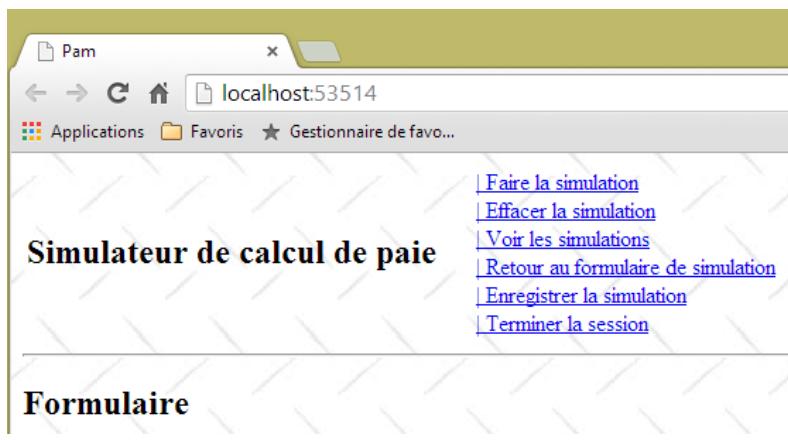
Modifiez [Index.cshtml] de la façon suivante :

```

1.  @{
2.    ViewBag.Title = "Pam";
3.  }
4.  <h2>Formulaire</h2>

```

Exécutez l'application par [Ctrl-F5]. Vous devez obtenir la page suivante :



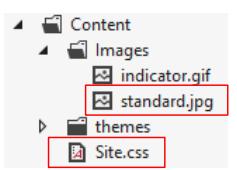
**Travail :** Expliquez ce qui s'est passé.

L'application utilise une feuille de style référencée dans la page maître [`_Layout.cshtml`] :

```
<link rel="stylesheet" href("~/Content/Site.css" />
```

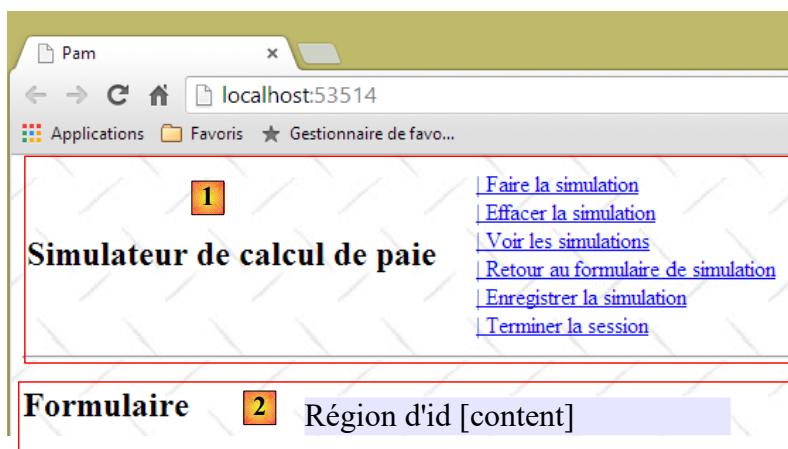
La feuille de style [/Content/Site.css] définit une image de fond pour les pages de l'application :

```
1. body {
2.   background-image: url("/Content/Images/standard.jpg");
3. }
```



## 2.9 Étape 3 : mise en place du modèle APU

Nous voulons écrire une application suivant le modèle APU (Application à Page Unique) décrit au paragraphe 1.7.5, page 200 ainsi qu'au paragraphe 1.7.6, page 204. La page unique est celle chargée par le navigateur au démarrage de l'application :



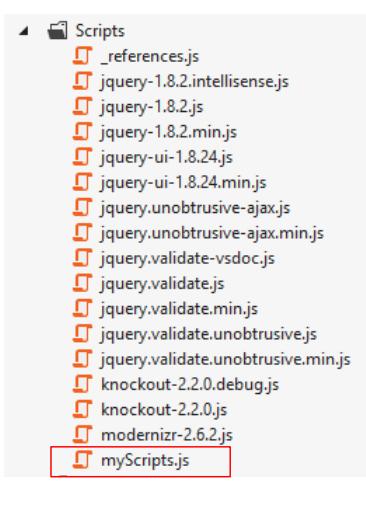
- la partie [1] ci-dessus est la partie fixe de la page unique. Nous avons vu qu'elle était fournie par la page maître [`_Layout.cshtml`] ;
- la partie [2] est la partie variable de la page unique. Elle s'inscrit dans la région d'**id** [content] de la page maître [`_Layout.cshtml`] :

```

1.  <!DOCTYPE html>
2.  <html>
3.  <head>
4.    <title>@ViewBag.Title</title>
5.    ...
6.    <script type="text/javascript" src="~/Scripts/myScripts.js"></script>
7.  </head>
8.  <body>
9.    <table>
10.   ...
11.   </table>
12.   <hr />
13.   <div id="content">
14.     @RenderBody()
15.   </div>
16. </body>
17. </html>

```

Les différents fragments de page de l'application vont venir s'afficher dans la région d'**id** [content] de la ligne 13. Ils seront affichés via des appels Ajax. Les scripts Javascript exécutant ces appels sont dans la fichier [myScripts.js] référencé ligne 6. Créez ce fichier dont nous allons avoir besoin :



Nous suivons désormais le modèle APU décrit au paragraphe 1.7.6, page 204. Relisez ce paragraphe si vous l'avez oublié. Nous allons maintenant mettre en place les différents fragments de page affichés par l'application.

### 2.9.1 Les outils du développeur Javascript

Nous rappelons qu'avec le navigateur Chrome, vous disposez d'une palette d'outils pour déboguer le HTML, CSS, Javascript de vos pages. Ces outils ont été présentés partiellement au paragraphe 1.7.2, page 181. Dans le modèle APU, les navigateurs gardent en cache les scripts Javascript référencés par la première page de l'application. Aussi, faut-il penser à vider ce cache lorsque vous modifiez vos scripts, sinon les modifications peuvent ne pas être prises en compte. Voici comment faire avec Chrome :

- faire [Ctrl-Maj-I] pour afficher l'environnement de développement



- cliquer sur l'icône [1] en bas à droite de la fenêtre de développement ;

- puis cocher l'option [2] qui inhibe le cache en mode développement.

## 2.9.2 Utilisation d'une vue partielle pour afficher le formulaire

Le formulaire de saisie est l'un des fragments affichés par l'application. Pour l'instant ce formulaire est affiché par la vue [Index.cshtml] qui est une vue complète :

```
1. @{
2.   ViewBag.Title = "Pam";
3. }
4. <h2>Formulaire</h2>
```

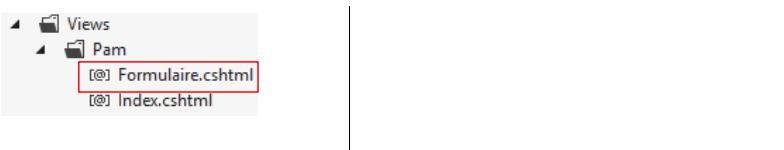
Cette vue est affichée par l'action [Index] :

```
1. [HttpGet]
2. public ViewResult Index()
3. {
4.   return View();
5. }
```

Ligne 4 ci-dessus, c'est bien une vue [View] et non une vue partielle [PartialView] qui est affichée. Nous avons besoin d'une vue partielle pour le formulaire qui sera un fragment de page. Nous faisons évoluer la vue [Index.cshtml] de la façon suivante :

```
1. @{
2.   ViewBag.Title = "Pam";
3. }
4. @Html.Partial("Formulaire")
```

Ligne 4, le formulaire ne fait plus partie de la page [Index.cshtml]. Il est désormais logé dans une vue partielle [Formulaire.cshtml] :



Le code de [Formulaire.cshtml] est simplement le suivant :

```
1. <h2>Formulaire</h2>
```

Faites ces modifications et vérifiez que vous obtenez toujours la vue suivante au démarrage de l'application :



## 2.9.3 L'appel Ajax [faireSimulation]

Nous nous intéressons au fragment affiché lorsque l'utilisateur clique sur le lien [Faire la simulation] :

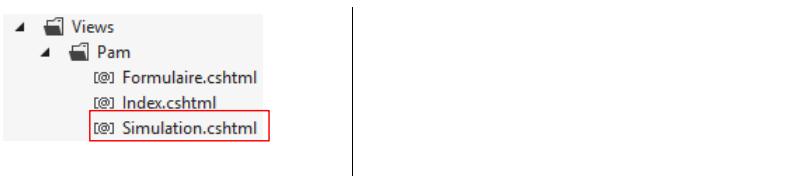
- en [1], l'utilisateur clique sur le lien [Faire la simulation] ;
- en [2], la simulation apparaît sous le formulaire.

Nous faisons évoluer de la façon suivante la vue partielle [Formulaire.cshtml] qui affiche le formulaire :

```
1. <h2>Formulaire</h2>
2. <div id="simulation" />
```

Ligne 3, nous créons une région d'**id** [simulation] pour loger le fragment de la simulation.

Nous créons la vue partielle [Simulation.cshtml] suivante :



Le contenu de la vue [Simulation.cshtml] est le suivant :

```
1. <hr />
2. <h2>Simulation</h2>
```

Il nous faut maintenant écrire le code Javascript qui gère le clic sur le lien [Faire la simulation]. Nous allons suivre la démarche exposée au paragraphe 1.7.6.5, page 211. Tout d'abord, regardons le code HTML du lien dans [\_Layout.cshtml] :

```
1. <a id="lnkFaireSimulation" href="javascript:faireSimulation()">| Faire la simulation<br />
2. </a>
```

On voit qu'un clic sur le lien [Faire la simulation] va lancer l'exécution de la fonction JS [faireSimulation]. Cette fonction va être écrite dans le fichier [myScripts.js] ainsi que les autres fonctions JS nécessaires à l'application :

```
1. // variables globales
2. var loading;
3. var content;
4.
5. function faireSimulation() {
6.   // on fait un appel Ajax à la main
7.   ...
8. }
9.
10. function effacerSimulation() {
11.   // on efface les saisies du formulaire
12.   ...
13. }
14.
15. function enregistrerSimulation() {
16.   // on fait un appel Ajax à la main
17.   ...
18. }
19.
20. function voirSimulations() {
```

```

21. // on fait un appel Ajax à la main
22. ...
23. }
24.
25. function retourFormulaire() {
26.   // on fait un appel Ajax à la main
27.   ...
28. }
29.
30. function terminerSession() {
31.   ...
32. }
33.
34. // au chargement du document
35. $(document).ready(function () {
36.   // on récupère les références des différents composants de la page
37.   loading = $("#loading");
38.   content = $("#content");
39. });

```

- lignes 35-39 : la fonction JQuery exécutée au démarrage de l'application ;
- lignes 37-38 : on initialise les variables globales des lignes 2 et 3.

On rappelle que les éléments d'**id** [loading] et [content] sont définis dans la page maître [`_Layout.cshtml`] (lignes 14 et 21 ci-dessous) :

```

1. <!DOCTYPE html>
2. <html>
3.   <head>
4.   ...
5.   </head>
6.   <body>
7.     <table>
8.       <tbody>
9.         <tr>
10.           <td>
11.             <h2>Simulateur de calcul de paie</h2>
12.           </td>
13.           <td style="width: 20px">
14.             
15.           </td>
16. ...
17.           </td>
18.         </tbody>
19.       </table>
20.       <hr />
21.       <div id="content">
22.         @RenderBody()
23.       </div>
24.     </body>
25.   </html>

```

---

**Travail** : en suivant la démarche exposée au paragraphe 1.7.6.5, page 211, écrivez la fonction JS [faireSimulation]. Celle-ci émettra un appel Ajax de type POST vers l'action [/Pam/FaireSimulation]. Il n'y aura pas de données postées pour l'instant. L'action [/Pam/FaireSimulation] renverra la vue partielle [Simulation.cshtml] à la fonction JS [faireSimulation] qui placera alors ce flux HTML dans la région d'**id** [simulation] du formulaire.

---

Testez le lien [Faire la simulation] de votre application.

## 2.9.4 L'appel Ajax [enregistrerSimulation]

Le lien [Enregistrer la simulation] est défini de la façon suivante dans [`_Layout.cshtml`] :

```

1. <a id="lnkEnregistrerSimulation" href="javascript:enregistrerSimulation()">| Enregistrer la simulation<br />
2. </a>

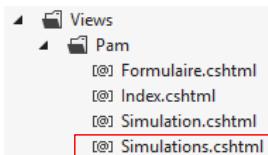
```

---

**Travail** : en suivant la démarche précédente, écrivez la fonction JS [enregistrerSimulation]. Celle-ci émettra un appel Ajax de type POST vers l'action [/Pam/EnregistrerSimulation]. Il n'y aura pas de données postées pour l'instant. L'action [/Pam/EnregistrerSimulation] renverra la vue partielle [Simulations.cshtml] à la fonction JS [enregistrerSimulation] qui placera alors ce flux HTML dans la région d'**id** [content] de la page maître.

---

La vue [Simulations.cshtml] est la suivante :



Son contenu est le suivant :

1. <h2>Simulations</h2>

Voici un exemple d'exécution :

## 2.9.5 L'appel Ajax [voirSimulations]

Le lien [Voir les simulations] est défini de la façon suivante dans [\_Layout.cshtml] :

1. <a id="lnkVoirSimulations" href="javascript:voirSimulations()">| Voir les simulations<br />
2. </a>

**Travail** : en suivant la démarche précédente, écrivez la fonction JS [voirSimulations]. Celle-ci émettra un appel Ajax de type POST vers l'action [/Pam/VoirSimulations]. Il n'y aura pas de données postées pour l'instant. L'action [/Pam/VoirSimulations] renverra la vue partielle [Simulations.cshtml] à la fonction JS [voirSimulations] qui placera alors ce flux HTML dans la région d'**id** [content] de la page maître.

La vue [Simulations.cshtml] est celle déjà utilisée dans la question précédente.

Voici un exemple d'exécution :

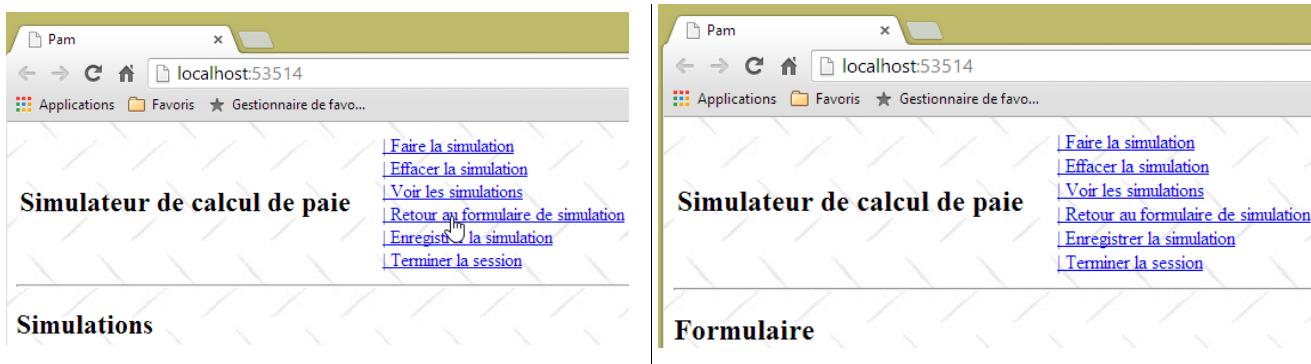
## 2.9.6 L'appel Ajax [retourFormulaire]

Le lien [Retour au formulaire de simulation] est défini de la façon suivante dans [\_Layout.cshtml] :

```
1. <a id="lnkRetourFormulaire" href="javascript:retourFormulaire()">| Retour au formulaire de simulation<br />
2. </a>
```

**Travail** : en suivant la démarche précédente, écrivez la fonction JS [retourFormulaire]. Celle-ci émettra un appel Ajax de type POST vers l'action [/Pam/Formulaire]. Il n'y aura pas de données postées pour l'instant. L'action [/Pam/Formulaire] renverra la vue partielle [Formulaire.cshtml] à la fonction JS [retourFormulaire] qui placera alors ce flux HTML dans la région d'**id** [content] de la page maître.

La vue [Formulaire .cshtml] a déjà été définie. Voici un exemple d'exécution :



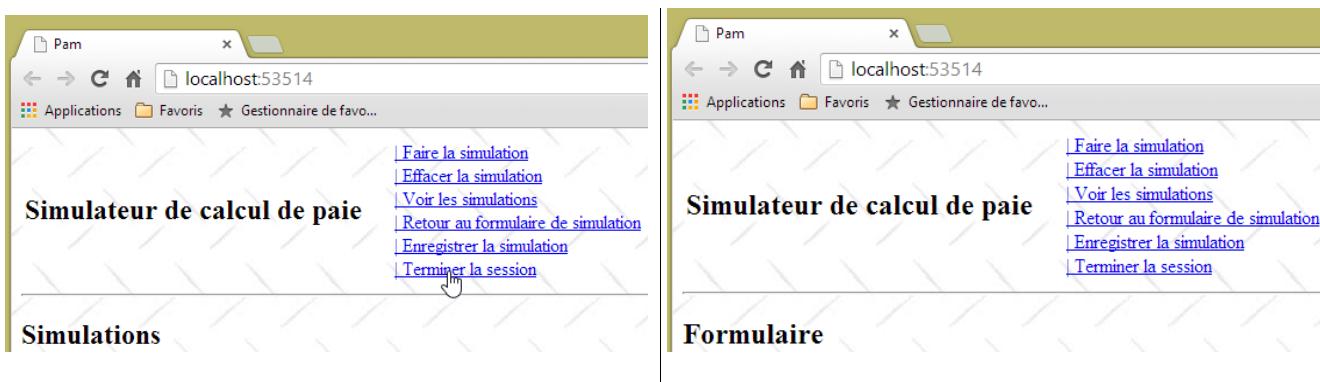
## 2.9.7 L'appel Ajax [terminerSession]

Le lien [Terminer la session] est défini de la façon suivante dans [\_Layout.cshtml] :

```
1. <a id="lnkTerminerSession" href="javascript:terminerSession()">| Terminer la session<br />
2. </a>
```

**Travail** : en suivant la démarche précédente, écrivez la fonction JS [terminerSession]. Celle-ci émettra un appel Ajax de type POST vers l'action [/Pam/TerminerSession]. Il n'y aura pas de données postées pour l'instant. L'action [/Pam/TerminerSession] renverra la vue partielle [Formulaire.cshtml] à la fonction JS [terminerSession] qui placera alors ce flux HTML dans la région d'**id** [content] de la page maître.

Voici un exemple d'exécution :



## 2.9.8 La fonction JS [effacerSimulation]

Le lien [Effacer la simulation] est défini de la façon suivante dans [\_Layout.cshtml] :

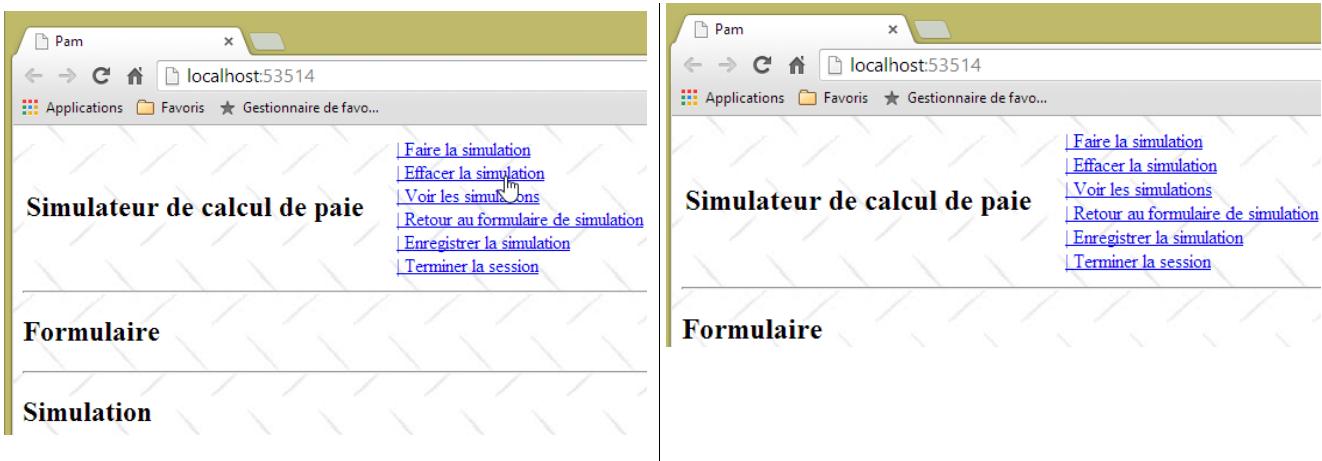
```
1. <a id="lnkEffacerSimulation" href="javascript:effacerSimulation()">| Effacer la simulation<br />
2. </a>
```

La fonction JS [effacerSimulation] a pour but :

- de cacher le fragment [Simulation] s'il existe ;
- de remettre les champs de saisie du formulaire dans l'état où ils étaient au chargement initial de l'application (lorsqu'il y aura des champs de saisie - pour l'instant il n'y en a pas).

**Travail** : écrivez la fonction JS [effacerSimulation]. Il n'y a pas ici d'appel Ajax. Ce qui se passe est interne au navigateur et n'implique pas le serveur.

Voici un exemple d'exécution :



## 2.9.9 Gestion de la navigation entre écrans

Pour l'instant, les liens sont toujours affichés. Nous allons maintenant gérer leur affichage avec une fonction Javascript. Rappelons tout d'abord le code des six liens Javascript dans [\_Layout.cshtml] :

```

1. <a id="lnkFaireSimulation" href="javascript:faireSimulation()">| Faire la simulation<br />
2. </a>
3. <a id="lnkEffacerSimulation" href="javascript:effacerSimulation()">| Effacer la simulation<br />
4. </a>
5. <a id="lnkVoirSimulations" href="javascript:voirSimulations()">| Voir les simulations<br />
6. </a>
7. <a id="lnkRetourFormulaire" href="javascript:retourFormulaire()">| Retour au formulaire de simulation<br />
8. </a>
9. <a id="lnkEnregistrerSimulation" href="javascript:enregistrerSimulation()">| Enregistrer la simulation<br />
10. </a>
11. <a id="lnkTerminerSession" href="javascript:terminerSession()">| Terminer la session<br />
12. </a>

```

Tous les liens ont un attribut [id] qui va nous permettre de les gérer en Javascript. Nous modifions la méthode JS exécutée au chargement de la page de la façon suivante :

```

1. // variables globales
2. var loading;
3. var content;
4. var lnkFaireSimulation;
5. var lnkEffacerSimulation
6. var lnkEnregistrerSimulation;
7. var lnkTerminerSession;
8. var lnkVoirSimulations;
9. var lnkRetourFormulaire;
10. var options;
11.
12. ...
13. // au chargement du document
14. $(document).ready(function () {
15.   // on récupère les références des différents composants de la page
16.   loading = $("#loading");
17.   content = $("#content");
18.   // les liens du menu
19.   lnkFaireSimulation = $("#lnkFaireSimulation");
20.   lnkEffacerSimulation = $("#lnkEffacerSimulation");
21.   lnkEnregistrerSimulation = $("#lnkEnregistrerSimulation");

```

```

22.   lnkVoirSimulations = $("#lnkVoirSimulations");
23.   lnkTerminerSession = $("#lnkTerminerSession");
24.   lnkRetourFormulaire = $("#lnkRetourFormulaire");
25.   // on les met dans un tableau
26.   options = [lnkFaireSimulation, lnkEffacerSimulation, lnkEnregistrerSimulation, lnkVoirSimulations, lnkTerminerSession,
    lnkRetourFormulaire];
27.   // on cache certains éléments de la page
28.   loading.hide();
29.   // on fixe le menu
30.   setMenu([lnkFaireSimulation, lnkVoirSimulations, lnkTerminerSession]);
31. });
32.

```

- lignes 19-24 : on récupère les références sur les six liens. Ces références sont définies en variables globales aux lignes 4-9 ;
- ligne 26 : le tableau [options] est initialisé avec les six références. Ce tableau est défini en variable globale ligne 10 ;
- ligne 28 : on cache l'image animée de l'attente de fin des appels Ajax ;
- ligne 30 : on affiche les liens [lnkFaireSimulation, lnkVoirSimulations, lnkTerminerSession]. Les autres seront cachés.

La fonction JS [setMenu] est la suivante :

```

1.   function setMenu(show) {
2.     // on affiche les liens du tableau [show]
3.     ...
4. }

```

---

**Travail** : écrire la fonction JS [setMenu].

---

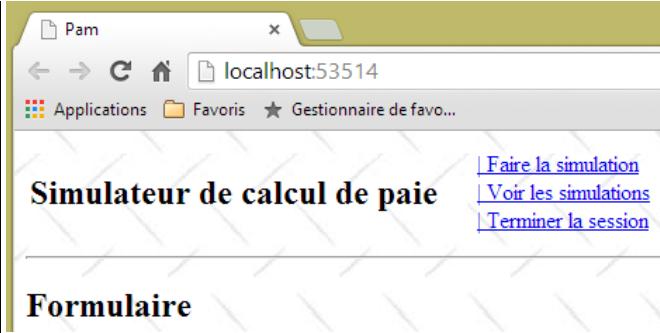
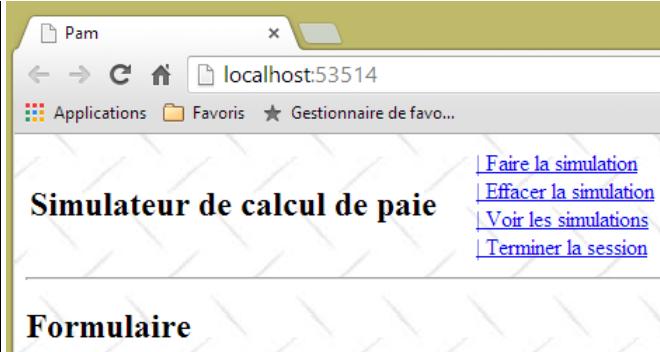
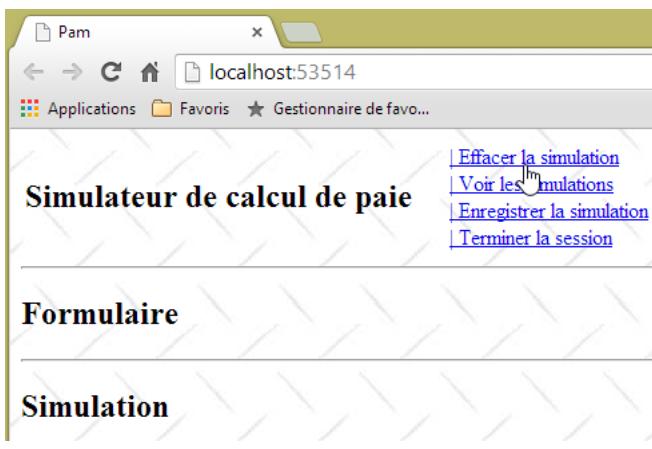
Si **T** est un tableau de liens :

- **T.length** est le nombre de liens ;
- **T[i]** est le lien n° i ;
- **T[i].show()** affiche le lien n° i ;
- **T[i].hide()** cache le lien n° i.

Avec ces nouvelles fonctions JS, la page affichée au démarrage est la suivante :



Adaptez les fonctions JS [faireSimulation, effacerSimulation, enregistrerSimulation, voirSimulations, retourFormulaire, terminerSession] afin d'avoir les écrans suivants :



Maintenant que le modèle APU et les liens de navigation sont en place, nous pouvons passer à l'écriture des actions et des vues côté serveur. Au fil des étapes, vous allez découvrir que certains des liens Ajax qui fonctionnent maintenant ne fonctionnent plus, ceci parce que vous allez modifier les vues partielles envoyées au client. Au fur et à mesure que vous allez construire les différentes actions et vues côté serveur, les liens Ajax côté client vont retrouver le fonctionnement que vous leur avez donné.

## 2.10 Étape 4 : écriture de l'action serveur [Index]

Actuellement au démarrage de l'application, nous avons l'écran suivant :

Au lieu d'avoir cet écran, nous voudrions avoir le suivant :

C'est l'action [Index] qui doit produire cette page. Faisons quelques remarques :

- la page présente un formulaire avec trois champs de saisie :
  - l'employé dont on calcule le salaire,
  - son nombre d'heures travaillées,
  - son nombre de jours travaillés ;
- le formulaire est posté par le lien [Faire la simulation] ;

- la validité des champs de saisie [Heures travaillées] et [Jours travaillés] doit être vérifiée ;
- la liste des employés vient de la couche [métier] que nous avons construite précédemment.

Rappelons le code actuel de l'action [Index] :

```
1.     [HttpGet]
2.     public ViewResult Index()
3.     {
4.         return View();
5.     }
```

celui de la vue [Index.cshtml] que cette action affiche :

```
1. @{
2.     ViewBag.Title = "Pam";
3. }
4. @Html.Partial("Formulaire")
```

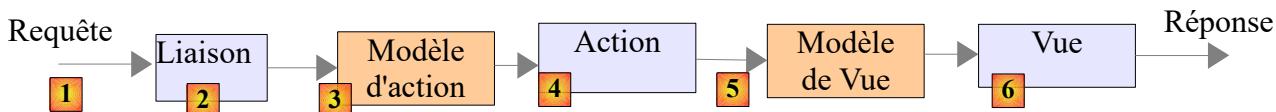
et celui de la vue partielle [Formulaire.cshtml] :

```
1. <h2>Formulaire</h2>
```

Des modifications vont avoir lieu dans ces trois endroits.

## 2.10.1 Le modèle du formulaire

Revenons à la chaîne de traitement de l'URL [/Pam/Index] :

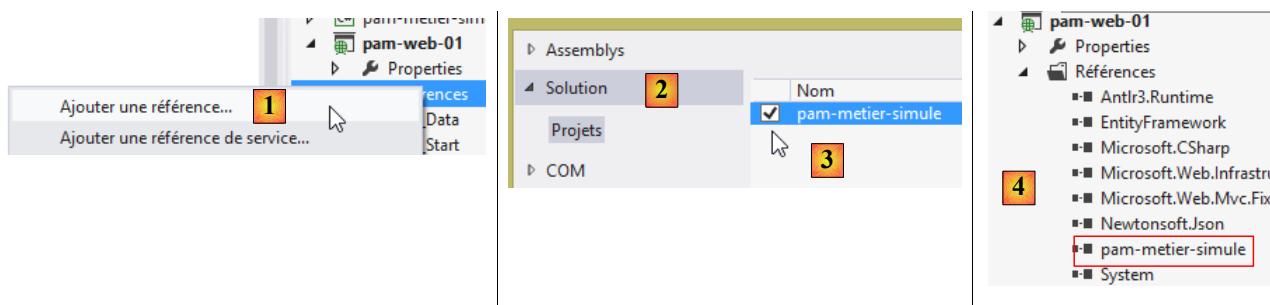


- la requête HTTP du client arrive en [1] ;
- en [2], les informations contenues dans la requête vont être transformées en modèle d'action [3] qui servira d'entrée à l'action [4] ;
- en [4], l'action, à partir de ce modèle, va générer une réponse. Celle-ci aura deux composantes : une vue V [6] et le modèle M de cette vue [5] ;
- la vue V [6] va utiliser son modèle M [5] pour générer la réponse HTTP destinée au client.

L'action à laquelle nous nous intéressons est l'action [Index] qui pour l'instant est la suivante :

```
1.     [HttpGet]
2.     public ViewResult Index()
3.     {
4.         return View();
5.     }
```

L'action [Index] ne passe aucun modèle à la vue [Index.cshtml]. Celle-ci ne pourra donc pas afficher la liste des employés. Celle-ci peut être demandée à la couche [métier]. Pour cela, il faut que le projet [pam-web-01] ait une référence sur le projet [pam-metier-simule]. Nous créons cette référence maintenant :



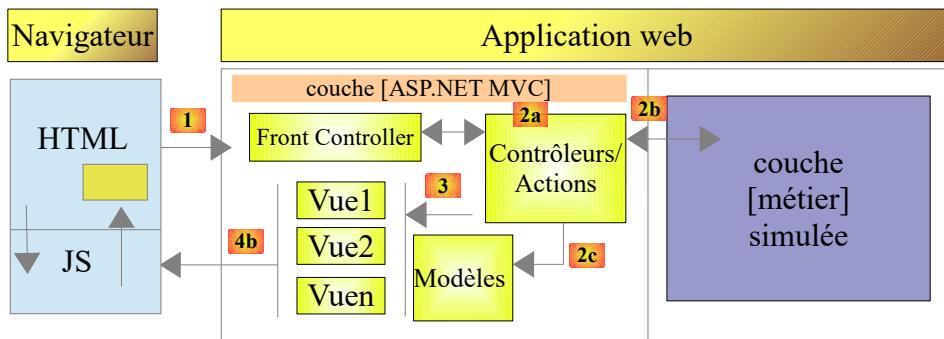
- en [1], clic droit sur [References] du projet [pam-web-01] puis [Ajouter une référence] ;
- en [2], sélectionner l'option [Solution] puis le projet [pam-metier-simule] en [3] ;
- en [4], le projet [pam-metier-simule] a été ajouté aux références du projet [pam-web-01].

## 2.10.2 Le modèle de l'application

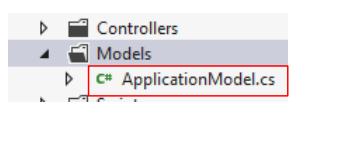
Nous avons introduit les notions importantes de **modèle d'application** et de **modèle de session** au paragraphe 1.4.10, page 84. Nous allons les utiliser maintenant. On rappelle qu'on met dans le modèle :

- **d'application** des données en lecture seule pour tous les utilisateurs. Ce modèle constitue une mémoire partagée par toutes les requêtes de tous les utilisateurs ;
- **de session** des données en lecture et écriture pour un utilisateur donné. Ce modèle constitue une mémoire partagée par toutes les requêtes de cet utilisateur.

Qu'allons-nous mettre dans le modèle d'application ? Revenons à l'architecture de celle-ci :



La couche [web] détient une référence sur la couche [métier]. Celle-ci peut être partagée par tous les utilisateurs. On peut donc la mettre dans le modèle de l'application. Par ailleurs, nous allons faire l'hypothèse que la liste des employés ne change pas. Elle peut donc être lue une unique fois puis partagée entre tous les utilisateurs. Nous proposons alors le modèle d'application suivant :



Le code de la classe [ApplicationModel] pourrait être le suivant :

```

1.  using Pam.Metier.Entites;
2.  using Pam.Metier.Service;
3.  namespace PamWeb.Models
4.  {
5.      public class ApplicationModel
6.      {
7.          // --- données de portée application ---
8.          public Employe[] Employes { get; set; }
9.          public IPamMetier PamMetier { get; set; }
10.     }
11. }

```

Pour afficher une liste déroulante dans une vue, on écrit quelque chose comme suit (cf page 125) :

```

1.      <!-- la liste déroulante -->
2.      <tr>
3.          <td>Liste déroulante</td>
4.          <td>@Html.DropDownListFor(m => m.DropDownListField,
5.              new SelectList(@Model.DropDownListFieldItems, "Value", "Label"))
6.          </td>
7.      </tr>

```

La méthode [DropDownListFor] attend pour second paramètre un type *SelectListItem[]* qui avait été fourni ci-dessus par un type *SelectList*. Il nous faut construire un tel tableau avec la liste des employés. Puisque les employés ne changent pas, ce tableau peut lui aussi être placé dans le modèle de l'application. Nous faisons évoluer celui-ci comme suit :

```

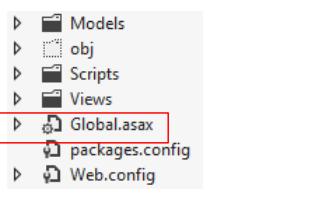
1.  using Pam.Metier.Entites;
2.  using Pam.Metier.Service;
3.  using System.Web.Mvc;
4.
5.  namespace PamWeb.Models
6.  {

```

```

7.     public class ApplicationModel
8.    {
9.        // --- données de portée application ---
10.       public Employe[] Employes { get; set; }
11.       public IPamMetier PamMetier { get; set; }
12.       public SelectListItems[] EmployesItems { get; set; }
13.    }
14. }
```

A quel moment ce modèle doit-il être construit ? Nous l'avons montré au paragraphe 1.4.10, page 84. C'est lors de l'exécution de la méthode [Application\_Start] du fichier [Global.asax] :



La méthode [Application\_Start] est pour l'instant la suivante :

```

1.  using System.Web.Http;
2.  using System.Web.Mvc;
3.  using System.Web.Optimization;
4.  using System.Web.Routing;
5.
6.  namespace pam_web_01
7.  {
8.      public class MvcApplication : System.Web.HttpApplication
9.      {
10.          protected void Application_Start()
11.          {
12.              AreaRegistration.RegisterAllAreas();
13.
14.              WebApiConfig.Register(GlobalConfiguration.Configuration);
15.              FilterConfig.RegisterGlobalFilters(GlobalFilters.Filters);
16.              RouteConfig.RegisterRoutes(RouteTable.Routes);
17.              BundleConfig.RegisterBundles(BundleTable.Bundles);
18.          }
19.      }
20. }
```

Nous la faisons évoluer comme suit :

```

1.  using Pam.Metier.Entites;
2.  using Pam.Metier.Service;
3.  using PamWeb.Infrastructure;
4.  using PamWeb.Models;
5.  using System.Web.Http;
6.  using System.Web.Mvc;
7.  using System.Web.Optimization;
8.  using System.Web.Routing;
9.
10. namespace pam_web_01
11. {
12.     public class MvcApplication : System.Web.HttpApplication
13.     {
14.         protected void Application_Start()
15.         {
16.             // -----Auto-généré
17.             AreaRegistration.RegisterAllAreas();
18.             WebApiConfig.Register(GlobalConfiguration.Configuration);
19.             FilterConfig.RegisterGlobalFilters(GlobalFilters.Filters);
20.             RouteConfig.RegisterRoutes(RouteTable.Routes);
21.             BundleConfig.RegisterBundles(BundleTable.Bundles);
22.             // -----
23.             // ----- configuration spécifique
24.             // -----
25.             // données de portée application
26.             ApplicationModel application = new ApplicationModel();
27.             Application["data"] = application;
28.             // instanciation couche [métier]
29.             application.PamMetier = ...
30.             // tableau des employés
31.             application.Employes = ...
32.             // éléments du combo des employés
33.             application.EmployesItems = ...
34.             // model binder pour [ApplicationModel]
```

```
35.     ...
36.   }
37. }
38. }
```

**Travail** : compléter le code de la méthode [Application\_Start]. Tout ce dont vous avez besoin se trouve au paragraphe 1.4.10, page 84. Prenez le temps de relire ce paragraphe long mais important.

La ligne 33 comporte en fait plusieurs lignes. Pour construire un objet de type [SelectListItem] vous pouvez utiliser la méthode suivante :

```
new SelectListItem() { Text = unTexte, Value = uneValeur };
```

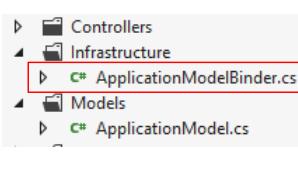
Ce [SelectListItem] servira à générer la balise HTML <option> suivante :

```
<option value='uneValeur'>unTexte</option>
```

de la liste déroulante. On fera en sorte que :

- **unTexte** soit le **prénom** suivi du **nom** de l'employé ;
- **uneValeur** soit le n° **SS** de l'employé.

Ligne 35, ci-dessus, vous aurez besoin de la classe [ApplicationModelBinder] décrite au paragraphe 5, page 88 :



### 2.10.3 Le code de l'action [Index]

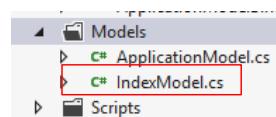
Maintenant que nous avons défini un modèle pour l'application, nous pouvons faire évoluer le code de l'action [Index] de la façon suivante :

```
1.   [HttpGet]
2.   public ViewResult Index(ApplicationModel application)
3.   {
4.     return View();
5. }
```

- ligne 4 : le modèle de l'application est désormais un paramètre de l'action [Index]. Nous avons expliqué au paragraphe 1.4.10, page 84, comment ce paramètre était initialisé par le framework.

### 2.10.4 Le modèle de la vue [Index.cshtml]

Maintenant, l'action [Index] a accès aux employés qui sont enregistrés dans le modèle de l'application. Il faut maintenant qu'elle les passe à la vue [Index.cshtml] qu'elle va afficher. On pourrait passer un type [ApplicationModel] comme modèle de la vue [Index.cshtml] mais nous verrons rapidement que cette vue a besoin d'autres informations qui ne sont pas dans [ApplicationModel]. Nous allons utiliser le modèle de vue [IndexModel] suivant :



```
1. namespace Pam.Web.Models
2. {
3.   public class IndexModel
4.   {
5.     // données de portée application
6.     public ApplicationModel Application { get; set; }
7.   }
8. }
```

- ligne 6 : [IndexModel] embarque le modèle de l'application.

L'action [Index] devient la suivante :

```

1.      [HttpGet]
2.      public ViewResult Index(ApplicationModel application)
3.      {
4.          return View(new IndexModel() { Application = application });
5.      }

```

- ligne 4, la vue par défaut [Index.cshtml] est affichée avec comme modèle un type [IndexModel] initialisé avec les données du modèle d'application.

Nous savons que la vue [Index.cshtml] doit afficher un formulaire :



Revenons à la chaîne de traitement d'une requête :



Pour la requête [GET /Pam/Index] :

- l'action est [Index] ;
- le modèle de cette action est [ApplicationModel] ;
- la vue est [Index.cshtml] ;
- le modèle de cette vue est [IndexModel].

Lorsque le formulaire va être posté on aura une chaîne de traitement analogue :

- l'action est celle qui traite le POST ;
- son modèle rassemble les valeurs postées, ici :
  - le n° SS de l'employé sélectionné ;
  - le nombre d'heures travaillées ;
  - le nombre de jours travaillés ;

On pourrait créer un modèle d'action rassemblant ces trois valeurs. Il est également fréquent de réutiliser le modèle qui a servi à afficher le formulaire. C'est ce que nous allons faire ici. La classe [IndexModel] évolue comme suit :

```

1.  using System.ComponentModel.DataAnnotations;
2.  using System.Web.Mvc;
3.  namespace Pam.Web.Models
4.  {
5.      [Bind(Exclude = "Application")]
6.      public class IndexModel
7.      {
8.          // données de portée application
9.          public ApplicationModel Application { get; set; }
10.
11.         // valeurs postées
12.         [Display(Name = "Employé")]
13.         public string SS { get; set; }
14.         [Display(Name = "Heures travaillées")]
15.         [UIHint("Decimal")]

```

```

16.     public double HeuresTravaillées { get; set; }
17.     [Display(Name = "Jours travaillés")]
18.     public double JoursTravaillés { get; set; }
19.   }
20. }
```

- lignes 13, 16, 18 : les trois valeurs postées. On notera que [joursTravaillés] a été déclaré de type [double] alors qu'en réalité on attend un entier. Le type [double] a été introduit pour faciliter la validation de ce champ côté client, la validation d'un type [int] ayant posé problème ;
- lignes 12, 14, 17 : des libellés pour les méthodes [Html.LabelFor] de la vue associée au modèle ;
- ligne 15 : une annotation pour avoir un affichage du champ [HeuresTravaillées] avec deux décimales ;
- ligne 5 : on indique que la propriété nommée [Application] ne fait pas partie des valeurs postées.

## 2.10.5 Les vues [Index.cshtml] et [Formulaire.cshtml]

La vue [Index.cshtml] est affichée par l'action [Index] suivante :

```

1.     [HttpGet]
2.     public ViewResult Index(ApplicationModel application)
3.     {
4.       return View(new IndexModel() { Application = application });
5.     }
```

De façon intéressante, la vue [Index.cshtml] reste inchangée :

```

1. @{
2.   ViewBag.Title = "Pam";
3. }
4. @Html.Partial("Formulaire")
```

- la vue ne déclare aucun modèle ;
- ligne 4 : elle intègre la vue partielle [Formulaire.cshtml], là encore sans passer à celle-ci de modèle. Au cours des tests, il a été constaté que le modèle [IndexModel] passé à la vue [Index.cshtml] se propageait implicitement à la vue partielle [Formulaire.cshtml]. Cette dernière vue pourrait maintenant avoir la forme suivante :

```

1. @model Pam.Web.Models.IndexModel
2.
3. @using (Html.BeginForm("FaireSimulation", "Pam", FormMethod.Post, new { id = "formulaire" }))
4. {
5.   <table>
6.     <thead>
7.       <tr>
8. ...
9.       </tr>
10.    </thead>
11.    <tbody>
12.      <tr>
13. ...
14.      </tr>
15.    </tbody>
16. ...
17.      </tr>
18.    </tbody>
19.  </table>
20. }
21. <div id="simulation" />
```

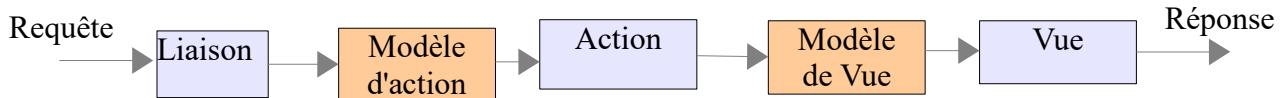
- ligne 1 : la vue reçoit un modèle de type [IndexModel] ;
- ligne 3 : le formulaire ;
- lignes 6-10 : les entêtes du tableau des saisies ;
- lignes 12-14 : la ligne des saisies ;
- lignes 15-17 : les éventuels messages d'erreur.

---

**Travail** : compléter le code de la vue [Formulaire.cshtml]. On utilisera les méthodes [DropDownListFor, EditorFor, LabelFor, ValidationMessageFor] décrites au paragraphe 1.5.7, page 125.

## 2.10.6 Test de l'action [Index]

Nous avons écrit tous les éléments de la chaîne de traitement de l'URL [/Pam/Index] :



Nous testons l'application avec [Ctrl-F5] :

Vous devez vérifier que votre liste déroulante a bien été remplie avec la liste des employés que nous avions définie dans la couche [métier] simulée.

## 2.11 Étape 5 : mise en place de la validation des saisies

### 2.11.1 Le problème

Bien que nous n'ayons rien fait pour cela, des validations côté client sont déjà à l'œuvre :

La validation côté client est à l'oeuvre par défaut à cause de la ligne 3 ci-dessous dans le fichier [Web.config] de l'application.

```
5.    <appSettings>
6.      ...
7.        <add key="ClientValidationEnabled" value="true" />
8.    </appSettings>
```

Cependant parce que dans [IndexModel], on a déclaré le champ [JoursTravaillés] de type [double] :

```
public double JoursTravaillés { get; set; }
```

on peut saisir un nombre réel dans ce champ :

Par ailleurs, on peut entrer des valeurs fantaisistes dans les deux champs :

Le modèle [IndexModel] du formulaire est actuellement le suivant :

```
1.  using System.ComponentModel.DataAnnotations;
```

```

2.  using System.Web.Mvc;
3.  namespace Pam.Web.Models
4.  {
5.      [Bind(Exclude = "Application")]
6.      public class IndexModel
7.      {
8.          // données de portée application
9.          public ApplicationModel Application { get; set; }
10.
11.         // valeurs postées
12.         [Display(Name = "Employé")]
13.         public string SS { get; set; }
14.         [Display(Name = "Heures travaillées")]
15.         [UIHint("Decimal")]
16.         public double HeuresTravaillées { get; set; }
17.         [Display(Name = "Jours travaillés")]
18.         public double JoursTravaillés { get; set; }
19.     }
20. }
```

Travail : améliorez ce modèle pour :

- avoir des messages d'erreur personnalisés ;
- n'accepter que des valeurs réelles dans l'intervalle [0,400] pour le champ [HeuresTravaillées] ;
- n'accepter que des valeurs entières dans l'intervalle [0,31] pour le champ [JoursTravaillées] ;

On pourra s'aider de l'exemple du paragraphe 1.7.6.2, page 209. Pour vérifier que le nombre de jours travaillés est un entier, on pourra utiliser une expression régulière (cf exemples du paragraphe 1.5.9.1, page 145).

Voici des exemples de ce qui est attendu :

The screenshot shows a web page titled "Simulateur de calcul de paie". At the top right, there are three links: "Faire la simulation", "Voir les simulations", and "Terminer la session". Below the title, there are three input fields: "Employé" (containing "Marie Jouveinal"), "Heures travaillées" (containing "x"), and "Jours travaillés" (containing "y"). A red error message at the bottom states: "Le champ Heures travaillées doit être un nombre. Le nombre de jours doit avoir un ou deux chiffres !".

The screenshot shows the same web page. The "Employé" field contains "Marie Jouveinal". The "Heures travaillées" field contains "-1" and has a red border. The "Jours travaillés" field contains "44" and also has a red border. A red error message at the bottom states: "Le nombre d'heures doit être dans l'intervalle [0,400] ! Le nombre de jours doit être dans l'intervalle [0,31] !".

A screenshot of a web browser window titled "Pam". The address bar shows "localhost:53514". The page content is a "Simulateur de calcul de paie" (Payroll calculator). It has three input fields: "Employé" (Employee) with value "Marie Jouveinal", "Heures travaillées" (Hours worked) with value "20,5", and "Jours travaillés" (Days worked) with value "4". Below the form, a red error message reads "Le nombre d'heures travaillées est requis ! Le nombre de jours travaillés est requis !" (Hours worked and Days worked are required!). At the top right, there are links: "Faire la simulation", "Voir les simulations", and "Terminer la session".

## 2.11.2 Saisie des nombres réels au format français

Dans la version actuelle de l'application, le nombre d'heures travaillées doit être un nombre décimal au format anglo-saxon (avec le point décimal). Le format français avec la virgule n'est pas accepté :

A screenshot of a web browser window titled "Pam". The address bar shows "localhost:65013". The page content is a "Simulateur de calcul de paie" (Payroll calculator). It has three input fields: "Employé" (Employee) with value "Marie Jouveinal", "Heures travaillées" (Hours worked) with value "20,5", and "Jours travaillés" (Days worked) with value "4". Below the form, a red error message reads "Le champ Heures travaillées doit être un nombre." (The Hours worked field must be a number.). At the top right, there are links: "Faire la simulation", "Voir les simulations", and "Terminer la session".

Ce problème a été identifié et traité au paragraphe 1.6.1, page 156.

---

**Travail** : en suivant la démarche du paragraphe sus-nommé, faites les modifications nécessaires pour qu'on puisse saisir les nombres réels avec le format décimal français. Testez votre application.

Maintenant, l'écran précédent devient :

A screenshot of a web browser window titled "Pam". The address bar shows "localhost:65013". The page content is a "Simulateur de calcul de paie" (Payroll calculator). It has three input fields: "Employé" (Employee) with value "Marie Jouveinal", "Heures travaillées" (Hours worked) with value "20,5", and "Jours travaillés" (Days worked) with value "4". There is no error message displayed. At the top right, there are links: "Faire la simulation", "Voir les simulations", and "Terminer la session".

### 2.11.3 Validation du formulaire par le lien Javascript [Faire la simulation]

Actuellement, on peut poster des valeurs invalides comme le montre la séquence suivante :

Simulateur de calcul de paie

Employé      Heures travaillées      Jours travaillés

Marie Jouveinal      a      b

Le champ Heures travaillées doit être un nombre. Le nombre de jours doit avoir un ou deux chiffres !

[Faire la simulation] [Voir les simulations] [Terminer la session]

Simulateur de calcul de paie

Employé      Heures travaillées      Jours travaillés

Marie Jouveinal      a      b

Le champ Heures travaillées doit être un nombre. Le nombre de jours doit avoir un ou deux chiffres !

[Effacer la simulation] [Voir les simulations] [Enregistrer la simulation] [Terminer la session]

Simulation [1]

La présence de la simulation en [1] et le changement de menu en [2] montrent que le clic sur le lien [Faire la simulation] a posté le formulaire alors même que les valeurs saisies étaient invalides. Ce problème a été identifié et traité au paragraphe 1.7.6.5, page 211.

---

**Travail** : en suivant la démarche exposée au paragraphe sus-nommé, faites en sorte que le POST du lien [Faire la simulation] ne puisse se faire si les valeurs saisies sont invalides. Pensez à vider le cache du navigateur avant de tester vos modifications.

---

On rappelle que la vue partielle [Formulaire.cshtml] génère un formulaire HTML d'id [formulaire] (ligne 1 ci-dessous) :

```
1. @using (Html.BeginForm("FaireSimulation", "Pam", FormMethod.Post, new { id = "formulaire" }))  
2. {  
3. ...  
4. }
```

Cela peut se vérifier en affichant le code source du formulaire dans le navigateur :

```
1. <div id="content">  
2. <form action="/Pam/FaireSimulation" id="formulaire" method="post">  
3. ...  
4. </form>  
5. <div id="simulation" />  
6. </div>
```

## 2.12 Étape 6 : faire une simulation

### 2.12.1 Le problème

Lorsque nous faisons une simulation, nous voulons obtenir le résultat suivant :

The screenshot shows a web browser window titled "Pam" with the URL "localhost:65013". The page is titled "Simulateur de calcul de paie". A sidebar on the right contains links: "Faire la simulation", "Effacer la simulation", "Enregistrer la simulation", and "Terminer la session". The main content area has three tabs: "Employé", "Heures travaillées", and "Jours travaillés". The "Employé" tab is selected, showing "Marie Jouveinal" with "150" hours worked and "20" days worked. Below this is a red-bordered section containing four groups of information: "Informations Employé", "Informations Cotisations", "Informations Indemnités", and "Informations Salaire".

Informations Employé		
Nom	Prénom	
Jouveinal	Marie	
Ville	Code Postal	Indice
St Corentin	49203	2

CGSRDS	CSGD	Retraite	Sécurité sociale
3,49 %	6,15 %	7,88 %	9,39 %

Salaire horaire	Entretien / Jour	Repas / Jour	Congés payés
2,10 €	2,10 €	3,10 €	15 %

Salaire de base	Cotisations sociales	Indemnités d'entretien	Indemnités de repas
362,25 €	97,48 €	42,00 €	62,00 €

Salaire net à payer : 368,77 €

La vue partielle [Simulation.cshtml] affiche désormais la feuille de salaire d'un employé.

### 2.12.2 Écriture de la vue [Simulation.cshtml]

La vue [Simulation.cshtml] évolue comme suit :

```
1. @model Pam.Metier.Entites.FeuilleSalaire
2. <hr />
3. <p><span class="info">Informations Employé</span></p>
4. <table>
5.   <tbody>
6.     <tr>
```

```

7.      <td><span class="libellé">Nom</span>
8.      </td>
9.      <td><span class="libellé">Prénom</span>
10.     </td>
11.     <td><span class="libellé">Adresse</span>
12.     </td>
13.   </tr>
14.   <tr>
15.     <td>
16.       <span class="valeur">@Model.Employe.Nom</span>
17.     </td>
18. ...
19.   </tr>
20.   <tr>
21.     <td><span class="libellé">Ville</span>
22.     </td>
23.     <td><span class="libellé">Code Postal</span>
24.     </td>
25.     <td><span class="libellé">Indice</span>
26.     </td>
27.   </tr>
28.   <tr>
29. ...
30.   </tr>
31. </tbody>
32. </table>
33. <br />
34. <p><span class="info">Informations Cotisations</span></p>
35. <table>
36. ...
37.   </tbody>
38. </table>
39. <br />
40. <p><span class="info">Informations Indemnités</span></p>
41. <table>
42. ...
43. </table>
44. <br />
45. <p><span class="info">Informations Salaire</span></p>
46. <table>
47. ...
48. </table>
49. <br />
50. <table>
51. ...
52. </table>

```

- ligne 1 : la vue [Simulation.cshtml] a pour modèle le type [FeuilleSalaire] défini au paragraphe 2.7.3, page 230 ;
- la vue utilise les classes [libellé, info, valeur] définies dans la feuille de style de l'application [Content / Site.css] :

```

1. .libellé {
2.   background-color: azure;
3.   margin: 5px;
4.   padding: 5px;
5. }
6.
7. .info {
8.   background-color: antiquewhite;
9.   margin: 5px;
10.  padding: 5px;
11. }
12.
13. .valeur {
14.   background-color: beige;
15.   padding: 5px;
16.   margin: 5px;
17. }

```

Par ailleurs, toujours dans [Site.css], on fixe la hauteur des lignes des différentes tables HTML de la région d'**id** [simulation], précisément là où est affichée la feuille de salaire :

```

1. #simulation table tr {
2.   height: 30px;
3. }

```

---

**Travail** : complétez la vue [Simulation.cshtml].

---

Pour afficher les euros d'une somme d'argent, on utilisera la méthode [string.Format] :

```
string.Format("{0:C2}", somme)
```

L'instruction ci-dessus affiche [somme] en valeur monétaire [C] (Currency) avec deux décimales [C2].

Pour tester cette vue, on doit lui fournir une feuille de salaire. Celle-ci doit lui être fournie par l'action [/Pam/FaireSimulation] qui est la cible de l'appel Ajax du lien [Faire la simulation]. Actuellement, cette action est la suivante :

```
1.      [HttpGet]
2.  public ViewResult Index(ApplicationModel application)
3.  {
4.      return View(new IndexModel() { Application = application });
5.  }
6.
7. // faire une simulation
8. [HttpPost]
9. public PartialViewResult FaireSimulation()
10. {
11.     return PartialView("Simulation");
12. }
```

Ci-dessus, l'action [FaireSimulation] ne passe aucun modèle à la vue [Simulation.cshtml]. Il faut qu'elle lui passe une feuille de salaire. On sait que c'est la couche [métier] qui fait le calcul des feuilles de salaire. Cette couche [métier] est accessible via le modèle de l'application [ApplicationModel] que nous avons défini au paragraphe 2.10.2, page 254 :

```
1.  public class ApplicationModel
2.  {
3.      // --- données de portée application ---
4.      public Employe[] Employes { get; set; }
5.      public IPamMetier PamMetier { get; set; }
6.      public SelectListItem[] EmployesItems { get; set; }
7.  }
```

La couche [métier] est accessible via la propriété de la ligne 5 ci-dessus. Pour que l'action [FaireSimulation] ait accès à la couche [métier], nous allons lui passer le modèle de l'application comme nous l'avons fait pour l'action [Index]. Le code évolue alors comme suit :

```
1.  // faire une simulation
2.  [HttpPost]
3.  public PartialViewResult FaireSimulation(ApplicationModel application)
4.  {
5.      return PartialView("Simulation");
6.  }
```

Maintenant, nous sommes capables, à l'intérieur de l'action, de calculer une feuille de salaire fictive. Le code évolue comme suit :

```
1.  // faire une simulation
2.  [HttpPost]
3.  public PartialViewResult FaireSimulation(ApplicationModel application)
4.  {
5.      FeuilleSalaire feuilleSalaire = application.PamMetier.GetSalaire("254104940426058", 150, 20);
6.      return PartialView("Simulation", feuilleSalaire);
7.  }
```

- ligne 5, on calcule un salaire fictif. Le 1er paramètre est un n° SS existant. Il a été défini dans la classe [métier] simulée au paragraphe 2.7.5, page 233. Le second paramètre est le nombre d'heures travaillées et le troisième le nombre de jours travaillés ;
- ligne 6 : cette feuille de salaire est passée comme modèle à la vue [Simulation.cshtml].

Nous sommes désormais prêts à tester la vue [Simulation.cshtml] :

On ne fait aucune saisie et on demande la simulation. On obtient alors le résultat suivant :

### 2.12.3 Calcul du salaire réel

Notre action [FaireSimulation] actuelle calcule toujours la même feuille de salaire :

```

1. // faire une simulation
2. [HttpPost]
3. public PartialViewResult FaireSimulation(ApplicationModel application)
4. {
5.     FeuilleSalaire feuilleSalaire = application.PamMetier.GetSalaire("254104940426058", 150, 20);
6.     return PartialView("Simulation", feuilleSalaire);
7. }
```

Elle ne tient pas compte des informations saisies :

- l'employé dont on calcule le salaire ;
- son nombre d'heures travaillées ;
- son nombre de jours travaillés.

Les valeurs saisies arrivent à l'action [FaireSimulation] de la façon suivante :

1. l'utilisateur clique sur le lien [Faire la simulation]. Cela déclenche l'exécution de la fonction JS [faireSimulation] que nous avons déjà écrite ;
2. la fonction JS [faireSimulation] fait ensuite un appel Ajax à l'action serveur [/Pam/FaireSimulation] sur laquelle nous travaillons actuellement. Pour l'instant, la fonction JS [faireSimulation] ne transmet aucune information à l'action serveur. Il faudra qu'elle lui transmette les valeurs saisies par l'utilisateur ;
3. l'action serveur [/Pam/FaireSimulation] va récupérer les valeurs saisies dans les valeurs postées par la fonction JS [faireSimulation].

Commençons par le point 2 : la fonction JS [faireSimulation] doit poster les valeurs saisies par l'utilisateur à l'action serveur [/Pam/FaireSimulation].

---

**Travail** : complétez la fonction JS [faireSimulation] afin qu'elle poste les valeurs saisies par l'utilisateur. On pourra s'aider de l'exemple du paragraphe 1.7.6.5, page 211 où ce problème a été traité.

---

Traitons maintenant le point 3 ci-dessus. L'action serveur [/Pam/FaireSimulation] doit récupérer les valeurs postées par la fonction JS [faireSimulation].

---

**Travail** : complétez la méthode serveur [FaireSimulation] afin qu'elle calcule le salaire avec les valeurs postées par la fonction JS [faireSimulation]. On pourra s'aider de nouveau de l'exemple du paragraphe 1.7.6.5, page 211 où ce problème a été traité. Pour le moment, on supposera que le modèle issu des valeurs postées est toujours valide.

---

**Hint** : l'action serveur [FaireSimulation] évolue comme suit :

```
1. // faire une simulation
2.     [HttpPost]
3.     public PartialViewResult FaireSimulation(ApplicationModel application, FormCollection data)
4.     {
5.         // création du modèle de l'action
6.         ...
7.         // on essaie de récupérer les valeurs postées dans ce modèle
8.         ...
9.         // on calcule le salaire
10.        FeuilleSalaire feuilleSalaire = ...
11.        // on affiche la feuille de salaire
12.        return PartialView("Simulation", feuilleSalaire);
13.    }
```

Voici un exemple d'exécution :

On choisit [Justine Laverti]. On obtient alors le résultat suivant :

Pam

localhost:65013

Applications Favoris Gestionnaire de favo...

[Effacer la simulation](#)  
[Voir les simulations](#)  
[Enregistrer la simulation](#)  
[Terminer la session](#)

**Simulateur de calcul de paie**

Employé	Heures travaillées	Jours travaillés
Justine Laverti	50	10

**Informations Employé**

Nom	Prénom	Adresse
Laverti	Justine	La brûlerie
Ville	Code Postal	Indice
St Marcel	49014	1

**Informations Cotisations**

CGSRDS	CSGD	Retraite	Sécurité sociale
3,49 %	6,15 %	7,88 %	9,38 %

**Informations Indemnités**

Salaire horaire	Entretien / Jour	Repas / Jour	Congés payés
1,93 €	2,00 €	3,00 €	12 %

**Informations Salaire**

Salaire de base	Cotisations sociales	Indemnités d'entretien	Indemnités de repas
100,00 €	100,00 €	100,00 €	100,00 €

Salaire net à payer : 100,00 €

On a bien obtenu la feuille de salaire fictive de [Justine Laverti]. Précédemment, la feuille de salaire unique qui était calculée était celle de [Marie Jouveinal]. Donc la valeur postée pour le choix de l'employé a été exploitée. Pour le nombre d'heures et le nombre de jours, on ne peut rien dire puisque notre couche [métier] simulée n'en tient pas compte.

#### 2.12.4 Gestion des erreurs

Regardons l'exemple suivant :

Pam

localhost:65013

Applications Favoris Gestionnaire de favo...

[Faire la simulation](#)  
[Voir les simulations](#)  
[Terminer la session](#)

**Simulateur de calcul de paie**

Employé	Heures travaillées	Jours travaillés
XX	50	10

- en [1], on choisit un employé qui n'existe pas (voir la définition de la couche [métier] simulée au paragraphe 2.7.5, page 233) ;
- en [2], on fait la simulation ;
- en [3] ci-dessous, on récupère une page d'erreur.

**Simulateur de calcul de paie**

- [Effacer la simulation](#)
- [Voir les simulations](#)
- [Enregistrer la simulation](#)
- [Terminer la session](#)

Employé	Heures travaillées	Jours travaillés
XX	0,00	0

3

**Erreur du serveur dans l'application '/'.**

*L'employé de n° SS [XX] n'existe pas*

**Description :** Une exception non gérée s'est produite au moment de l'exécution de la requête Web actuelle. Contrôlez la trace de la pile pour plus d'informations sur l'erreur et son origine dans le code.

**Détails de l'exception:** Pam.Metier.Entites.PamException: L'employé de n° SS [XX] n'existe pas

**Erreur source:**

```

Ligne 66 :         if (e == null)
Ligne 67 :         {
Ligne 68 :             throw new PamException(string.Format("L'employé de n° SS [{0}] n'existe pas", ss), 10);
Ligne 69 :         }
Ligne 70 :         // on rend une feuille de salaire fictive

```

Fichier source : d:\data\istia-1314\aspnet\pam-td\pam-metier\metier\service\PamMetier.cs Ligne : 68

Que s'est-il passé ?

La fonction JS [faireSimulation] a été exécutée. Son code ressemble à ceci :

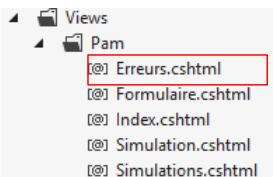
```

1. function faireSimulation() {
2. ...
3.     // on fait un appel Ajax à la main
4.     $.ajax({
5.         url: '/Pam/FaireSimulation',
6. ...
7.         beforeSend: function () {
8.             // signal d'attente allumé
9.             loading.show();
10.        },
11.        success: function (data) {
12. ...
13.        },
14.        error: function (jqXHR) {
15.            // affichage erreur
16.            simulation.html(jqXHR.responseText);
17.            simulation.show();
18.        },
19.        complete: function () {
20.            // signal d'attente éteint
21.            loading.hide();
22.        }
23.    });
24.    // menu
25.    setMenu([lnkEffacerSimulation, lnkEnregistrerSimulation, lnkTerminerSession, lnkVoirSimulations]);
26. }

```

L'appel Ajax a échoué et c'est la fonction des lignes 14-18 qui s'est exécutée. La page d'erreur [jqXHR.responseText] renvoyée par le serveur a été affichée. Celle-ci est assez précise. La couche [métier] simulée a lancé une exception parce que le n° SS qu'on lui a fourni n'est pas celui d'un employé existant (voir code de la couche [métier] simulée au paragraphe 2.7.5, page 233). Il nous faut gérer ce cas proprement.

Nous allons créer une vue partielle [Erreurs.chtml] qui sera renvoyée au client JS à chaque fois qu'une erreur sera détectée côté serveur :



Le code de la vue partielle [Erreurs.cshtml] est le suivant :

```

1. @model IEnumerable<string>
2.
3. <hr />
4. <h2>Les erreurs suivantes se sont produites</h2>
5. <ul>
6.   @foreach (string msg in Model)
7.   {
8.     <li>@msg</li>
9.   }
10. </ul>
```

- ligne 1 : la vue reçoit pour modèle une liste de messages d'erreur ;
- lignes 5-10 : qui sont affichés dans une liste HTML ;

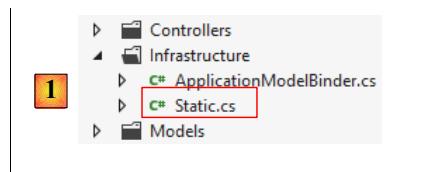
Maintenant,修改ons le code de l'action serveur [FaireSimulation] de la façon suivante :

```

1. // faire une simulation
2. [HttpPost]
3. public PartialViewResult FaireSimulation(ApplicationModel application, FormCollection data)
4. {
5. ...
6.   // on calcule le salaire
7.   FeuilleSalaire feuilleSalaire = null;
8.   Exception exception=null;
9.   try
10.  {
11.    // calcul salaire
12.    feuilleSalaire = ...
13.  }
14.  catch (Exception ex)
15.  {
16.    exception = ex;
17.  }
18.  // erreur ?
19.  if (exception == null)
20.  {
21.    // on affiche la feuille de salaire
22.    return PartialView("Simulation", feuilleSalaire);
23.  }
24.  else
25.  {
26.    // on affiche la page d'erreurs
27.    return PartialView("Erreurs", Static.GetErreursForException(exception));
28.  }
29. }
```

- lignes 9-17 : le calcul du salaire est désormais fait dans un try / catch ;
- ligne 27 : s'il y a eu erreur, on affiche la vue partielle [Erreurs.cshtml] avec pour modèle la liste de messages d'erreur fournie par la méthode statique [Static.GetErreursForException(exception)].

On regroupe dans la classe [Static] deux fonctions utilitaires statiques [1] :



```

1. using System;
2. using System.Collections.Generic;
3. using System.Web.Mvc;
4.
5. namespace PamWeb.Infrastructure
```

```

6.  {
7.    public class Static
8.    {
9.      // liste des messages d'erreur d'une exception
10.     public static List<string> GetErreursForException(Exception ex)
11.     {
12.       List<string> erreurs = new List<string>();
13.       while (ex != null)
14.       {
15.         erreurs.Add(ex.Message);
16.         ex = ex.InnerException;
17.       }
18.       return erreurs;
19.     }
20.
21.     // liste des messages d'erreur liés à un modèle invalide
22.     public static List<string> GetErreursForModel(ModelStateDictionary état)
23.     {
24.       List<string> erreurs = new List<string>();
25.       if (!état.IsValid)
26.       {
27.         foreach (ModelState ModelState in état.Values)
28.         {
29.           foreach (ModelError error in ModelState.Errors)
30.           {
31.             erreurs.Add(getErrorMessageFor(error));
32.           }
33.         }
34.       }
35.       return erreurs;
36.     }
37.
38.     // le message d'erreur lié à un élément du modèle de l'action
39.     static private string getErrorMessageFor(ModelError error)
40.     {
41.       if (error.ErrorMessage != null && error.ErrorMessage.Trim() != string.Empty)
42.       {
43.         return error.ErrorMessage;
44.       }
45.       if (error.Exception != null && error.Exception.InnerException == null && error.Exception.Message != string.Empty)
46.       {
47.         return error.Exception.Message;
48.       }
49.       if (error.Exception != null && error.Exception.InnerException != null && error.Exception.InnerException.Message !=
50.           string.Empty)
51.       {
52.         return error.Exception.InnerException.Message;
53.       }
54.       return string.Empty;
55.     }
56.   }
57. }
```

- lignes 10-19 : la fonction statique [GetErreursForException] rend la liste des erreurs d'une pile d'exceptions ;
- lignes 22-36 : la fonction statique [GetErreursForModel] rend la liste des erreurs d'un modèle d'action invalide. Le code de cette fonction ainsi que celui de la méthode privée [getErrorMessageFor] (lignes 39-54) a déjà été rencontré page 71.

Ceci fait, nous pouvons tester de nouveau le cas d'erreur :

**Simulateur de calcul de paie**

Employé : XX

Heures travaillées : 0,00

Jours travaillés : 0

[Faire la simulation](#) 2

[Voir les simulations](#)

[Terminer la session](#)

**Simulateur de calcul de paie**

Employé : XX

Heures travaillées : 0,00

Jours travaillés : 0

[Effacer la simulation](#)

[Voir les simulations](#)

[Enregistrer la simulation](#)

[Terminer la session](#)

**Les erreurs suivantes se sont produites**

- L'employé de n° SS [XX] n'existe pas 3

- en [1], on choisit l'employé qui n'existe pas ;
- en [2], on fait la simulation ;
- en [3], on récupère la nouvelle page d'erreurs.

Revenons à l'action serveur [FaireSimulation] :

```

1.    // faire une simulation
2.    [HttpPost]
3.    public PartialViewResult FaireSimulation(ApplicationModel application, FormCollection data)
4.    {
5.        // création du modèle de l'action
6.        IndexModel modèle = new IndexModel() { Application = application };
7.        // on essaie de récupérer les valeurs postées dans le modèle
8.        TryUpdateModel(modèle, data);
9.        // on calcule le salaire
10.    ...
11. }
```

En ligne 8, on met à jour le modèle de la ligne 6 avec les valeurs postées par l'appel Ajax. Nous ne vérifions pas la validité du modèle. Il faut le faire car on ne peut savoir d'où proviennent les valeurs postées. Quelqu'un a pu bricoler un POST et nous envoyer des données invalides.

**Travail** : en suivant le modèle que nous avons développé pour le cas de l'exception, modifiez l'action serveur [FaireSimulation] afin d'envoyer une page d'erreurs lorsque les données postées sont invalides. Pour cela, on utilisera la méthode statique [GetErreursForModel] de la classe [Static].

Comment tester cette modification ? Au paragraphe 2.11.3, page 263, vous avez fait en sorte que la fonction JS [faireSimulation] ne fasse pas le POST des valeurs saisies si celles-ci étaient invalides. Mettez en commentaires les lignes qui réalisent cela puis faites le test suivant :

**Simulateur de calcul de paie**

Employé : Marie Jouveinal

Heures travaillées :

Jours travaillés :

[Faire la simulation](#) 1

[Voir les simulations](#)

[Terminer la session](#)

**Simulateur de calcul de paie**

Employé : Marie Jouveinal

Heures travaillées :

Jours travaillés :

[Effacer la simulation](#)

[Voir les simulations](#)

[Enregistrer la simulation](#)

[Terminer la session](#)

**Les erreurs suivantes se sont produites**

- Le nombre d'heures travaillées est requis ! 2
- Le nombre de jours travaillés est requis !

- en [1], on fait la simulation avec des valeurs invalides ;
- en [2], on récupère bien la page d'erreurs que nous venons de construire, preuve que les validateurs côté serveur ont bien fonctionné.

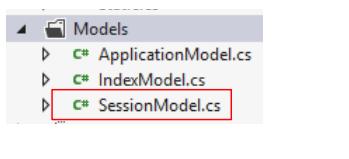
Pour la suite, pensez à décommenter les lignes que vous venez de mettre en commentaires dans la fonction JS [faireSimulation].

## 2.13 Étape 7 : mise en place d'une session utilisateur

L'application [Simulateur de calcul de paie] permet à l'utilisateur de faire diverses simulations de paie avec le lien [Faire la simulation], de les conserver avec le lien [Enregistrer la simulation], de les afficher avec le lien [Voir les simulations] et de les supprimer avec le lien [Retirer la simulation]. Nous savons qu'entre deux requêtes successives de l'utilisateur, il n'y a pas de mémoire sauf si on en crée une par le mécanisme de la session (cf paragraphe 1.4.10, page 84). Il est assez clair ici que nous devons conserver dans la session la liste des simulations enregistrées au fil du temps par l'utilisateur. Il y a d'autres données à mémoriser : lorsque l'utilisateur fait une simulation, celle-ci n'est enregistrée dans la liste des simulations que si l'utilisateur le demande avec le lien [Enregistrer la simulation]. Lorsqu'il le fait, on doit être capable de retrouver la simulation calculée dans la requête précédente. Pour cela, celle-ci sera mise également dans la session. Enfin, nous allons numérotter les simulations à partir de 1. Pour numérotter correctement une nouvelle simulation, il faut avoir conservé le n° de la simulation précédente, là encore dans la session.

Dans le paragraphe 1.4.10, page 84, nous avons introduit le concept de modèle de session en paramètre d'entrée d'une action afin que celle-ci ait accès à la session. Nous allons reprendre ce concept. Vous êtes invités à relire le paragraphe concerné si cette notion est obscure pour vous.

Nous créons la classe [SessionModel] suivante :



Son code est le suivant :

```

1.  using Pam.Web.Models;
2.  using System.Collections.Generic;
3.
4.  namespace Pam.Web.Models
5.  {
6.      public class SessionModel
7.      {
8.          // la liste des simulations
9.          public List<Simulation> Simulations { get; set; }
10.         // n° de la prochaine simulation
11.         public int NumNextSimulation { get; set; }
12.         // la dernière simulation
13.         public Simulation Simulation { get; set; }
14.
15.         // constructeur
16.         public SessionModel()
17.         {
18.             // liste de simulations vide
19.             Simulations = new List<Simulation>();
20.             // n° prochaine simulation
21.             NumNextSimulation = 1;
22.         }
23.     }
24. }
```

La classe [Simulation] des lignes 9 et 13, va enregistrer des informations sur une simulation. Qu'avons-nous besoin d'enregistrer ? Le lien [Faire la simulation] calcule une feuille de salaire de type [FeuilleSalaire]. Il paraît naturel de mettre celle-ci dans la simulation. Par ailleurs, il nous faut mémoriser les informations qui ont mené à cette feuille de salaire :

- l'employé sélectionné. On trouvera celui-ci dans le champ [FeuilleSalaire.Employe]. Il est donc inutile de le mémoriser une seconde fois ;
- le nombre d'heures et de jours travaillés. Ces informations ne sont pas dans le type [FeuilleSalaire]. Il nous faut donc les mémoriser.

Enfin chaque simulation est réperée par un numéro. On pourrait donc partir sur la classe [Simulation] suivante :

```

1.  using Pam.Metier.Entites;
2.
```

```

3.  namespace Pam.Web.Models
4.  {
5.      public class Simulation
6.      {
7.          // n° de la simulation
8.          public int Num { get; set; }
9.          // le nombre d'heures travaillées
10.         public double HeuresTravaillées { get; set; }
11.         // le nombre de jours travaillés
12.         public int JoursTravaillés { get; set; }
13.         // la feuille de salaire
14.         public FeuilleSalaire FeuilleSalaire { get; set; }
15.     }
16. }
```

L'action serveur [FaireSimulation] doit, outre calculer une feuille de salaire, créer une simulation et la mettre dans la session. Pour cela, elle va recevoir en paramètre le modèle de la session :

```

1.  // faire une simulation
2.  [HttpPost]
3.  public PartialViewResult FaireSimulation(ApplicationModel application, SessionModel session, FormCollection data)
4.  {
5.      // création du modèle de l'action
6.      IndexModel modèle = new IndexModel() { Application = application };
7.      // on essaie de récupérer les valeurs postées dans le modèle
8.      TryUpdateModel(modèle, data);
9.      // modèle valide ?
10.     if (!ModelState.IsValid)
11.     {
12.         // on affiche la page d'erreurs
13.         return PartialView("Erreurs", Static.GetErreursForModel(ModelState));
14.     }
15.     // on calcule le salaire
16.     FeuilleSalaire feuilleSalaire = null;
17.     Exception exception = null;
18.     try
19.     {
20.         // calcul salaire
21.         feuilleSalaire = application.PamMetier.GetSalaire(modèle.SS, modèle.HeuresTravaillées,
22.             (int)modèle.JoursTravaillés);
23.     }
24.     catch (Exception ex)
25.     {
26.         exception = ex;
27.     }
28.     // erreur ?
29.     if (exception != null)
30.     {
31.         // on affiche la page d'erreurs
32.         return PartialView("Erreurs", Static.GetErreursForException(exception));
33.     }
34.     // on crée une simulation et on la met dans la session
35.     session.Simulation = ...
36.     // on affiche la feuille de salaire
37.     return PartialView("Simulation", feuilleSalaire);
38. }
```

- ligne 3 : l'action reçoit en paramètre le modèle de la session ;

**Travail 1** : compléter le code de l'action, ligne 34

**Travail 2** : en suivant la démarche du paragraphe 1.4.10, page 84, faites ce qui est nécessaire pour que le paramètre [SessionModel session] de l'action soit bien initialisé par le framework. Si on ne fait rien, on aura un pointeur *null* pour ce paramètre.

## 2.14 Étape 8 : enregistrer une simulation

### 2.14.1 Le problème

Lorsque nous avons fait une simulation, nous pouvons l'enregistrer :

Pam

localhost:65012

Applications Favoris Gestionnaire de favo...

## Simulateur de calcul de paie

[Effacer la simulation](#)  
[Voir les simulations](#)  
[Enregistrer la simulation](#)  
[Terminer la session](#)

Employé	Heures travaillées	Jours travaillés
Marie Jouveinal	10	10

**Informations Employé** 1

Nom	Prénom	Adresse
Jouveinal	Marie	5 rue des oiseaux
Ville	Code Postal	Indice
St Corentin	49203	2

**Informations Cotisations**

CGSRDS	CSGD	Retraite	Sécurité sociale
3,49 %	6,15 %	7,88 %	9,38 %

**Informations Indemnités**

Salaire horaire	Entretien / Jour	Repas / Jour	Congés payés
2,10 €	2,10 €	3,10 €	15 %

**Informations Salaire**

Salaire de base	Cotisations sociales	Indemnités d'entretien	Indemnités de repas
100,00 €	100,00 €	100,00 €	100,00 €

Salaire net à payer : 100,00 €

Pam

localhost:65012

Applications Favoris Gestionnaire de favo...

## Simulateur de calcul de paie

[Retour au formulaire de simulation](#)  
[Terminer la session](#)

---

### Liste des simulations

N°	Nom	Prénom	Heures travaillées	Jours travaillés	Salaire de base	Indemnités	Cotisations sociales	Salaire net	Actions
1	Jouveinal	Marie	10	10	100,00 €	200,00 €	100,00 €	100,00 €	<a href="#">retirer</a>

La vue partielle [Simulations.cshtml] affiche désormais la liste des simulations faites par l'utilisateur. On rappelle que la feuille de salaire calculée est fictive.

## 2.14.2 Écriture de l'action serveur [EnregistrerSimulation]

Le lien Ajax [Enregistrer la simulation] appelle l'action serveur [EnregistrerSimulation] dont le code était jusqu'à maintenant le suivant :

```
1.     [HttpPost]
2.     public PartialViewResult EnregistrerSimulation()
3.     {
4.         return PartialView("Simulations");
5.     }
```

Il évolue comme suit :

```
1.     // enregistrer une simulation
2.     [HttpPost]
3.     public PartialViewResult EnregistrerSimulation(SessionModel session)
4.     {
5.         // on enregistre la dernière simulation faite dans la liste des simulations de la session
6.         ...
7.         // on incrémente dans la session le n° de la prochaine simulation
8.         ...
9.         // on affiche la liste des simulations
10.        ...
11.    }
```

- ligne 1 : l'action [EnregistrerSimulation] a besoin d'avoir accès à la session. C'est pourquoi elle a pour paramètre le modèle de la session.

---

**Travail** : compléter l'action serveur [EnregistrerSimulation].

## 2.14.3 Écriture de la vue partielle [Simulations.cshtml]

L'action précédente [EnregistrerSimulation] fait afficher la vue partielle [Simulations.cshtml] avec pour modèle la liste des simulations faites par l'utilisateur. Son code est le suivant :

```
1.     @model IEnumerable<Simulation>
2.
3.     @using Pam.Web.Models
4.
5.     @if (Model.Count() == 0)
6.     {
7.         <h2>Votre liste de simulations est vide</h2>
8.     }
9.     @if (Model.Count() != 0)
10.    {
11.        <h2>Liste des simulations</h2>
12.        ...
13.    }
```

---

**Travail 1** : compléter le code de la vue partielle [Simulations.cshtml]. On utilisera une table HTML pour l'affichage des simulations. On pourra s'aider des exemples du paragraphe 1.5.4, page 102.

**Note** : le lien [retirer] de chaque simulation de la table HTML sera un lien Javascript de la forme suivante :

```
<a href="javascript:retirerSimulation(N)">retirer</a>
```

où N est le n° de la simulation.

---

**Travail 2** : testez votre application en faisant des simulations. Pour faire celles-ci, faites la séquence suivante de façon répétée : 1) chargez la page de l'application par [F5], 2) faites une simulation, 3) enregistrez-la. Les simulations vont s'accumuler dans la session ce qui devrait être reflétée dans la vue [Simulations.cshtml].

---

**Travail 3** : améliorez la vue partielle [Simulations.cshtml] de telle sorte que les couleurs des lignes de la table HTML soient alternées.

Nº	Nom	Prénom	Heures travaillées	Jours travaillés	Salaire de base	Indemnités	Cotisations sociales	Salaire net	
1	Jouveinal	Marie	10	10	100,00 €	200,00 €	100,00 €	100,00 €	<a href="#">retirer</a>
2	Jouveinal	Marie	10	10	100,00 €	200,00 €	100,00 €	100,00 €	<a href="#">retirer</a>
3	Laverti	Justine	20	20	100,00 €	200,00 €	100,00 €	100,00 €	<a href="#">retirer</a>

On donnera de façon alternée aux lignes <tr> de la table HTML, les classes CSS [pair] et [impair] définies dans la feuille de style [/Content/Site.css] :

```

1. .impair {
2.   background-color: beige;
3. }
4.
5. .pair {
6.   background-color: lightsteelblue;
7. }

```

## 2.15 Étape 9 : retourner au formulaire de saisie

### 2.15.1 Le problème

Lorsque nous avons obtenu la liste des simulations, nous pouvons revenir au formulaire de saisie, ce qu'on ne pouvait plus faire depuis un moment :

Nº	Nom	Prénom	Heures travaillées	Jours travaillés	Salaire de base	Indemnités	Cotisations sociales	Salaire net	
1	Jouveinal	Marie	0	0	100,00 €	200,00 €	100,00 €	100,00 €	<a href="#">retirer</a>
2	Jouveinal	Marie	10	10	100,00 €	200,00 €	100,00 €	100,00 €	<a href="#">retirer</a>

## 2.15.2 Écriture de l'action serveur [Formulaire]

Le lien Ajax [Retour au formulaire de simulation] appelle l'action serveur [Formulaire] dont le code était jusqu'à maintenant le suivant :

```
1.     [HttpPost]
2.     public PartialViewResult Formulaire()
3.     {
4.         return PartialView("Formulaire");
5.     }
```

La vue partielle [Formulaire] qu'elle affiche attend un modèle [IndexModel] (ligne 1 ci-dessous) :

```
1. @model Pam.Web.Models.IndexModel
2.
3. @using (Html.BeginForm("FaireSimulation", "Pam", FormMethod.Post, new { id = "formulaire" }))
4. {
5. ...
6. }
7. <div id="simulation" />
```

C'est pour cette raison que le lien [Retour au formulaire de simulation] ne marchait plus.

---

**Travail** : écrire la nouvelle version de l'action serveur [Formulaire] (2 lignes à réécrire) puis faire les tests.

---

## 2.15.3 Modification de la fonction Javascript [retourFormulaire]

Avec la modification faite précédemment, on peut désormais revenir au formulaire mais une anomalie apparaît alors :

N°	Nom	Prénom	Heures travaillées	Jours travaillés	Salaire de base	Indemnité
1	Jouveinal	Marie	0	0	100,00 €	200,00 €
2	Jouveinal	Marie	10	10	100,00 €	200,00 €
3	Jouveinal	Marie	0	0	100,00 €	200,00 €
4	Jouveinal	Marie	10	10	100,00 €	200,00 €
5	Jouveinal	Marie	12	12	100,00 €	200,00 €

- en [1], on revient au formulaire de saisie ;

- en [2], on fait une simulation avec des saisies erronées. On découvre alors que les validateurs côté client ne fonctionnent plus. Ici, le serveur a été appelé et a renvoyé une page d'erreurs grâce au travail fait au paragraphe 2.12.4, page 269.

Cette anomalie a été identifiée et traitée au paragraphe 1.7.6.7, page 216.

**Travail** : en suivant la démarche du paragraphe 1.7.6.7, page 216, corrigez la fonction Javascript [retourFormulaire] puis faites des tests pour vérifier que les validateurs côté client fonctionnent de nouveau.

## 2.16 Étape 10 : voir la liste des simulations

### 2.16.1 Le problème

Lorsqu'on travaille avec le formulaire de simulation, on peut visualiser la liste des simulations qu'on a faites :

Nº	Nom	Prénom	Heures travaillées	Jours travaillés	Salaire de base	Indemnité
1	Jouveinal	Marie	10	10	100,00 €	200,00 €
2	Laverti	Justine	20	20	100,00 €	200,00 €

### 2.16.2 Écriture de l'action serveur [VoirSimulations]

Le lien Ajax [Voir les simulations] appelle l'action serveur [VoirSimulations] dont le code était jusqu'à maintenant le suivant :

```

1.    // voir les simulations
2.    [HttpPost]
3.    public PartialViewResult VoirSimulations()
4.    {
5.        return PartialView("Simulations");
6.    }

```

La vue partielle [Simulations] qu'elle affiche attend un modèle [IEnumerable<Simulation>] (ligne 1 ci-dessous) :

```

1. @model IEnumerable<Simulation>
2.
3. @using Pam.Web.Models
4.
5. @if (Model.Count() == 0)
6. {
7.     <h2>Votre liste de simulations est vide</h2>
8. }
9. @if (Model.Count() != 0)
10. {
11.     <h2>Liste des simulations</h2>
12. ...
13. }

```

C'est pour cette raison que le lien [Voir les simulations] ne marchait plus.

**Travail** : écrire la nouvelle version de l'action serveur [VoirSimulations] (2 lignes à réécrire) puis faire les tests.

## 2.17 Étape 11 : terminer la session

### 2.17.1 Le problème

On peut à tout moment terminer la session de l'utilisateur avec le lien [Ajax] [Terminer la session]. Ceci a pour effet d'abandonner la session courante pour en commencer une nouvelle. Par ailleurs, on revient à la vue du formulaire :

The screenshots illustrate the behavior of the application when a session is terminated:

- Screenshot 1: The application displays a list of simulations for employees Jouveinal and Laverti. A yellow box labeled '1' highlights the 'Terminer la session' link.
- Screenshot 2: After clicking 'Terminer la session', the application reloads. The employee dropdown is now set to 'Marie Jouveinal'. A yellow box labeled '2' highlights the 'Terminer la session' link again.
- Screenshot 3: The application reloads again, showing a message 'Votre liste de simulations est vide' (Your list of simulations is empty). A yellow box labeled '3' highlights this message.

- en [1], on a fait deux simulations puis on termine la session ;
- en [2], on est revenu au formulaire de saisies. On veut voir les simulations ;
- en [3], à cause du changement de session, la liste des simulations est désormais vide.

### 2.17.2 Écriture de l'action serveur [TerminerSession]

Le lien Ajax [Terminer la session] appelle l'action serveur [TerminerSession] dont le code était jusqu'à maintenant le suivant :

```
1. // terminer la session
2. [HttpPost]
3. public PartialViewResult TerminerSession()
4. {
5.     return PartialView("Formulaire");
6. }
```

La vue partielle [Formulaire] qu'elle affiche attend un modèle [IndexModel] (ligne 1 ci-dessous) :

```
1. @model Pam.Web.Models.IndexModel
2.
3. @using (Html.BeginForm("FaireSimulation", "Pam", FormMethod.Post, new { id = "formulaire" }))
4. {
5. ...
6. }
7. <div id="simulation" />
```

C'est pour cette raison que le lien [Terminer la session] ne marchait plus.

---

**Travail** : écrire la nouvelle version de l'action serveur [TerminerSession] (2 lignes à réécrire) puis faire les tests.

---

**Note** : pour abandonner la session dans l'action, on écrit :

```
Session.Abandon() ;
```

### 2.17.3 Modification de la fonction Javascript [terminerSession]

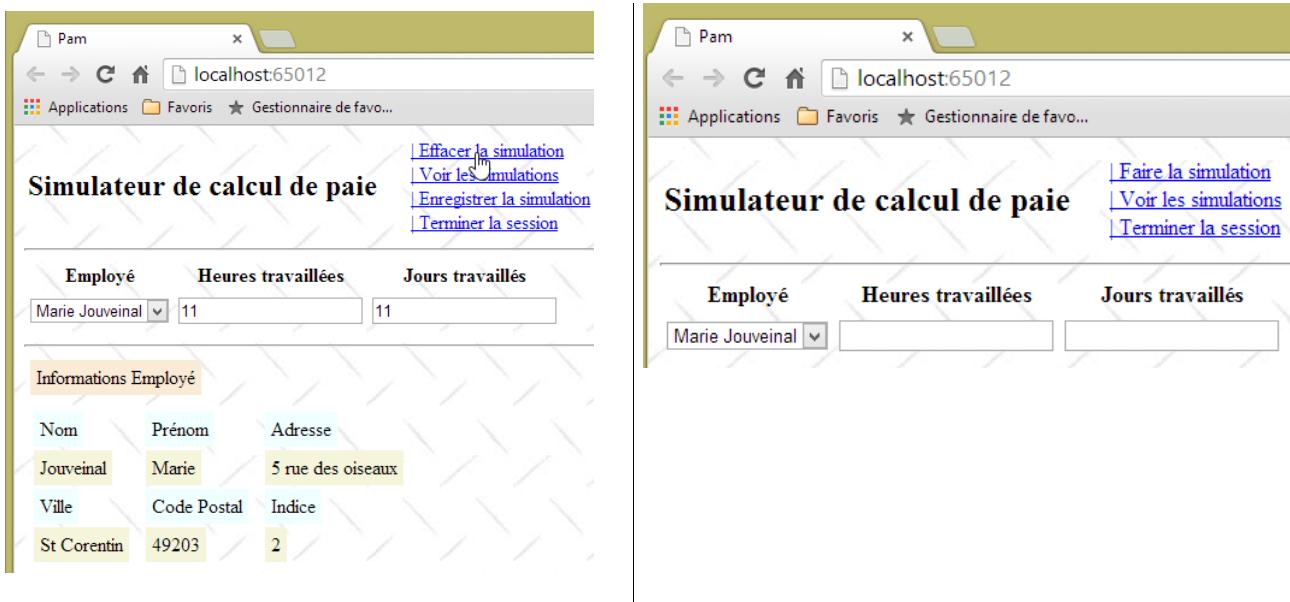
Avec la modification faite précédemment, on peut désormais revenir au formulaire mais une anomalie apparaît alors, celle qui a été décrite précédemment au paragraphe 2.15.3, page 279.

**Travail** : en suivant la démarche que vous avez suivie au paragraphe 2.15.3, page 279, corrigez la fonction Javascript [terminerSession] puis faites des tests pour vérifier que les validateurs côté client fonctionnent de nouveau.

## 2.18 Étape 12 : effacer la simulation

### 2.18.1 Le problème

Lorsqu'on a fait une simulation, on peut l'effacer avec le lien Javascript [Effacer la simulation] :



### 2.18.2 Écriture de l'action client [effacerSimulation]

La fonction Javascript [effacerSimulation] a pour l'instant le code suivant :

```
1. function effacerSimulation() {
2.   // on efface les saisies du formulaire
3.   // ...
4.   // on cache la simulation si elle existe
5.   $("#simulation").hide();
6.   // menu
7.   setMenu([lnkFaireSimulation, lnkTerminerSession, lnkVoirSimulations]);
8. }
```

**Travail** : complétez ce code. On pourra s'inspirer de l'exemple du paragraphe 1.7.6.6, page 215.

## 2.19 Étape 13 : retirer une simulation

### 2.19.1 Le problème

Lorsqu'on a la page des simulations, on peut en supprimer certaines avec le lien Javascript [retirer] :

N°	Nom	Prénom	Heures travaillées	Jours travaillés	Salaire de base	Indemnités	Cotisations sociales	Salaire net	
2	Jouveinal	Marie	10	10	100,00 €	200,00 €	100,00 €	100,00 €	<a href="#">Retirer</a>

Votre liste de simulations est vide

## 2.19.2 Écriture de l'action client [retirerSimulation]

Les liens [retirer] ont la forme HTML suivante :

```
<a href="javascript:retirerSimulation(N)">Retirer</a>
```

où N est le n° de la simulation.

**Travail** : en suivant la démarche des paragraphes 2.9.3 à 2.9.8, écrivez la fonction JS [retirerSimulation]. Celle-ci émettra un appel Ajax de type POST vers l'action [/Pam/RetirerSimulation]. Elle postera la donnée N sous la forme **num=N**.

**Note** : la fonction JS [retirerSimulation] est analogue aux autres fonctions JS que vous avez écrites et qui font un appel Ajax au serveur. La seule nouveauté ici est le POST d'une valeur qui n'est pas dans un formulaire. On sait que les valeurs postées sont rassemblées dans une chaîne de caractères sous la forme :

```
param1=val1&param2=val2&....
```

la fonction JS [retirerSimulation] aura donc la forme suivante :

```

1. function retirerSimulation(N) {
2.   // on fait un appel Ajax à la main
3.   $.ajax({
4.     url: '/Pam/RetirerSimulation',
5.     ...
6.     data:"num="+N,
7.     ...
8.   });
9.   // menu
10.  setMenu([lnkRetourFormulaire, lnkTerminerSession]);
11. }
```

- ligne 6 : la propriété [data] d'un appel Ajax JQuery représente la chaîne postée au serveur.

## 2.19.3 Écriture de l'action serveur [RetirerSimulation]

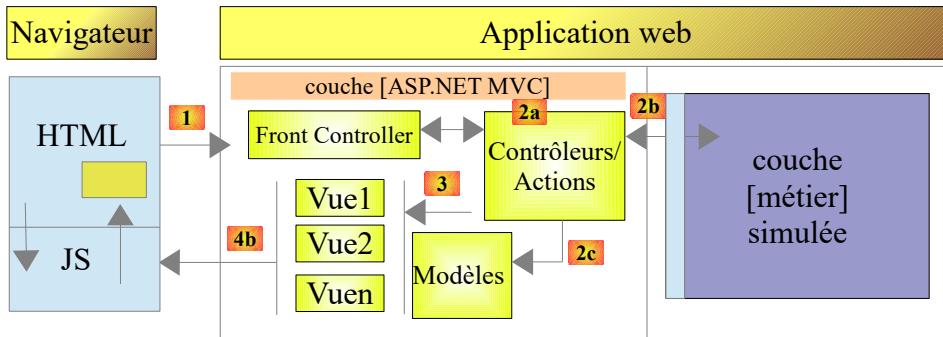
L'action serveur [RetirerSimulation] :

- reçoit un paramètre posté appelé [num] qui est le n° d'une simulation ;
- doit retirer de la liste des simulations enregistrée en session, la simulation qui a ce n° ;
- doit ensuite faire afficher la nouvelle liste de simulations.

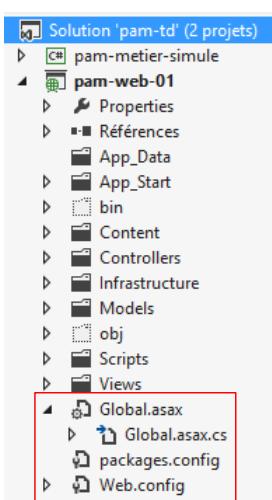
**Travail :** écrire l'action serveur [RetirerSimulation]. Revoyez le paragraphe 1.4.1, page 65, pour savoir comment récupérer le paramètre posté appelé [num].

## 2.20 Étape 14 : amélioration de la méthode d'initialisation de l'application

Notre application web est terminée. Elle est fonctionnelle avec une classe [métier] simulée. Rappelons l'architecture que nous avons développée :



Il reste quelques détails à régler avant de passer à l'implémentation réelle de la couche [métier] et cela se passe dans la méthode d'initialisation de l'application : la méthode [Application\_Start] dans [Global.asax] :



La méthode [Application\_Start] dans [Global.asax] est exécutée une unique fois au démarrage de l'application. C'est là que le fichier de configuration [Web.config] peut être exploité. Pour l'instant, notre méthode [Application\_Start] ressemble à ceci :

```

1. // application
2. protected void Application_Start()
3. {
4.     // -----Auto-généré
5.     AreaRegistration.RegisterAllAreas();
6.     WebApiConfig.Register(GlobalConfiguration.Configuration);
7.     FilterConfig.RegisterGlobalFilters(GlobalFilters.Filters);
8.     RouteConfig.RegisterRoutes(RouteTable.Routes);
9.     BundleConfig.RegisterBundles(BundleTable.Bundles);
10.    // -----
11.    // ----- configuration spécifique
12.    // -----
13.    // données de portée application
14.    ApplicationModel application = new ApplicationModel();
15.    Application["data"] = application;
16.    // instanciation couche [métier]

```

```

17.     application.PamMetier = new PamMetier();
18. ...
19.     // model binders
20. ...
21. }

```

En ligne 17, la couche métier est instanciée par un opérateur **new**. Par ailleurs, le modèle de l'application est défini comme suit :

```

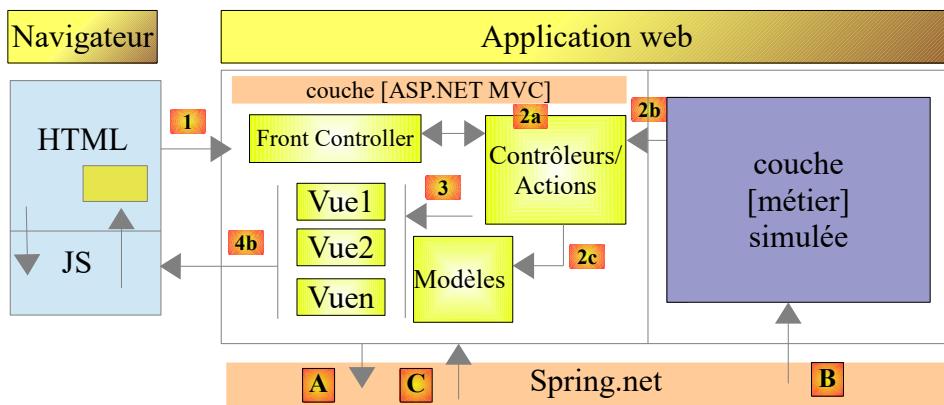
1.  public class ApplicationModel
2.  {
3.      // --- données de portée application ---
4.      public Employe[] Employes { get; set; }
5.      public IPamMetier PamMetier { get; set; }
6.      public SelectListItems[] EmployesItems { get; set; }
7.  }

```

Ligne 5 ci-dessus, on voit que le type de la propriété [PamMetier] est celui de l'interface [IPamMetier]. Ceci signifie que cette propriété peut être initialisée par tout objet implémentant cette interface. Or ligne 17 de [Application\_Start], nous avons écrit en dur le nom d'une classe d'implémentation de [IPamMetier]. Si donc la couche [métier] venait à être implémentée avec une nouvelle classe implémentant [IPamMetier], il faudrait changer cette ligne. Ce n'est pas bien important mais cela peut être évité. La définition de la classe d'implémentation de l'interface [IPamMetier] peut être déportée dans un fichier de configuration. Pour changer d'implémentation, on change alors le contenu de ce fichier de configuration. Le code .NET n'a pas à être changé.

Nous utiliserons ici le conteneur d'injections de dépendances [Spring.net]. Il existe d'autres frameworks .NET pour faire la même chose, peut-être mieux et plus simplement.

L'architecture du projet évolue comme suit :

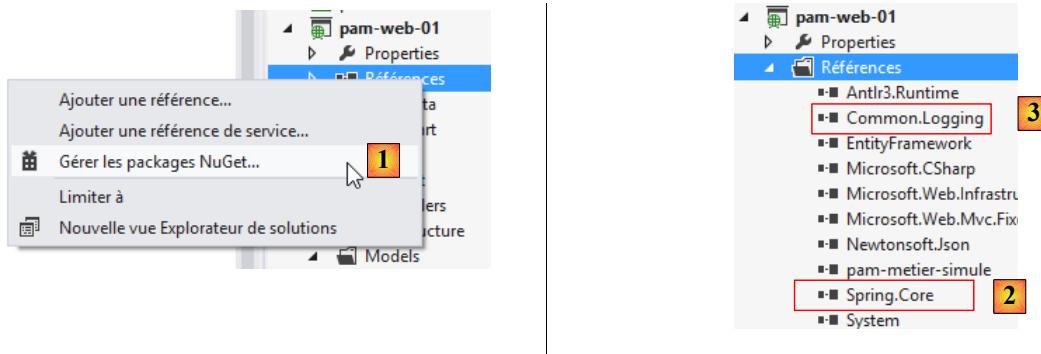


- en [A], la méthode d'initialisation de la couche [ASP.NET MVC] va demander à [Spring.net] une référence sur la couche [métier] simulée ;
- en [B], [Spring.net] va créer la couche [métier] simulée en exploitant son fichier de configuration pour savoir quelle classe il doit instancier ;
- en [C], [Spring.net] va rendre la référence de la couche [métier] simulée à la couche [ASP.NET MVC].

On notera que par défaut, les objets gérés par [Spring.net] sont des **singlenton** : ils n'existent qu'en un unique exemplaire. Ainsi si plus tard dans notre exemple, du code redemande à [Spring.net] une référence sur la couche [métier] simulée, [Spring.net] se contente de rendre la référence sur l'objet initialement créé.

## 2.20.1 Ajout des références [Spring] au projet web

Nous allons utiliser [Spring.net]. Ce framework arrive sous la forme de DLL qu'il faut ajouter aux références du projet. On pourra procéder ainsi :



En [1], cliquer droit sur la branche [References] du projet puis prendre l'option [Gérer les packages NuGet]. Il faut une connexion internet. Ensuite on procèdera comme il a été fait page 157 pour la bibliothèque JQuery [Globalize]. On cherchera le mot clé [Spring.core] et on installera ce package. L'installation amène deux DLL : [Spring.core] [2] et [Common.Logging] [3]. Dans les exemples qui suivent, c'est la version 1.3.2 de Spring qui a été utilisée.

**Note :** si vous n'avez pas de connexion Internet, vous trouverez ces DLL dans un dossier [lib] du support de cette étude de cas.

## 2.20.2 Configuration de [web.config]

La définition de la classe d'implémentation de l'interface [IPamMetier] se fait dans le fichier [web.config].

```

1. <configuration>
2.   <configSections>
3.   ...
4.     <sectionGroup name="spring">
5.       <section name="objects" type="Spring.Context.Support.DefaultSectionHandler, Spring.Core" />
6.       <section name="context" type="Spring.Context.Support.ContextHandler, Spring.Core" />
7.     </sectionGroup>
8.   </configSections>
9.   <!-- configuration Spring -->
10.  <spring>
11.    <context>
12.      <resource uri="config://spring/objects" />
13.    </context>
14.    <objects xmlns="http://www.springframework.net">
15.      <object id="pammetier" type="Pam.Metier.Service.PamMetier, pam-metier-simule"/>
16.    </objects>
17.  </spring>
18. ...

```

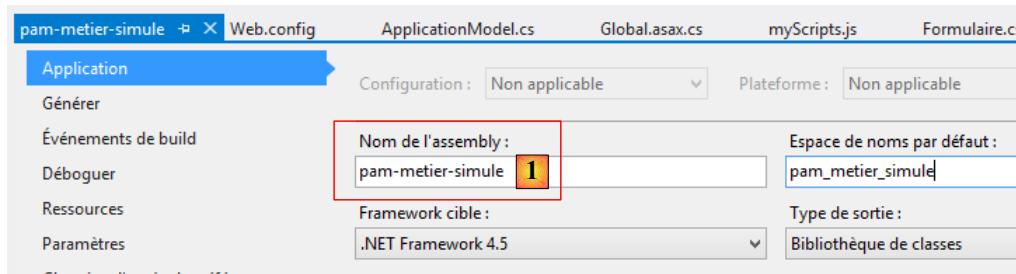
- lignes 2-8 : repérez la balise <configSections> du fichier et insérez dedans les lignes 4-7 ;
- ligne 4 : l'attribut [name="spring"] indique des informations concernant la section [spring] des lignes 10-17 ;
- ligne 5 : définit la classe [Spring.Context.Support.DefaultSectionHandler] située dans la DLL [Spring.Core] comme celle capable de traiter la section [objects] des lignes 14-16 ;
- ligne 6 : définit la classe [Spring.Context.Support.ContextHandler] située dans la DLL [Spring.Core] comme celle capable de traiter la section [context] des lignes 11-13 ;
- lignes 11-13 : cette section apporte l'information [<resource uri="config://spring/objects" />] qui indique que les objets Spring se trouvent dans le fichier de configuration dans la section [/spring/objects], c-à-d aux lignes 14-16 ;
- lignes 14-16 : la balise [objects] introduit les objets Spring ;
- ligne 15 : définit un objet identifié par [id="pammetier"] qui est une instance de la classe [Pam.Metier.Service.PamMetier] située dans la DLL [pam-metier-simule]. Là, il ne faut pas se tromper. Pour l'attribut [id], vous pouvez mettre ce que vous voulez. Vous allez utiliser cet identifiant dans [Global.asax]. La classe [Pam.Metier.Service.PamMetier] est celle de notre couche [métier] simulée. Il faut revenir à sa définition pour connaître son nom complet :

```

namespace Pam.Metier.Service
{
    public class PamMetier : IPamMetier
    {
        ...

```

Pour la DLL [pam-metier-simule], il faut regarder les propriétés du projet C# [pam-metier-simule] :



Il faut utiliser le nom indiqué en [1].

### 2.20.3 Modification de [Application\_Start]

La méthode [Application\_Start] évolue comme suit :

```

1.  using Spring.Context.Support;
2.
3.  // application
4.  protected void Application_Start()
5.  {
6.      // -----Auto-généré
7.      AreaRegistration.RegisterAllAreas();
8.      WebApiConfig.Register(GlobalConfiguration.Configuration);
9.      FilterConfig.RegisterGlobalFilters(GlobalFilters.Filters);
10.     RouteConfig.RegisterRoutes(RouteTable.Routes);
11.     BundleConfig.RegisterBundles(BundleTable.Bundles);
12.     // -----
13.     // ----- configuration spécifique
14.     // -----
15.     // données de portée application
16.     ApplicationModel application = new ApplicationModel();
17.     Application["data"] = application;
18.     // instanciation couche [métier]
19.     application.PamMetier = ContextRegistry.GetContext().GetObject("pammetier") as IPamMetier;
20. ...
21.     // model binders
22. ...
23. }
```

- ligne 19 : on utilise la classe Spring [ContextRegistry] qui est une classe capable d'exploiter le fichier [web.config]. Pour cela, on a besoin d'importer l'espace de noms de la ligne 1. La méthode statique [GetContext] permet d'avoir le contenu des balises [context] qui indiquent où se trouvent les objets Spring. La méthode statique[GetObject] permet ensuite d'avoir un objet particulier identifié par son attribut **id**. On notera que maintenant, le nom de la classe d'implémentation de l'interface [IPamMetier] n'est plus inscrit en dur dans le code. Il est maintenant dans le fichier [web.config].

Après avoir fait toutes ces modifications, testez votre application. Elle doit marcher.

### 2.20.4 Gérer une erreur d'initialisation de l'application

Dans la méthode [Application\_Start] nous avons écrit :

```
application.PamMetier = ContextRegistry.GetContext().GetObject("pammetier") as IPamMetier;
```

L'instruction à droite du signe = peut échouer. Il y a diverses raisons à cela :

- la plus évidente est qu'on se trompe dans le nom de l'objet à instancier ;
- l'autre est que l'instanciation de la couche [métier] se passe mal. Ce ne peut être le cas pour notre couche [métier] simulée mais ça pourra l'être pour notre couche [métier] réelle qui sera connectée à une base de données. Le SGBD peut ne pas être lancé, les informations sur la base à gérer peuvent être incorrectes, etc...

Nous allons gérer une éventuelle exception dans un try / catch. Le code évolue comme suit :

```

1.  // application
2.  protected void Application_Start()
3.  {
4.      // -----Auto-généré
5.  ...
6.      // -----
7.      // ----- configuration spécifique
8.      // -----
```

```

9.     // données de portée application
10.    ApplicationModel application = new ApplicationModel();
11.    Application["data"] = application;
12.    application.InitException = null;
13.    try
14.    {
15.        // instantiation couche [métier]
16.        application.PamMetier = ContextRegistry.GetContext().GetObject("pammetier") as IPamMetier;
17.    }
18.    catch (Exception ex)
19.    {
20.        application.InitException = ex;
21.    }
22.    //si pas d'erreur
23.    if (application.InitException == null)
24.    {
25. ....
26.    }
27.    // model binders
28. ...
29. }
```

- ligne 12, nous introduisons une nouvelle propriété nommée [InitException] dans le modèle de l'application :

```

1.  public class ApplicationModel
2.  {
3.      // --- données de portée application ---
4.      public Employe[] Employes { get; set; }
5.      public IPamMetier PamMetier { get; set; }
6.      public SelectListitem[] EmployesItems { get; set; }
7.      public Exception InitException { get; set; }
8. }
```

- ligne 7 ci-dessus, l'exception qui se produit éventuellement lors de l'initialisation de l'application ;
- lignes 13-21 de [Application\_Start] : l'instanciation de la couche [métier] se fait désormais dans un try / catch ;
- ligne 20 : on mémorise l'exception ;
- lignes 23-26 : s'il n'y a pas eu d'erreur, on exécute le code qu'on avait précédemment ;
- ligne 28 : les [ModelBinders] sont créés qu'il y ait eu erreur ou non. C'est important. On veut s'assurer que le modèle de l'application [ApplicationModel] va bien être lié par le framework.

On sait qu'au démarrage de l'application, l'action serveur [Index] est exécutée. Pour l'instant c'est la suivante :

```

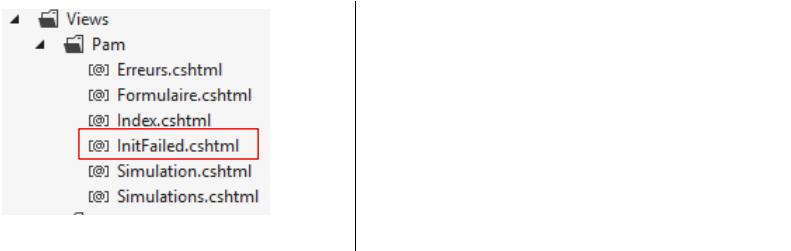
1.  [HttpGet]
2.  public ViewResult Index(ApplicationModel application)
3.  {
4.      return View(new IndexModel() { Application = application });
5. }
```

Ligne 2, l'action [Index] reçoit le modèle de l'application. Elle peut donc savoir si l'initialisation s'est bien passée ou non et afficher une page d'erreurs si l'initialisation a échoué d'une façon ou d'une autre. Nous faisons évoluer le code de la façon suivante :

```

1.  [HttpGet]
2.  public ViewResult Index(ApplicationModel application)
3.  {
4.      // erreur d'initialisation ?
5.      if (application.InitException != null)
6.      {
7.          // page d'erreurs sans menu
8.          return View("InitFailed", Static.GetErreursForException(application.InitException));
9.      }
10.     // pas d'erreur
11.     return View(new IndexModel() { Application = application });
12. }
```

Ligne 8, en cas d'erreur d'initialisation, nous affichons la vue [InitFailed.cshtml] avec pour modèle la liste des messages d'erreur de l'exception qui s'est produite lors de l'initialisation. La méthode [Static.GetErreursForException] a été présentée et expliquée au paragraphe 2.12.4, page 269. La vue [InitFailed.cshtml] sera la suivante :



Son code est le suivant :

```

1. @model IEnumerable<string>
2. @{
3.     Layout = null;
4. }
5. <!DOCTYPE html>
6. <html>
7. <head>
8.     <title>@ViewBag.Title</title>
9.     <meta charset="utf-8" />
10.    <meta name="viewport" content="width=device-width" />
11.    <link rel="stylesheet" href("~/Content/Site.css" />
12. </head>
13. <body>
14.     <table>
15.         <tbody>
16.             <tr>
17.                 <td>
18.                     <h2>Simulateur de calcul de paie</h2>
19.                 </td>
20.             </tbody>
21.         </table>
22.         <hr />
23.         <h2>Les erreurs suivantes se sont produites à l'initialisation de l'application : </h2>
24.     <ul>
25.         @foreach (string msg in Model)
26.         {
27.             <li>@msg</li>
28.         }
29.     </ul>
30. </body>
31. </html>
```

- ligne 1 : le modèle de la vue est une liste de messages d'erreur. Ceux-ci sont affichés dans une liste HTML aux lignes 24-29 ;
- ligne 3 : cette vue n'utilise pas la page maître [\_Layout.cshtml]. En effet, on ne veut pas du menu amené par ce document. On construit donc une page HTML complète (lignes 5-23).

Pour tester, il suffit de modifier dans [Application\_Start], linstanciation de la couche [métier] comme suit :

```

1.     try
2.     {
3.         // instanciation couche [métier]
4.         application.PamMetier = ContextRegistry.GetContext().GetObject("xx") as IPamMetier;
5.     }
6.     catch (Exception ex)
7.     {
8.         application.InitException = ex;
9.     }
```

Ligne 4, on cherche un objet qui n'existe pas dans les objets Spring.

Lorsqu'on valide ces modifications et qu'on lance lapplication, on obtient la page suivante :

The screenshot shows a browser window with the URL `localhost:65012`. The title bar says "Simulateur de calcul de paie". Below the title bar, the address bar also shows `localhost:65012`. The page content is a standard error message:

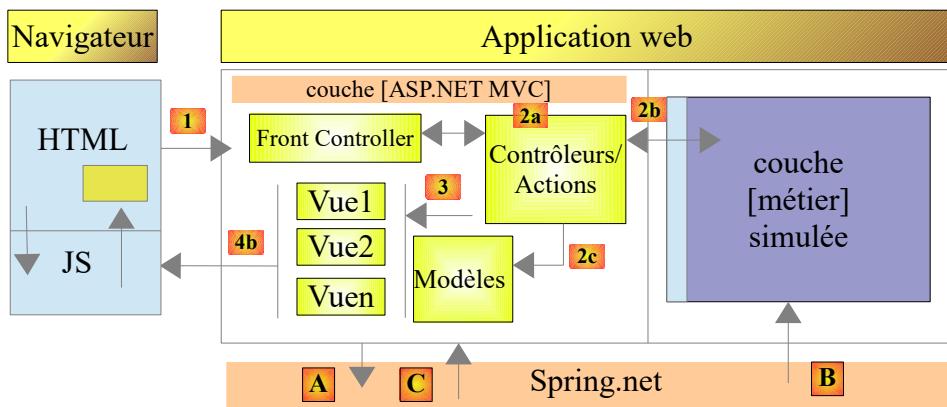
**Les erreurs suivantes se sont produites à l'initialisation de l'application :**

- No object named 'xx' is defined : Cannot find definition for object [xx]

On obtient une page d'erreurs sans menu. L'utilisateur ne peut rien faire d'autre que constater l'erreur. C'est ce qui était souhaité.

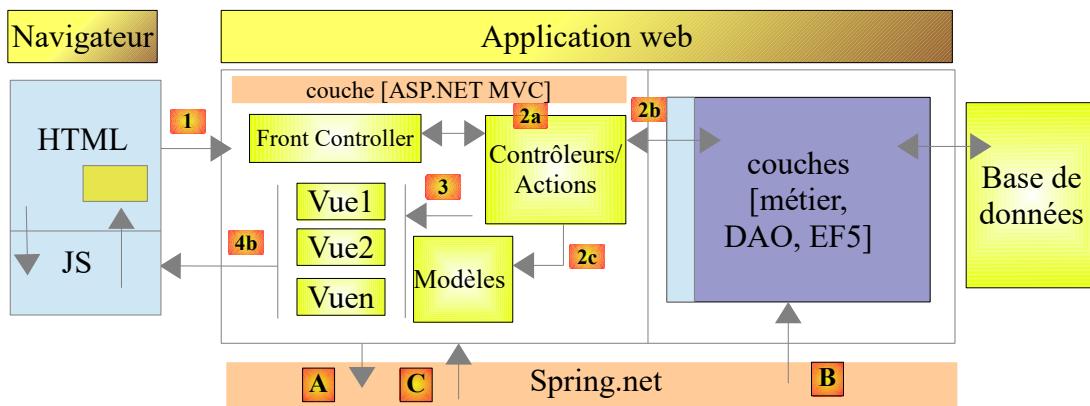
## 2.21 Où en sommes – nous ?

Nous avons désormais une application web opérationnelle qui travaille avec une couche métier simulée. Son architecture est la suivante :



La couche [ASP.NET MVC] travaille avec la couche métier simulée au-travers de l'interface [IPamMetier]. Si nous remplaçons cette couche métier simulée par une couche métier réelle qui respecte cette interface, nous n'aurons pas à modifier le code de la couche web. Grâce à [Spring.net], nous aurons juste à changer dans [web.config] la classe d'implémentation de l'interface [IPamMetier]. Nous partons sur cette voie.

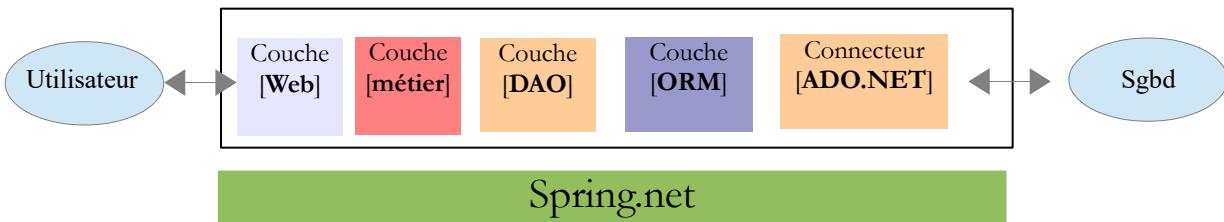
La nouvelle architecture sera la suivante :



Nous allons décrire successivement :

- la couche [EF5] connectée au SGBD. Elle sera implémentée avec Entity Framework 5 (EF5) ;
- la couche [DAO] qui gère l'accès aux données via la couche [EF5]. Cela lui permet d'ignorer l'existence du SGBD. Cette couche se contente de manipuler les entités de l'application [Employe, Cotisations, Indemnites] ;
- la couche [métier] qui implémente le calcul du salaire.

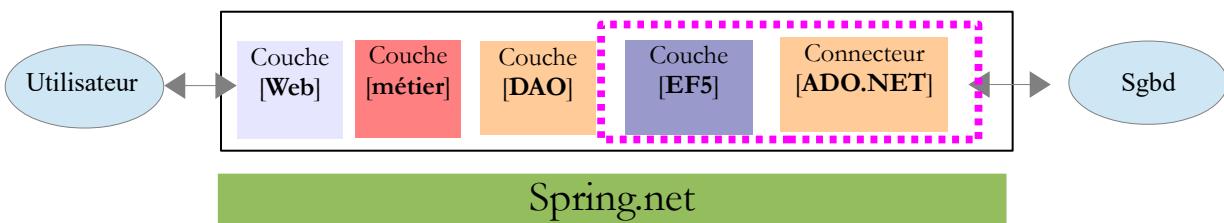
La nouvelle architecture est celle présentée tout au début de ce document au paragraphe 1.1.1, page 9 et que nous rappelons maintenant :



- la couche [Web] est la couche en contact avec l'utilisateur de l'application Web. Celui-ci interagit avec l'application Web au travers de pages Web visualisées par un navigateur. **C'est dans cette couche que se situe ASP.NET MVC et uniquement dans cette couche** ;
- la couche [métier] implémente les règles de gestion de l'application, tels que le calcul d'un salaire ou d'une facture. Cette couche utilise des données provenant de l'utilisateur via la couche [Web] et du SGBD via la couche [DAO] ;
- la couche [DAO] (Data Access Objects), la couche [ORM] (Object Relational Mapper) et le connecteur ADO.NET gèrent l'accès aux données du SGBD. La couche [ORM] fait un pont entre les objets manipulés par la couche [DAO] et les lignes et les colonnes des données d'une base de données relationnelle. Deux ORM sont couramment utilisés dans le monde .NET, NHibernate (<http://sourceforge.net/projects/nhibernate/>) et Entity Framework (<http://msdn.microsoft.com/en-us/data/ef.aspx>) ;
- l'intégration des couches peut être réalisée par un conteneur d'injection de dépendances (Dependency Injection Container) tel que Spring (<http://www.springframework.net/>) ;

Les couches [métier], [DAO], [EF5] vont être implémentées à l'aide de projets C#. A partir de maintenant, nous travaillons avec Visual Studio Express 2012 pour le bureau.

## 2.22 Étape 15 : mise en place de la couche Entity Framework 5



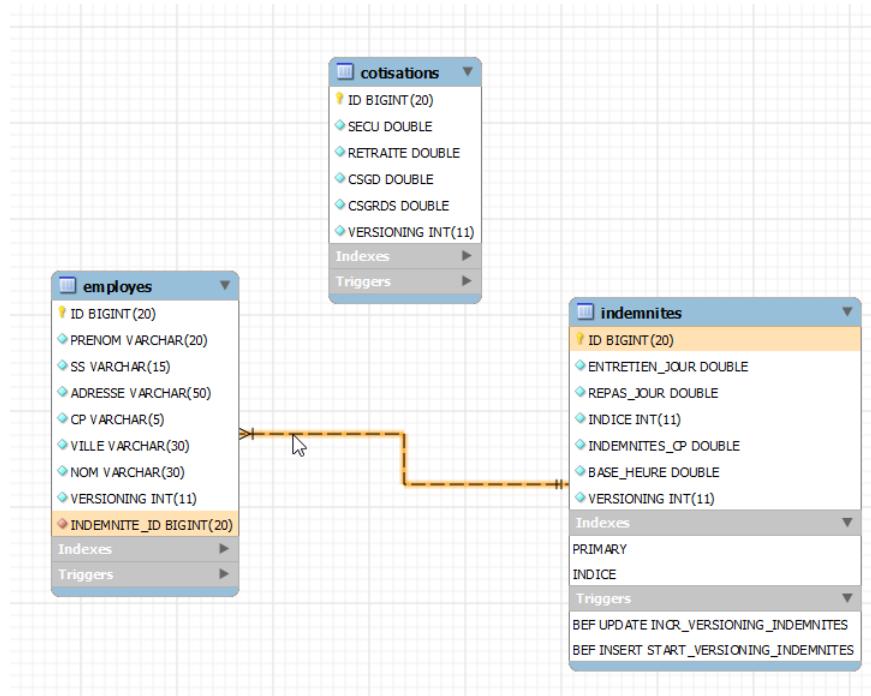
La création de la couche [EF5] est moins affaire de codage que de configuration. Pour apprêhender l'écriture de cette couche, on lira le document **[Introduction à Entity Framework 5 Code First]** disponible à l'URL [<http://tahe.developpez.com/dotnet/ef5cf-02/>]. C'est un document assez volumineux. Les fondamentaux sont dans les quatre premiers chapitres. Les paragraphes à lire plus particulièrement seront précisés. Lorsque nous ferons référence à ce document nous utiliserons la notation **[refEF5]**.

Par ailleurs, nous aurons parfois besoin de concepts C#. On référencera alors le cours **[Introduction au langage C#]** disponible à l'URL [<http://tahe.developpez.com/dotnet/csharp/>] avec la notation **[refC#]**.

### 2.22.1 La base de données

La base de données de l'application a été présentée au paragraphe 2.4, page 223. C'est une base de données MySQL nommée **[dbpam\_ef5]** (**pam**=Paie Assistante Maternelle). Cette base a un administrateur appelé **root** sans mot de passe.

Rappelons le schéma de la base de données. Elle a trois tables :



Il y a une relation de clé étrangère entre la colonne EMPLOYES(INDEMNITE\_ID) et la colonne INDEMNITES(ID). Une partie de la structure de cette base est dictée par son utilisation avec EF5.

Le script SQL de création de la base est le suivant :

```

1. -- phpMyAdmin SQL Dump
2. -- version 3.5.1
3. -- http://www.phpmyadmin.net
4. --
5. -- Client: localhost
6. -- Généré le: Lun 04 Novembre 2013 à 09:34
7. -- Version du serveur: 5.5.24-log
8. -- Version de PHP: 5.4.3
9.
10. SET SQL_MODE="NO_AUTO_VALUE_ON_ZERO";
11. SET time_zone = "+00:00";
12.
13.
14. /*!40101 SET @OLD_CHARACTER_SET_CLIENT=@@CHARACTER_SET_CLIENT */;
15. /*!40101 SET @OLD_CHARACTER_SET_RESULTS=@@CHARACTER_SET_RESULTS */;
16. /*!40101 SET @OLD_COLLATION_CONNECTION=@@COLLATION_CONNECTION */;
17. /*!40101 SET NAMES utf8 */;
18.
19. --
20. -- Base de données: `dbpam_ef5`
21. --
22.
23. --
24.
25. --
26. -- Structure de la table `cotisations`
27. --
28.
29. CREATE TABLE IF NOT EXISTS `cotisations` (
30.   `ID` bigint(20) NOT NULL AUTO_INCREMENT,
31.   `SECU` double NOT NULL,
32.   `RETRAITE` double NOT NULL,
33.   `CSGD` double NOT NULL,
34.   `CSGRDS` double NOT NULL,
35.   `VERSIONING` int(11) NOT NULL,
36.   PRIMARY KEY (`ID`)
37. ) ENGINE=InnoDB DEFAULT CHARSET=utf8 AUTO_INCREMENT=12 ;
38.
39. --
40. -- Contenu de la table `cotisations`
41. --
42.
43. INSERT INTO `cotisations` (`ID`, `SECU`, `RETRAITE`, `CSGD`, `CSGRDS`, `VERSIONING`) VALUES
44. (11, 9.39, 7.88, 6.15, 3.49, 1);

```

```

45.
46. --
47. -- Déclencheurs `cotisations`
48. --
49. DROP TRIGGER IF EXISTS `INCR_VERSIONING_COTISATIONS`;
50. DELIMITER //
51. CREATE TRIGGER `INCR_VERSIONING_COTISATIONS` BEFORE UPDATE ON `cotisations`
52. FOR EACH ROW BEGIN
53.     SET NEW.VERSIONING:=OLD.VERSIONING+1;
54. END
55. //
56. DELIMITER ;
57. DROP TRIGGER IF EXISTS `START_VERSIONING_COTISATIONS`;
58. DELIMITER //
59. CREATE TRIGGER `START_VERSIONING_COTISATIONS` BEFORE INSERT ON `cotisations`
60. FOR EACH ROW BEGIN
61.     SET NEW.VERSIONING:=1;
62. END
63. //
64. DELIMITER ;
65.
66. -- -----
67.
68. --
69. -- Structure de la table `employes`
70. --
71.
72. CREATE TABLE IF NOT EXISTS `employes` (
73.     `ID` bigint(20) NOT NULL AUTO_INCREMENT,
74.     `PRENOM` varchar(20) CHARACTER SET latin1 NOT NULL,
75.     `SS` varchar(15) CHARACTER SET latin1 NOT NULL,
76.     `ADRESSE` varchar(50) CHARACTER SET latin1 NOT NULL,
77.     `CP` varchar(5) CHARACTER SET latin1 NOT NULL,
78.     `VILLE` varchar(30) CHARACTER SET latin1 NOT NULL,
79.     `NOM` varchar(30) CHARACTER SET latin1 NOT NULL,
80.     `VERSIONING` int(11) NOT NULL,
81.     `INDEMNITE_ID` bigint(20) NOT NULL,
82.     PRIMARY KEY (`ID`),
83.     UNIQUE KEY `SS` (`SS`),
84.     KEY `FK_EMPLOYES_INDEMNITE_ID` (`INDEMNITE_ID`)
85. ) ENGINE=InnoDB DEFAULT CHARSET=utf8 AUTO_INCREMENT=26 ;
86.
87. --
88. -- Contenu de la table `employes`
89. --
90.
91. INSERT INTO `employes` (`ID`, `PRENOM`, `SS`, `ADRESSE`, `CP`, `VILLE`, `NOM`, `VERSIONING`, `INDEMNITE_ID`) VALUES
92. (24, 'Marie', '254104940426058', '5 rue des oiseaux', '49203', 'St Corentin', 'Jouveinal', 1, 93),
93. (25, 'Justine', '260124402111742', 'La Brûlerie', '49014', 'St Marcel', 'Laverti', 1, 94);
94.
95. --
96. -- Déclencheurs `employes`
97. --
98. DROP TRIGGER IF EXISTS `INCR_VERSIONING_EMPLOYES`;
99. DELIMITER //
100. CREATE TRIGGER `INCR_VERSIONING_EMPLOYES` BEFORE UPDATE ON `employes`
101. FOR EACH ROW BEGIN
102.     SET NEW.VERSIONING:=OLD.VERSIONING+1;
103. END
104. //
105. DELIMITER ;
106. DROP TRIGGER IF EXISTS `START_VERSIONING_EMPLOYES`;
107. DELIMITER //
108. CREATE TRIGGER `START_VERSIONING_EMPLOYES` BEFORE INSERT ON `employes`
109. FOR EACH ROW BEGIN
110.     SET NEW.VERSIONING:=1;
111. END
112. //
113. DELIMITER ;
114.
115. -- -----
116.
117. --
118. -- Structure de la table `indemnites`
119. --
120.
121. CREATE TABLE IF NOT EXISTS `indemnites` (
122.     `ID` bigint(20) NOT NULL AUTO_INCREMENT,
123.     `ENTRETIEN_JOUR` double NOT NULL,
124.     `REPAS_JOUR` double NOT NULL,
125.     `INDICE` int(11) NOT NULL,
126.     `INDEMNITES_CP` double NOT NULL,
127.     `BASE_HEURE` double NOT NULL,
128.     `VERSIONING` int(11) NOT NULL,
129.     PRIMARY KEY (`ID`),
130.     UNIQUE KEY `INDICE` (`INDICE`)
131. ) ENGINE=InnoDB DEFAULT CHARSET=utf8 AUTO_INCREMENT=95 ;

```

```

132.
133. --
134. -- Contenu de la table `indemnites` 
135. --
136.
137. INSERT INTO `indemnites` (`ID`, `ENTRETIEN_JOUR`, `REPAS_JOUR`, `INDICE`, `INDEMNITES_CP`, `BASE_HEURE`, `VERSIONING`)
   VALUES
138. (93, 2.1, 3.1, 2, 15, 2.1, 1),
139. (94, 2, 3, 1, 12, 1.93, 1);
140.
141. --
142. -- Déclencheurs `indemnites`
143. --
144. DROP TRIGGER IF EXISTS `INCR_VERSIONING_INDEMNITES`;
145. DELIMITER //
146. CREATE TRIGGER `INCR_VERSIONING_INDEMNITES` BEFORE UPDATE ON `indemnites`
147. FOR EACH ROW BEGIN
148.   SET NEW.VERSIONING:=OLD.VERSIONING+1;
149. END
150. //
151. DELIMITER ;
152. DROP TRIGGER IF EXISTS `START_VERSIONING_INDEMNITES`;
153. DELIMITER //
154. CREATE TRIGGER `START_VERSIONING_INDEMNITES` BEFORE INSERT ON `indemnites`
155. FOR EACH ROW BEGIN
156.   SET NEW.VERSIONING:=1;
157. END
158. //
159. DELIMITER ;
160.
161. --
162. -- Contraintes pour les tables exportées
163. --
164.
165. --
166. -- Contraintes pour la table `employes`
167. --
168. ALTER TABLE `employes`
169. ADD CONSTRAINT `FK_EMPLOYES_INDEMNITE_ID` FOREIGN KEY (`INDEMNITE_ID`) REFERENCES `indemnites` (`ID`);
170.
171. /*!40101 SET CHARACTER_SET_CLIENT=@OLD_CHARACTER_SET_CLIENT */;
172. /*!40101 SET CHARACTER_SET_RESULTS=@OLD_CHARACTER_SET_RESULTS */;
173. /*!40101 SET COLLATION_CONNECTION=@OLD_COLLATION_CONNECTION */;
```

On notera les points suivants :

- lignes 30, 73, 122 : les clés primaires des tables sont en mode [AUTO\_INCREMENT]. C'est MySQL qui les gère et non EF5 ;
- ligne 83 : le n° SS a une contrainte d'unicité ;
- ligne 130 : l'indice de l'employé a une contrainte d'unicité ;
- lignes 168-169 : la clé étrangère de la table [employes] vers la table [indemnites] ;
- ligne 49 : un déclencheur ou [Trigger] est un script SQL embarqué par le SGBD et qui est exécuté à certains moments ;
- lignes 51-54 : le déclencheur [INCR\_VERSIONING\_COTISATIONS] se déclenche avant toute modification d'une ligne de la table [cotisations]. Il incrémente alors d'une unité la colonne [VERSIONING] ;
- lignes 59-62 : le déclencheur [START\_VERSIONING\_COTISATIONS] se déclenche avant toute insertion d'une nouvelle ligne dans la table [cotisations]. Il initialise alors à 1 la colonne [VERSIONING] ;
- au final, la colonne [VERSIONING] vaut 1 lorsqu'une ligne est créée dans la table [cotisations] puis est incrémentée de 1 à chaque modification faite sur cette ligne. Ce mécanisme permet à EF5 de gérer la concurrence d'accès à une ligne de la table [cotisations] de la façon suivante :

- x       un processus P1 lit une ligne L de la table [cotisations] au temps T1. La ligne a la colonne [VERSIONING] V1 ;
- x       un processus P2 lit la même ligne L de la table [cotisations] au temps T2. La ligne a la colonne [VERSIONING] V1 parce que le processus P1 n'a pas encore validé sa modification ;
- x       le processus P1 modifie la ligne L et valide sa modification. La colonne [VERSIONING] de la ligne L passe alors à V1+1 à cause du déclencheur [INCR\_VERSIONING\_COTISATIONS] ;
- x       le processus P2 fait ensuite de même. EF5 lance alors une exception car le processus P2 a une ligne avec une colonne [VERSIONING] ayant une valeur V1 différente de celle trouvée en base qui est V1+1. On ne peut modifier une ligne que si on a la même valeur de [VERSIONING] que dans la base.

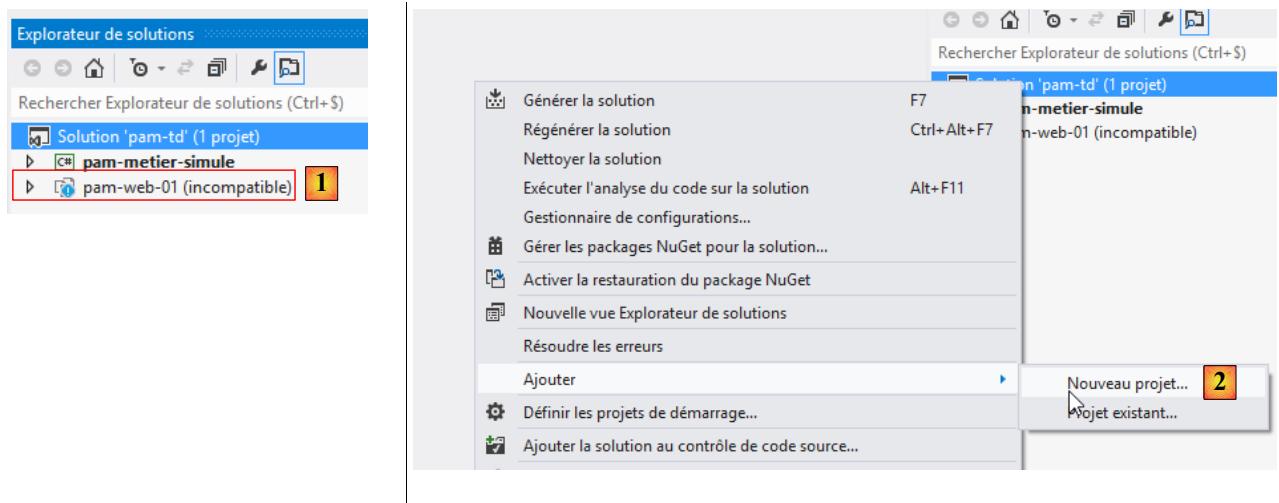
On appelle cela la **gestion optimiste** des accès concurrents. Avec EF5, un champ jouant ce rôle doit avoir l'annotation [**ConcurrencyCheck**].

- un mécanisme analogue est créé pour la table [employes] (lignes 98-113) et la table [indemnites] (lignes 144-159).

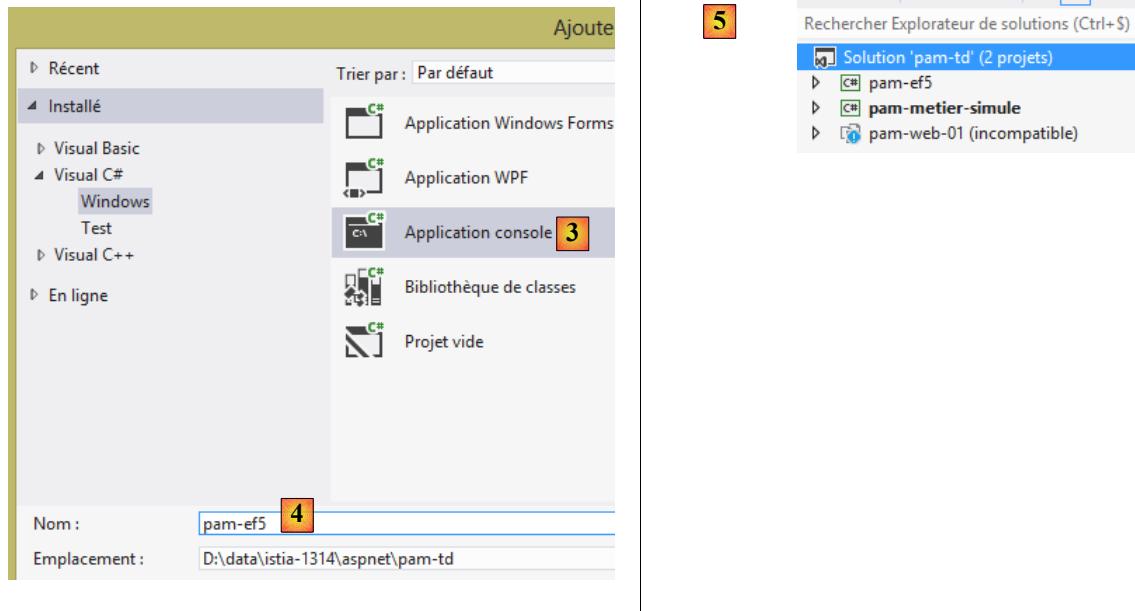
**Travail** : créez la base de données MySQL [dbpam\_ef5] à l'aide du script SQL précédent. La base [dbpam\_ef5] doit être créée auparavant car le script ne la crée pas. On jouera ensuite le script SQL sur cette base.

## 2.22.2 Le projet Visual Studio

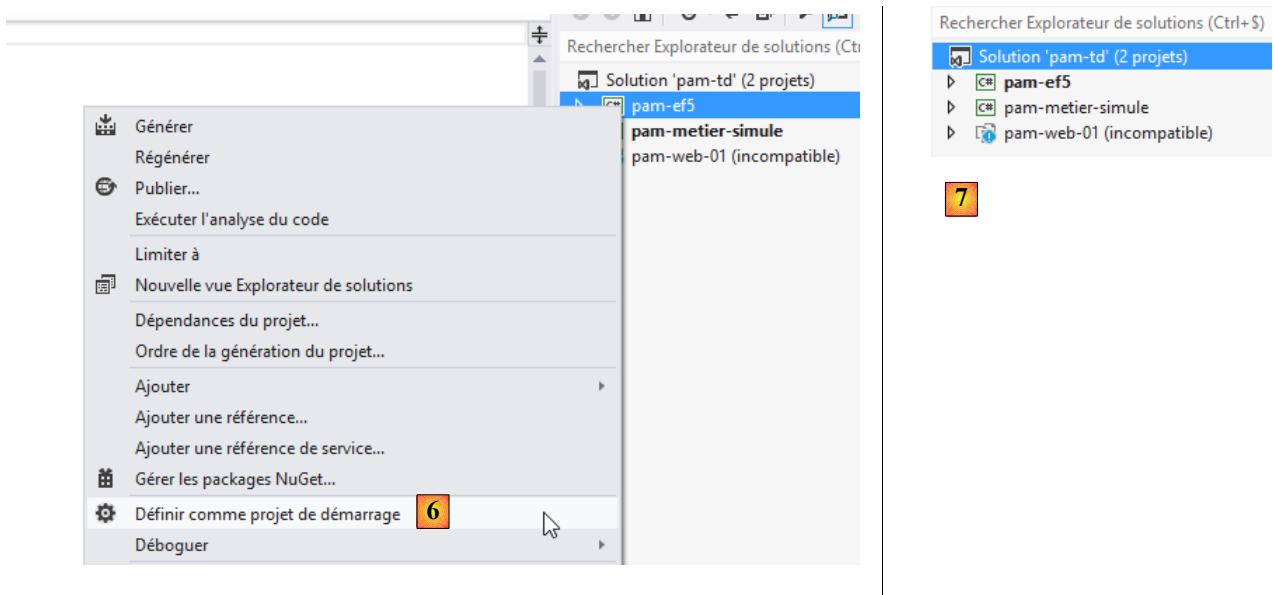
Avec Visual Studio Express 2012 pour le bureau, nous chargeons la solution [pam-td] utilisée lors de la construction de la couche [web] :



- en [1], VS 2012 Express pour le bureau n'arrive pas à charger le projet web [pam-web-01]. C'est normal et ce n'est pas gênant ;
- en [2], on ajoute un nouveau projet à la solution [pam-td] ;



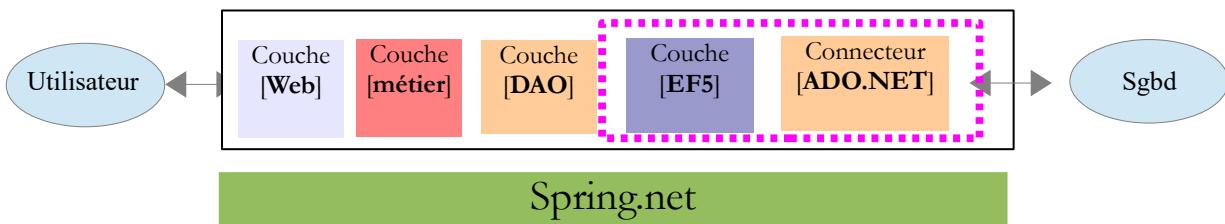
- en [3], le projet est de type [console] et s'appelle [4] [pam-ef5] ;
- en [5], le projet créé. Son nom n'est pas en gras donc ce n'est pas le projet de démarrage de la solution ;



- en [6] et [7], on définit le nouveau projet comme projet de démarrage.

### 2.22.3 Ajout des références nécessaires au projet

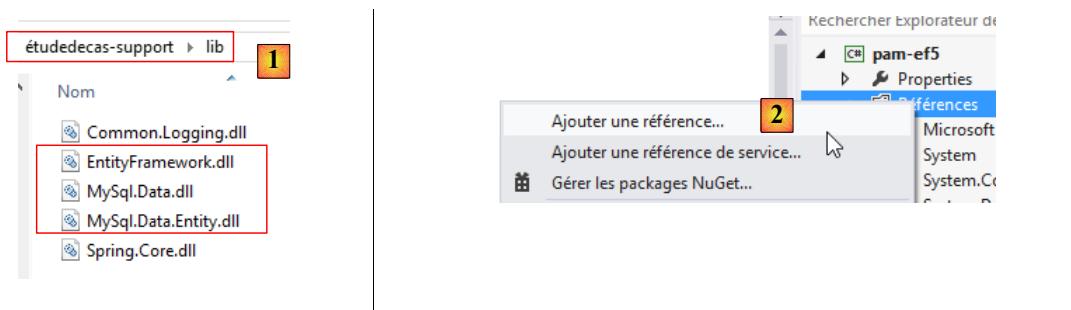
Situons le projet dans son ensemble :



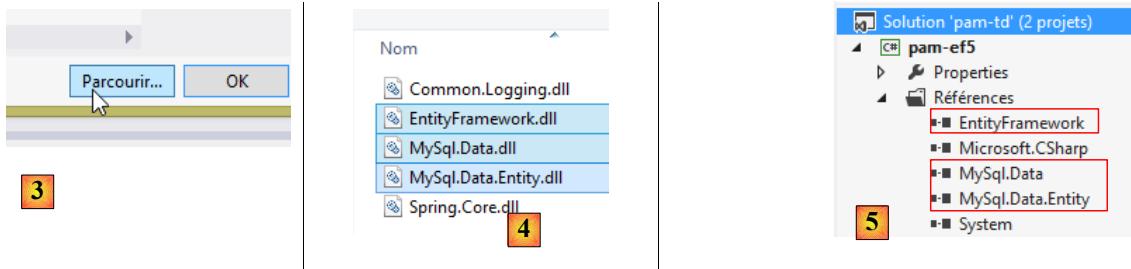
Notre projet a besoin d'un certain nombre de DLL :

- la DLL d'Entity Framework 5 ;
- la DLL du connecteur ADO.NET du SGBD MySQL.

La paragraphe 4.2 de [refEF5] explique comment installer ces DLL à l'aide de l'outil [NuGet]. Actuellement (nov 2013), la version disponible d'Entity Framework est la version 6 (EF6). Malheureusement, il semble que le connecteur ADO.NET du SGBD MySQL disponible (nov 2013) via [NuGet] ne soit pas compatible avec EF6. Aussi a-t-on placé dans un dossier [lib] [1] la DLL d'EF5 ainsi que les autres DLL nécessaires au projet [pam-ef5]

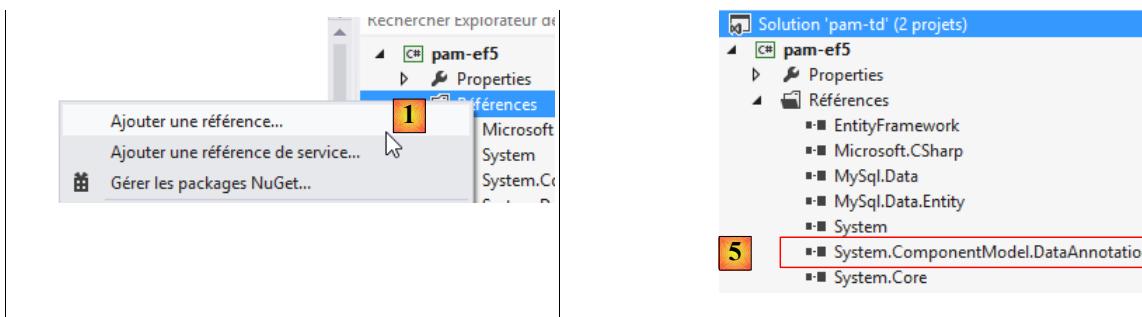


Nous avons placé d'autres DLL dans le dossier [lib]. Nous les utiliserons ultérieurement. En [2], nous ajoutons ces nouvelles DLL au projet.

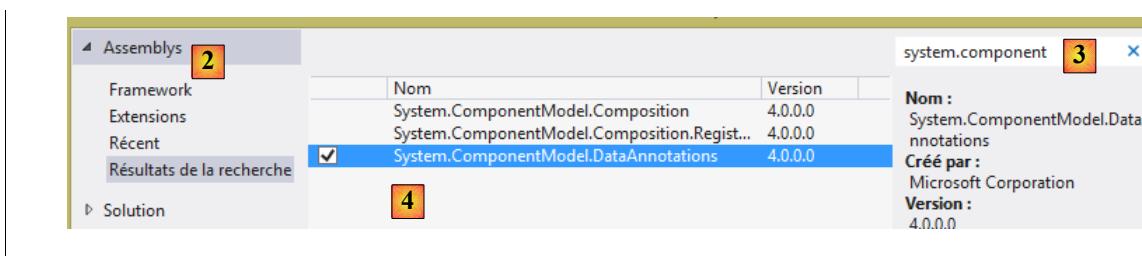


- en [3], on parcourt le système de fichiers jusqu'au dossier [lib] ;
- en [4], on sélectionne les trois DLL puis on valide deux fois ;
- en [5], les trois DLL ont été ajoutées au références du projet.

Il nous faut une autre DLL. Celle-ci sera trouvée parmi celles du framework .NET de la machine.



- en [1], ajoutez une nouvelle référence au projet ;

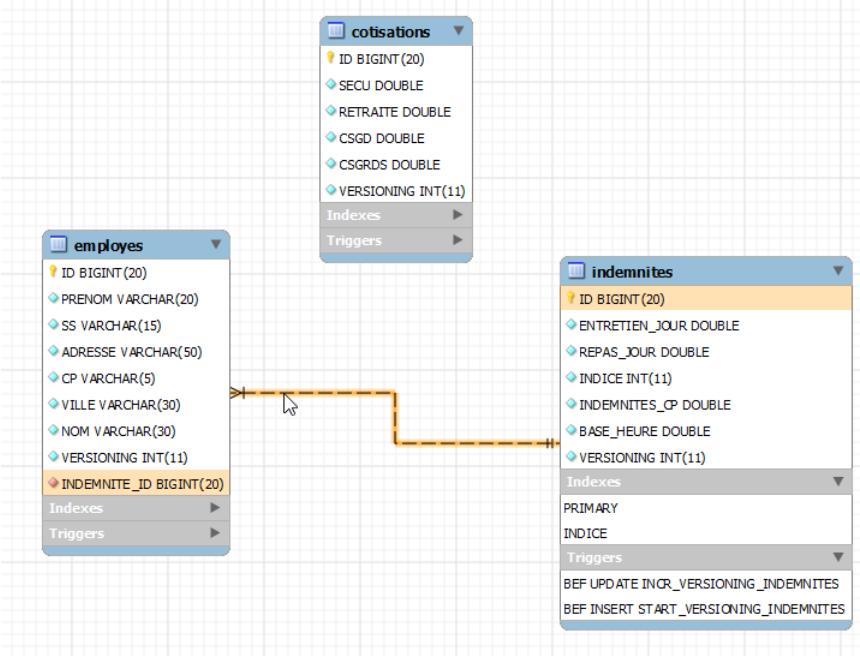


- en [2], sélectionnez [Assemblies] ;
- en [3], tapez [system.component] ;
- en [4], sélectionnez l'assembly [System.ComponentModel.DataAnnotations] ;
- en [5], la référence a été ajoutée.

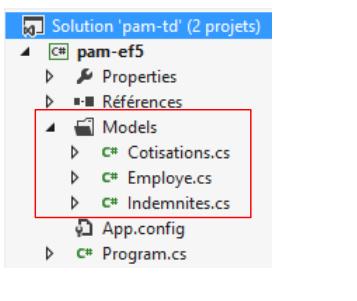
Nous sommes désormais prêts pour coder et configurer.

## 2.22.4 Les entités Entity Framework

Les entités Entity framework sont des classes dans lesquelles on encapsule les lignes des différentes tables de la base de données. Rappelons celles-ci :



Dans la couche [web], nous avions utilisé les entités [Employe, Cotisations, Indemnités] (voir paragraphe 2.7.3, page 230). Elles n'étaient pas des images fidèles des tables. Ainsi les colonnes [ID, VERSIONING] avaient été ignorées. Ici, ce ne va pas être le cas car elles sont utilisées par l'ORM EF5. Nous allons donc leur ajouter les propriétés manquantes. Nous créons ces entités dans un dossier [Models] du projet :



Leur nouveau code est désormais le suivant :

### Classe [Cotisations]

```

1.  using System;
2.
3.  namespace Pam.EF5.Entites
4.  {
5.      public class Cotisations
6.      {
7.          public int Id { get; set; }
8.          public double CsgRds { get; set; }
9.          public double Csgd { get; set; }
10.         public double Secu { get; set; }
11.         public double Retraite { get; set; }
12.         public int Versioning { get; set; }
13.
14.         // signature
15.         public override string ToString()
16.         {
17.             return string.Format("Cotisations[{0},{1},{2},{3}, {4}, {5}]", Id, Versioning, CsgRds, Csgd, Secu, Retraite);
18.         }
19.     }
20. }
```

- ligne 3 : l'espace de noms a été adapté au nouveau projet ;

- les propriétés des lignes 7 et 12 ont été rajoutées pour refléter la structure de la table [cotisations] ;
- ligne 17 : la méthode [ToString] affiche maintenant les deux nouveaux champs.

### Classe [Indemnites]

```

1.  using System;
2.
3.  namespace Pam.EF5.Entites
4.  {
5.      public class Indemnites
6.      {
7.          public int Id { get; set; }
8.          public int Indice { get; set; }
9.          public double BaseHeure { get; set; }
10.         public double EntretienJour { get; set; }
11.         public double RepasJour { get; set; }
12.         public double IndemnitesCp { get; set; }
13.         public int Versioning { get; set; }
14.
15.         // signature
16.         public override string ToString()
17.         {
18.             return string.Format("Indemnités[{0},{1},{2},{3},{4}, {5}, {6}]", Id, Versioning, Indice, BaseHeure, EntretienJour,
RepasJour, IndemnitesCp);
19.         }
20.     }
21. }
```

- ligne 3 : l'espace de noms a été adapté au nouveau projet ;
- les propriétés des lignes 7 et 13 ont été rajoutées pour refléter la structure de la table [indemnites] ;
- ligne 18 : la méthode [ToString] affiche maintenant les deux nouveaux champs.

### Classe [Employe]

```

1.  using System;
2.
3.  namespace Pam.EF5.Entites
4.  {
5.
6.      public class Employe
7.      {
8.          public int Id { get; set; }
9.          public string SS { get; set; }
10.         public string Nom { get; set; }
11.         public string Prenom { get; set; }
12.         public string Adresse { get; set; }
13.         public string Ville { get; set; }
14.         public string CodePostal { get; set; }
15.         public Indemnites Indemnites { get; set; }
16.         public int Versioning { get; set; }
17.
18.         // signature
19.         public override string ToString()
20.         {
21.             return string.Format("Employé[{0},{1},{2},{3},{4}, {5}, {6}, {7}]", Id, Versioning, SS, Nom, Prenom, Adresse, Ville,
CodePostal);
22.         }
23.     }
24. }
```

- ligne 3 : l'espace de noms a été adapté au nouveau projet ;
- les propriétés des lignes 8 et 16 ont été rajoutées pour refléter la structure de la table [employes] ;
- ligne 21 : la méthode [ToString] affiche maintenant les deux nouveaux champs.

Pour être utilisables par l'ORM EF5, les propriétés de ces classes doivent être décorées par des annotations.

---

**Travail** : en vous aidant du paragraphe 3.4 [Création de la base à partir des entités] de [[refEF5](#)], ajoutez aux entités [Employe, Cotisations, Indemnites] les annotations nécessaires à EF5.

---

### **Conseils :**

- il s'agit seulement de créer des annotations. Ne suivez pas la partie [création de base] du paragraphe référencé ;
- pour l'annotation [Table], vous suivrez l'exemple MySQL du paragraphe 4.2 de [[refEF5](#)] ;
- pour l'annotation [ConcurrencyCheck] sur la propriété [Versioning], vous suivrez l'exemple Oracle du paragraphe 5.2 de [[refEF5](#)] ;
- pour la clé étrangère que possède la table [employes] sur la table [indemnités], vous suivrez l'exemple 3.4.2 de [[refEF5](#)]. Vous ajouterez ainsi une nouvelle propriété à l'entité [Employe] :

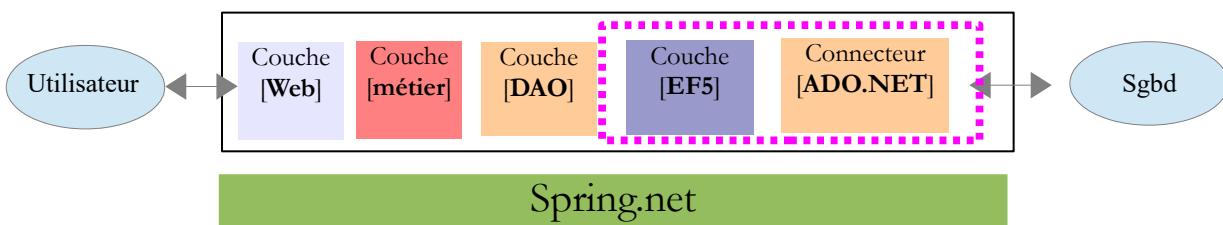
```
public int IndemniteId { get; set; }
```

dont la valeur sera celle de la colonne [INDEMNITES\_ID] de la table [employes]. Vous mettrez aux propriétés [IndemniteId] et [Indemnites] de l'entité [Employe] les annotations de clé étrangère. Pour cela, suivez l'exemple 3.4.2 de [refEF5] ;

- vous ne gérerez pas les relations inverses des clés étrangères ;
- ce travail nécessite un peu de lecture de [refEF5].

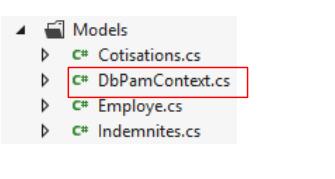
## 2.22.5 Configuration de l'ORM EF5

Resituons le projet dans son ensemble :



La couche [EF5] va accéder à la base de données via le connecteur [ADO.NET] du SGBD MySQL. Elle a besoin d'un certain nombre de renseignements pour accéder à cette base. Ceux-ci sont placés à divers endroits du projet.

Nous devons tout d'abord créer le contexte de la base de données. Ce contexte est une classe dérivée de la classe système [System.Data.Entity.DbContext]. Elle sert à définir les images objets des tables de la base de données. Nous placerons cette classe dans le dossier [Models] du projet avec les entités EF5 :



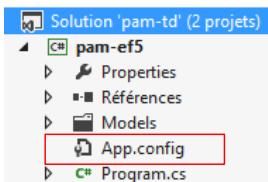
La classe [DbPamContext] sera la suivante :

```
1. using Pam.EF5.Entites;
2. using System.Data.Entity;
3.
4. namespace Pam.Models
5. {
6.     public class DbPamContext : DbContext
7.     {
8.         public DbSet<Employe> Employes { get; set; }
9.         public DbSet<Cotisations> Cotisations { get; set; }
10.        public DbSet<Indemnites> Indemnites { get; set; }
11.    }
12. }
```

- ligne 6 : la classe [DbPamContext] dérive de la classe système [DbContext] ;
- lignes 8-10 : les images objets des trois tables de la base de données. Leur type est [DbSet<Entity>] où [Entity] est une des entités Entity Framework que nous venons de définir. On peut voir le type [DbSet] comme une collection d'entités. Elle peut être requêtée avec LINQ (Language INtegrated Query). Le lecteur ne connaissant pas LINQ est invité à lire le paragraphe 3.5.4 [Apprentissage de LINQ avec LINQPad] de [refEF5].

Nous appellerons par la suite la classe [DbPamContext] **contexte de persistance** de la base de données [dbpam\_ef5]. C'est une terminologie habituelle dans les ORM (Object Relational Mapper). Ce contexte de persistance est une image objet de la base de données. On parle également de **synchronisation du contexte de persistance** avec la base de données : les modifications, ajouts, suppressions faits sur le contexte de persistance sont répercutés sur la base de données. Cette synchronisation se fait à des moments précis : à la fermeture du contexte de persistance, à la fin d'une transaction ou avant une requête SQL SELECT sur la base.

Les informations sur le SGBD et la base de données sont placées dans [app.config].



La configuration nécessaire dans [app.config] est expliquée aux paragraphes suivants de [refEF5] :

- 3.4 pour le SGBD SQL Server. C'est là que les grands principes de la configuration d'EF5 sont posés ;
- 4.2 pour le SGBD MySQL.

Nous suivons ce dernier paragraphe et nous configurons le fichier [app.config] de la façon suivante :

```

1.  <?xml version="1.0" encoding="utf-8" ?>
2.  <configuration>
3.    <startup>
4.      <supportedRuntime version="v4.0" sku=".NETFramework,Version=v4.5" />
5.    </startup>
6.    <!-- configuration EF5 -->
7.    <!-- chaîne de connexion à la base de données [dbpam_ef5] -->
8.    <connectionStrings>
9.      <add name="DbPamContext"
10.        connectionString="Server=localhost;Database=dbpam_ef5;Uid=root;Pwd=";
11.        providerName=" MySql.Data.MySqlClient" />
12.    </connectionStrings>
13.    <!-- le factory provider de MySQL -->
14.    <system.data>
15.      <DbProviderFactories>
16.        <remove invariant=" MySql.Data.MySqlClient" />
17.        <add name="MySQL Data Provider" invariant=" MySql.Data.MySqlClient" description=".Net Framework Data Provider for
MySQL"
18.          type=" MySql.Data.MySqlClient.MySqlClientFactory, MySql.Data, Version=6.5.4.0, Culture=neutral,
PublicKeyToken=C5687FC88969C44D"
19.        />
20.      </DbProviderFactories>
21.    </system.data>
22.  </configuration>
```

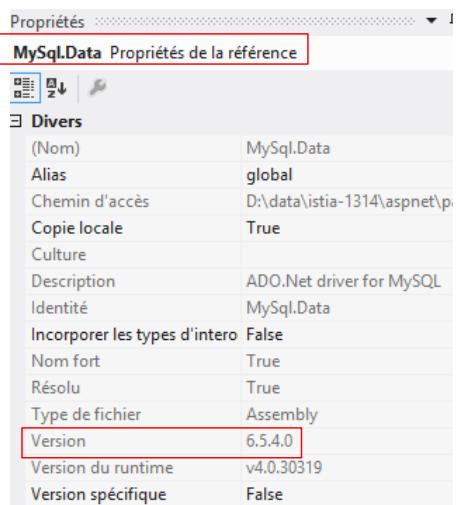
- les lignes 6-21 ont été ajoutées. Elles doivent s'insérer dans la balise <configuration> des lignes 2 et 22 ;
- lignes 8-12 : définissent des chaînes de connexion à des bases de données, un concept ADO.NET (voir paragraphe 7.3.5 dans [refC#]) ;
- lignes 9-11 : définissent la chaîne de connexion à la base de données MySQL [dbpam\_ef5] ;
- ligne 9 : le nom de la chaîne de connexion. **Ici, on ne peut pas mettre n'importe quoi.** Par défaut, il faut mettre le nom de la classe implémentant le contexte de la base de données :

```

1.  public class DbPamContext : DbContext
2.  {
3.    public DbSet<Employe> Employes { get; set; }
4.    public DbSet<Cotisations> Cotisations { get; set; }
5.    public DbSet<Indemnites> Indemnites { get; set; }
6. }
```

La classe s'appelle [DbPamContext]. Ligne 9 de [app.config], il faut alors mettre [name="DbPamContext"] ;

- ligne 10 : une chaîne de connexion propre au SGBD MySQL :
  - [Server=localhost] : adresse IP de la machine hébergeant le SGBD. Ici c'est la machine locale [localhost] ;
  - [Database=dbpam\_ef5] : nom de la base de données,
  - [Uid=root] : login avec lequel on va se connecter à la base,
  - [Pwd=] : mot de passe de ce login. Ici pas de mot de passe ;
- ligne 10 : [providerName=" MySql.Data.MySqlClient"] est le nom du connecteur ADO.NET à utiliser. Ce nom est celui de l'attribut [invariant] de la ligne 17. On peut mettre n'importe quoi tant qu'on respecte la règle précédente et qu'un provider de même invariant n'ait pas déjà été enregistré ;
- lignes 15-20 : définissent une usine (factory) de connecteurs (provider) ADO.NET. Le [DbProviderFactory] est un concept un peu nébuleux pour moi. Si j'en crois son nom, ce serait une classe capable de générer le connecteur ADO.NET qui donne accès au SGBD, ici MySQL5. On fait en général du copier / coller de ces lignes. Elles sont nécessaires. On fera attention à l'attribut [Version=6.5.4.0] de la ligne 16. Ce n° de version doit correspondre au n° de version de la DLL [MySql.Data] que vous avez ajoutée aux références du projet :

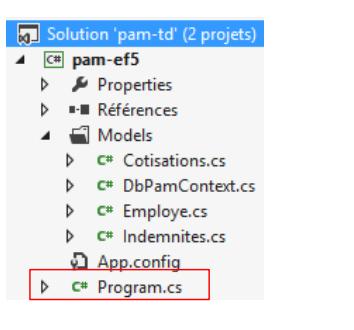


- la ligne 16 est importante. Parce qu'on ne peut pas installer deux providers de mêmes noms, on commence par supprimer un éventuel provider installé qui porterait le nom de celui qu'on installe ligne 17 ;

C'est tout. C'est compliqué et obscur lorsqu'on le fait la première fois, puis au fil du temps cela devient simple car c'est toujours la même chose que l'on répète.

## 2.22.6 Test de la couche [EF5]

Nous sommes prêts à tester notre couche [EF5]. On le fait à l'aide du programme [Program.cs] déjà présent :



Nous allons afficher le contenu de la base. Si on y arrive ce sera un début d'indication que notre configuration est correcte. Un exemple de code est disponible au paragraphe 3.5.3 de [refEF5]. Le code de [Program.cs] sera le suivant :

```

1.  using Pam.EF5.Entites;
2.  using Pam.Models;
3.  using System;
4.
5.  namespace Pam
6.  {
7.      class Program
8.      {
9.          static void Main(string[] args)
10.         {
11.             try
12.             {
13.                 using (var context = new DbPamContext())
14.                 {
15.                     // on affiche le contenu des tables
16.                     Console.WriteLine("Liste des employés -----");
17.                     foreach (Employe employe in context.Employes)
18.                     {
19.                         Console.WriteLine(employe);
20.                     }
21.                     Console.WriteLine("Liste des indemnités -----");
22.                     foreach (Indemnites indemnite in context.Indemnites)
23.                     {
24.                         Console.WriteLine(indemnite);
25.                     }

```

```

26.         Console.WriteLine("Liste des cotisations -----");
27.         foreach (Cotisations cotisations in context.Cotisations)
28.         {
29.             Console.WriteLine(cotisations);
30.         }
31.     }
32. }
33. catch (Exception e)
34. {
35.     Console.WriteLine(e);
36.     return;
37. }
38. }
39. }
40. }

```

- ligne 13 : toute opération sur la BD se fait au travers du contexte de cette base. Nous avons implémenté ce contexte avec la classe [DbPamContext]. Nous l'avons appelé également **contexte de persistance** de la base ;
- lignes 13, 31 : les opérations sur le contexte de persistance se font dans une clause [using]. Le contexte de persistance est ouvert au début de la clause [using] et automatiquement fermé à la sortie de cette clause. Cela implique que toute modification faite sur le contexte de persistance dans la clause [using] sera répercutée sur la base de données à la sortie de la clause. Une série d'ordres SQL est alors envoyée à la BD à l'intérieur d'une transaction. Ce qui veut dire que si un ordre SQL échoue, tous les ordres SQL émis précédemment sont annulés. Une exception est alors lancée par EF5 ;
- ligne 17 : l'expression [context.Employees] désigne l'image objet de la table [employees]. On rappelle que [Employees] est une propriété du contexte de persistance [DbPamContext] :

```

1.     public class DbPamContext : DbContext
2.     {
3.         public DbSet<Employe> Employees { get; set; }
4.         public DbSet<Cotisations> Cotisations { get; set; }
5.         public DbSet<Indemnites> Indemnites { get; set; }
6.     }

```

- ligne 17 : le fait que le [foreach] parcourt la collection [context.Employees] va ramener tous les employés de la base de données dans le contexte de persistance. Un ordre SQL SELECT va donc être émis par EF5 ;
- lignes 17-20 : on parcourt la collection des employés et ligne 19, on utilise la méthode [ToString] de la classe [Employe] pour afficher les employés sur la console ;
- lignes 21-25 : idem pour la collection des indemnités ;
- lignes 27-30 : idem pour la collection des cotisations.

Revenons sur la définition de l'entité [Employe] :

```

1. using System;
2.
3. namespace Pam.EF5.Entites
4. {
5.
6.     public class Employe
7.     {
8.         public int Id { get; set; }
9.         public string SS { get; set; }
10.        public string Nom { get; set; }
11.        public string Prenom { get; set; }
12.        public string Adresse { get; set; }
13.        public string Ville { get; set; }
14.        public string CodePostal { get; set; }
15.        public Indemnites Indemnites { get; set; }
16.        public int Versioning { get; set; }
17.
18.        // signature
19.        public override string ToString()
20.        {
21.            return string.Format("Employé[{0},{1},{2},{3},{4},{5}, {6}, {7}]", Id, Versioning, SS, Nom, Prenom, Adresse, Ville,
22.                CodePostal);
23.        }
24.    }

```

- ligne 15 : un employé a une référence sur une indemnité.

Lorsqu'on ramène un employé dans le contexte de persistance, ramène-t-on son indemnité avec ? La réponse est **non par défaut**. C'est la notion de **[Lazy Loading]**. Les entités référencées au sein d'une autre entité ne sont pas amenées dans le contexte de persistance avec cette autre entité. Elles ne le sont que lorsqu'elles sont **demandées** par le code au sein d'un contexte de persistance **ouvert**. Si le contexte de persistance est fermé, une exception est alors lancée.

Ainsi, si la méthode [ToString] avait référencé la propriété [Indemnites] comme ci-après :

```

1.     // signature
2.     public override string ToString()
3.     {
4.         return string.Format("Employé[{0},{1},{2},{3},{4},{5},{6},{7},{8}]", Id, Versioning, SS, Nom, Prenom, Adresse,
5.         Ville, CodePostal, Indemnités);
    }

```

l'opération suivante dans [Program.cs] :

```

1.     foreach (Employé employé in context.Employés)
2.     {
3.         Console.WriteLine(employé);
4.     }

```

aurait ramené dans le contexte de persistance non seulement les employés mais également leurs indemnités, parce que ligne 3, la méthode [Employé.ToString] est appelée et qu'elle référence l'entité [Indemnités].

L'exécution de [Program.cs] donne les résultats suivants :

```

1. Liste des employés -----
2. Employé[24,1,254104940426058,Jouveinal,Marie,5 rue des oiseaux,St Corentin,49203]
3. Employé[25,1,260124402111742,Laverti,Justine,La Brûlerie,St Marcel,49014]
4. Liste des indemnités -----
5. Indemnités[93,1,2,2,1,2,1,3,1,15]
6. Indemnités[94,1,1,1,93,2,3,12]
7. Liste des cotisations -----
8. Cotisations[11,1,3,49,6,15,9,39,7,88]

```

Que faire si ça ne marche pas ? Vous êtes mal... Il y a de nombreuses sources d'erreur possibles :

- vérifiez la configuration d'EF5 (paragraphe 2.22.5, page 300) ;
- vérifiez vos entités Entity Framework (paragraphe 2.22.4, page 297).

## 2.22.7 DLL de la couche [EF5]

Nous faisons de notre projet une bibliothèque de classes afin qu'à la génération, un assembly .dll soit généré plutôt qu'un .exe. Cela se fait dans les propriétés du projet comme il a été vu au paragraphe 2.7.6, page 235 pour la couche métier simulée.

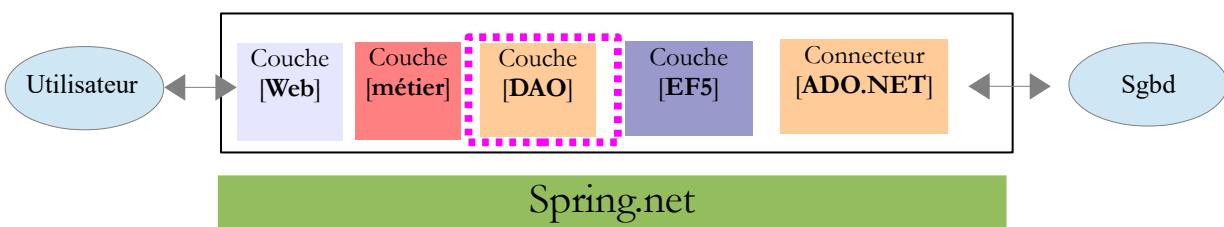
---

**Travail** : transformez le type du projet [pam-ef5] en bibliothèque de classes puis régénérez le projet.

---

## 2.23 Étape 16 : mise en place de la couche [DAO]

### 2.23.1 L'interface de la couche [DAO]



Comme nous l'avions fait pour la couche [métier] simulée, la couche [DAO] sera accessible via une interface. Quelle sera-t-elle ?

Regardons l'interface [IPamMetier] de la couche [métier] simulée que nous avons construite :

```

1.     public interface IPamMetier {
2.         // liste de toutes les identités des employés
3.         Employé[] GetAllIdentitesEmployés();
4.
5.         // ----- le calcul du salaire
6.         FeuilleSalaire GetSalaire(string ss, double heuresTravaillées, int joursTravaillés);
7.     }

```

Ligne 3, la méthode [GetAllIdentitesEmployés] sert à alimenter la liste déroulante de la page d'accueil :



Ces employés devront être cherchés dans la base de données.

Ligne 6, la méthode [GetSalaire] permet de calculer la feuille de salaire d'un employé dont on a le n° SS. Rappelons la définition du type [FeuilleSalaire] :

```

1.  public class FeuilleSalaire
2.  {
3.
4.      // propriétés automatiques
5.      public Employe Employe { get; set; }
6.      public Cotisations Cotisations { get; set; }
7.      public ElementsSalaire ElementsSalaire { get; set; }
8.  }
```

Les informations des lignes 5 et 6 viendront de la base de données. Rappelons qu'un employé a une propriété [Indemnites]. Cette information devra être ramenée également.

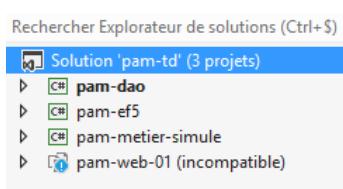
On pourrait donc partir avec l'interface suivante pour la couche [DAO] :

```

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

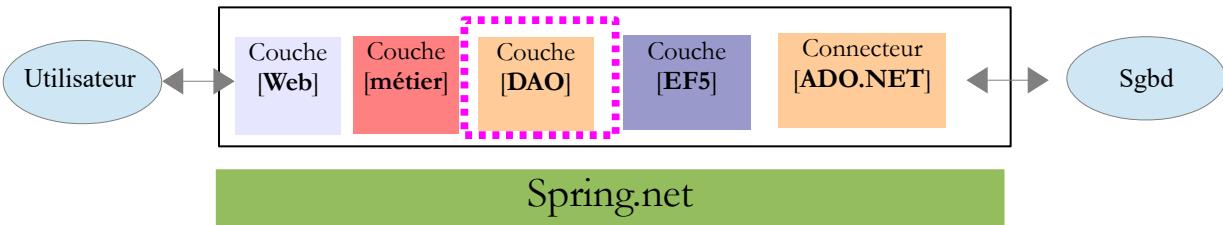
## 2.23.2 Le projet Visual Studio

**Travail** : ajouter à la solution [pam-td] un nouveau projet de type [console] appelé [pam-dao]. Faites-en le projet de démarrage de la solution.



## 2.23.3 Ajout des références nécessaires au projet

Situons le projet dans son ensemble :



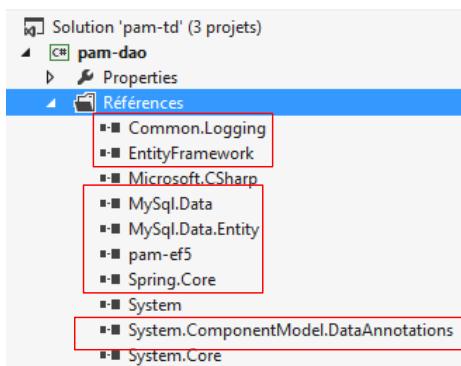
Le projet [pam-dao] a besoin d'un certain nombre de DLL :

- toutes celles référencées par le projet [pam-ef5] ;
- celle du projet [pam-ef5] lui-même.

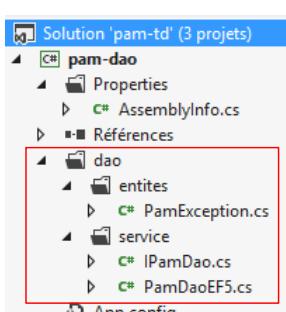
Par ailleurs, nous allons utiliser [Spring.net] pour instancier la couche [DAO]. Pour cela, nous avons besoin des DLL [Spring.core] et [Common.Logging]. Ces DLL sont dans le dossier [lib] du support de l'étude de cas.

---

**Travail** : ajoutez ces différentes références au projet [pam-dao].



#### 2.23.4 Implémentation de la couche [DAO]



Ci-dessus, la classe [PamException] est celle qui a été définie au paragraphe 2.7.4, page 233. On change simplement son espace de noms (ligne 1 ci-dessous) :

```

1.  namespace Pam.Dao.Entites
2.  {
3.      // classe d'exception
4.      public class PamException : Exception
5.  {
6.      ....
7.  }
8. }
```

L'interface [IPamDao] est celle que nous venons de définir au paragraphe 2.23.1, page 304 :

```
1.  using Pam.EF5.Entities;
2.
3.  namespace Pam.Dao.Service
4.  {
5.      public interface IPamDao
6.      {
7.          // liste de toutes les identités des employés
8.          Employe[] GetAllIdentitesEmployes();
9.          // un employé particulier avec ses indemnités
10.         Employe GetEmploye(string ss);
11.         // liste de toutes les cotisations
12.         Cotisations GetCotisations();
13.     }
14. }
```

La classe [PamDaoEF5] implémente cette interface à l'aide de l'ORM EF5. Son code est le suivant :

```
1.  using Pam.Dao.Entities;
2.  using Pam.EF5.Entities;
3.  using Pam.Models;
4.  using System;
5.  using System.Linq;
6.
7.  namespace Pam.Dao.Service
8.  {
9.
10.     public class PamDaoEF5 : IPamDao
11.     {
12.         // champs privés
13.         private Cotisations cotisations;
14.         private Employe[] employes;
15.
16.         // Constructeur
17.         public PamDaoEF5()
18.         {
19.             // cotisation
20.             try
21.             {
22.                 ....
23.             }
24.             catch (Exception e)
25.             {
26.                 throw new PamException("Erreur système lors de la construction de la couche [DAO]", e, 1);
27.             }
28.         }
29.
30.         // GetCotisations
31.         public Cotisations GetCotisations()
32.         {
33.             return cotisations;
34.         }
35.
36.         // GetAllIdentitesEmploye
37.         public Employe[] GetAllIdentitesEmployes()
38.         {
39.             return employes;
40.         }
41.
42.         // GetEmploye
43.         public Employe GetEmploye(string ss)
44.         {
45.             try
46.             {
47.                 ....
48.                 catch (Exception e)
49.                 {
50.                     throw new PamException(string.Format("Erreur système lors de la recherche de l'employé [{0}]", ss), e, 2);
51.                 }
52.             }
53.         }
54.     }
```

A savoir :

- ligne 10 : la classe [PamDaoEF5] implémente l'interface [IPamDao] ;
- les tables [cotisations] et [employes] sont mises en cache dans les propriétés des lignes 13-14. Les employés sont **sans leurs indemnités** ;
- lignes 17-28 : c'est le constructeur qui initialise les lignes 13-14 ;
- lignes 43-52 : la méthode [GetEmploye] ramène un employé **avec ses indemnités**. Elle reçoit en paramètre le n° de sécurité sociale de cet employé. Si l'employé n'existe pas dans la base, la méthode rendra le pointeur **null**.

---

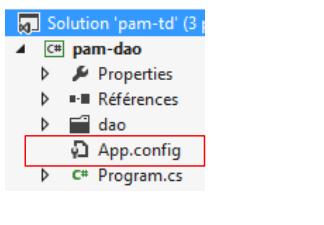
Travail : compléter le code de la classe [PamDaoEF5].

---

Pour le constructeur, on s'inspirera du code de test de la couche [EF5] présenté au paragraphe 2.22.6, page 302. Pour la méthode [GetEmploye] on s'inspirera de l'exemple du paragraphe 3.5.7 [Eager and Lazy loading] de [refEF5].

### 2.23.5 Configuration de la couche [DAO]

Comme il a été fait au paragraphe 2.22.5, page 300, il nous faut configurer EF5 dans le fichier [app.config] du projet :



---

Travail 1 : configurez EF5 dans [app.config]. Il suffit de reprendre ce qui a été fait dans le fichier [app.config] de la couche [EF5].

---

Notre programme de test va utiliser [Spring.net] pour obtenir une référence sur la couche [DAO].

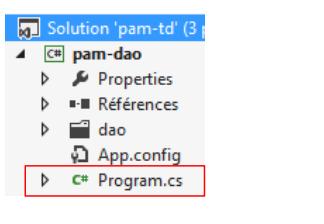
---

Travail 2 : en vous aidant de ce qui a été fait au paragraphe 2.20.2, page 286, modifiez le fichier de configuration [app.config] du projet [pam-dao] afin qu'il définisse un objet Spring appelé [pamdao] associé à la classe [PamDaoEF5] que nous venons de construire. Les fichiers [app.config] et [web.config] ont la même structure. Il faut faire attention à ce que la balise <configSections> soit la première balise rencontrée derrière la balise racine <configuration>.

---

### 2.23.6 Test de la couche [DAO]

Nous sommes prêts à tester notre couche [DAO]. On le fait à l'aide du programme [Program.cs] déjà présent :



Nous allons tester les différentes fonctionnalités de l'interface de la couche [DAO]. Le code de [Program.cs] sera le suivant :

```
1.  using Pam.Dao.Service;
2.  using Pam.EF5.Entites;
3.  using Spring.Context.Support;
4.  using System;
5.
6.  namespace Pam.Dao.Tests
7.  {
8.      public class Program
9.      {
10.          public static void Main()
11.          {
12.              try
13.              {
14.                  // instanciation couche [dao]
15.                  IPamDao pamDao = (IPamDao)ContextRegistry.GetContext().GetObject("pamdao");
16.                  // liste des identités des employés
17.                  foreach (Employe Employe in pamDao.GetAllIdentitesEmployes())
18.                  {
19.                      Console.WriteLine(Employe.ToString());
20.                  }
21.                  // un employé avec ses indemnités
22.                  Console.WriteLine("-----");
23.                  Employe e = pamDao.GetEmploye("254104940426058");
24.                  Console.WriteLine("employé= {0}, indemnités={1}", e, e.Indemnites);
25.                  Console.WriteLine("-----");
26.                  // un employé qui n'existe pas
```

```
27.         Employe employe = pamDao.GetEmploye("xx");
28.         Console.WriteLine("Employé n° xx");
29.         Console.WriteLine(employe == null ? "null" : employe.ToString());
30.         Console.WriteLine("-----");
31.         // liste des cotisations
32.         Cotisations cotisations = pamDao.GetCotisations();
33.         Console.WriteLine(cotisations.ToString());
34.     }
35.     catch (Exception ex)
36.     {
37.         // affichage exception
38.         Console.WriteLine(ex.ToString());
39.     }
40.     //pause
41.     Console.ReadLine();
42. }
43. }
44. }
```

- ligne 15 : on obtient une référence sur la couche [DAO] grâce à [Spring.net].

Les résultats de l'exécution de ce programme sont les suivants :

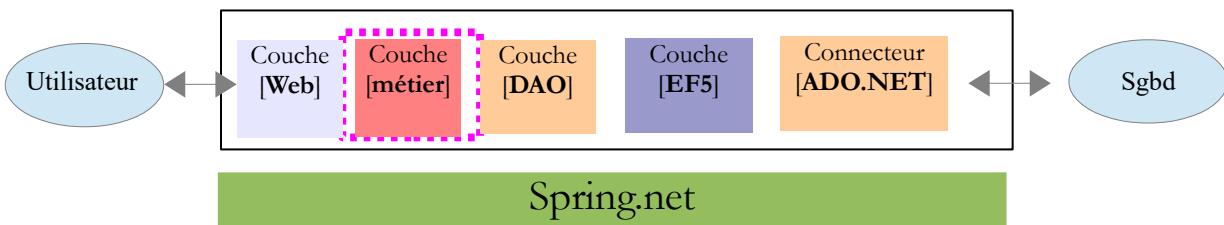
```
1. Employé[22,1,254104940426058,Jouveinal,Marie,5 rue des oiseaux,St Corentin,49203]
2. Employé[23,1,260124402111742,Laverti,Justine,La Brûlerie,St Marcel,49014]
3. -----
4. employé= Employé[22,1,254104940426058,Jouveinal,Marie,5 rue des oiseaux,St Corentin,49203],
indemnités=Indemnités[91,1,2,2,1,2,1,3,1,15]
5. -----
6. Employé n° xx
7. null
8. -----
9. Cotisations[10,1,3,49,6,15,9,39,7,88]
```

### 2.23.7 DLL de la couche [DAO]

**Travail :** transformez le type du projet [pam-dao] en bibliothèque de classes puis régénérez le projet (refaire ce qui a été fait au paragraphe 2.22.7, page 304).

## 2.24 Étape 17 : mise en place de la couche [métier]

### 2.24.1 L'interface de la couche [métier]

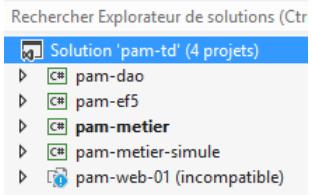


L'interface de la couche [métier] sera l'interface [IPamMetier] de la couche [métier] simulée que nous avons construite au paragraphe 2.7.2, page 228.

```
8.     public interface IPamMetier {
9.         // liste de toutes les identités des employés
10.        Employe[] GetAllIdentitesEmployes();
11.
12.        // ----- le calcul du salaire
13.        FeuilleSalaire GetSalaire(string ss, double heuresTravaillées, int joursTravaillés);
14.    }
```

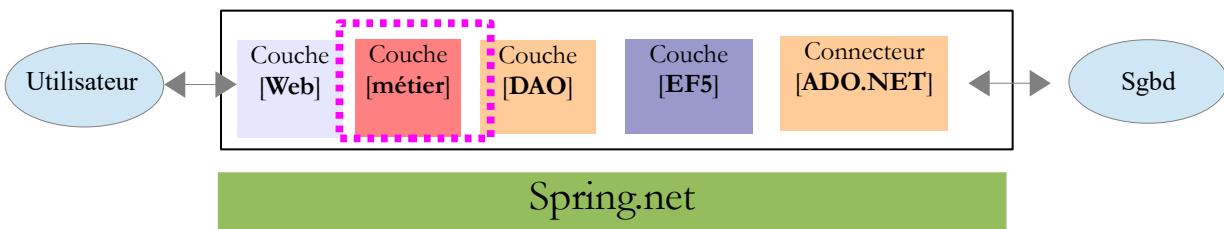
## 2.24.2 Le projet Visual Studio

**Travail** : ajouter à la solution [pam-td] un nouveau projet de type [console] appelé [pam-metier]. Faites-en le projet de démarrage de la solution.



### 2.24.3 Ajout des références nécessaires au projet

Situons le projet dans son ensemble :



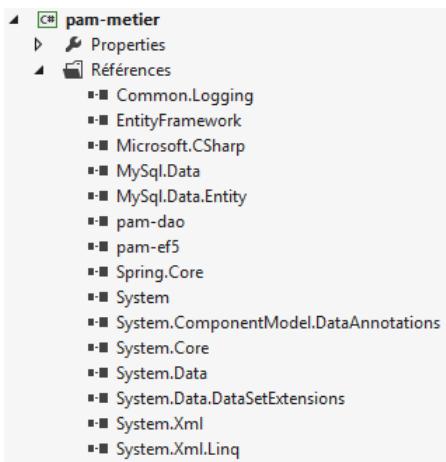
Le projet [pam-metier] a besoin d'un certain nombre de DLL :

- toutes celles référencées par les projets [pam-dao] et [pam-ef5] ;
- celles des projets [pam-dao] et [pam-ef5] eux-mêmes.

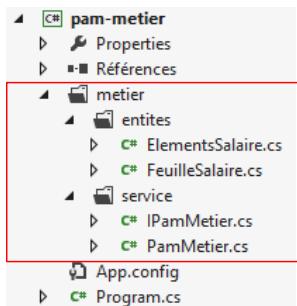
---

**Travail :** ajoutez ces différentes références au projet [pam-metier].

---



### 2.24.4 Implémentation de la couche [métier]



Ci-dessus, on retrouve quatre éléments déjà utilisés dans la couche [métier] simulée (voir paragraphe 2.7, page 227). Il peut y avoir des changements pour les espaces de noms importés par ces différentes classes. Gérez-les. La classe [PamMetier] implémente l'interface [IPamMetier] de la façon suivante :

```

1.  using Pam.Dao.Service;
2.  using Pam.EF5.Entites;
3.  using Pam.Metier.Entites;
4.  using System;
5.
6.  namespace Pam.Metier.Service
7.  {
8.
9.      public class PamMetier : IPamMetier
10.     {
11.
12.         // référence sur la couche [DAO] initialisée par Spring
13.         public IPamDao PamDao { get; set; }
14.
15.         // liste de toutes les identités des employés
16.         public Employe[] GetAllIdentitesEmployes()
17.         {
18.             ...
19.         }
20.
21.         // un employé particulier avec ses indemnités
22.         public Employe GetEmploye(string ss)
23.         {
24.             ...
25.         }
26.
27.         // les cotisations
28.         public Cotisations GetCotisations()
29.         {
30.             ...
31.         }
32.
33.         // calcul du salaire
34.         public FeuilleSalaire GetSalaire(string ss, double heuresTravaillées, int joursTravaillés)
35.         {
36.             // SS : n° SS de l'employé
37.             // HeuresTravaillées : le nombre d'heures travaillées
38.             // Jours Travaillos : nbre de jours travaillés
39.             ...
40.         }
41.     }

```

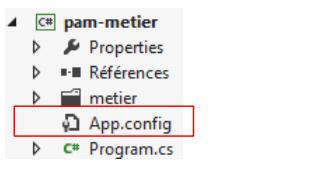
- ligne 13 : on a une référence sur la couche [DAO]. Elle sera initialisée par Spring lors de l'instanciation de la classe [PamMetier]. Donc lorsque les différentes méthodes s'exécutent, la ligne 13 a déjà été initialisée.

---

**Travail** : compléter le code de la classe [PamMetier]. Si dans [GetSalaire] on découvre que l'employé de n° ss n'existe pas, on lancera une [PamException]. Le mode de calcul du salaire est expliqué au paragraphe 2.5, page 225. On fera attention d'arrondir tous les calculs intermédiaires à deux chiffres après la virgule.

## 2.24.5 Configuration de la couche [métier]

Comme il a été fait au paragraphe 2.22.5, page 300, il nous faut configurer EF5 dans le fichier [app.config] du projet :



**Travail 1 :** configurez EF5 dans [app.config]. Il suffit de reprendre ce qui a été fait dans le fichier [app.config] de la couche [EF5].

Notre programme de test va utiliser [Spring.net] pour obtenir une référence sur la couche [métier].

**Travail 2 :** en vous aidant de ce que vous avez fait précédemment au paragraphe 2.23.5, page 308, modifiez le fichier de configuration [app.config] du projet [pam-metier] afin qu'il définisse un objet Spring appelé [pammetier] associé à la classe [PamMetier] que nous venons de construire. Le plus simple est de recopier le fichier [app.config] du projet [pam-dao] et d'ajouter ce qui manque.

Il y a une difficulté ici. Non seulement, il faut instancier la couche [métier] avec la classe [PamMetier] mais il faut également initialiser sa propriété [PamDao] :

```
// référence sur la couche [DAO] initialisée par Spring
public IPamDao PamDao { get; set; }
```

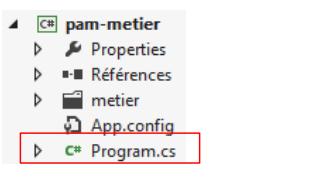
La configuration de Spring dans [app.config] est alors la suivante :

```
1. <spring>
2.   <context>
3.     <resource uri="config://spring/objects" />
4.   </context>
5.   <objects xmlns="http://www.springframework.net">
6.     <object id="pamdao" type="Pam.Dao.Service.PamDaoEF5, pam-dao"/>
7.     <object id="pammetier" type="Pam.Metier.Service.PamMetier, pam-metier">
8.       <property name="PamDao" ref="pamdao" />
9.     </object>
10.   </objects>
11. </spring>
```

- ligne 6 : définit l'objet [pamdao] associé à la classe [PamDaoEF5] ;
- ligne 7 : définit l'objet [pammetier] associé à la classe [PamMetier] ;
- ligne 8 : la balise [property] sert à initialiser une propriété publique de la classe [PamMetier]. L'attribut [name="PamDao"] correspond au nom de la propriété à initialiser dans la classe [PamMetier]. L'attribut [ref="pamdao"] indique que la propriété est initialisée avec une référence, celle de l'objet [pamdao] de la ligne 6, donc avec la référence de la couche [DAO]. C'est ce que nous voulions.

## 2.24.6 Test de la couche [métier]

Nous sommes prêts à tester notre couche [métier]. On le fait à l'aide du programme [Program.cs] déjà présent :



Nous allons tester les différentes fonctionnalités de l'interface de la couche [métier]. Le code de [Program.cs] sera le suivant :

```
1. using System;
2. using Pam.Dao.Entites;
3. using Pam.Metier.Service;
4. using Spring.Context.Support;
5. using Pam.EF5.Entites;
6.
7. namespace Pam.Metier.Tests
8. {
9.   public class Program
10.  {
```

```

11.    public static void Main()
12.    {
13.        try
14.        {
15.            // instanciation couche [métier]
16.            IPamMetier pamMetier = ContextRegistry.GetContext().GetObject("pammetier") as IPamMetier;
17.            // liste des identités des employés
18.            Console.WriteLine("Employés -----");
19.            foreach (Employe Employe in pamMetier.GetAllIdentitesEmployes())
20.            {
21.                Console.WriteLine(Employe);
22.            }
23.
24.            // calculs de feuilles de salaire
25.            Console.WriteLine("salaires -----");
26.            Console.WriteLine(pamMetier.GetSalaire("260124402111742", 30, 5));
27.            Console.WriteLine(pamMetier.GetSalaire("254104940426058", 150, 20));
28.            try
29.            {
30.                Console.WriteLine(pamMetier.GetSalaire("xx", 150, 20));
31.            }
32.            catch (PamException ex)
33.            {
34.                Console.WriteLine(string.Format("PamException : {0}", ex.Message));
35.            }
36.        }
37.        catch (Exception ex)
38.        {
39.            Console.WriteLine(string.Format("Exception : {0}, Exception interne : {1}", ex.Message, ex.InnerException == null ?
40.                "" : ex.InnerException.Message));
41.        }
42.        // pause
43.        Console.ReadLine();
44.    }
45. }

```

- ligne 16 : on obtient une référence sur la couche [métier] grâce à [Spring.net].

Les résultats de l'exécution de ce programme sont les suivants :

```

1. Employés -----
2. Employé[24,1,254104940426058,Jouveinal,Marie,5 rue des oiseaux,St Corentin,49203]
3. Employé[25,1,260124402111742,Laverti,Justine,La Brûlerie,St Marcel,49014]
4. salaires -----
5. [Employé[25,1,260124402111742,Laverti,Justine,La Brûlerie,St Marcel,49014],Cotisations[11,1,3,49,6,15,9,39,7,88],[64,85 :
17,45 : 10 : 15 : 72,4]]
6. [Employé[24,1,254104940426058,Jouveinal,Marie,5 rue des oiseaux,St Corentin,49203],Cotisations[11,1,3,49,6,15,9,39,7,88],
[362,25 : 97,48 : 42 : 62 : 368,77]]
7. PamException : L'employé de n° [xx] n'existe pas

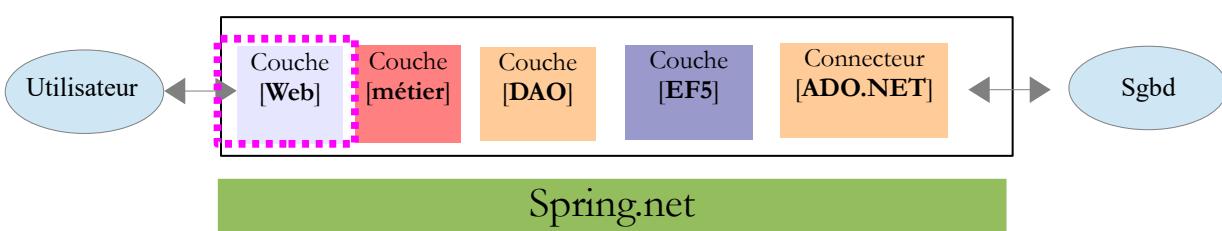
```

## 2.24.7 DLL de la couche [métier]

**Travail** : transformez le type du projet [pam-metier] en bibliothèque de classes puis régénérez le projet (refaire ce qui a été fait au paragraphe 2.22.7, page 304).

## 2.25 Étape 18 : mise en place de la couche [web]

Nous arrivons à la dernière couche de notre architecture, la couche [web] :

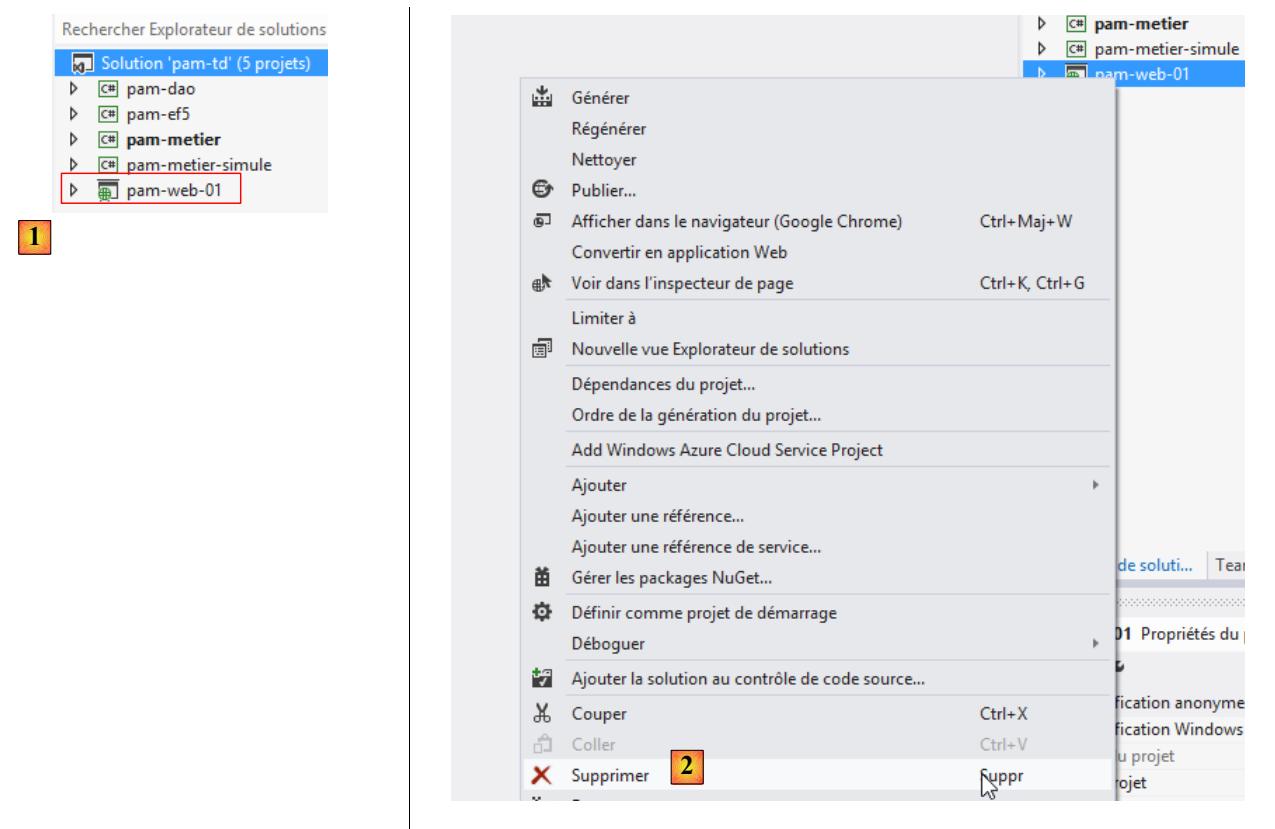


Nous allons réutiliser la couche [web] que nous avions développée avec l'aide d'une couche [métier] simulée.

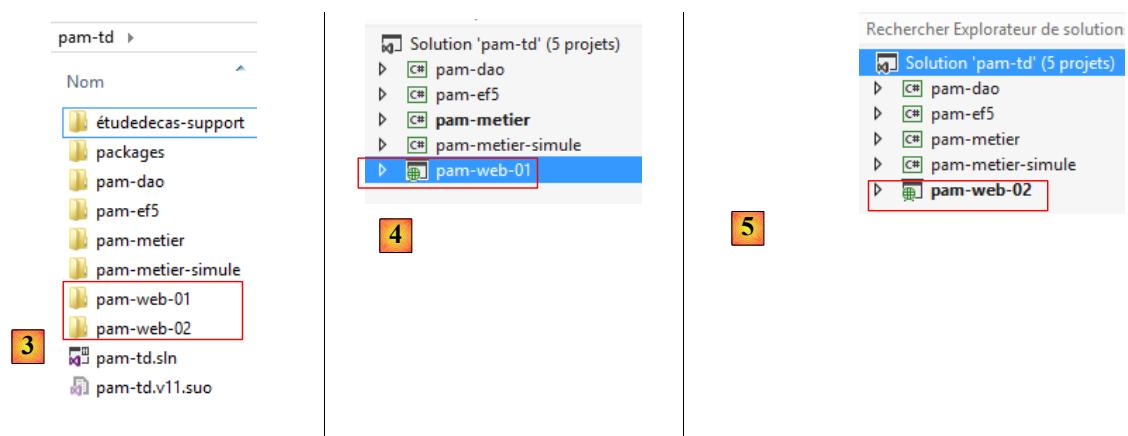
## 2.25.1 Le projet Visual Studio

Nous revenons à Visual Studio Express 2012 pour le web afin de brancher notre couche web aux couches [métier, DAO, EF5] que nous venons de développer. Il y a surtout de la configuration à faire et quelques changements d'espaces de noms.

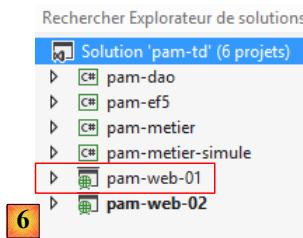
Avec Visual Studio Express 2012 pour le web, chargez la solution [pam-td] :



- en [1], la solution [pam-td] dans VS Studio pour le web. Le projet web [pam-web-01] redevient visible. On l'avait perdu dans VS Studio pour le bureau.
- la configuration du projet web [pam-web-01] va devoir être modifiée. Plutôt que de modifier un projet qui fonctionne, on va faire les modifications sur une copie de ce projet. Tout d'abord en [2], on supprime le projet de la solution (ça ne supprime rien dans le système de fichiers).



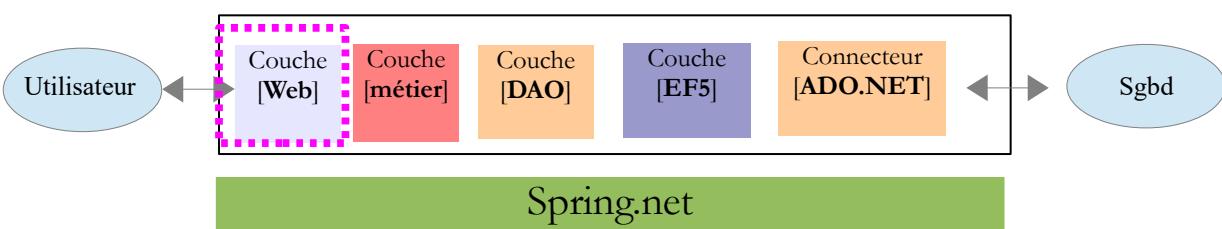
- en [3], avec l'explorateur windows, on duplique le dossier [pam-web-01] dans [pam-web-02] ;
- en [4], on charge le projet [pam-web-02] dans la solution [pam-td]. Il arrive avec le nom [pam-web-01] ;
- en [5], changez ce nom en [pam-web-02] et faites de ce projet le projet de démarrage ;



- en [6], chargez l'ancien projet [pam-web-01]. Vous avez désormais tous vos projets. Faites attention de travailler avec [pam-web-02].

## 2.25.2 Ajout des références nécessaires au projet

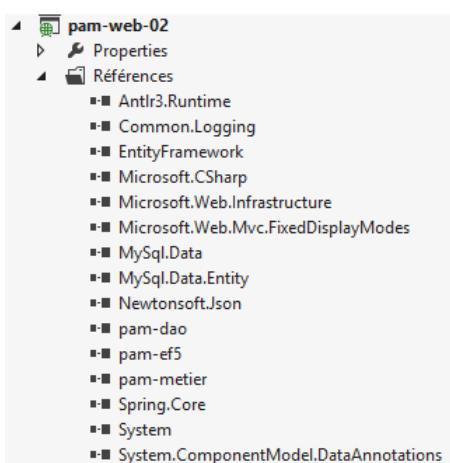
Situons le projet dans son ensemble :



Le projet [pam-web-02] a besoin d'un certain nombre de DLL :

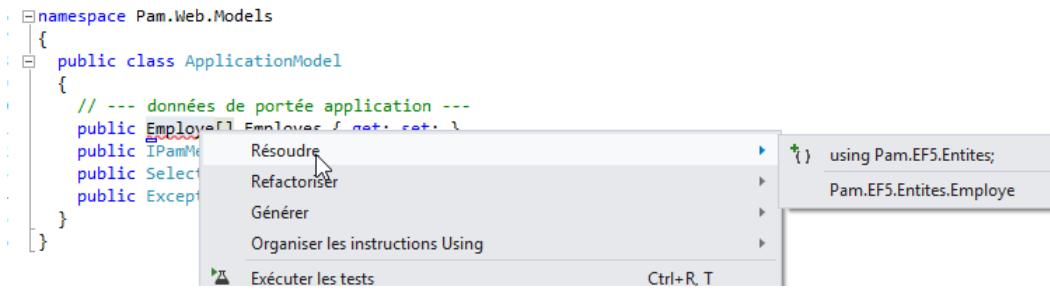
- toutes celles référencées par les projets [pam-metier], [pam-dao] et [pam-ef5] ;
- celles des projets [pam-metier], [pam-dao] et [pam-ef5] eux-mêmes.

**Travail** : ajoutez ces différentes références au projet [pam-web-02]. La référence sur le projet [pam-metier-simule] doit elle être enlevée. On change de couche [métier]. Certaines DLL sont déjà présentes dans les références. Supprimez-les puis faites vos ajouts.



## 2.25.3 Implémentation de la couche [web]

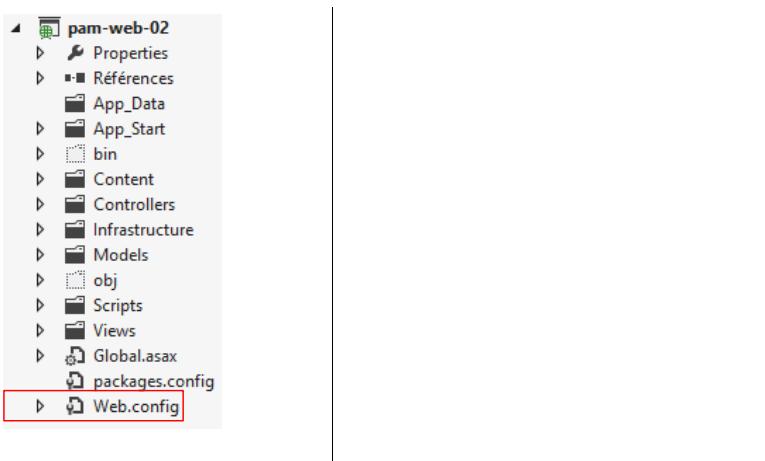
Générez le projet [pam-web-02]. Des erreurs vont apparaître telles que la suivante :



La classe [ApplicationModel] utilise le type [Employe]. Avec la couche [métier] simulée, ce type était défini dans l'espace de noms [Pam.Metier.Entites]. Il est désormais dans l'espace de noms [Pam.EF5.Entites]. Corrigez ces erreurs comme montré ci-dessus.

## 2.25.4 Configuration de la couche [web]

Comme il a été fait au paragraphe 2.24.5, page 311, il nous faut configurer EF5 dans le fichier [web.config] du projet :




---

**Travail 1 :** remplacez tout le contenu actuel de [web.config] par celui du fichier [app.config] du projet [pam-metier].

---

Le fichier [Global.asax] de notre application web utilise [Spring.net] pour récupérer une référence sur la couche [métier] :

```

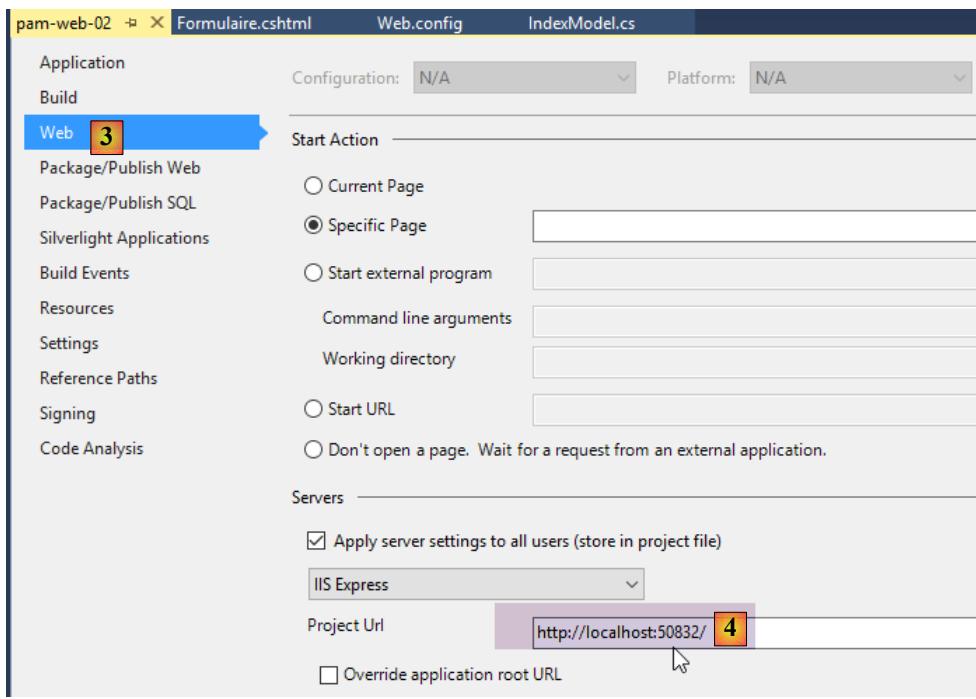
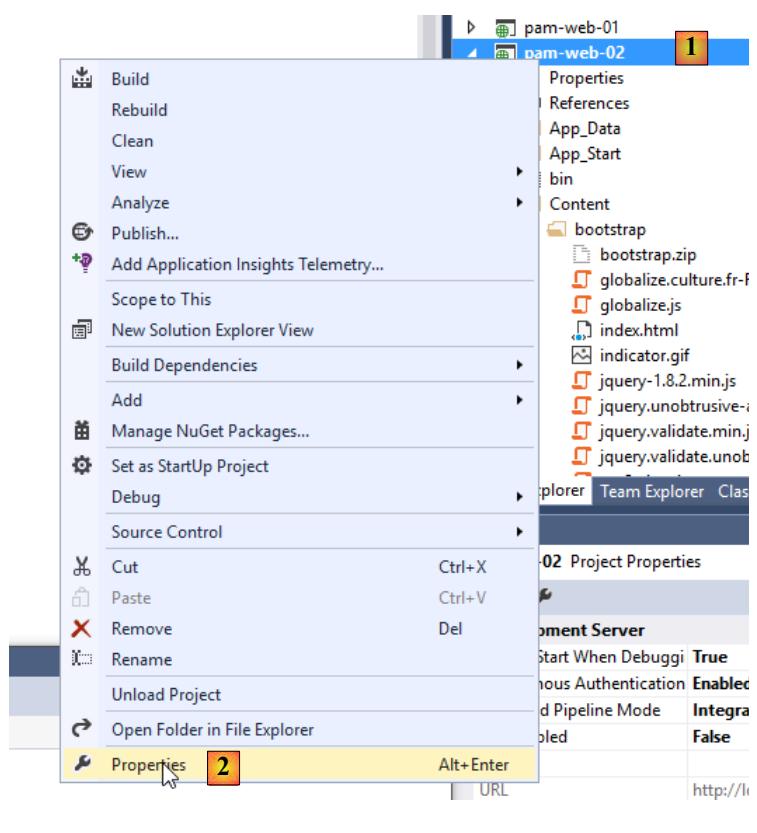
1.      try
2.      {
3.          // instantiation couche [métier]
4.          application.PamMetier = ContextRegistry.GetContext().GetObject("pammetier") as IPamMetier;
5.      }
6.      catch (Exception ex)
7.      {
8.          application.InitException = ex;
9.      }

```

Ligne 4, on demande une référence sur l'objet Spring nommé [pammetier]. C'est bien ce nom qui a été donné à la couche [métier] (vérifiez-le dans votre fichier [web.config]).

## 2.25.5 Test de la couche [web]

Nous sommes prêts à tester notre couche [web]. Nous allons d'abord changer son port de travail. Par défaut, [pam-web-02] a la configuration de [pam-web-01] et donc travaille sur le même port. L'expérience montre que cela pose problème : IIS continue alors à utiliser les codes du projet [pam-web-01]. Procédez de la façon suivante :



En [4], changez le n° du port, par exemple en changeant le chiffre des unités.

On exécute le projet [pam-web-02] par [Ctrl-F5]. On obtient alors la page d'accueil suivante :

Pam

localhost:65010

Applications Favoris Gestionnaire de favo...

## Simulateur de calcul de paie

[Faire la simulation](#)  
[Voir les simulations](#)  
[Terminer la session](#)

Employé	Heures travaillées	Jours travaillés
Marie Jouveinal	0,00	0
Marie Jouveinal		
Justine Laverdi		

1

En [1], on obtient les employés de la base de données [dbpam\_ef5]. On remarquera qu'on n'a plus l'employé [X X] que nous avions avec la couche [métier] simulée. Faisons une simulation :

Pam

localhost:65010

Applications Favoris Gestionnaire de favo...

## Simulateur de calcul de paie

[Effacer la simulation](#)  
[Voir les simulations](#)  
[Enregistrer la simulation](#)  
[Terminer la session](#)

Employé	Heures travaillées	Jours travaillés
Marie Jouveinal	150	20

**Informations Employé**

Nom	Prénom	Adresse
Jouveinal	Marie	5 rue des oiseaux
Ville	Code Postal	Indice
St Corentin	49203	2

**Informations Cotisations**

CGSRDS	CSGD	Retraite	Sécurité sociale
3,49 %	6,15 %	7,88 %	9,39 %

**Informations Indemnités**

Salaire horaire	Entretien / Jour	Repas / Jour	Congés payés
2,10 €	2,10 €	3,10 €	15 %

**Informations Salaire**

Salaire de base	Cotisations sociales	Indemnités d'entretien	Indemnités de repas
362,25 €	97,48 €	42,00 €	62,00 €
2			
Salaire net à payer : 368,77 €			

En [2], on obtient bien le salaire réel et non plus un salaire fictif. Maintenant arrêtons le SGBD MySQL5 et faisons une autre simulation :

Pam

localhost:65010

Simulateur de calcul de paie

Effacer la simulation  
Voir les simulations  
Enregistrer la simulation  
Terminer la session

Employé	Heures travaillées	Jours travaillés
Marie Jouveinal	150	20

**Les erreurs suivantes se sont produites** 3

- Erreur système lors de la recherche de l'employé [254104940426058]
- Échec du fournisseur sous-jacent sur Open.
- Unable to connect to any of the specified MySQL hosts.

En [3], on a obtenu une page d'erreurs lisible même si certains messages sont en anglais. Maintenant arrêtons de nouveau MySQL et réexécutons l'application dans VS par [Ctrl-F5] :

localhost:65010

Simulateur de calcul de paie

**Les erreurs suivantes se sont produites à l'initialisation de l'application :**

- Error creating context 'spring.root': Unable to connect to any of the specified MySQL hosts.
- Error creating object with name 'pamdao' defined in 'config [D:\data\istia-1314\aspnet\pam-td\pam-web-02\web.config#spring\objects] line 1' : Initialization of object failed : Cannot instantiate Type [Pam.Dao.Service.PamDaoEF5] using ctor [Void .ctor()]: 'Erreur système lors de la construction de la couche [DAO]'
- Cannot instantiate Type [Pam.Dao.Service.PamDaoEF5] using ctor [Void .ctor()]: 'Erreur système lors de la construction de la couche [DAO]'
- Erreur système lors de la construction de la couche [DAO]
- An error occurred while getting provider information from the database. This can be caused by Entity Framework using an incorrect connection string. Check the inner exceptions for details and ensure that the connection string is correct.
- Le fournisseur n'a pas retourné de chaîne ProviderManifestToken.
- Unable to connect to any of the specified MySQL hosts.

On obtient la vue [initFailed.cshtml] construite au paragraphe 2.20.4, page 287. Elle affiche les messages d'erreurs de la pile d'exceptions. Le lecteur est invité à faire d'autres tests.

## 2.26 Étape 19 : rendre accessible sur Internet une application ASP.NET

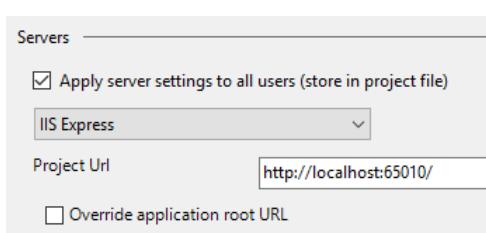
Lorsqu'on développe une application ASP.NET avec Visual Studio, la configuration utilisée par défaut fait que l'application développée n'est accessible qu'à l'adresse [localhost]. Toute autre adresse est refusée par le serveur embarqué de Visual Studio qui renvoie alors l'erreur [400 Bad Request].

On peut le voir de la façon suivante ;

- dans une fenêtre DOS, notez l'adresse IP de votre machine de développement :

L'adresse IP est ici indiquée ligne 14. Si vous avez une connexion wifi, l'adresse wifi du poste apparaîtra lignes 20 et suivantes.

- vérifiez les propriétés du projet [clic droit sur projet / propriétés / onglet web] :



L'application va s'exécuter sur le port [65010] de la machine [localhost].

- exécutez votre projet par [Ctrl-F5]



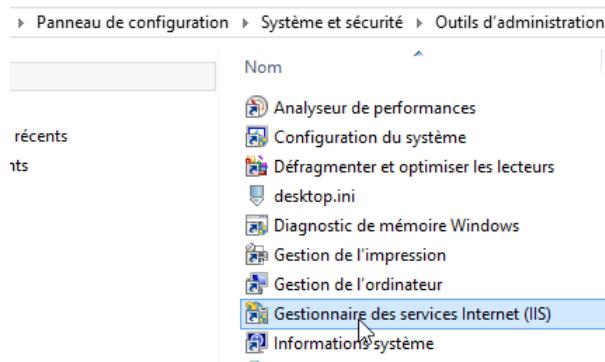
- remplacez [localhost] par l'adresse IP du poste :

## Bad Request - Invalid Hostname

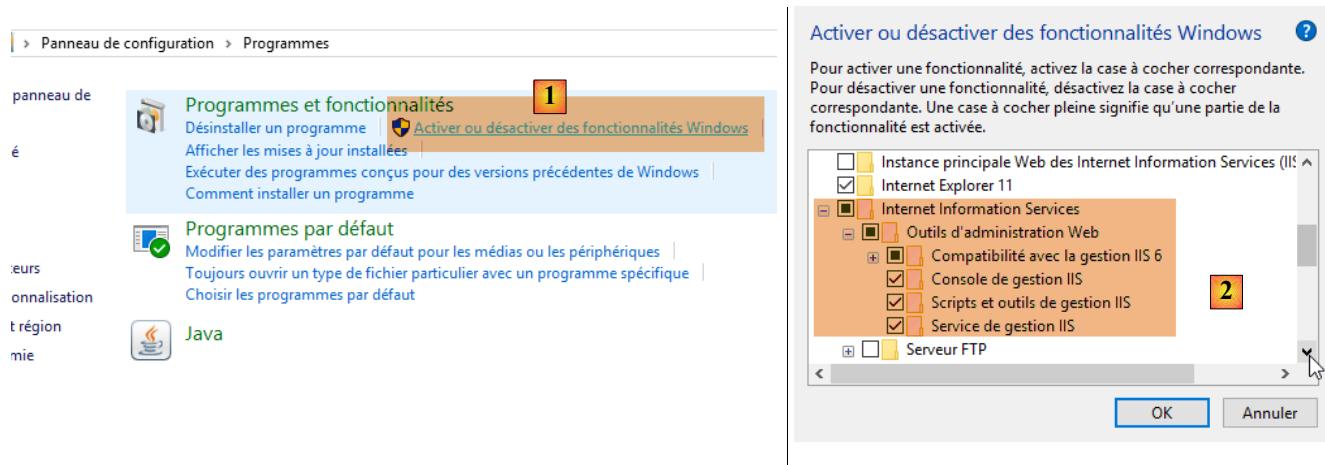
HTTP Error 400. The request hostname is invalid.

Le serveur a renvoyé une réponse [400 Bad Request]. Le serveur IIS Express utilisé par Visual Studio n'accepte que le nom [localhost].

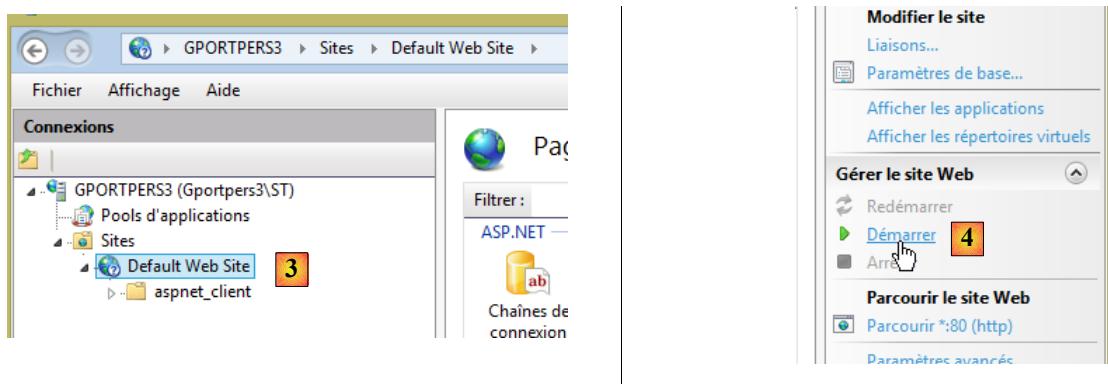
Pour rendre accessible l'application développée, à une URL du type [http://adresseIP/contexte/...], il faut utiliser un autre serveur que IIS Express, par exemple un serveur IIS (pas Express). Pour vérifier la présence de celui-ci (normalement dans les versions Pro de windows), il faut aller dans le panneau de configuration [Panneau de configuration\Système et sécurité\Outils d'administration] :



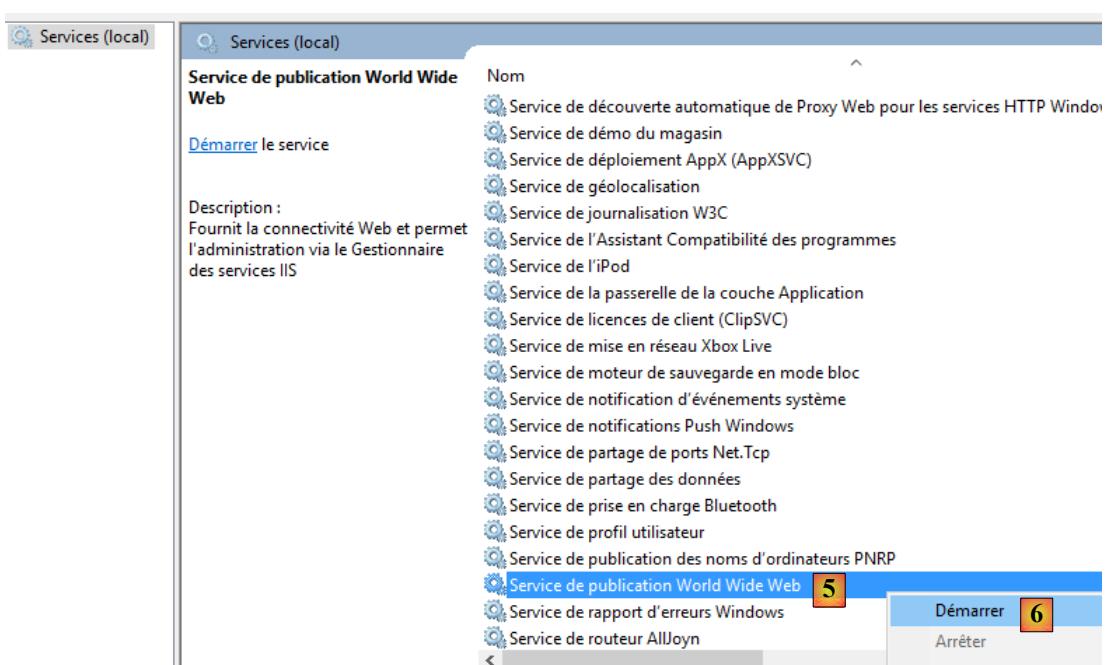
Cette option n'est pas toujours présente. Il faut alors aller dans [ Panneau de configuration \ Programmes] et installer les Outils d'administration web.



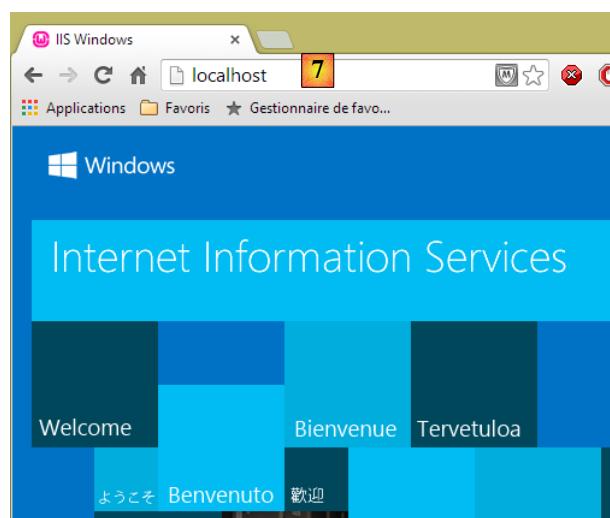
Une fois l'option [Gestionnaire des services internet (IIS)] présente, on l'active :



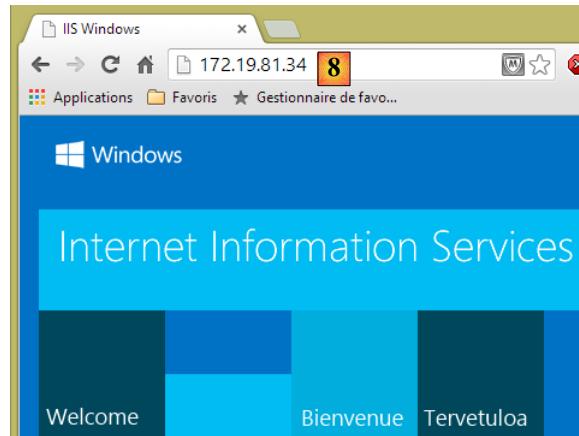
On démarre le site web par défaut. Pour cela, il faut que le service [Service de publication World Wide Web] soit auparavant lancé :



Ceci fait, demandez l'URL [<http://localhost>] avec un navigateur. Vérifiez auparavant qu'un autre serveur web n'occupe pas déjà le port 80. Si oui, arrêtez le.

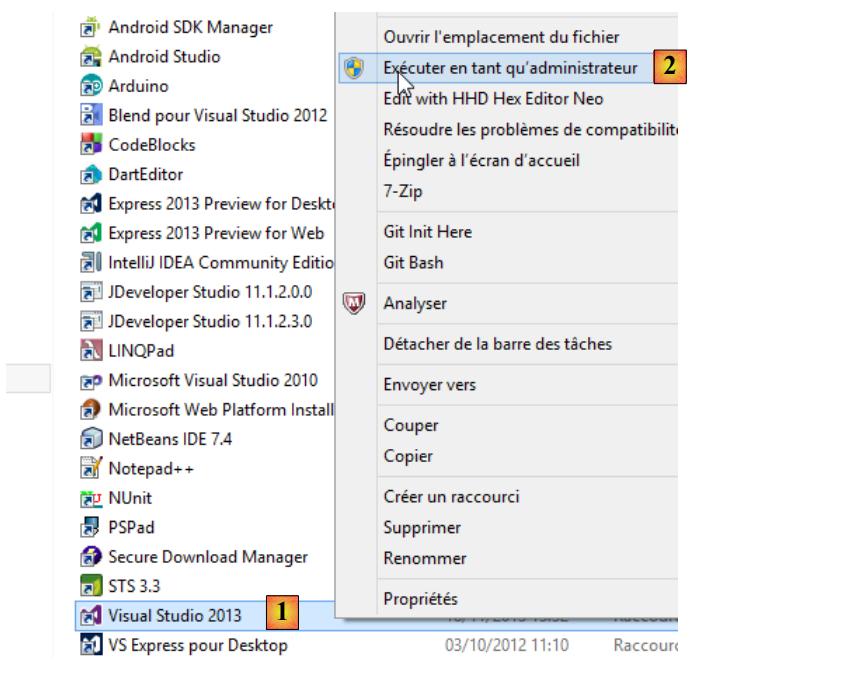


Le serveur IIS nous a répondu. Maintenant remplacez [localhost] par l'adresse IP de votre poste :

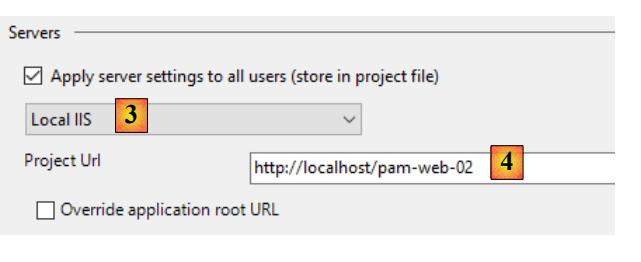


Ca marche. Revenons maintenant à Visual Studio :

- tout d'abord, il faut lancer Visual studio en mode [administrateur]



Ceci fait, il faut changer la configuration du projet web qu'on veut déployer [clic droit sur projet / propriétés / onglet web] :



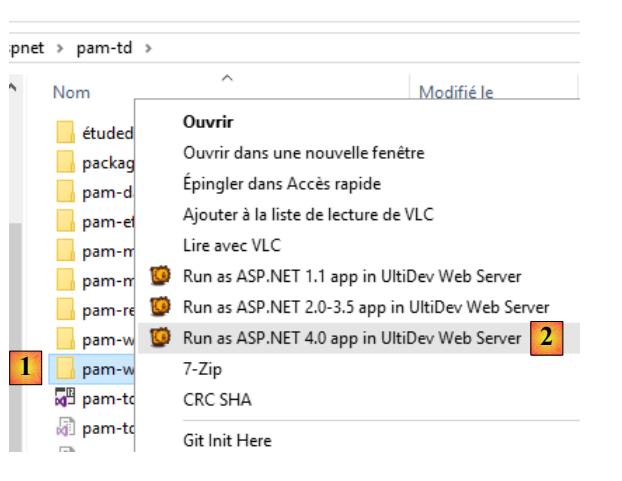
Il faut choisir le serveur IIS local comme serveur de déploiement. Visual studio fixe l'URL de l'application. On peut la changer. Exécutez le projet par [Ctrl-F5] :

Remplacez maintenant [localhost] par l'adresse IP de votre poste :

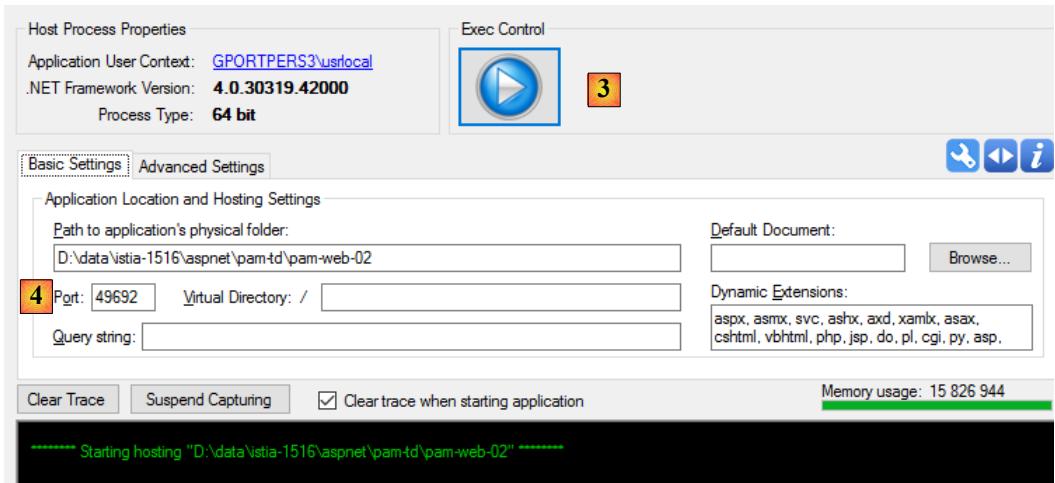
Si on ne dispose pas du serveur IIS, on pourra utiliser un serveur ASP.NET gratuit tel [Ultidev Web Server Pro] disponible à l'URL [<http://ultidev.com/Download/> ]. Une fois installé, il y a deux méthodes pour lancer une application web avec ce serveur :

### la manière rapide

Prenez un explorateur windows et sélectionnez le dossier de l'application ASP.NET à déployer :

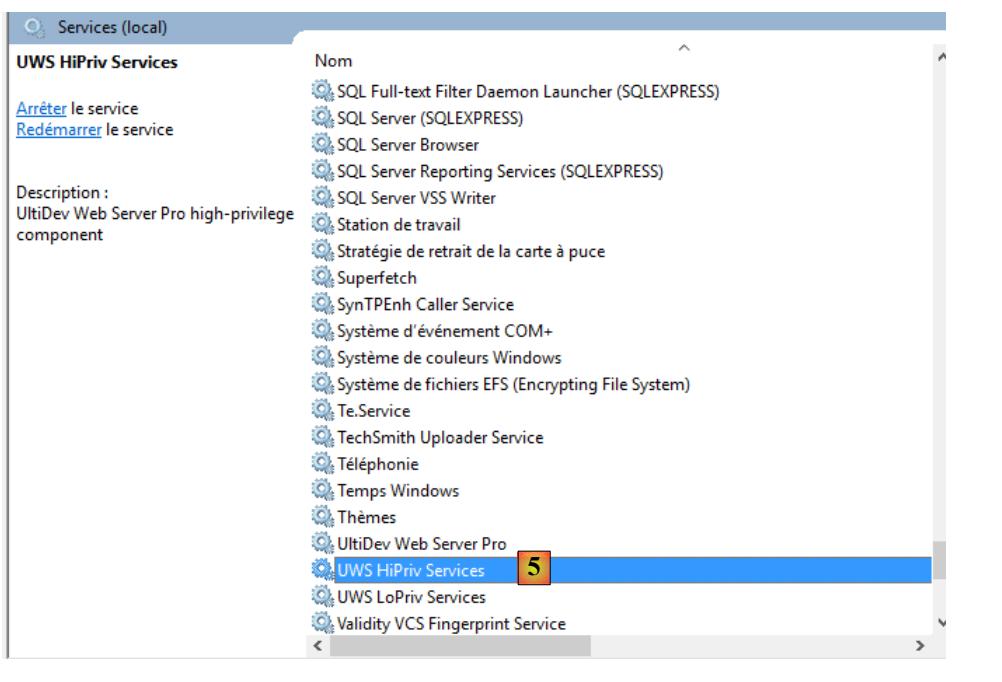


Le serveur web est alors lancé et l'application web affichée dans un navigateur :

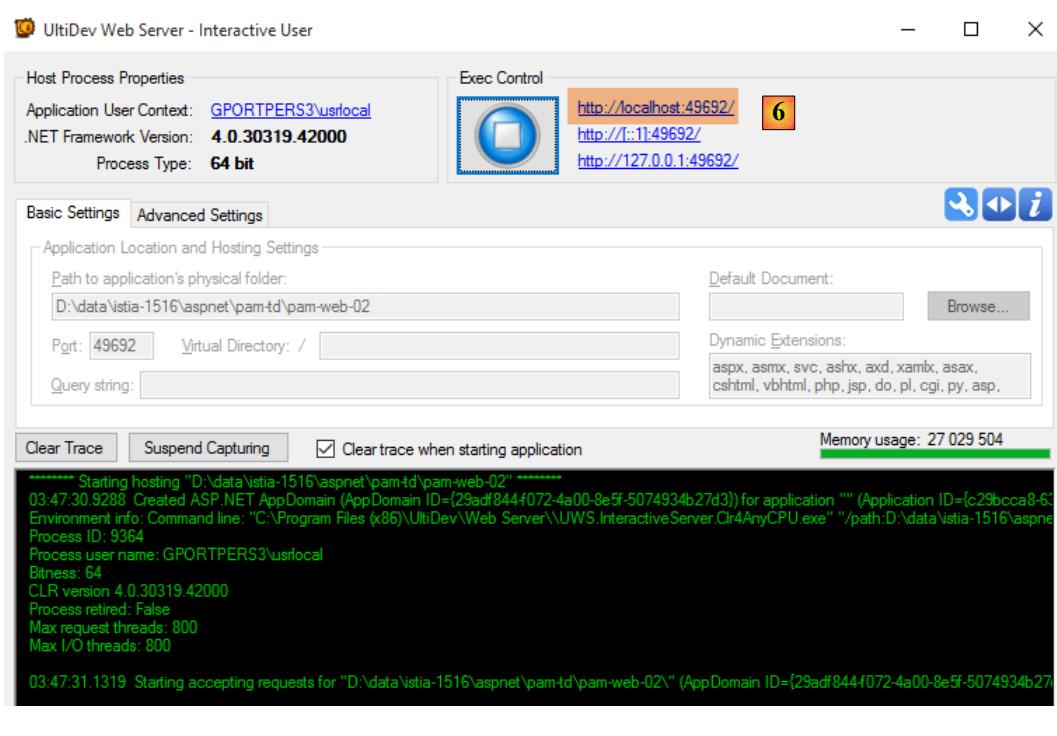


- en [3], on peut arrêter / lancer le serveur web ;
- en [4], on peut changer le port de service de l'application web ;

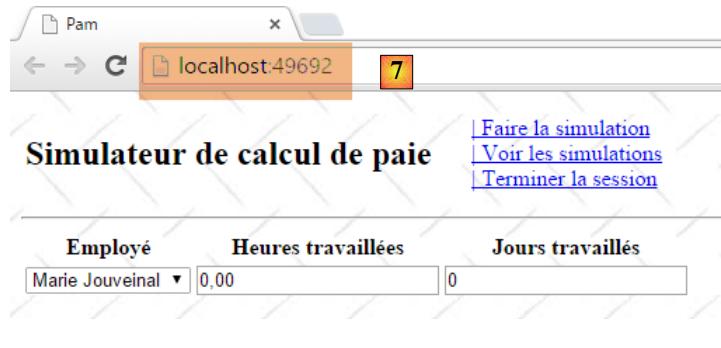
Avant de lancer le serveur, il faut que le service [UWS HiPriv Services] ci-dessous soit lancé :



Une fois le serveur lancé, l'interface se présente de la façon suivante :



Un clic sur le lien [6] affiche la 1ère page de l'application :



On peut alors mettre l'adresse IP de la machine à la place de [localhost] :



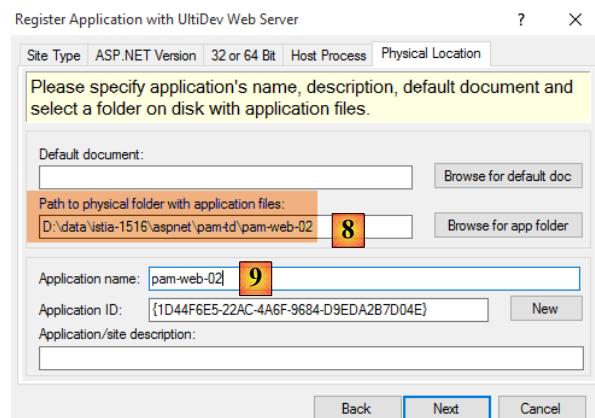
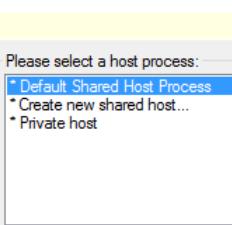
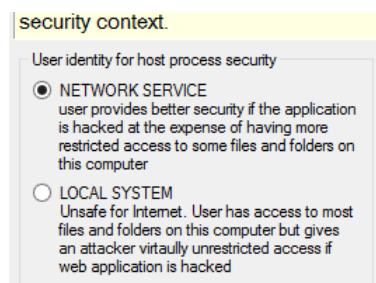
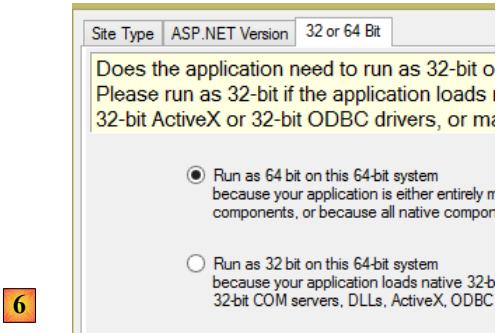
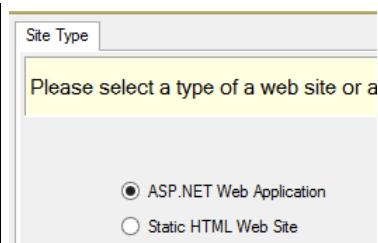
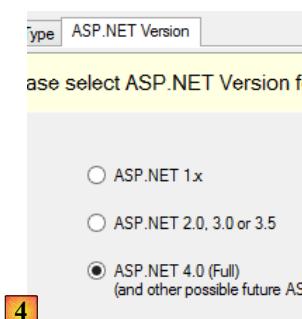
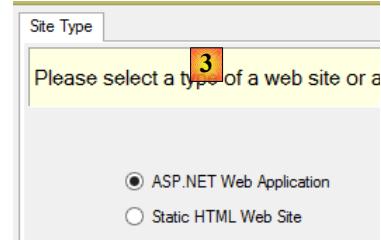
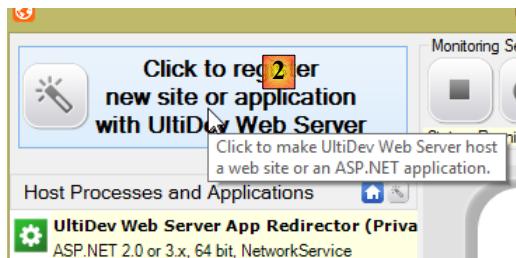
Donc là également, seul le nom [localhost] est accepté.

### la manière longue

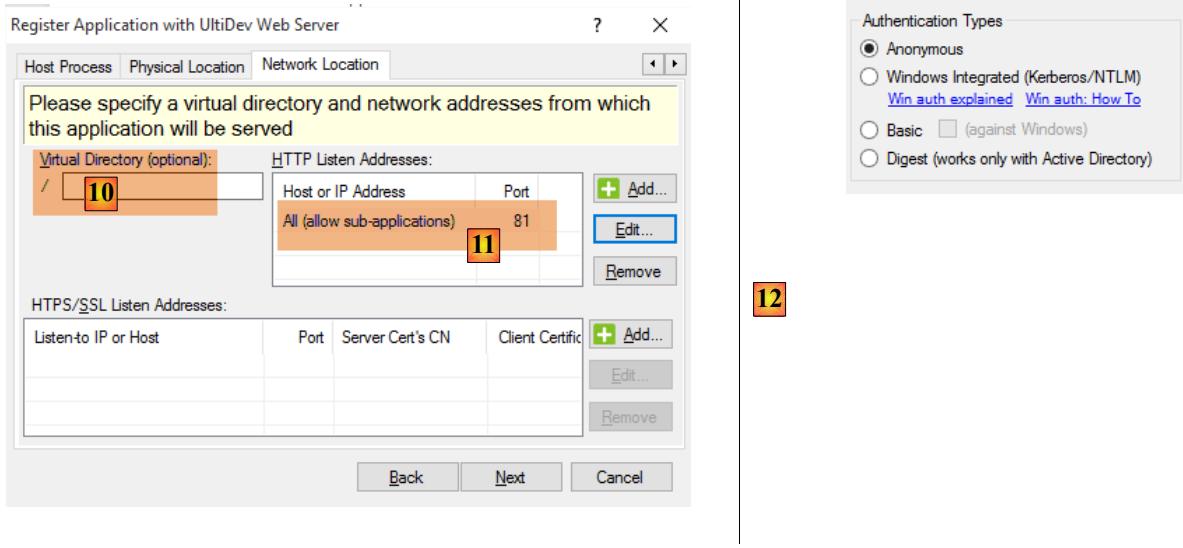
Lancez l'application Ultidev Web Explorer



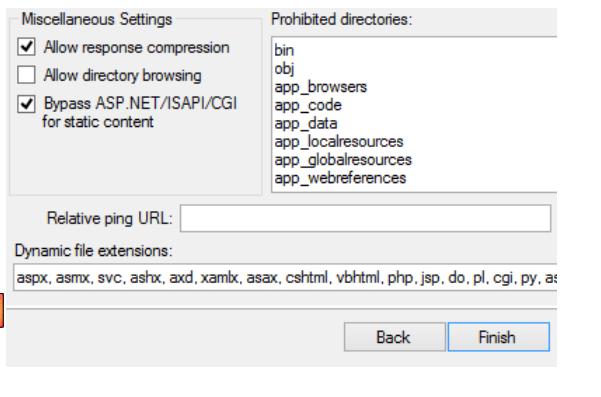
puis suivez les étapes suivantes :

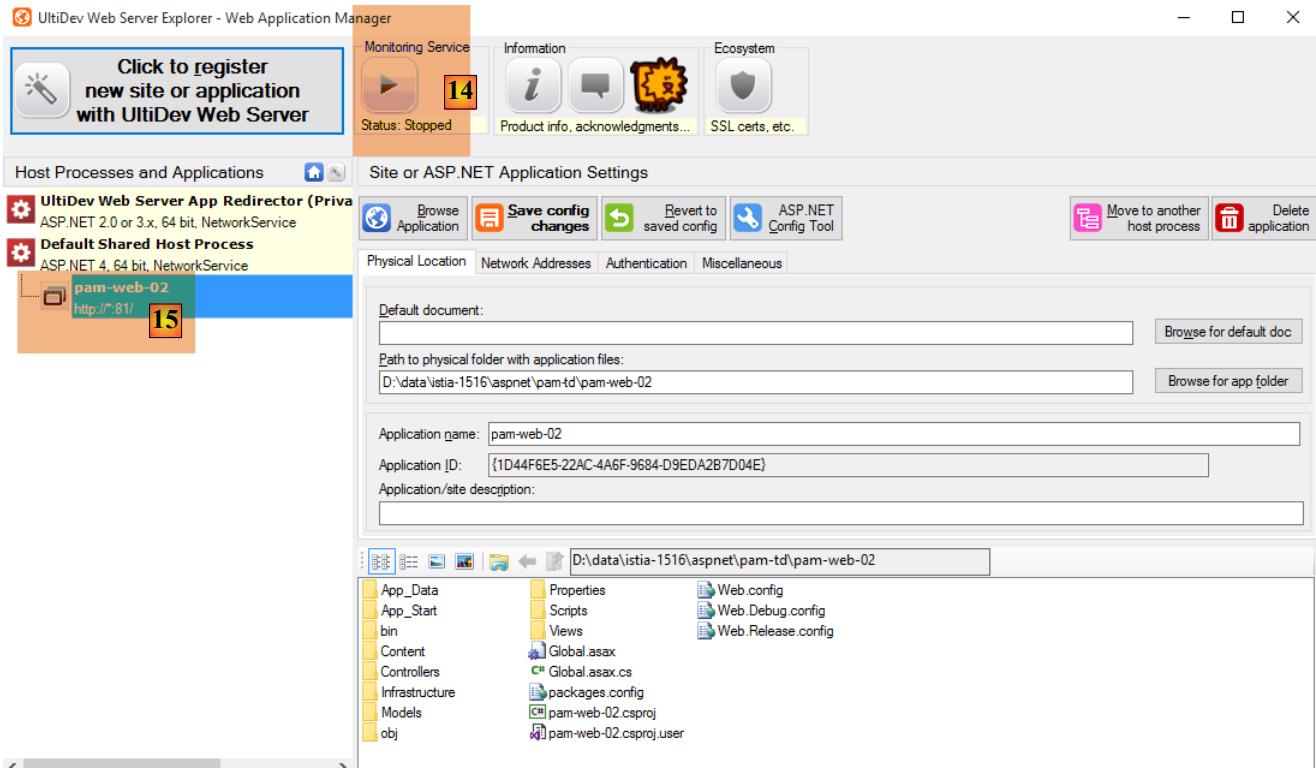


- en [8], désignez le dossier de l'application web à déployer ;

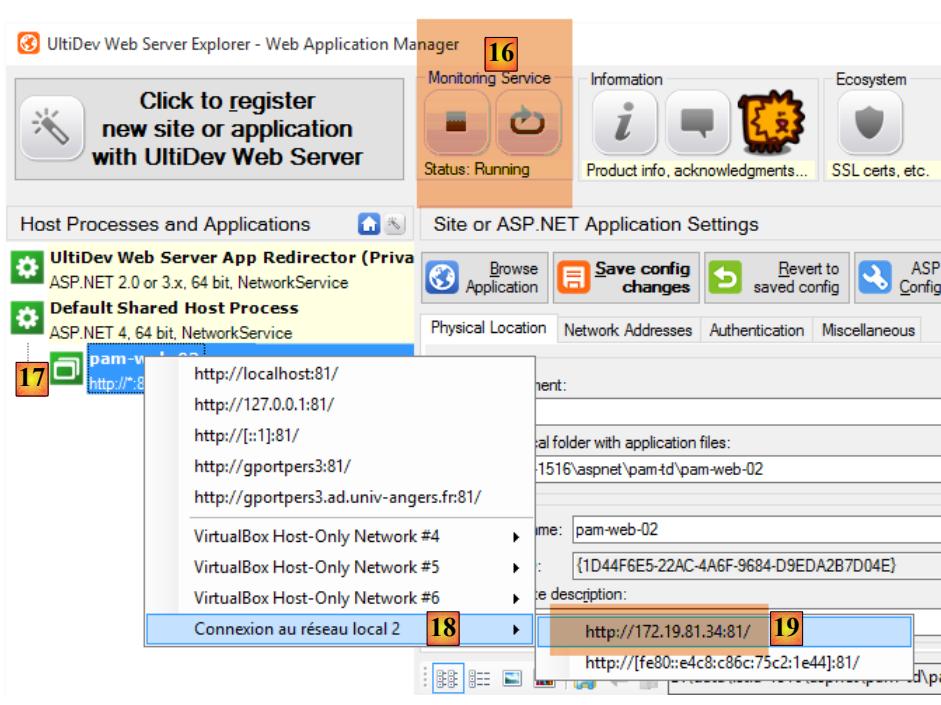


- à cause de [10-11], l'application web devra être demandée avec l'URL [<http://localhost:81/>];





- lancez le serveur web avec [14] ;

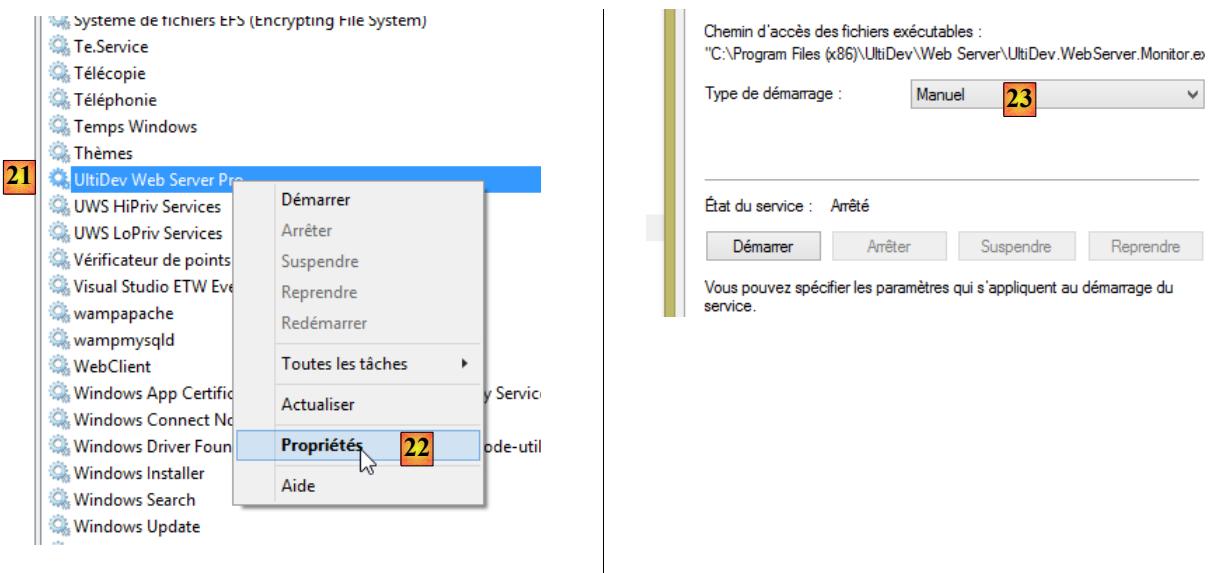


- demandez l'URL [19] ;

- en [20], on a obtenu la page désirée en utilisant l'adresse IP locale de la machine plutôt que le nom [localhost]. C'est ce que nous cherchions ;

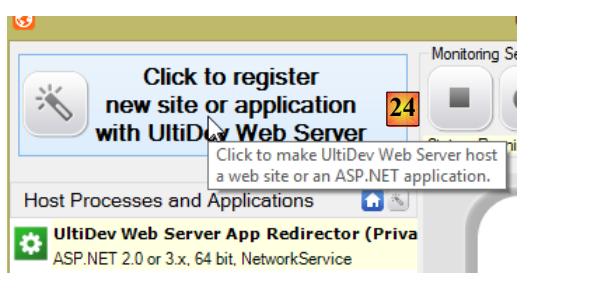
Le serveur Ultidev s'est installé sous la forme d'un service Windows qui se lance automatiquement. Vous pouvez inhiber le démarrage automatique du serveur Ultidev de la façon suivante :

- prendre l'option [Panneau de configuration\Système et sécurité\Outils d'administration] ;



- [1, 2] : sélectionnez les propriétés du service [Ultidev Web Server Pro] ;
- [3] : le mettre en démarrage manuel.

Pour lancer manuellement le serveur, utilisez par exemple l'application [Ultidev Web Explorer] :

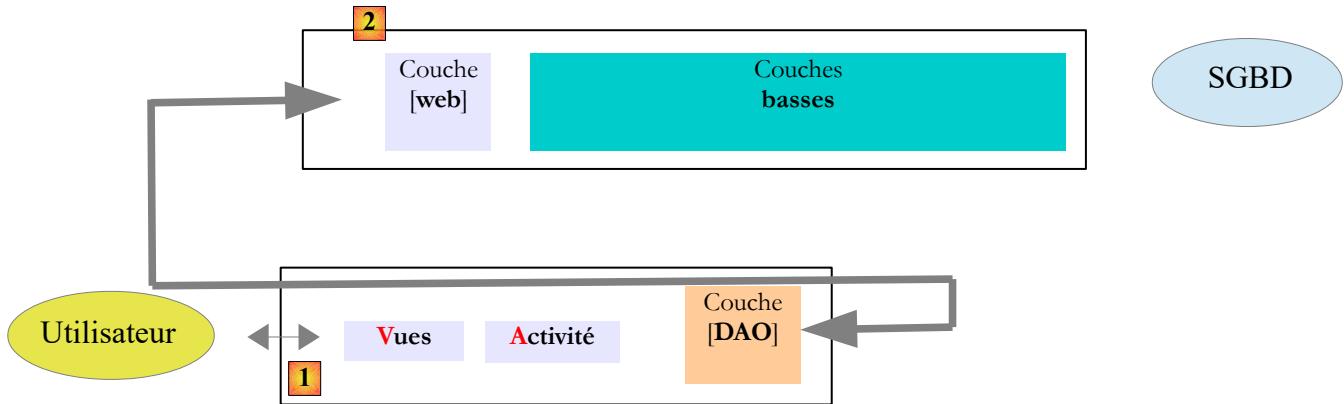


## 2.27 Étape 20 : génération d'une application native pour Android

Lorsqu'on a une application web de type APU (Application à Page Unique), il est possible de produire un exécutable pour mobile (Android, IoS, Windows 8, ...) avec l'outil [Phonegap] [<http://phonegap.com/>]. Il y a d'autres façons de faire, notamment avec le produit Open Source Apache Cordova [<https://cordova.apache.org/>]. L'outil présent en ligne sur le site de Phonegap [<http://build.phonegap.com/apps>] 'uploadé' le fichier zip du site à convertir. La page d'accueil doit s'appeler [index.html] et doit être une page statique, c-à-d ne pas être générée par un framework web (ASP.NET, JEE, PHP, ...). Nous allons commencer par construire celle-ci.

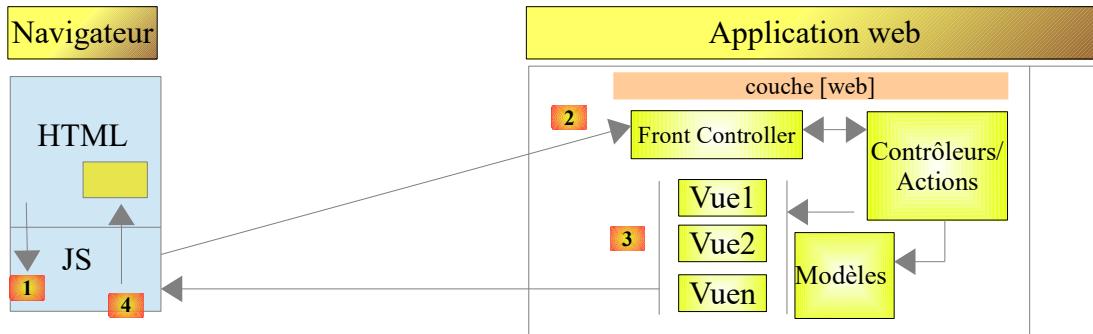
### 2.27.1 L'architecture de l'application

Il faut se rappeler ici que l'on veut créer une application Android. Une telle application a souvent l'architecture suivante :



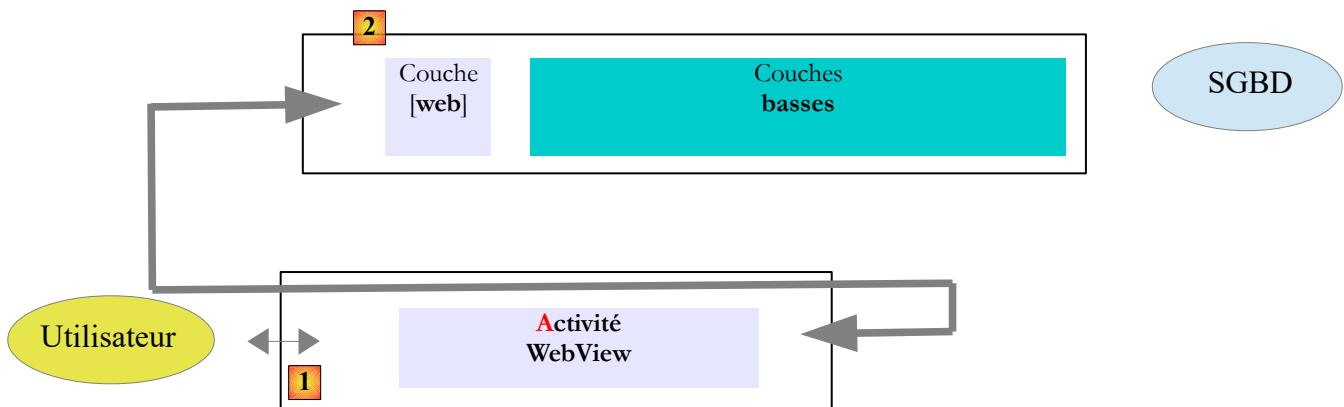
- en [1], l'utilisateur utilise une tablette Android qui communique avec un ou plusieurs services web [2] ;

Revenons au modèle APU :



- une page initiale est chargée dans le navigateur (le schéma ci-dessus ne dit pas d'où elle vient) ;
- les vues suivantes sont obtenues par des appels Ajax [1-4]. Aucune nouvelle page ne sera chargée par le navigateur ;

La vue initiale peut être fournie ou non par le même serveur que les autres vues obtenues par des appels Ajax. Si elle n'est pas fournie par le même serveur, le Javascript de la page initiale doit connaître l'URL du serveur web qui va délivrer les autres vues. Ce sera le cas dans l'application Android que nous allons construire :



- la page statique [index.html] va être encapsulée dans une application native Android [1] qui a les capacités d'un navigateur, capable donc d'exécuter le Javascript embarqué dans la page [index.html] ;
- cette page va obtenir les autres vues par des appels Ajax au serveur [2]. Pour cela, elle a besoin de connaître l'URL du serveur web ;

Nous allons refactoriser l'application [pam-web-02] pour qu'elle fonctionne sur ce mode. Ainsi la première page sera la suivante :



- en [1], l'URL de la page initiale de l'application. Elle nous sera fournie par le serveur Ultidev étudié au paragraphe 2.26, page 321;
- en [2], l'utilisateur devra entrer l'URL du simulateur de paie. On pourrait la rentrer en dur dans le code Javascript de la page initiale, mais cela compliquerait les tests : dès qu'on changerait le simulateur d'adresse IP (ou de port), il faudrait alors la changer dans le code Javascript ;
- en [3], le lien [Connexion] qui va aller chercher la vue suivante :

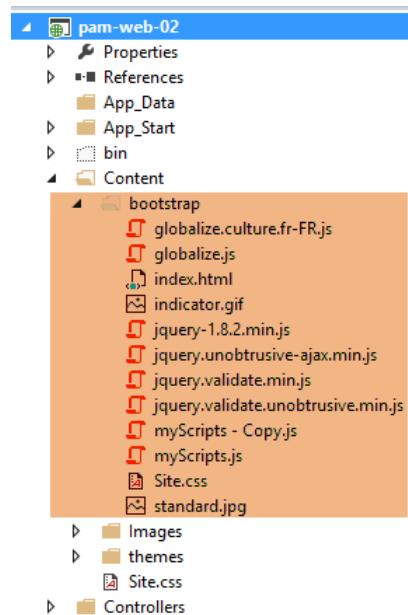


- on notera qu'en [4], l'URL du navigateur n'a pas changé. C'est toujours celle de la page initiale et restera ainsi pendant toute la durée de vie de l'application.

Une fois cette vue obtenue, tout fonctionne comme précédemment : les différentes vues sont obtenues par des appels Ajax. Nous allons voir que très peu de code doit être modifié.

## 2.27.2 Refactorisation du projet [pam-web-02]

A l'intérieur du dossier [Content] du projet [pam-web-02], nous construisons le dossier [bootstrap] (le nom n'importe pas) suivant :



Nous y avons inclus la page statique [index.html] et toutes les ressources dont elle a besoin (fichiers CSS et JS). La page [index.html] reprend le code de la page maître [\_Layout.cshtml] du projet Visual Studio en éliminant tout ce qui n'est pas statique. Cela donne le code suivant :

```
1. <!DOCTYPE html>
2. <html xmlns="http://www.w3.org/1999/xhtml">
3. <head>
4.     <title>Simulateur de paie</title>
5.     <meta charset="utf-8" />
6.     <meta name="viewport" content="width=device-width" />
7.     <link rel="stylesheet" href="Site.css" />
8.     <script type="text/javascript" src="jquery-1.8.2.min.js"></script>
9.     <script type="text/javascript" src="jquery.validate.min.js"></script>
10.    <script type="text/javascript" src="jquery.unobtrusive.min.js"></script>
11.    <script type="text/javascript" src="globalize.js"></script>
12.    <script type="text/javascript" src="globalize.culture.fr-FR.js"></script>
13.    <script type="text/javascript" src="jquery.unobtrusive-ajax.min.js"></script>
14.    <script type="text/javascript" src="myScripts.js"></script>
15. </head>
16. <body>
17.     <table>
18.         <tbody>
19.             <tr>
20.                 <td>
21.                     <h2>Simulateur de calcul de paie</h2>
22.                 </td>
23.                 <td style="width: 20px">
24.                     
25.                 </td>
26.                 <td>
27.                     <a id="lnkConnexion" href="javascript:connexion()">
28.                         | Connexion<br />
29.                     </a>
30.                     <a id="lnkFaireSimulation" href="javascript:faireSimulation()">
31.                         | Faire la simulation<br />
32.                     </a>
33.                     <a id="lnkEffacerSimulation" href="javascript:effacerSimulation()">
34.                         | Effacer la simulation<br />
35.                     </a>
36.                     <a id="lnkVoirSimulations" href="javascript:voirSimulations()">
37.                         | Voir les simulations<br />
38.                     </a>
39.                     <a id="lnkRetourFormulaire" href="javascript:retourFormulaire()">
40.                         | Retour au formulaire de simulation<br />
41.                     </a>
42.                     <a id="lnkEnregistrerSimulation" href="javascript:enregistrerSimulation()">
43.                         | Enregistrer la simulation<br />
44.                     </a>
45.                     <a id="lnkTerminerSession" href="javascript:terminerSession()">
```

```

46.           | Terminer la session<br />
47.           </a>
48.       </td>
49.   </tbody>
50. </table>
51. <hr />
52. <div id="content">
53.     <table>
54.         <tr>
55.             <td>URL du simulateur</td>
56.             <td><input type="text" id="urlServiceWeb" name="urlServiceWeb" size="80"></td>
57.         </tr>
58.     </table>
59.     <div id="erreur">
60.         <h3>Réponse du serveur :</h3>
61.         <div id="erreur1"></div>
62.         <div id="erreur2"></div>
63.     </div>
64. </div>
65. </body>
66. </html>

```

Nous avons ajouté les points suivants :

- lignes 27-29 : on a ajouté l'option de menu [Connexion] pour permettre la connexion au service de simulation ;
- lignes 55-56 : la saisie de l'URL du simulateur ;
- lignes 59-63 : une zone d'erreur si la connexion échoue ;

La refactorisation du code se fait uniquement dans le code [[myScripts.js](#)] de la ligne 14 ci-dessus. Rien d'autre ne change. Le code évolue de la façon suivante :

```

1. // au chargement du document
2. $(document).ready(function () {
3.     // on récupère les références des différents composants de la page
4.     loading = $("#loading");
5.     content = $("#content");
6.     erreur = $("#erreur");
7.     erreur1 = $("#erreur1");
8.     erreur2 = $("#erreur2");
9.     // les liens du menu
10.    lnkConnexion = $("#lnkConnexion");
11.    lnkFaireSimulation = $("#lnkFaireSimulation");
12.    lnkEffacerSimulation = $("#lnkEffacerSimulation");
13.    lnkEnregistrerSimulation = $("#lnkEnregistrerSimulation");
14.    lnkVoirSimulations = $("#lnkVoirSimulations");
15.    lnkTerminerSession = $("#lnkTerminerSession");
16.    lnkRetourFormulaire = $("#lnkRetourFormulaire");
17.    // on les met dans un tableau
18.    options = [lnkConnexion, lnkFaireSimulation, lnkEffacerSimulation, lnkEnregistrerSimulation, lnkVoirSimulations,
19.    lnkTerminerSession, lnkRetourFormulaire];
20.    // on cache certains éléments de la page
21.    loading.hide();
22.    erreur.hide();
23.    // on fixe le menu
24.    setMenu([lnkConnexion]);
25. });

```

- lignes 6-8 : les identifiants de la zone qui affiche les erreurs de connexion dans la page [index.html] ;
- ligne 10 : le nouveau lien pour la connexion au simulateur ;
- ligne 21 : la zone d'erreur est initialement cachée ;
- ligne 23 : on n'affiche que le lien de connexion ;

Dans la page [index.html], le lien de connexion est défini de la façon suivante :

```

1. <a id="lnkConnexion" href="javascript:connexion()">
2. | Connexion<br />
3. </a>

```

La fonction JS [connexion] (ligne 1) est la suivante :

```

1. var urlServiceWeb;
2. var erreur, erreur1, erreur2;
3.
4.
5. function connexion() {
6.     // on récupère l'urlServiceWeb du service web
7.     urlServiceWeb = $("#urlServiceWeb").val();
8.     // on récupère le formulaire de saisie
9.     $.ajax({
10.         url: urlServiceWeb + '/Pam/Formulaire',
11.         type: 'POST',

```

```

12.     dataType: 'html',
13.     beforeSend: function () {
14.         // signal d'attente allumé
15.         loading.show();
16.     },
17.     success: function (data) {
18.         // affichage résultats
19.         content.html(data);
20.         // menu
21.         setMenu([lnkFaireSimulation]);
22.     },
23.     error: function (jqXHR) {
24.         erreur2.html(jqXHR.responseText);
25.         erreur1.html(jqXHR.getAllResponseHeaders().replace(/\r\n/g, "<br/>").replace(/\r/g, "<br/>").replace(/\n/g,
26.             "<br/>"));
27.         erreur.show();
28.     },
29.     complete: function () {
30.         // signal d'attente éteint
31.         loading.hide();
32.     });
33. }

```

- ligne 7 : on récupère l'URL saisie par l'utilisateur. Elle est mise dans la variable globale de la ligne 1. Ainsi elle sera connue dans les autres fonctions du fichier ;
- ligne 10 : on fait un appel Ajax à l'URL [/Pam/Formulaire] du simulateur. Cette URL rend la vue partielle de la saisie des informations de la simulation (employés, heures travaillées, jours travaillés). Dans la version initiale de [pam-web-02], cette URL était suffisante. Elle était automatiquement préfixée par l'URL qui avait amené la page initiale. Maintenant, on fait l'hypothèse que la page initiale peut être fournie par un autre serveur que celui qui supporte le simulateur. Il faut alors préfixer l'URL [/Pam/Formulaire] par la variable [urlServiceWeb] de la ligne 1, qui est l'URL du simulateur (par exemple, <http://172.19.81.34/pam-web-02>). **Cela devra être fait pour tous les appels Ajax du fichier** ;
- lignes 17-22 : en cas de succès de la connexion, la vue partielle [Formulaire.cshtml] est affichée et on affiche un menu avec le seul lien [Faire la simulation] (ligne 21) ;
- lignes 23-27 : en cas d'échec de la connexion :
  - en ligne 24, on affiche la réponse HTML envoyée par le serveur web (s'il y en a une) ;
  - en ligne 25, on affiche les entêtes HTTP envoyés par le serveur web (s'il a répondu) ;

C'est tout. En cas de succès, on obtient la page suivante :



On est alors dans la situation précédente où désormais les vues sont obtenues par des appels Ajax. Ainsi ci-dessus, le clic sur le lien [Faire la simulation] va être exécuté par le code suivant du fichier [myScripts.js] :

```

1. function faireSimulation() {
2.     // on récupère des références
3.     var simulation = $("#simulation");
4.     var formulaire = $("#formulaire");
5.     // formulaire valide ?
6.     var formValid = formulaire.validate().form();
7.     if (!formValid) return;
8.     // on fait un appel Ajax à la main
9.     $.ajax({
10.         url: urlServiceWeb + '/Pam/FaireSimulation',
11.         type: 'POST',
12.         data: formulaire.serialize(),
13.         dataType: 'html',
14.         ...
15.     });
16.     // menu
17.     setMenu([lnkEffacerSimulation, lnkEnregistrerSimulation, lnkTerminerSession, lnkVoirSimulations]);

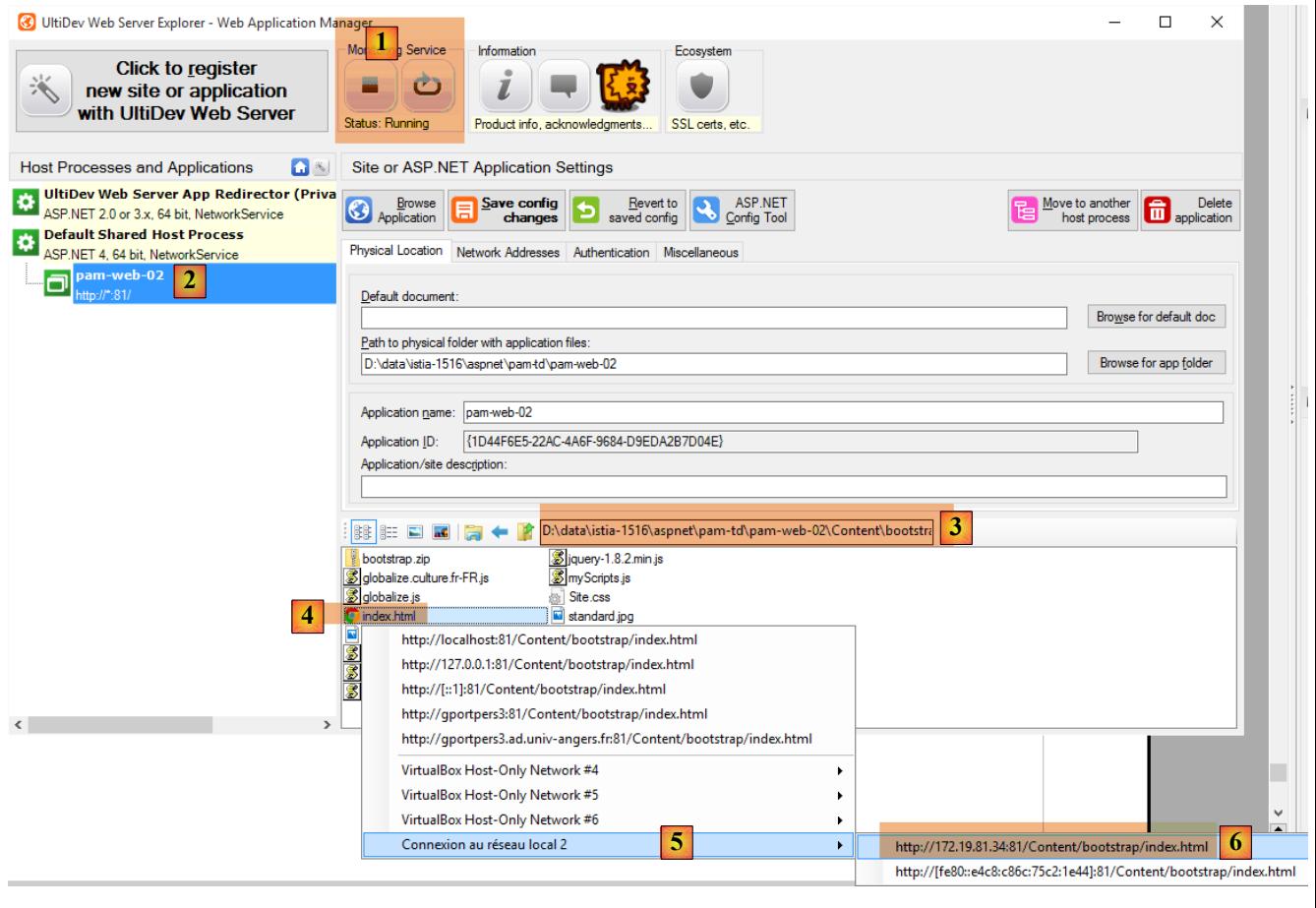
```

18. }

- une unique modification a été apportée, celle de la ligne 10 où la précédente URL est désormais préfixée par celle du simulateur ;

### 2.27.3 Test du projet refactorisé

Au paragraphe 2.26, page 321, nous avons montré comment installer l'application [pam-web-02] sur le serveur Ultidev. Nous allons partir de là :



- en [6], nous demandons l'affichage de la page [bootstrap/index.html]. On obtient la vue suivante :



Tapons une URL erronée :

The screenshot shows a web browser window titled "Simulateur de paie". The address bar displays the URL "172.19.81.34:81/Content/bootstrap/index.html" and has a yellow box labeled [8] next to it. The main content area is titled "Simulateur de calcul de paie" with a "Connexion" link. A red box labeled [9] highlights the "URL du simulateur" input field, which contains "http://172.19.81.34:81". Below it, a section titled "Réponse du serveur :" lists server headers: Date, Cache-Control, Server, X-AspNet-Version, Content-Encoding, Content-Length, and Content-Type. A yellow box labeled [10] highlights the "Content-Type: text/html; charset=utf-8" header. The next section, "Erreur du serveur dans l'application '/'.", is in red. A red box labeled [11] highlights the error message: "Le verbe HTTP POST utilisé pour accéder au chemin d'accès '/Content/bootstrap/x/Pam/Formulaire' n'est pas autorisé." Below this, a "Description" section states: "Une exception non gérée s'est produite au moment de l'exécution de la requête Web actuelle. Contrôlez la trace de la pile pour plus d'informations sur l'erreur et son origine dans le code." A "Détails de l'exception" section shows: "System.Web.HttpException: Le verbe HTTP POST utilisé pour accéder au chemin d'accès '/Content/bootstrap/x/Pam/Formulaire' n'est pas autorisé.". A "Erreur source:" section contains a yellow box with the text: "Une exception non gérée s'est produite lors de l'exécution de la requête Web actuelle. Les informations relatives à l'origine et l'emplacement de l'exception peuvent être identifiées en utilisant la trace de la pile d'exception ci-dessous." A "Trace de la pile:" section shows a stack trace in a yellow box: "[HttpException (0x80004005): Le verbe HTTP POST utilisé pour accéder au chemin d'accès '/Content/bootstrap/x/Pam/Formulaire' n'est pas autorisé.] System.Web.DefaultHttpHandler.BeginProcessRequest(HttpContext context, AsyncCallback callback, Object state) +8 System.Web.CallHandlerExecutionStep.System.Web.HttpApplication.IExecutionStep.Execute() +1115 System.Web.HttpApplication.ExecuteStep(IExecutionStep step, Boolean& completedSynchronously) +146". At the bottom, an "Informations sur la version" section states: "Version Microsoft .NET Framework :4.0.30319; Version ASP.NET :4.6.114.0".

- en [10], les entêtes HTTP de la réponse du serveur ;
- en [11], le document HTML de la réponse du serveur ;

Si on tape la bonne URL :

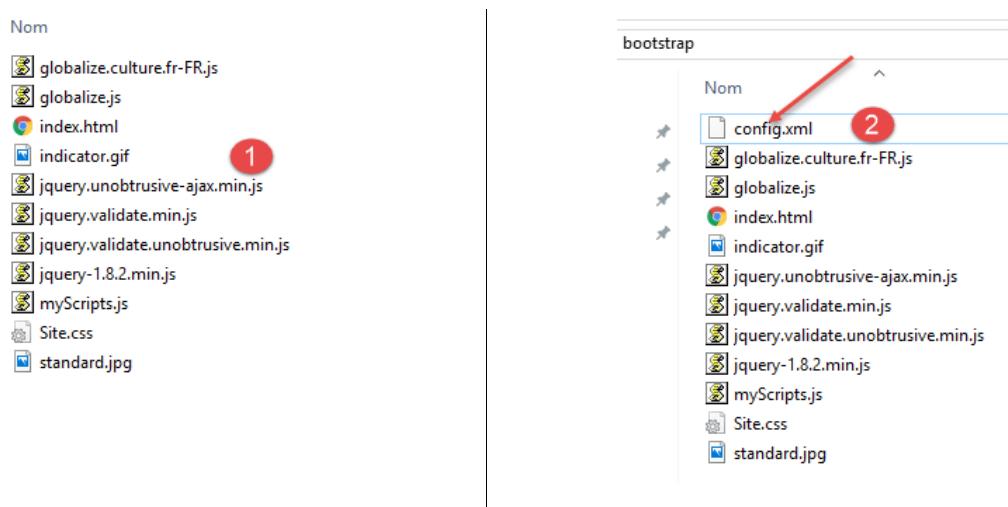
The screenshot shows a web browser window titled "Simulateur de paie". The address bar displays the URL "172.19.81.34:81/Content/bootstrap/index.html". The main content area is titled "Simulateur de calcul de paie" with a "Connexion" link. A red box labeled [12] highlights the "URL du simulateur" input field, which now contains "http://172.19.81.34:81".

on obtient la réponse suivante :



## 2.27.4 Crédation du binaire Android

Nous allons créer le binaire Android à partir du site statique que nous venons de créer et tester[1] :



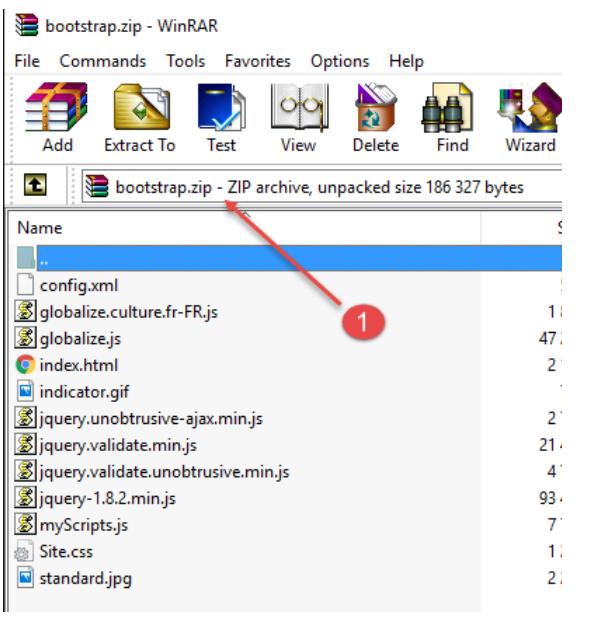
Nous ajoutons en [2], un fichier [config.xml] qui va servir à configurer le plugin [Phonegap] qui va générer le binaire Android. Son code est le suivant :

```

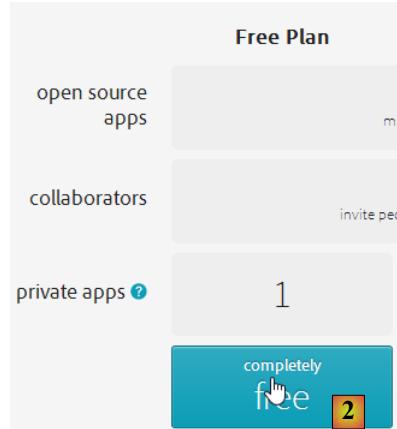
1.  <?xml version='1.0' encoding='utf-8'?>
2.  <widget id="android.exemples.pam" version="0.0.1" xmlns="http://www.w3.org/ns/widgets"
   xmlns:cdv="http://cordova.apache.org/ns/1.0">
3.    <name>Pam</name>
4.    <description>
5.      Istia - Université d'Angers
6.    </description>
7.    <author email="serge.tahe@univ-angers.fr">
8.      Serge Tahé
9.    </author>
10.   <content src="index.html" />
11.   <access origin="*" />
12.   <allow-navigation href="*" />
13.   <allow-intent href="*" />
14.   <plugin name="cordova-plugin-whitelist" />
15. </widget>
```

- lignes 7-9 : mettez ici vos coordonnées ;
- lignes 11-13 : ces lignes permettent au Javascript embarqué dans l'application web qui va s'exécuter au sein du périphérique Android de requérir des URL extérieures à ce périphérique ;

Nous zippsons le contenu du dossier [Content/bootstrap] :



Ensuite nous allons sur le site de Phonegap [<http://build.phonegap.com/apps>] :



- avant [1], vous aurez peut-être à créer un compte ;
- en [1], on démarre ;
- en [2], on choisit un plan gratuit n'autorisant qu'une application Phonegap ;
- en [3], on télécharge l'application zippée [4] ;



App ID: 2436261 Version: 0.01 Owned by: sergetahe@gmail.com

PhoneGap (iOS / Android / Windows)  
cli-6.3.0 (4.2.0 / 5.2.1 / 4.4.1)

Source: .zip package  
Last built: 1 minute

+ new app

delete Ready to build

- en [5], le nom à l'application ;
- cliquez sur le lien [6] pour construire les binaires des OS IoS, Android et Windows. Cela peut prendre quelques secondes ;

App ID: 2436261 Version: 0.01 Owned by: sergetahe@gmail.com

PhoneGap (iOS / Android / Windows)  
cli-6.3.0 (4.2.0 / 5.2.1 / 4.4.1)

Source: .zip package  
Last built (2): 2 minutes

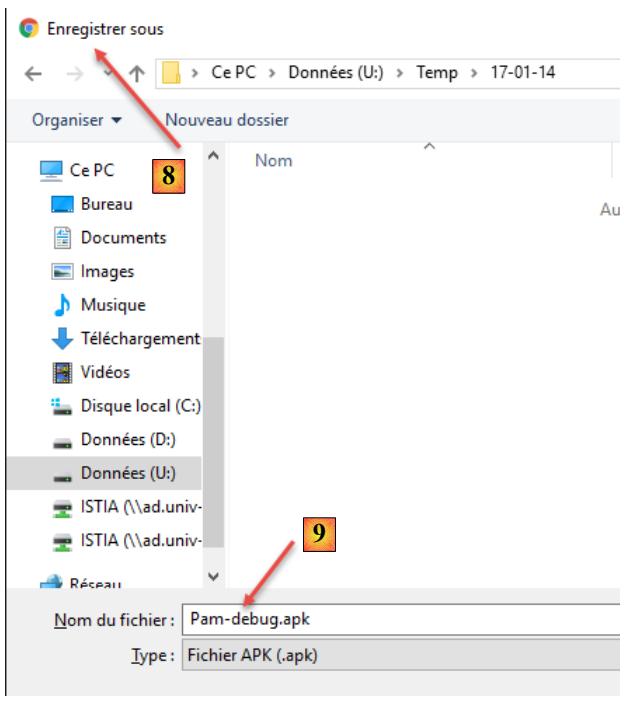
+ new app

iOS Android Windows

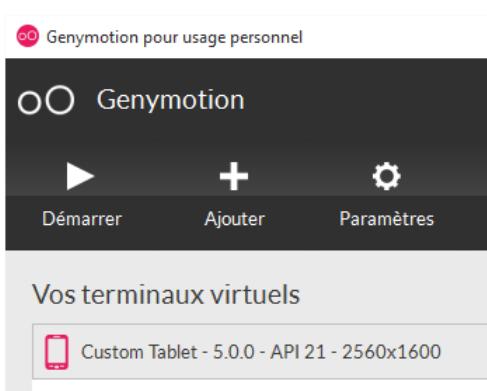
private

Update code Rebuild all

- en [7-9], téléchargez le binaire Android ;



Lancez un émulateur [GenyMotion] pour une tablette Android (voir paragraphe 2.28.1, page 343) :



Ci-dessus, on lance un émulateur de tablette avec l'API 21 d'Android. Une fois l'émulateur lancé,

- déverrouillez-le en tirant le verrou (s'il est présent) sur le côté puis en le lâchant ;
- avec la souris, tirez le fichier [Pam\_web\_02-debug.apk] que vous avez téléchargé et déposez-le sur l'émulateur. Il va être alors installé et exécuté ;



Mettez en [1], l'URL du simulateur comme il a été décrit au paragraphe 2.27.3, page 337. Ceci fait, connectez-vous au simulateur avec le lien [2] :

Employé	Heures travaillées	Jours travaillés
Marie Jouveinal ▾	0,00	0

[Faire la simulation](#)

Testez l'application sur l'émulateur. Elle doit fonctionner.

## 2.28 Annexes

### 2.28.1 Installation du gestionnaire d'émulateurs Genymotion

Les émulateurs fournis avec le SDK d'Android sont lents ce qui décourage de les utiliser. L'entreprise [[Genymotion](#)] offre un émulateur performant. Celui-ci est disponible à l'URL [<https://cloud.genymotion.com/page/launchpad/download/>] (octobre 2014).

Vous aurez à vous enregistrer pour obtenir une version à usage personnel. Téléchargez le produit [Genymotion] avec la machine virtuelle VirtualBox ainsi que le plugin [Genymotion] pour l'IDE [IntelliJIDEA] :

#### Download ready-to-run Genymotion installer for Windows

*This version includes Oracle VirtualBox 4.2.12 dependency, so that you don't need to download and install VirtualBox manually*

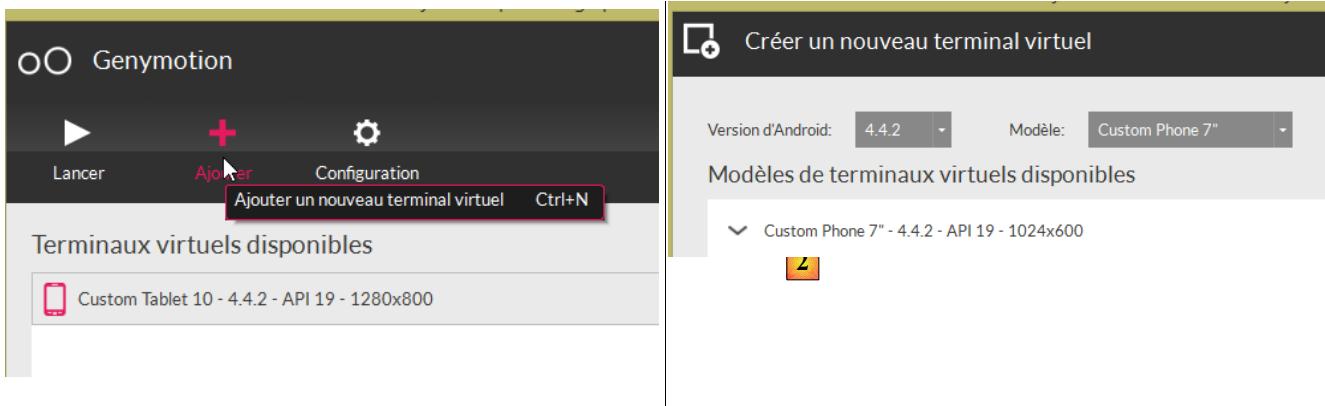
[Windows 32/64 bits \(with VirtualBox\)](#) v2.3.0

#### Download IntelliJ IDEA Plugin

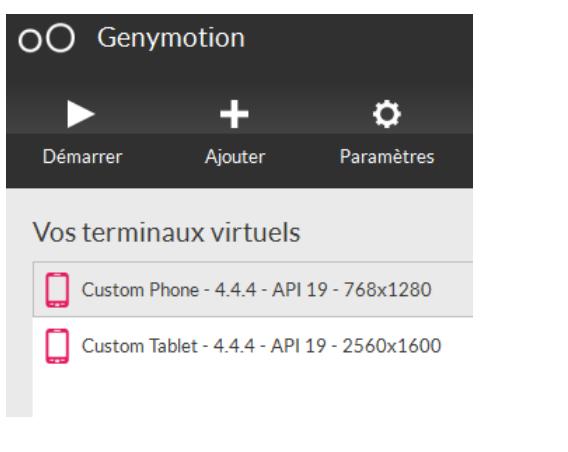
[Genymotion plugin for IntelliJ IDEA](#) v1.0.3

*The installation of the plugin can be done by launching IDEA and going to "File / Settings" menu, then go to "IDE Settings" section, then "Plugins". Click on "Browse repositories" button and search "Genymotion" entry. Follow the steps indicated by IDEA.*  
*Warning: to use this plugin, Genymotion must be installed on your system.*

Nous appellerons par la suite <genymotion-install> le dossier d'installation de [Genymotion]. Lancez [Genymotion]. Téléchargez ensuite une image pour une tablette ou un téléphone :



- en [1], ajoutez un terminal virtuel ;
- en [2], choisissez un ou plusieurs terminaux à installer. Vous pouvez affiner la liste affichée en précisant la version d'Android désirée [3] ainsi que le modèle de terminal [4] ;

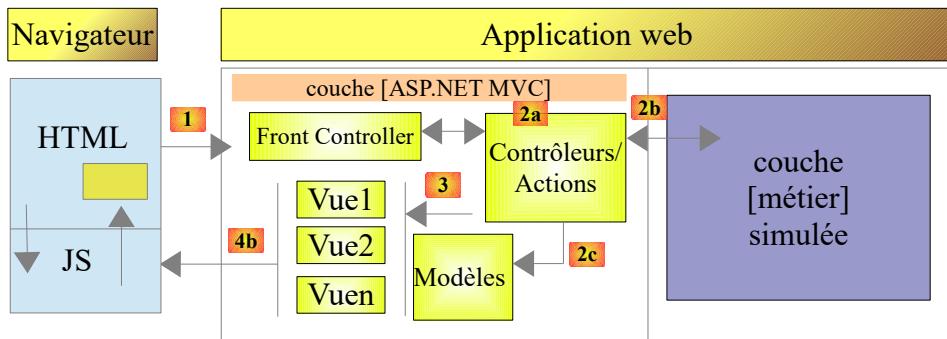


une fois le téléchargement terminé, vous obtenez en [5] la liste des terminaux virtuels dont vous disposez pour tester vos applications Android ;

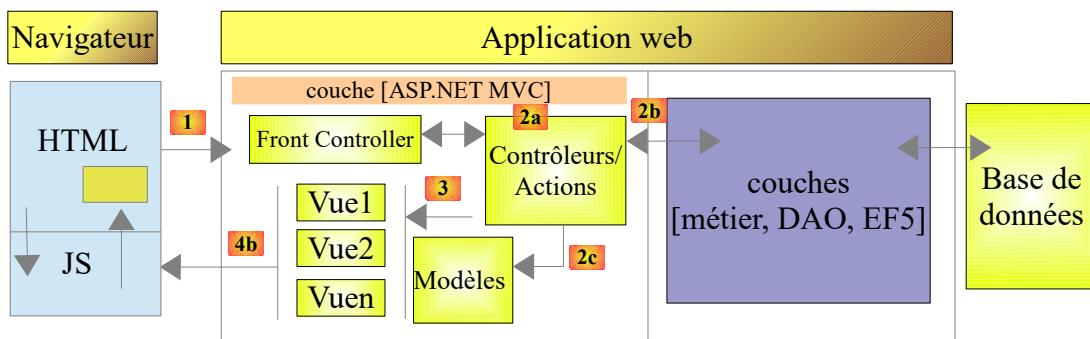
## 2.29 Conclusion

Rappelons ce qui a été fait dans ce document jusqu'à maintenant :

- les chapitres 1 à 8 nous ont présenté les fondamentaux du framework ASP.NET MVC ;
- le chapitre 9 a été consacré à une étude de cas, d'abord avec une architecture simplifiée :



Cette architecture simplifiée nous a permis de nous concentrer uniquement sur la couche [web] et a par ailleurs facilité les tests. Puis nous avons utilisé l'architecture plus complexe suivante :



Nous avons pu voir que les couches [métier], [DAO] et [EF5] amenaient une vraie complexité à l'ensemble de l'application, ce qui a justifié, a posteriori, l'usage d'une architecture simplifiée pour développer la couche [web].

Le lecteur ayant fait cette étude de cas devrait avoir acquis une bonne maîtrise d'ASP.NET MVC et du concept APU, Application à Page Unique.

Il manque certainement une chose dans ce document, ce sont **les tests unitaires**. Ceux-ci auraient dû être faits à différents endroits :

- tests de la couche [DAO] ;
- tests de la couche [métier] réelle ;
- tests des actions de la couche [web].

**Adam Freeman** dans son livre "**Pro ASP.NET MVC 4**" aux éditions **Apress** insiste avec raison sur l'importance de ces tests. On en trouvera de très nombreux dans son livre.

Serge Tahé, novembre 2013

### 3 Étude de cas n° 2 : gestion de bilans de formation

#### 3.1 Les compétences mises en oeuvre

Les compétences mises en oeuvre par cet exercice sont les suivantes :

- savoir écrire une page ASP.NET MVC ;
- savoir gérer le modèle et les événements d'une page ASP.NET MVC ;
- savoir faire des appels AJAX avec JQuery ;
- connaître et savoir utiliser les différentes portées des modèles d'une application web : application, session, request ;
- connaître le rôle des [ModelBinder] ;
- maîtriser Visual Studio et son environnement de développement et notamment savoir utiliser le débogueur ;
- savoir utiliser WampServer, PhpMyAdmin, MySQL ;

#### 3.2 Le problème

On désire gérer les bilans des étudiants d'une formation à l'aide d'une application web. Celle-ci présentera les deux vues suivantes :

##### La vue de connexion à l'application

Gestion des bilans

Login: Form0-Etudiant

Mot de passe: .....  
Messages:

Connect

EI5 AGI - Session 2 2014/2015 - ASP.NET

##### La vue de gestion des bilans

Gestion de bilans de formation

Etudiant : Form0-Etudiant

Formation: Form0 Note : 4,00

Cours: Form0-Module0-Cours0

Show Save Delete

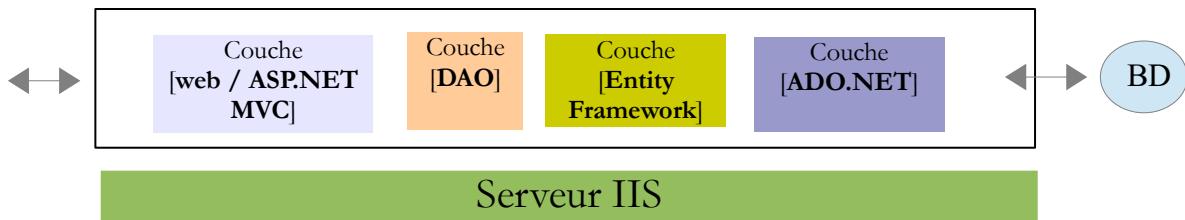
Note : 0  
Plus : 000  
Moins : 000  
Messages:

Disconnect

EI5 AGI - Février 2017 - ASP.NET MVC

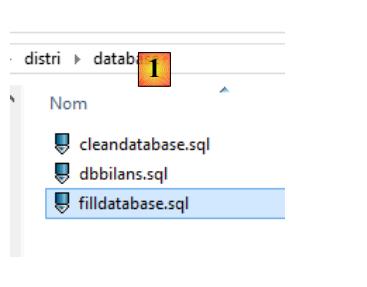
### 3.3 L'architecture de l'application

L'application à écrire a l'architecture en couches suivante :



### 3.4 La base de données

La base de données est une base de données MySQL. Son script SQL de génération vous est donné [1] :

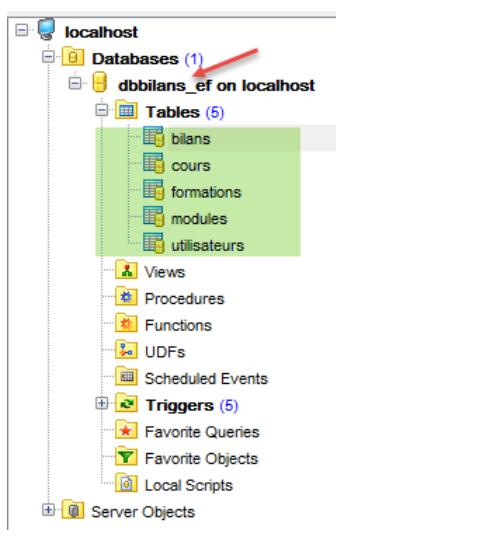


Avec [PhpMyAdmin] :

- exécutez le script [dbbilans.sql] qui crée la base [**dbbilans\_ef**] ;

- exécutez le script [filldatabase.sql] qui vide puis remplit les tables avec des données ;
- utilisez le script [cleandatabase.sql] lorsque vous souhaitez vider les tables sans les remplir ;

La base [dbbilans\_ef] a cinq tables :



La table [**formations**] contient la liste des formations à évaluer :

Properties Fields Indices Foreign Keys Triggers Data Dependencies DDL										
Field Name	Field Type	Size	Precision	Not Null	Unsigned	Zerofill	AutoInc	Default	Description	
ID	BIGINT	20	0	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	Null		
NOM	VARCHAR	30	0	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	Null		
VERSIONING	BIGINT	20	0	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	0		

- [ID] : clé primaire générée automatiquement par le SGBD MySQL ;
- [VERSIONING] : le n° de version de la formation. Est incrémenté de 1 à chaque modification. Chaque table a cette colonne et vous n'avez pas à vous en préoccuper ;

Les deux colonnes précédentes sont présentes dans toutes les tables. Nous ne les présenterons plus.

- [NOM] : nom de la formation ;

La table [**formations**] contient au départ les éléments suivants :

ID	NOM	VERSIONING
32	Form0	0
33	Form1	0

- une formation a des **utilisateurs** et des **modules** ;
- un **module** a des **cours** ;
- un **bilan** est l'évaluation d'un **étudiant** donné pour un **cours** donné ;

La table [**utilisateurs**] contient la liste des utilisateurs, étudiants et enseignants :

Properties Fields Indices Foreign Keys Triggers Data Dependencies DDL								
Field Name	Field Type	Size	Precision	Not Null	Unsigned	Zerofill	Autolnc	Default
ID	BIGINT	20	0	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	Null
LOGIN	VARCHAR	15	0	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	Null
NOM	VARCHAR	30	0	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	Null
PASSWORD	VARCHAR	60	0	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	Null
PRENOM	VARCHAR	30	0	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	Null
VERSIONING	BIGINT	20	0	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	0
ROLE	INTEGER	11	0	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	Null
FORMATION_ID	BIGINT	20	0	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	Null

- [LOGIN] : login de l'utilisateur ;
- [PASSWORD] : mot de passe en clair de l'utilisateur ;
- [NOM, PRENOM] : ses nom et prénom ;
- [ROLE] : il y en a deux, 1 : étudiant, 2 : enseignant ;
- [FORMATION\_ID] : n° de la formation suivie par l'étudiant ou dirigée par l'enseignant. Cette colonne est clé étrangère sur la colonne FORMATIONS.ID ;

La table [utilisateurs] contient au départ les éléments suivants :

ID	LOGIN	NOM	PASSWORD	PRENOM	VERSIONING	ROLE	FORMATION_ID
80	Form0-Etudiant	Form0-Etudiant	Form0-Etudiant	Form0-Etudiant	0	1	32
81	Form1-Etudiant	Form1-Etudiant	Form1-Etudiant	Form1-Etudiant	0	1	33
82	Form0-Prof	Form0-Prof	Form0-Prof	Form0-Prof	0	2	32
83	Form1-Prof	Form1-Prof	Form1-Prof	Form1-Prof	0	2	33

La table [modules] contient la liste des modules des formations de la table [formations]

Properties Fields Indices Foreign Keys Triggers Data Dependencies DDL								
Field Name	Field Type	Size	Precision	Not Null	Unsigned	Zerofill	Autolnc	Default
ID	BIGINT	20	0	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	Null
NOM	VARCHAR	30	0	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	Null
VERSIONING	BIGINT	20	0	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	0
FORMATION_ID	BIGINT	20	0	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	Null

- [NOM] : nom du module ;
- [FORMATION\_ID] : n° de la formation à laquelle appartient le module. Cette colonne est clé étrangère sur la colonne FORMATIONS.ID ;

La table [modules] contient au départ les éléments suivants :

ID	NOM	VERSIONING	FORMATION_ID
85	Form0-Module0	1	32
86	Form0-Module1	1	32
87	Form0-Module2	1	32
88	Form1-Module0	1	33
89	Form1-Module1	1	33
90	Form1-Module2	1	33

La table [cours] contient la liste des cours des modules de la table [modules]

Properties Fields Indices Foreign Keys Triggers Data Dependencies DDL								
Field Name	Field Type	Size	Precision	Not Null	Unsigned	Zerofill	Autonc	Default
ID	BIGINT	20	0	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	Null
NOM	VARCHAR	30	0	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	Null
VERSIONING	BIGINT	20	0	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	0
MODULE_ID	BIGINT	20	0	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	Null
ECTS	INTEGER	11	0	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	Null

- [NOM] : nom du cours ;
- [ECTS] : nombre d'ECTS du cours ;
- [MODULE\_ID] : n° du module auquel appartient le cours. Cette colonne est clé étrangère sur la colonne MODULES.ID ;

La table [**cours**] contient au départ les éléments suivants :

ID	NOM	VERSIONING	MODULE_ID	ECTS
166	Form0-Module0-Cours0	0	85	5
167	Form0-Module0-Cours1	0	85	10
168	Form0-Module1-Cours0	0	86	5
169	Form0-Module1-Cours1	0	86	10
170	Form0-Module2-Cours0	0	87	5
171	Form0-Module2-Cours1	0	87	10
172	Form1-Module0-Cours0	0	88	5
173	Form1-Module0-Cours1	0	88	10
174	Form1-Module1-Cours0	0	89	5
175	Form1-Module1-Cours1	0	89	10
176	Form1-Module2-Cours0	0	90	5
177	Form1-Module2-Cours1	0	90	10

La table [**bilans**] contient les bilans faits par les étudiants de la table [**utilisateurs**] sur les cours de la table [**cours**] :

Properties Fields Indices Foreign Keys Triggers Data Dependencies DDL								
Field Name	Field Type	Size	Precision	Not Null	Unsigned	Zerofill	Autonc	Description
ID	BIGINT	20	0	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	Null
MOINS	LONGTEXT	0	0	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	Null
NOTE	INTEGER	11	0	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	Null
PLUS	LONGTEXT	0	0	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	Null
VERSIONING	BIGINT	20	0	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	0
COURS_ID	BIGINT	20	0	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	Null
ETUDIANT_ID	BIGINT	20	0	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	Null

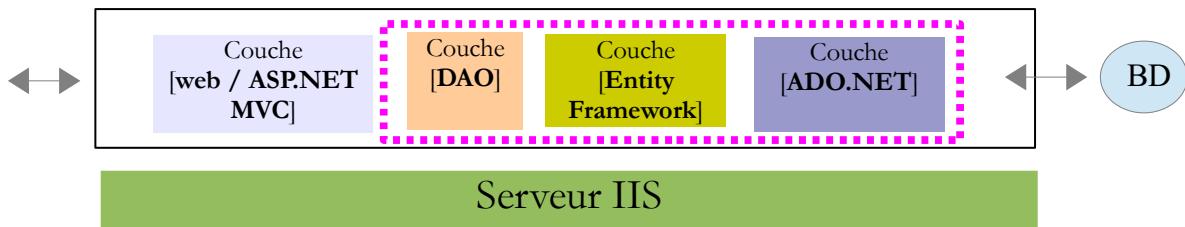
- [NOTE] : note donnée par l'étudiant au cours évalué ;
- [PLUS] : les points forts du cours évalué ;
- [MOINS] : les points faibles du cours évalué ;
- [COURS\_ID] : n° du cours évalué. Est colonne étrangère sur la colonne COURS.ID ;
- [ETUDIANT\_ID] : n° de l'étudiant ayant fait l'évaluation. Est colonne étrangère sur la colonne UTILISATEURS.ID ;

La table [**bilans**] contient au départ les éléments suivants :

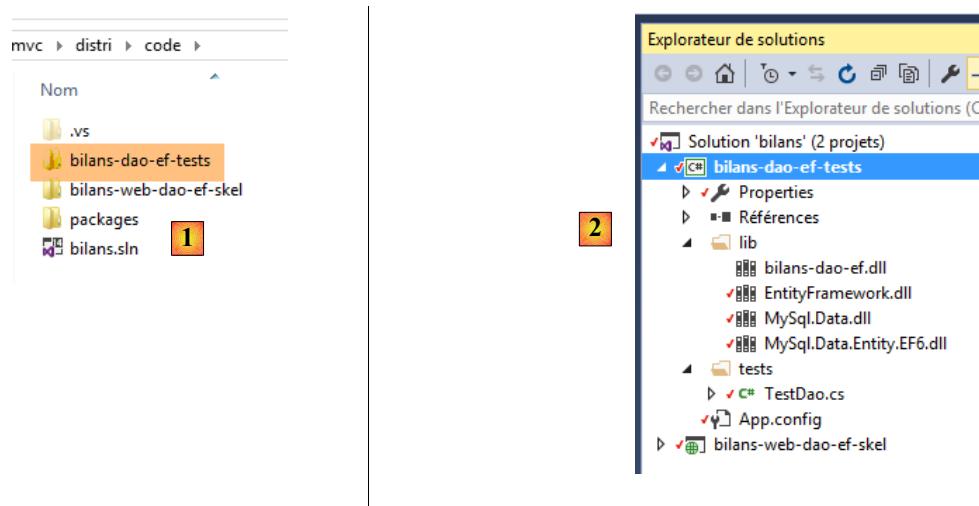
ID	MOINS	NOTE	PLUS	VERSIONING	COURS_ID	ETUDIANT_ID
268	020	4 020	0	0	170	80
269	021	7 021	0	0	171	80

## 3.5 Les couches [DAO, Entity Framework]

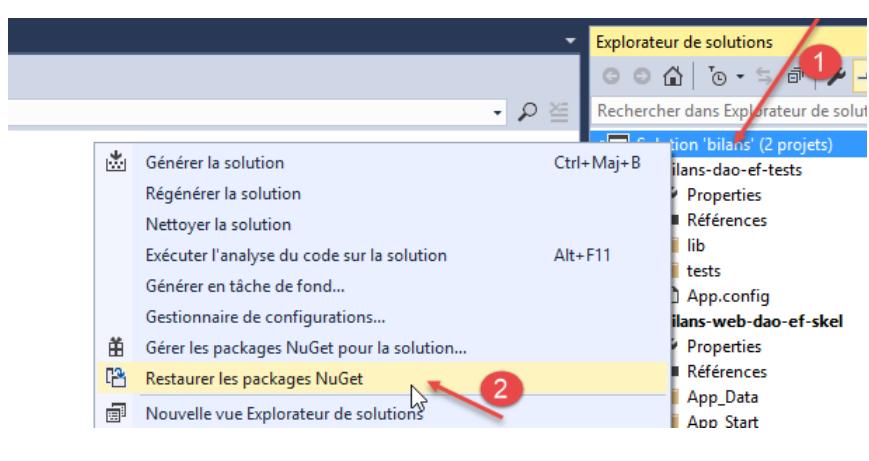
### 3.5.1 Installation et tests



L'implémentation des couches [DAO, Entity Framework] vous est donnée ainsi qu'un programme de test de celle-ci :



- chargez le projet [bilans.sln] ci-dessus [1-2] ;
- restaurez les packages [nuget] comme indiqué ci-dessous :



- [bilans-dao-ef-tests] est un projet de test des couches [DAO-EF] ;
- [bilans-web-dao-ef] est le projet web que vous devez construire dans cet exercice ;
- dans le projet [bilans-dao-ef-tests] :
  - [lib] contient les DLL nécessaires à ce projet :
    - [bilans-dao-ef.dll] est la DLL de la couche [DAO-EF] ;
    - [EntityFramework.dll] est la DLL d'Entity Framework ;
    - [MySql.Data.\*] sont les DLL permettant l'accès à la base de données MySQL ;

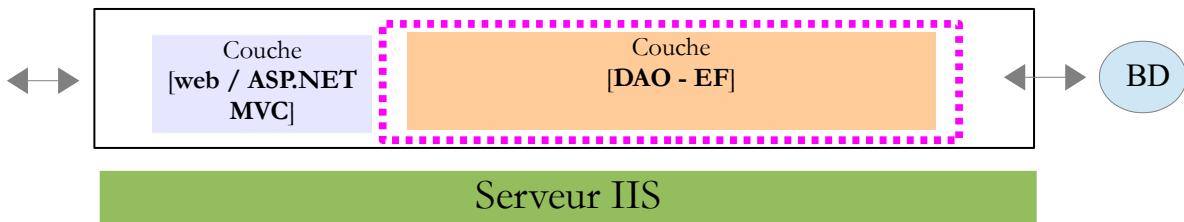
- [TestDao.cs] est le programme de test ;
- [AppConfig] est le fichier de configuration de l'application ;

Pour le test, procédez ainsi :

- videz la base de données en exécutant le script SQL [cleandatabase.sql]. Vérifiez que la base est désormais vide ;
- exécutez le programme [TestDao.cs] (Ctrl-F5). C'est une application console qui va remplir la base MySQL [dbbilans\_ef] ;
- vérifiez ensuite que la base de données a été remplie ;

### 3.5.2 L'interface de la couche [DAO]

Désormais, l'architecture de l'application est la suivante :



Le bloc [DAO-EF] est une boîte noire dont vous devez connaître l'interface pour l'exploiter. Cette interface est la suivante :

```

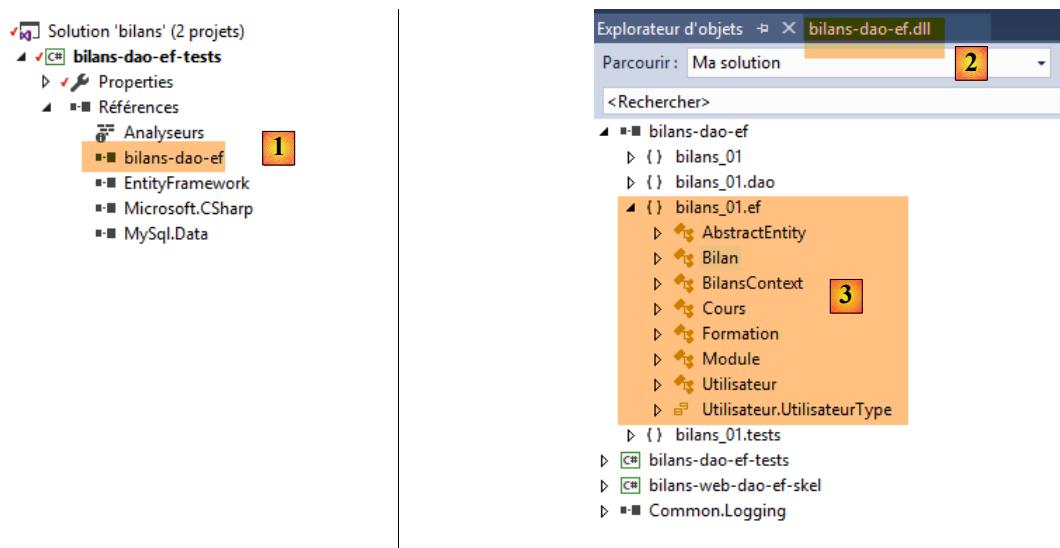
1. 1. using bilans_01.ef;
2. 2. using System;
3. 3. using System.Collections.Generic;
4.
5.
6. 6. namespace bilans_01.dao
7. 7. {
8.
9. 9.     public interface IDao
10. 10.    {
11.
12. 12.        // ----- DAO
13. 13.        // utilisateur identifié par login / password
14. 14.        Utilisateur GetLongUtilisateurByCredentials(string login, string password);
15.
16. 16.        Utilisateur GetShortUtilisateurByCredentials(string login, string password);
17.
18. 18.        // liste des formations
19. 19.        List<Formation> GetAllShortFormations();
20.
21. 21.        // liste des formations d'un utilisateur donné
22. 22.        List<Formation> GetShortFormationsByUtilisateurId(long utilisateurId);
23.
24. 24.        // liste des modules d'une formation
25. 25.        List<Module> GetShortModulesByFormationId(long formationId);
26.
27. 27.        // liste des cours d'un module
28. 28.        List<Cours> GetShortCoursByModuleId(long moduleId);
29.
30. 30.        // liste des cours d'une formation
31. 31.        List<Cours> GetShortCoursByFormationId(long formationId);
32.
33. 33.        // liste des utilisateurs d'une formation
34. 34.        List<Utilisateur> GetShortUtilisateursByFormationId(long formationId, Utilisateur.UtilisateurType
utilisateurType);
35.
36. 36.        // persistance d'un bilan
37. 37.        Bilan SaveBilan(Bilan bilan);
38.
39. 39.        // suppression d'un bilan
40. 40.        void DeleteBilan(long bilanId);
41.
42. 42.        // liste des bilans d'une formation
43. 43.        List<Bilan> GetShortBilansByFormationId(long formationId);
43.

```

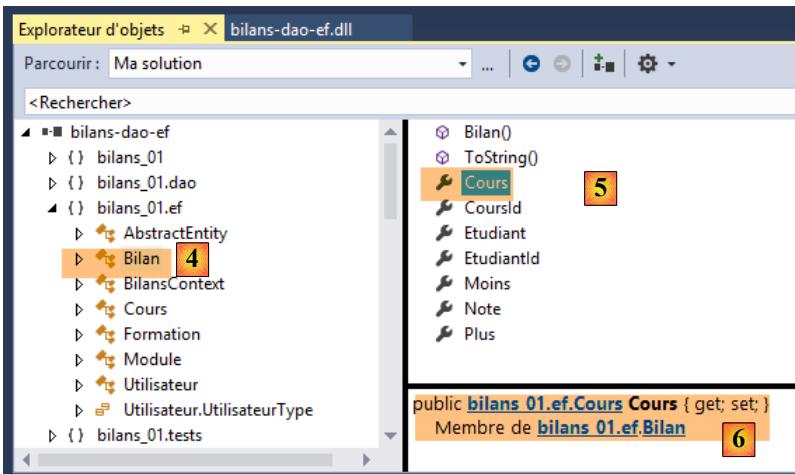
```

44.    // liste des bilans d'un cours
45.    List<Bilan> GetShortBilansByCoursId(long coursId);
46.
47.    // liste des bilans d'un étudiant
48.    List<Bilan> GetShortBilansByEtudiantId(long etudiantId);
49.
50.    // liste des bilans d'un étudiant pour une formation
51.    List<Bilan> GetShortBilansByEtudiantIdByFormationId(long etudiantId, long formationId);
52.
53.    // requêtes.byId
54.    Utilisateur GetShortUtilisateurById(long utilisateurId);
55.
56.    Formation GetShortFormationById(long formationId);
57.
58.    Cours GetShortCoursById(long coursId);
59.
60.    Module GetShortModuleById(long moduleId);
61.
62.    Bilan GetShortBilanById(long bilanId);
63.
64.    // requêtes.byName
65.    Formation GetShortFormationByName(string name);
66.
67.    // gestion base de données
68.    void CleanDatabase();
69.
70.    void FillDataBase();
71.
72. }
73.
74. }
```

Nous donnons dans ce qui suit, une définition **simplifiée** des entités manipulées par cette interface. Si vous avez des doutes, vous pouvez connaître la définition exacte de ces entités en examinant la référence [bilans-dao-ef] du projet de test [1] :



- en [1], double-cliquez sur la référence [bilans-dao-ef]. Vous obtenez la vue [2] ;
- les entités que nous allons décrire sont en [3] ;



- en [4], sélectionnez l'entité que vous voulez examiner ;
- en [5], sélectionnez le champ que vous voulez examiner ;
- en [6], les propriétés du champ sélectionné ;

La classe [Formation] encapsule une ligne de la table [FORMATIONS] de la base de données :

```

1.  namespace bilans_01.ef
2.  {
3.
4.      public abstract class AbstractEntity
5.      {
6.
7.          [Key]
8.          [Column("ID")]
9.          public long? Id { get; set; }
10.
11.         [ConcurrencyCheck]
12.         [Column("VERSIONING")]
13.         public long? Version;
14.
15.     }
16.
17.     [Table("FORMATIONS")]
18.     public class Formation : AbstractEntity
19.     {
20.
21.         [Column("NOM")]
22.         public string Nom { get; set; }
23.
24.         // signature
25.         public override string ToString()
26.         {
27.             return String.Format("Formation [id:{0},nom:{1}]", Id, Nom);
28.         }
29.     }
30.
31. ...

```

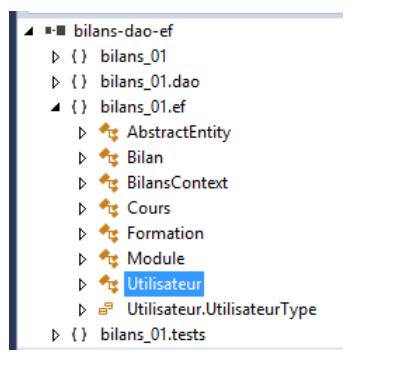
- ligne 1 : toutes les entités sont dans l'espace de noms [bilans\_01.ef] ;
- lignes 4-15 : la classe [AbstractEntity] est parente de toutes les entités. Elle définit deux propriétés :
  - [Id] : la clé primaire de l'entité. Le type [long?] est un type [long] avec la valeur *null* possible en plus. Donc ci-dessus, la propriété [Id] ci-dessus est un entier long qui peut recevoir la valeur *null*. Dans l'exercice, vous pouvez être amenés à écrire des choses telles que :

```
long valeur=(long) formation.Id ;
```

En effet, pour affecter un type [long?] à un type [long], il faut faire un transtypage. Le compilateur vous le signalera.

- [Version] : le n° de version de l'entité. Il est incrémenté à chaque modification de l'entité. Vous n'avez pas à vous préoccuper de ce champ dans l'exercice. L'annotation [ConcurrencyCheck] indique que ce champ sera utilisé par EF pour gérer les accès concurrents à une même entité de la base de données ;
- ligne 18 : la classe [Formation] étend la classe [AbstractEntity] et donc hérite des propriétés [Id, Version] ;
- ligne 17 : la classe [Formation] encapsule une ligne de la table [FORMATIONS] ;
- lignes 21-22 : le nom de la formation ;

La classe [Utilisateur] encapsule une ligne de la table [UTILISATEURS] de la base de données :



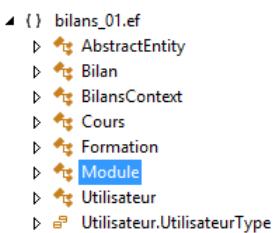
```

1.  [Table("UTILISATEURS")]
2.  public class Utilisateur : AbstractEntity
3.  {
4.
5.      [Column("NOM")]
6.      public string Nom { get; set; }
7.
8.      [Column("PRENOM")]
9.      public string Prenom { get; set; }
10.
11.     [Column("LOGIN")]
12.     public string Login { get; set; }
13.
14.     [Column("PASSWORD")]
15.     public string Password { get; set; }
16.
17.     public enum UtilisateurType { ALL, ETUDIANT, PROF };
18.
19.     [Column("ROLE")]
20.     public UtilisateurType Role { get; set; }
21.
22.     [Column("FORMATION_ID")]
23.     public long? FormationId { get; set; }
24.
25.     [ForeignKey("FormationId")]
26.     public Formation Formation { get; set; }
27.
28.     // signature
29.     public override string ToString()
30.     {
31.         return String.Format("Utilisateur [id:{0},nom:{1},utilisateurType:{2},formationId:{3}]", Id, Nom,
            Role, FormationId);
32.     }
33. }
```

- ligne 17 : une énumération qui définit trois entiers :
  - [UtilisateurType.ALL] qui vaut 0 ;
  - [UtilisateurType.ETUDIANT] qui vaut 1 ;
  - [UtilisateurType.PROFESSEUR] qui vaut 2 ;
- ligne 20 : pour attribuer une valeur à la propriété [Role], vous pouvez écrire quelque chose d'analogue à ceci :
  - role= UtilisateurType.ETUDIANT ;

- ligne 23 : le n° de la formation de l'utilisateur ;
- ligne 26 : la formation de l'utilisateur ;

La classe [Module] encapsule une ligne de la table [MODULES] de la base de données :

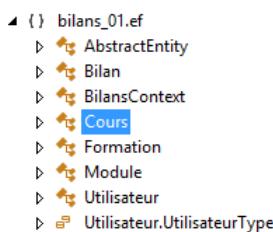


```

1.  [Table("MODULES")]
2.  public class Module : AbstractEntity
3.  {
4.
5.    [Column("NOM")]
6.    public string Nom { get; set; }
7.
8.    [Column("FORMATION_ID")]
9.    public long? FormationId { get; set; }
10.
11.   [ForeignKey("FormationId")]
12.   public Formation Formation { get; set; }
13.
14.   // signature
15.   public override string ToString()
16.   {
17.     return String.Format("Module [id:{0},nom:{1},formationId:{2}]", Id, Nom, FormationId);
18.   }
19. }
  
```

- ligne 9 : le n° de la formation à laquelle appartient le module ;
- ligne 12 : la formation à laquelle appartient le module ;

La classe [Cours] encapsule une ligne de la table [COURS] de la base de données :



```

1.  [Table("COURS")]
2.  public class Cours : AbstractEntity
3.  {
4.
5.    [Column("NOM")]
6.    public string Nom { get; set; }
7.
8.    [Column("ECTS")]
9.    public int Ects { get; set; }
10.
11.   [Column("MODULE_ID")]
12.   public long? ModuleId { get; set; }
13.
14.   [ForeignKey("ModuleId")]
15.   public Module Module { get; set; }
16.
  
```

```

17.     // signature
18.     public override string ToString()
19.     {
20.         return String.Format("Cours [id:{0},nom:{1},moduleId:{2}]", Id, Nom, ModuleId);
21.     }
22. }

```

- ligne 12 : le n° du module auquel appartient le cours ;
- ligne 15 : le module auquel appartient le cours ;

La classe [Bilan] encapsule une ligne de la table [BILANS] de la base de données :

```

▲ {} bilans_01.ef
  ▷ AbstractEntity
  ▷ Bilan
  ▷ BilansContext
  ▷ Cours
  ▷ Formation
  ▷ Module
  ▷ Utilisateur
  ▷ Utilisateur.UtilisateurType

```

```

1. [Table("BILANS")]
2.     public class Bilan : AbstractEntity
3.     {
4.
5.         [Column("NOTE")]
6.         public int Note { get; set; }
7.
8.         [Column("PLUS")]
9.         public string Plus { get; set; }
10.
11.        [Column("MOINS")]
12.        public string Moins { get; set; }
13.
14.        [Column("COURS_ID")]
15.        public long? CoursId { get; set; }
16.
17.        [ForeignKey("CoursId")]
18.        public Cours Cours { get; set; }
19.
20.        [Column("ETUDIANT_ID")]
21.        public long? EtudiantId { get; set; }
22.
23.        [ForeignKey("EtudiantId")]
24.        public Utilisateur Etudiant { get; set; }
25.
26.        // signature
27.        public override string ToString()
28.        {
29.            return String.Format("Bilan [id:{0},note:{1},plus:{2},moins:{3},coursId:{4},etudiantId:{5}]", Id,
   Note, Plus, Moins, CoursId, EtudiantId);
30.        }
31.    }

```

- ligne 21 : le n° de l'étudiant qui a fait le bilan ;
- ligne 24 : l'étudiant qui a fait le bilan ;
- ligne 15 : le n° du cours évalué ;
- ligne 18 : le cours évalué ;
- ligne 6 : la note du bilan, un nombre dans l'intervalle [0,10] ;
- ligne 9 : les points à conserver ;
- ligne 12 : les points à améliorer ;

### 3.5.3 Versions courtes et longues des entités

Dans l'interface [IDao] on trouve deux sortes de méthodes, par exemple :

```
1. // utilisateur identifié par login / password
```

```

2. Utilisateur GetLongUtilisateurByCredentials(string login, string password);
3.
4. Utilisateur GetShortUtilisateurByCredentials(string login, string password);

```

La méthode de la ligne 1 rend la version longue de l'utilisateur alors que la ligne 3 rend la version courte. Revenons à la définition simplifiée de la classe [Utilisateur] :

```

1. [Table("UTILISATEURS")]
2. public class Utilisateur : AbstractEntity
3. {
4.
5.     [Column("NOM")]
6.     public string Nom { get; set; }
7.
8.     [Column("PRENOM")]
9.     public string Prenom { get; set; }
10.
11.    [Column("LOGIN")]
12.    public string Login { get; set; }
13.
14.    [Column("PASSWORD")]
15.    public string Password { get; set; }
16.
17.    public enum UtilisateurType { ALL, ETUDIANT, PROF };
18.
19.    [Column("ROLE")]
20.    public UtilisateurType Role { get; set; }
21.
22.    [Column("FORMATION_ID")]
23.    public long? FormationId { get; set; }
24.
25.    [ForeignKey("FormationId")]
26.    public Formation Formation { get; set; }
27.
28.    // signature
29.    public override string ToString()
30.    {
31.        return String.Format("Utilisateur [id:{0},nom:{1},utilisateurType:{2},formationId:{3}]", Id, Nom,
32.        Role, FormationId);
32.    }
33. }

```

- lignes 25-26 : selon la requête faite à la couche [Entity Framework], le champ [Formation] peut être initialisé ou non. On appellera :
  - version **courte** d'un utilisateur, une instance [Utilisateur] où le champ [Formation Formation] n'a pas été initialisé avec la formation de l'utilisateur mais avec autre chose appelée un PROXY. **Accéder à ce champ peut provoquer alors une exception.** Cela dépend de la présence ou non de la formation associée à l'utilisateur dans le contexte de persistance d'Entity Framework. Si par une opération sur ce contexte, on a ramené **auparavant** la formation F de l'utilisateur U, la notation [U.Formation] rendra bien cette formation F, sinon on aura une exception. Du coup, l'ordre des demandes faites à Entity Framework a une importance. Un code fonctionnel à un moment donné peut ne plus l'être si les demandes faites à EF le sont dans un ordre différent. Pour éviter des erreurs difficiles à détecter, il vous est conseillé dans un premier temps de toujours supposer qu'avec la version courte de [Utilisateur], le champ [Utilisateur.Formation] n'est pas utilisé sauf si vous êtes sûr que cette formation a déjà été obtenue par une requête précédente. Le plus simple est d'utiliser le débogueur pour inspecter l'instance de la classe [Utilisateur] obtenue ;
  - version **longue** d'un utilisateur, une instance [Utilisateur] où le champ [Formation Formation] a été initialisé avec la formation de l'utilisateur ;

Ainsi si nous revenons aux deux méthodes présentées :

```

1. Utilisateur getLongUtilisateurByCredentials(String login, String password);
2.
3. Utilisateur getShortUtilisateurByCredentials(String login, String password);

```

- la méthode de la ligne 1 rend un utilisateur U où la propriété [U.Formation] peut être utilisée ;
- la méthode de la ligne 3 rend un utilisateur U où la propriété [U.formation] peut ou pas être utilisée (ça dépend) ;

Il en est ainsi pour toutes les entités de cet exercice. Lorsqu'une méthode à le mot clé [Short], cela signifie qu'elle rend une entité E dans laquelle la propriété correspondant à l'entité pointée par sa clé étrangère n'est peut-être pas accessible.

Revenons au cas de l'utilisateur :

```

1. [Table("UTILISATEURS")]
2. public class Utilisateur : AbstractEntity
3. {
4. ...
5.
6.     [Column("FORMATION_ID")]
7.     public long? FormationId { get; set; }
8.
9.     [ForeignKey("FormationId")]
10.    public Formation Formation { get; set; }
11. ...
12. }
```

Lorsqu'on a la version courte de l'utilisateur, la propriété [Formation] (ligne 10) n'est peut-être pas utilisable. Mais la propriété [FormationId] de la ligne 7 elle, l'est. Elle représente la clé primaire de la formation à laquelle appartient l'étudiant. Si on voulait cette formation, elle pourrait être obtenue par la méthode suivante de l'interface [IDao] :

```
Formation GetShortFormationById(long formationId);
```

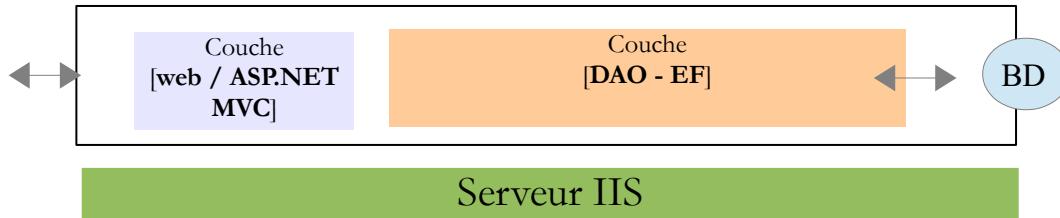
Ces méthodes existent pour toutes les entités :

```

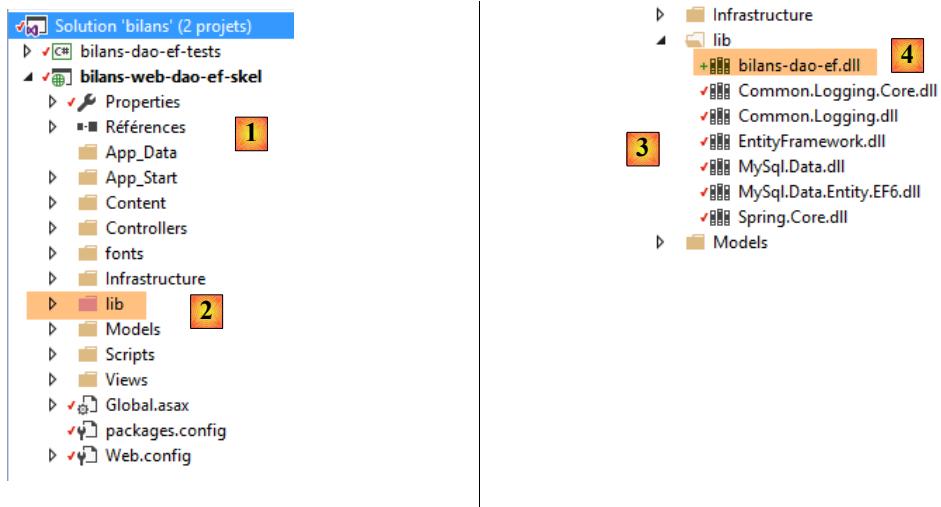
1. // requêtes getById
2. Utilisateur GetShortUtilisateurById(long utilisateurId);
3.
4. Formation GetShortFormationById(long formationId);
5.
6. Cours GetShortCoursById(long coursId);
7.
8. Module GetShortModuleById(long moduleId);
9.
10. Bilan GetShortBilanById(long bilanId);
```

## 3.6 La couche [web / ASP.NET MVC]

### 3.6.1 L'implémentation de démarrage

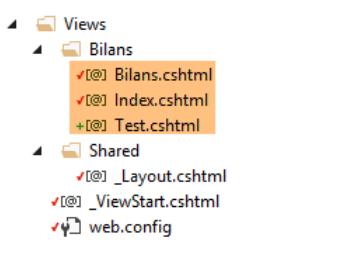


Une implémentation de démarrage de la couche [web / ASPNET MVC] vous est fournie :



- en [1], l'application [bilans-web-dao-ef-skel] que vous allez développer ;
- en [2-3], le dossier [lib] contient les DLL dont vous avez besoin. Elles sont déjà présentes dans les références du projet ;
- en [4], la DLL de la couche [DAO-EF] ;

L'application [1] a trois vues :



- [Test.cshtml] est une page de test ;
- [Index.cshtml] est la page de connexion à l'application ;
- [Bilans.cshtml] est la page de gestion des bilans de l'utilisateur connecté ;

Lorsqu'on exécute l'application, on obtient la page suivante :



Si vous obtenez cette page, alors l'application est correctement installée. Si la connexion à la base de données n'a pu se faire, vous obtiendrez la page suivante :

# Gestion de bilans de formation

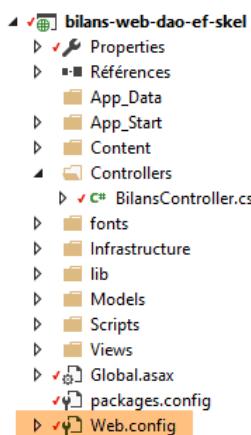
## Liste des formations :

```
System.Data.Entity.Core.EntityException: The underlying provider failed on Open. --> MySql.Data.MySqlClient.MySqlException: Unable to connect to any of the specified MySQL hosts. à MySql.Data.MySqlClient.NativeDriver.Open() à MySql.Data.MySqlClient.Driver.Open() à MySql.Data.MySqlClient.Driver.Create(MySQLConnectionStringBuilder settings) à MySql.Data.MySqlClient.MySqlPool.GetPooledConnection() à MySql.Data.MySqlClient.MySqlPool.TryToGetDriver() à MySql.Data.MySqlClient.MySqlPool.GetConnection() à MySql.Data.MySqlClient.MySqlConnection.Open() à System.Data.Entity.Infrastructure.Interception.DbConnectionDispatcher.<Open>b__36(DbConnection t, DbConnectionInterceptionContext c) à System.Data.Entity.Infrastructure.Interception.InternalDispatcher`1.Dispatch(TTarget, TInterceptionContext)(TTarget target, Action`2 operation, TInterceptionContext interceptionContext, Action`3 executing, Action`3 executed) à System.Data.Entity.Infrastructure.Interception.DbConnectionDispatcher.Open(DbConnection connection, DbInterceptionContext interceptionContext) à System.Data.Entity.Core.EntityClient.EntityConnection.<Open>b__2() à System.Data.Entity.Infrastructure.DefaultExecutionStrategy.Execute(Action operation) à System.Data.Entity.Core.EntityClient.EntityConnection.Open() --- Fin de la trace de la pile d'exception interne --- à System.Data.Entity.Core.EntityClient.EntityConnection.Open() à System.Data.Entity.Core.Objects.ObjectContext.EnsureConnection(Boolean shouldMonitorTransactions) à System.Data.Entity.Core.Objects.ObjectContext.ExecuteInTransaction[T](Func`1 func, IExecutionStrategy executionStrategy, Boolean startLocalTransaction, Boolean releaseConnectionOnSuccess) à System.Data.Entity.Core.Objects.ObjectQuery`1.<>c__DisplayClass7_<GetResults>b__5() à System.Data.Entity.Infrastructure.DefaultExecutionStrategy.Execute[TResult](Func`1 operation) à System.Data.Entity.Core.Objects.ObjectQuery`1.GetResults(Nullable`1 forMergeOption) à System.Data.Entity.Core.Objects.ObjectQuery`1.MoveNext() <System.Collections.Generic.List`1..ctor(IEnumerable`1 collection) à System.Linq.Enumerable.ToList[TSource](IEnumerable`1 source) à bilans_01.dao.Dao.GetAllShortFormations() à bilans_01.web.BilansController.Test(ApplicationModel application) dans D:\data\listia-1617\ei5-agilaspnet\contrôles\cclasnetmvcldistri\code\bilans-web-dao-ef-skel\Controllers\BilansController.cs:ligne 20
```

EI5 AGI - Février 2017 - ASP.NET MVC

Vérifiez alors que le sgbd MySQL est bien lancé et refaites le test.

### 3.6.2 Configuration de l'application



Le fichier [Web.config] configure l'application web. Son code est le suivant :

```
1.  <?xml version="1.0" encoding="utf-8"?>
2.  <!--
3.  Pour plus d'informations sur la configuration de votre application ASP.NET, rendez-vous sur
4.  http://go.microsoft.com/fwlink/?LinkId=301880
5.  -->
6.  <configuration>
7.    <configSections>
8.      <sectionGroup name="spring">
9.        <section name="objects" type="Spring.Context.Support.DefaultSectionHandler, Spring.Core" />
10.       <section name="context" type="Spring.Context.Support.ContextHandler, Spring.Core" />
11.     </sectionGroup>
12.     <!-- For more information on Entity Framework configuration, visit http://go.microsoft.com/fwlink/?LinkID=237468 -->
13.     <section name="entityFramework" type="System.Data.Entity.Internal.ConfigFile.EntityFrameworkSection,
14. EntityFramework, Version=6.0.0.0, Culture=neutral, PublicKeyToken=b77a5c561934e089"
15. requirePermission="false" />
16.   </configSections>
17.   <!-- configuration Spring -->
18.   <spring>
19.     <context>
20.       <resource uri="config://spring/objects" />
21.     </context>
```

```

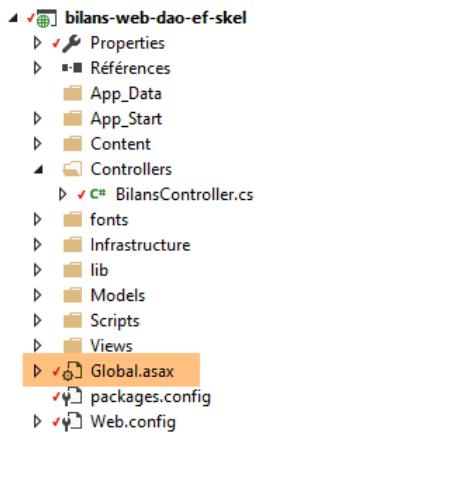
20.      <objects xmlns="http://www.springframework.net">
21.          <object id="dao" type="bilans_01.dao.Dao, bilans-dao-ef" />
22.      </objects>
23.  </spring>
24.  <entityFramework>
25.      <providers>
26.          <provider invariantName=" MySql.Data.MySqlClient " type=" MySql.Data.MySqlClient.MySqlProviderServices,
    MySql.Data.Entity.EF6, Version=6.9.6.0, Culture=neutral, PublicKeyToken=c5687fc88969c44d ">
27.              </provider>
28.          </providers>
29.      </entityFramework>
30.  <system.data>
31.      <DbProviderFactories>
32.          <remove invariant=" MySql.Data.MySqlClient " />
33.          <add name=" MySQL Data Provider " invariant=" MySql.Data.MySqlClient " description=".Net Framework Data
    Provider for MySQL " type=" MySql.Data.MySqlClient.MySqlClientFactory, MySql.Data, Version=6.9.6.0,
    Culture=neutral, PublicKeyToken=c5687fc88969c44d " />
34.      </DbProviderFactories>
35.  </system.data>
36.  <connectionStrings>
37.      <add name=" BilansContext " connectionString=" Server=localhost;Database=dbbilans_ef;Uid=root;Pwd=; "
    providerName=" MySql.Data.MySqlClient "/>
38.  </connectionStrings>
39.  <!-- autogénéré-->
40.  <appSettings>
41.      <add key=" webpages:Version " value=" 3.0.0.0 " />
42.      <add key=" webpages:Enabled " value=" false " />
43.      <add key=" ClientValidationEnabled " value=" true " />
44.      <add key=" UnobtrusiveJavaScriptEnabled " value=" true " />
45.  </appSettings>
46.  <system.web>
47.      <compilation debug=" true " targetFramework=" 4.5 " />
48.      <httpRuntime targetFramework=" 4.5 " />
49.  </system.web>
50.  <runtime>
51.      <assemblyBinding xmlns=" urn:schemas-microsoft-com:asm.v1 ">
52.          <dependentAssembly>
53.              <assemblyIdentity name=" System.Web.Helpers " publicKeyToken=" 31bf3856ad364e35 " />
54.              <bindingRedirect oldVersion=" 1.0.0.0-3.0.0.0 " newVersion=" 3.0.0.0 " />
55.          </dependentAssembly>
56.          <dependentAssembly>
57.              <assemblyIdentity name=" System.Web.WebPages " publicKeyToken=" 31bf3856ad364e35 " />
58.              <bindingRedirect oldVersion=" 1.0.0.0-3.0.0.0 " newVersion=" 3.0.0.0 " />
59.          </dependentAssembly>
60.          <dependentAssembly>
61.              <assemblyIdentity name=" System.Web.Mvc " publicKeyToken=" 31bf3856ad364e35 " />
62.              <bindingRedirect oldVersion=" 1.0.0.0-5.2.3.0 " newVersion=" 5.2.3.0 " />
63.          </dependentAssembly>
64.      </assemblyBinding>
65.  </runtime>
66. </configuration>

```

De cette configuration, il faut retenir les points suivants :

- la ligne 37 définit la base de données à exploiter [dbbilans\_ef]. Le pilote ADO.NET utilisé [providerName] est défini ligne 33. C'est le provider du SGBD MySQL ;
- lignes 16-23 : définissent les objets gérés par Spring. On retiendra l'id [dao] (ligne 21) de l'objet implémentant la couche [DAO-EF] ;

### 3.6.3 Initialisation de l'application



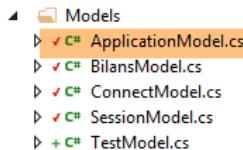
Le fichier [Global.asax] est exécuté au démarrage de l'application et assure l'initialisation de celle-ci. Son code est le suivant :

```

1.  using System;
2.  using System.Web.Mvc;
3.  using System.Web.Routing;
4.  using Spring.Context.Support;
5.  using bilans_01.dao;
6.
7.  namespace bilans_01.web
8.  {
9.      public class MvcApplication : System.Web.HttpApplication
10.     {
11.         protected void Application_Start()
12.         {
13.             // auto-généré
14.             AreaRegistration.RegisterAllAreas();
15.             RouteConfig.RegisterRoutes(RouteTable.Routes);
16.             // -----
17.             // ----- configuration spécifique
18.             // -----
19.             // données de portée application
20.             ApplicationModel application = new ApplicationModel();
21.             Application["data"] = application;
22.             // instanciation couche [dao]
23.             application.Dao = ContextRegistry.GetContext().GetObject("dao") as IDao;
24.             // model binders
25.             ModelBinders.Binders.Add(typeof(ApplicationModel), new ApplicationModelBinder());
26.             ModelBinders.Binders.Add(typeof(SessionModel), new SessionModelBinder());
27.         }
28.
29.         // session
30.         protected void Session_Start()
31.         {
32.             Session["data"] = new SessionModel();
33.         }
34.     }
35. }
```

- lignes 20-21 : un modèle de portée [Application] est créé et associé à la clé [data] ;
- ligne 23 : le fichier [Web.config] est exploité pour instancier la couche [DAO-EF] grâce au framework Spring. Une référence de la couche est stockée dans le modèle de portée [Application] ;
- lignes 25-26 : définissent les deux [Binders] qui instancieront les modèles de portée [Application] et [Session] ;
- lignes 30-33 : le modèle de portée [Session] est créé et associé à la clé [data] ;

### 3.6.4 Le modèle de portée [Application]



La classe [ApplicationModel] est instanciée une unique fois, au démarrage de l'application par [Global.asax]. Elle est de portée [Application] ce qui signifie que son contenu est accessible à toutes les requêtes HTTP de tous les utilisateurs de l'application. Son code est le suivant :

```
1. using bilans_01.dao;
2. using bilans_01.ef;
3. using System.Collections.Generic;
4. using System;
5.
6. namespace bilans_01.web
7. {
8.     public class ApplicationModel : IDao
9.     {
10.         // la couche [DAO]
11.         public IDao Dao { get; set; }
12.
13.         // ----- interface IDao
14.         public void CleanDatabase()
15.         {
16.             Dao.CleanDatabase();
17.         }
18.
19.         public void DeleteBilan(long bilanId)
20.         {
21.             Dao.DeleteBilan(bilanId);
22.         }
23.
24.         public void FillDataBase()
25.         {
26.             Dao.FillDataBase();
27.         }
28.
29.         public List<Formation> GetAllShortFormations()
30.         {
31.             return Dao.GetAllShortFormations();
32.         }
33.
34.         public Bilan GetShortBilanById(long bilanId)
35.         {
36.             return Dao.GetShortBilanById(bilanId);
37.         }
38.
39.         public Cours GetShortCoursById(long coursId)
40.         {
41.             return Dao.GetShortCoursById(coursId);
42.         }
43.
44.         public Formation GetShortFormationById(long formationId)
45.         {
46.             return Dao.GetShortFormationById(formationId);
47.         }
48.
49.         public Module GetShortModuleById(long moduleId)
50.         {
51.             return Dao.GetShortModuleById(moduleId);
52.         }
53.
54.         public Utilisateur GetLongUtilisateurByCredentials(string login, string password)
55.         {
56.             return Dao.GetLongUtilisateurByCredentials(login, password);
57.         }
58.
59.         public Utilisateur GetShortUtilisateurById(long utilisateurId)
60.         {
61. }
```

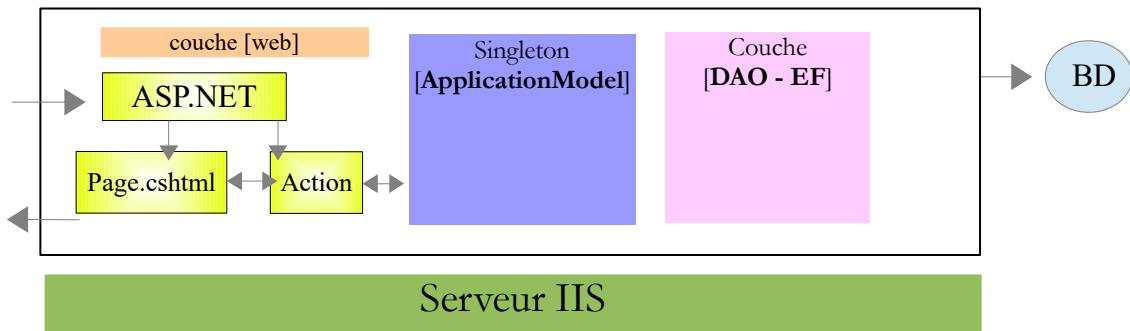
```

62.     return Dao.GetShortUtilisateurById(utilisateurId);
63. }
64.
65. public List<Bilan> GetShortBilansByCoursId(long coursId)
66. {
67.     return Dao.GetShortBilansByCoursId(coursId);
68. }
69.
70. public List<Bilan> GetShortBilansByEtudiantId(long etudiantId)
71. {
72.     return Dao.GetShortBilansByEtudiantId(etudiantId);
73. }
74.
75. public List<Bilan> GetShortBilansByEtudiantIdByFormationId(long etudiantId, long formationId)
76. {
77.     return Dao.GetShortBilansByEtudiantIdByFormationId(etudiantId, formationId);
78. }
79.
80. public List<Bilan> GetShortBilansByFormationId(long formationId)
81. {
82.     return Dao.GetShortBilansByFormationId(formationId);
83. }
84.
85. public List<Cours> GetShortCoursByFormationId(long formationId)
86. {
87.     return Dao.GetShortCoursByFormationId(formationId);
88. }
89.
90. public List<Cours> GetShortCoursByModuleId(long moduleId)
91. {
92.     return Dao.GetShortCoursByModuleId(moduleId);
93. }
94.
95. public Formation GetShortFormationByName(string name)
96. {
97.     return Dao.GetShortFormationByName(name);
98. }
99.
100. public List<Formation> GetShortFormationsByUtilisateurId(long utilisateurId)
101. {
102.     return Dao.GetShortFormationsByUtilisateurId(utilisateurId);
103. }
104.
105. public List<Module> GetShortModulesByFormationId(long formationId)
106. {
107.     return Dao.GetShortModulesByFormationId(formationId);
108. }
109.
110. public Utilisateur GetShortUtilisateurByCredentials(string login, string password)
111. {
112.     return Dao.GetShortUtilisateurByCredentials(login, password);
113. }
114.
115. public List<Utilisateur> GetShortUtilisateursByFormationId(long formationId,
    Utilisateur.UtilisateurType utilisateurType)
116. {
117.     return Dao.GetShortUtilisateursByFormationId(formationId, utilisateurType);
118. }
119.
120. public Bilan SaveBilan(Bilan bilan)
121. {
122.     return Dao.SaveBilan(bilan);
123. }
124. }
125. }

```

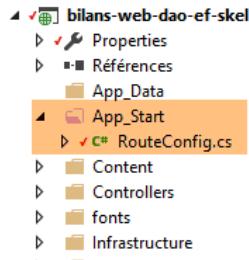
- ligne 11 : une référence sur la couche [DAO-EF]. Nous avons vu que celle-ci était initialisée par Spring lors de l'instanciation de la classe [ApplicationModel]. Donc lorsqu'on utilise l'unique instance de la classe [ApplicationModel], ce champ a été initialisé ;
- ligne 8 : la classe [ApplicationModel] implémente l'interface [IDao] de la couche [DAO-EF] ;
- lignes 13-124 : implémentation de cette interface ;

Du fait que le singleton [ApplicationModel] implémente l'interface [IDao], l'architecture MVC de notre application est la suivante :



Les actions du contrôleur de l'application dialogueront avec le singleton [ApplicationModel] au lieu de dialoguer directement avec la couche [DAO-EF].

### 3.6.5 Le routage des URL



Le fichier [RouteConfig.cs] configure le routage des URL, c-à-d qu'à chaque URL demandée par un client, il affecte un contrôleur et une action dans celui-ci pour traiter cette demande. Son contenu est ici le suivant :

```

1.  using System.Web.Mvc;
2.  using System.Web.Routing;
3.
4.  namespace bilans_01.web
5.  {
6.      public class RouteConfig
7.      {
8.          public static void RegisterRoutes(RouteCollection routes)
9.          {
10.              routes.IgnoreRoute("{resource}.axd/{*pathInfo}");
11.
12.              routes.MapRoute(
13.                  name: "Default",
14.                  url: "{action}",
15.                  defaults: new { controller = "Bilans", action = "Test" }
16.              );
17.          }
18.      }
19.  }

```

- lignes 12-16 : l'unique configuration de routage ;
- ligne 14 : une URL de type / {action} sera traitée par l'action [action] du contrôleur [BilansController] (valeur par défaut du contrôleur, ligne 15) ;
- ligne 15 : une URL vide sera traitée par l'action [Test] (valeur par défaut de l'action ligne 15) du contrôleur [BilansController]. C'est ce qui se passe notamment lorsqu'on lance l'application à partir de Visual Studio. Une URL vide est alors demandée et c'est donc l'action [Test] qui s'exécute ;

### 3.6.6 La page [Test.cshtml] et son modèle [TestModel]

The left side shows a file explorer with the following structure:

```

    ▲ Models
        ▷ ✓ C# ApplicationModel.cs
        ▷ ✓ C# BilansModel.cs
        ▷ ✓ C# ConnectModel.cs
        ▷ ✓ C# SessionModel.cs
        ▷ + C# TestModel.cs [2]
    ▷ Scripts
    ▲ Views
        ▲ Bilans
            ✓[@] Bilans.cshtml
            ✓[@] Index.cshtml
            +[@] Test.cshtml [1]
        ▲ Shared
            ✓[@] _Layout.cshtml
            ✓[@] _ViewStart.cshtml
    ▷ web.config

```

The right side shows a browser window at [localhost:51358](http://localhost:51358) displaying the "Gestion de bilans de formation" page. The title is "Gestion de bilans de formation". Below it is a section titled "Liste des formations:" containing a bulleted list: "Form0" and "Form1". At the bottom of the page is the footer "E15 AGI - Février 2017 - ASP.NET MVC".

La page [Test.cshtml] affiche la vue [3] ou bien un texte d'erreur. Son code est le suivant :

```

1. @model bilans_01.web.TestModel
2. @using bilans_01.ef
3.
4. <!DOCTYPE html>
5.
6. <html>
7. <head>
8.     <meta name="viewport" content="width=device-width" />
9.     <title>Gestion des bilans</title>
10. </head>
11. <body>
12.     <h3>Liste des formations : </h3>
13.     @if (Model.Msg != null)
14.     {
15.         <span style="color: red">@Model.Msg</span>
16.     }
17.     else
18.     {
19.         <ul>
20.             @foreach (Formation f in Model.Formations)
21.             {
22.                 <li>@f.Nom</li>
23.             }
24.         </ul>
25.     }
26. </body>
27. </html>

```

- ligne 1 : le modèle de la page ;
- ligne 2 : import de l'espace de noms [bilans\_01.ef] ;

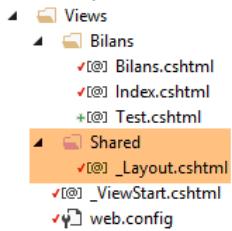
Le modèle [TestModel] est le suivant :

```

1. using bilans_01.ef;
2. using System.Collections.Generic;
3.
4. namespace bilans_01.web
5. {
6.     public class TestModel
7.     {
8.         // la liste des formations
9.         public List<Formation> Formations { get; set; }
10.        // le msg d'erreur
11.        public string Msg { get; set; }
12.    }
13. }

```

### 3.6.7 Le patron des vues



La page [\_Layout.cshtml] encapsule toutes les vues affichées par l'application. Son code est le suivant :

```

1.  <!DOCTYPE html>
2.  <html>
3.  <head>
4.      <meta charset="utf-8" />
5.      <meta name="viewport" content="width=device-width, initial-scale=1.0">
6.      <title>Gestion de bilans de formation</title>
7.      <link href("~/Content/Site.css" rel="stylesheet" type="text/css" />
8.      <link href("~/Content/bootstrap.min.css" rel="stylesheet" type="text/css" />
9.      <script src "~/Scripts/modernizr-2.6.2.js"></script>
10.     </head>
11.     <body>
12.         <div class="container body-content">
13.             <h1>Gestion de bilans de formation</h1>
14.             <span style="width: 20px">
15.                 <img id="loading" style="display: none" src "~/Content/images/indicator.gif" />
16.             </span>
17.             @using (Html.BeginForm("Action", "Bilans", FormMethod.Post, new { id = "formulaire" }))
18.             {
19.                 <div id="content">
20.                     @RenderBody()
21.                 </div>
22.             }
23.             <hr />
24.             <footer>
25.                 <p>EIS AGI - Février 2017 - ASP.NET MVC</p>
26.             </footer>
27.         </div>
28.         <!-- scripts JS de l'application--&gt;
29.         &lt;script src "~/Scripts/jquery-1.10.2.min.js"&gt;&lt;/script&gt;
30.         &lt;script src "~/Scripts/bootstrap.min.js"&gt;&lt;/script&gt;
31.         &lt;script src "~/Scripts/bilans.js"&gt;&lt;/script&gt;
32.     &lt;/body&gt;
33. &lt;/html&gt;</pre>

```

- la ligne 20 affiche la vue demandée. Celle-ci s'inscrit donc dans une page qui l'englobe et qui permet d'avoir la même mise en page pour toutes les vues ;
- ligne 13 : ce titre sera sur toutes les vues ;
- lignes 14-16 : une image que l'on utilisera lorsqu'on fera des appels Ajax. On notera son identifiant [loading] ;
- lignes 17-22 : un formulaire. Toutes les vues de cette application sont à l'intérieur de ce formulaire. On notera son identifiant [formulaire] (ligne 17). Il nous sera nécessaire pour les appels Ajax ;
- lignes 19-21 : une zone identifiée par [content]. C'est dans cette zone que s'afficheront toutes les vues ;
- lignes 24-26 : un pied de page qui sera lui aussi sur toutes les vues de l'application ;
- ligne 29 : la bibliothèque Javascript jQuery ;
- ligne 31 : nous mettrons le code Javascript des appels Ajax dans [bilans.js] ;

Au final, la page [\_Layout.cshtml] génère les éléments suivants (surlignés) dans la page de test :

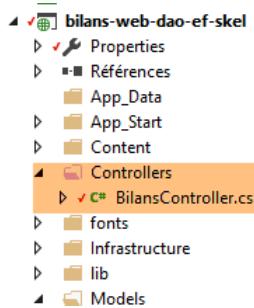
Gestion de bilans de formation

Liste des formations :

- Form0
- Form1

EI5 AGI - Février 2017 - ASP.NET MVC

### 3.6.8 L'action [Test]



La contrôleur [BilansController] est l'unique contrôleur de l'application. Pour l'instant, son code est le suivant :

```

1. using bilans_01.ef;
2. using System;
3. using System.Collections.Generic;
4. using System.Web.Mvc;
5. using System.Linq;
6.
7. namespace bilans_01.web
8. {
9.     public class BilansController : Controller
10.    {
11.
12.        // GET: Bilans
13.        [HttpGet]
14.        public ViewResult Test(ApplicationModel application)
15.        {
16.            // on récupère la liste des formations
17.            List<Formation> formations = null;
18.            string msg = null;
19.            try
20.            {
21.                formations = application.GetAllShortFormations();
22.            }
23.            catch (Exception e)
24.            {
25.                msg = e.ToString();
26.            }
27.            // affichage de la vue avec son modèle
28.            return View("Test", new TestModel { Formations = formations, Msg = msg });
29.        }
30.    }
31. }
```

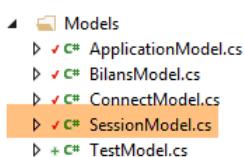
- ligne 9 : la classe [BilansController] étend la classe [System.Web.Mvc.Controller], ce qui est obligatoire ;
- ligne 14 : l'action [Test] qui est exécutée lorsqu'on interroge l'application avec une URL vide ou avec l'URL [/Test]. Elle admet pour paramètre une instance de la classe [ApplicationModel] que nous avons présentée. Grâce aux [ModelBinders] initialisés dans [Global.asax], ce paramètre est initialisé avec l'unique instance [ApplicationModel] créée au démarrage de l'application ;

- ligne 21 : on utilise le paramètre [application] pour récupérer toutes les formations de la base de données ;
- ligne 25 : en cas d'erreur, on enregistre le message d'erreur ;
- ligne 28 : on fait afficher la page [Test.cshtml] en lui passant le modèle [TestModel] qu'elle attend ;

A chaque fois que vous aurez une nouvelle vue à définir, vous suivrez les mêmes étapes que précédemment :

- définir la vue (=page) à afficher et son modèle ;
- définir l'action du contrôleur qui va faire afficher cette vue ;

### 3.6.9 Le bean [SessionModel]



La classe [SessionModel] est un modèle de portée [Session]. Cela signifie :

- qu'il est créé à l'arrivée d'un nouvel utilisateur ;
- qu'il permet de mémoriser des informations produites par les actions du contrôleur traitant les demandes de cet utilisateur ;
- qu'il est supprimé au bout d'un certain temps d'inactivité de cet utilisateur, typiquement une vingtaine de minutes ;

Son code est le suivant :

```

1. using bilans_01.ef;
2. using System.Collections.Generic;
3.
4. namespace bilans_01.web
5. {
6.     public class SessionModel
7.     {
8.         // l'utilisateur connecté
9.         public Utilisateur Utilisateur { get; set; }
10.        // formation de l'utilisateur
11.        public Formation Formation { get; set; }
12.        // cours de la formation
13.        public List<Cours> Cours { get; set; }
14.        // bilans de l'utilisateur pour la formation
15.        public List<Bilan> Bilans { get; set; }
16.        // bilan actuellement affiché
17.        public Bilan Bilan { get; set; }
18.        // cours actuellement sélectionné
19.        public Cours LeCours { get; set; }
20.        // note de la formation
21.        public double NoteFormation;
22.    }
23. }
```

- lignes 8-21 : les informations mémorisées pour un utilisateur donné ;

La classe [SessionModel] joue un rôle central dans cet exercice. Elle permet de se souvenir des choix faits par l'utilisateur connecté, au fil de ses actions sur les pages de l'application. Vous êtes libre d'ajouter ou de supprimer des informations de cette classe.

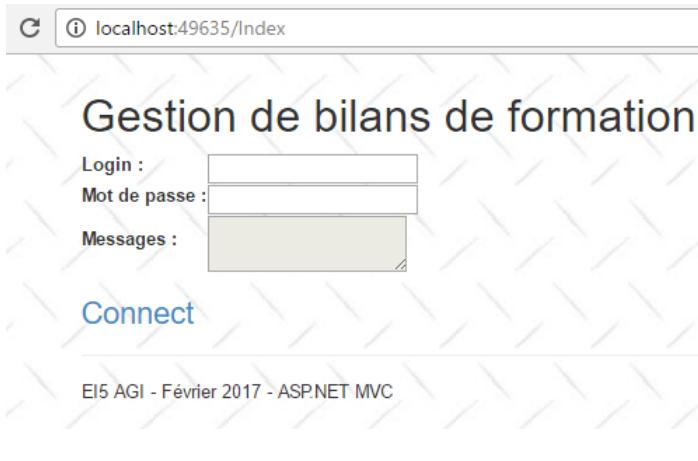
## 3.7 Implémentation de l'application

Nous allons construire l'application étape par étape. A chaque fois, il s'agit de modifier / créer :

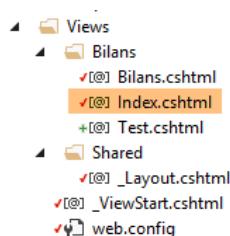
- la vue (=page) à afficher et son modèle ;
- l'action du contrôleur qui va faire afficher cette vue ;
- le script JS qui va permettre d'interroger le serveur web avec un appel Ajax ;

### 3.7.1 Étape 1

La page [Index.cshtml] permet de se connecter à l'application :



Son code est le suivant :



```

1. @model bilans_01.web.IndexModel
2.
3. <!DOCTYPE html>
4.
5. <html>
6. <head>
7.   <meta name="viewport" content="width=device-width" />
8.   <title>Connect</title>
9. </head>
10. <body>
11.   <table>
12.     <tbody>
13.       <tr>
14.         <td>@Html.Label("Login : ")</td>
15.         <td><input type="text" value="@Model.Login" name="Login" id="Login" /></td>
16.         <td style="color:red">@Html.ValidationMessage("Login")</td>
17.       </tr>
18.       <tr>
19.         <td>@Html.Label("Mot de passe : ")</td>
20.         <td><input type="password" value="@Model.Password" name="Password" id="Password" /></td>
21.       </tr>
22.       <tr>
23.         <td>@Html.Label("Messages: ")</td>
24.         <td width="300px"><textarea id="Messages" name="Messages" disabled="disabled" size="80">@Model.Messages</textarea></td>
25.       </tr>
26.     </tbody>
27.   </table>
28.   <h3>
29.     <a href="javascript:connect()">Connect</a>
30.   </h3>
31. </body>
32. </html>
  
```

- ligne 1 : le modèle de la page est la classe [IndexModel] ;
- ligne 15 : saisie du login. Cette saisie est associée au champ [Login] du modèle. Pour cela, les attributs [name] et [id] doivent porter le nom exact de ce champ ;

- ligne 16 : message d'erreur associé au champ [Login] du modèle. Ce message apparaît lorsque l'utilisateur laisse le champ [Login] vide ;
- ligne 20 : saisie du mot de passe. Cette saisie est associée au champ [Password] du modèle. Pour cela, les attributs [name] et [id] doivent porter le nom exact de ce champ. L'utilisateur peut laisser ce champ vide. Aussi n'a-t-on pas prévu de message d'erreur ;
- ligne 24 : la zone multi-lignes qui affiche un éventuel message d'erreur. C'est le cas par exemple de l'utilisateur qui se connecte et qui n'est pas reconnu ;
- ligne 29 : lorsque l'utilisateur clique sur le lien [Connect], la fonction Javascript [connect] est exécutée. Celle-ci se trouve dans le fichier [bilans.js] présenté au paragraphe 3.6.7, page 367 ;

Le modèle [IndexModel] associé à la page [Index.cshtml] est le suivant :

```
▲ Models
  ▷ ✓ C# ApplicationModel.cs
  ▷ ✓ C# BilansModel.cs
  ▷ ✓ C# IndexModel.cs
  ▷ ✓ C# SessionModel.cs
  ▷ + C# TestModel.cs
```

```
1. using System.ComponentModel.DataAnnotations;
2. using System.Web.Mvc;
3.
4. namespace bilans_01.web
5. {
6.     [Bind(Exclude = "Application,Messages")]
7.     public class IndexModel
8.     {
9.
10.        // données de portée application
11.        public ApplicationModel Application { get; set; }
12.
13.        // valeurs postées
14.        [Required(ErrorMessage = "Le login est requis !")]
15.        public string Login { get; set; }
16.        public string Password { get; set; }
17.
18.        // messages d'erreur
19.        public string Messages { get; set; }
20.    }
21. }
```

Lorsqu'on demande l'URL [/Index], la vue présentée est la suivante :

Gestion de bilans de formation

Login :

Mot de passe :

Messages :

Connect

EI5 AGI - Février 2017 - ASP.NET MVC

```
▷ Scripts
  ▷ bilans.js [2]
  ▷ bootstrap.js
  ▷ bootstrap.min.js
  ▷ jquery-1.10.2.intellisense.js
  ▷ jquery-1.10.2.js
  ▷ jquery-1.10.2.min.js
  ▷ jquery-1.10.2.min.map
  ▷ jquery.validate-vsdoc.js
  ▷ jquery.validate.js
  ▷ jquery.validate.min.js
  ▷ jquery.validate.unobtrusive.js
  ▷ jquery.validate.unobtrusive.min.js
  ▷ modernizr-2.6.2.js
▷ Views
```

Lorsqu'on clique sur le lien [Connect] ci-dessus, la fonction JS [connect] du fichier [bilans.js] [2] est exécutée :

```
1. function connect() {
2.     // on récupère des références
3.     var formulaire = $("#formulaire");
```

```

4.  var content = $("#content");
5.  var loading = $("#loading");
6.  // on fait un appel Ajax à la main
7.  var data = formulaire.serialize();
8.  loading.show();
9.  $.ajax({
10.    url: '/Connect',
11.    type: 'POST',
12.    data: data,
13.    dataType: 'html',
14.    success: function (data) {
15.      // affichage résultats
16.      content.html(data);
17.      loading.hide();
18.    },
19.    error: function (jqXHR) {
20.      // affichage erreur
21.      content.html(jqXHR.responseText);
22.      loading.hide();
23.    },
24.  });
25. }

```

- ligne 3 : on récupère la référence de la zone [formulaire] de la vue. Nous avons vu (cf paragraphe 3.6.7) que cet identifiant désignait un formulaire HTML ;
- ligne 5 : on récupère la référence de la zone [content] de la vue. Nous avons vu (cf paragraphe 3.6.7) que cet identifiant désignait la zone du patron dans laquelle sont affichées toutes les vues ;
- ligne 5 : on récupère la référence de la zone [loading] de la vue. Nous avons vu (cf paragraphe 3.6.7) que cet identifiant désignait une image ;
- ligne 7 : les entrées HTML du formulaire sont sérialisées afin d'être envoyées au serveur par une opération POST ;
- ligne 8 : l'image d'attente est affichée ;
- lignes 9-24 : un appel Ajax est fait vers le serveur ;
- ligne 10 : l'URL [/Connect] est demandée par un POST (ligne 11). Nous savons que cette URL sera traitée par la méthode [Connect] du contrôleur [BilansController] ;
- ligne 12 : les valeurs postées ;
- ligne 13 : la nature de la réponse du serveur. C'est du HTML ;
- lignes 14-17 : code exécuté en cas de réussite de la requête ;
- ligne 16 : la réponse HTML du serveur est mise dans la zone [content] ;
- ligne 17 : arrêt de l'animation d'attente ;
- lignes 19-23 : code exécuté en cas d'échec de la requête (absence ou plantage du serveur en général) ;
- ligne 21 : la réponse HTML du serveur est mise dans la zone [content] ;
- ligne 22 : arrêt de l'animation d'attente ;

La méthode [Connect] du contrôleur [BilansController] est pour l'instant la suivante :

```

1.  [HttpPost]
2.  public PartialViewResult Connect(ApplicationModel application, SessionModel session,
   FormCollection post)
3.  {
4.    // on récupère les valeurs postées
5.    IndexModel indexModel = new IndexModel() { };
6.    TryUpdateModel(indexModel, post);
7.    // des erreurs ?
8.    if (!ModelState.IsValid)
9.    {
10.      // on retourne le formulaire
11.      return PartialView("Index", new IndexModel());
12.    }
13.    // on renvoie les valeurs reçues
14.    indexModel.Messages = string.Format("Valeurs saisies : Login=[{0}], Password=[{1}]",
   indexModel.Login, indexModel.Password);
15.    return PartialView("Index", indexModel);
16.  }

```

- ligne 1 : la méthode gère uniquement un POST ;
- ligne 2 : elle reçoit les paramètres suivants :
  - le singleton [ApplicationModel] qui permet à la méthode d'avoir accès aux méthodes de l'interface [IDao] permettant l'accès à la base de données ;
  - le singleton [SessionModel] qui est la mémoire de l'utilisateur ;

- les valeurs postées encapsulées dans [FormCollection post] ;

Voici deux exemples. Dans les deux cas, l'URL initialement appelée est [/Index] :

## Gestion de bilans de formation

Login :

Mot de passe :

Messages :

Connect

EI5 AGI - Février 2017 - ASP.NET MVC

## Gestion de bilans de formation

Login :

Mot de passe :

Messages :

Le login est requis !

Connect

EI5 AGI - Février 2017 - ASP.NET MVC

## Gestion de bilans de formation

Login :

Mot de passe :

Messages :

Connect

EI5 AGI - Février 2017 - ASP.NET MVC

## Gestion de bilans de formation

Login :

Mot de passe :

Messages :

Connect

EI5 AGI - Février 2017 - ASP.NET MVC

---

**Question 1 :** compléter la méthode [/Index] du contrôleur [BilansController] pour avoir le comportement décrit par les copies d'écran suivantes.

Voici des exemples d'erreur de connexion :

## Gestion de bilans de formation

Login :

Mot de passe :

Messages :

Connect

EI5 AGI - Février 2017 - ASP.NET MVC

## Gestion de bilans de formation

Login :

Mot de passe :

Messages : Vous n'avez pas été reconnu

Connect

EI5 AGI - Février 2017 - ASP.NET MVC

## Gestion de bilans de formation

Login : Form0-Prof  
Mot de passe : .....  
Messages :

[Connect](#)

EI5 AGI - Février 2017 - ASP.NET MVC

## Gestion de bilans de formation

Login : Form0-Prof  
Mot de passe : .....  
Messages : Pour l'instant, seuls les étudiants peuvent se connecter

[Connect](#)

EI5 AGI - Février 2017 - ASP.NET MVC

Ci-dessus, l'utilisateur qui se connecte est un enseignant, c'est à dire un utilisateur avec le champ [role=Utilisateur.UtilisateurType.PROF]. Il est refusé car seuls les étudiants peuvent se connecter à l'application.

Voici un exemple de réussite :

## Gestion de bilans de formation

Login : Form0-Etudiant  
Mot de passe : .....  
Messages :

[Connect](#)

EI5 AGI - Février 2017 - ASP.NET MVC

## Gestion de bilans de formation

Etudiant : Form0-Etudiant

[Disconnect](#)

EI5 AGI - Février 2017 - ASP.NET MVC

- en [1], c'est la page [Bilans.cshtml] qui a été affichée ;

Il y a deux étudiants de logins [Form0-Etudiant] et [Form1-Etudiant] (cf base de données). Leurs mots de passe sont égaux à leurs logins. Pour vous connecter, utilisez par exemple les codes *Form0-Etudiant* / *Form0-Etudiant* qui désignent l'unique étudiant de la formation [Form0].

### Conseils et contraintes :

- utilisez la méthode [GetLongUtilisateurByCredentials(login, password)] de la classe [ApplicationModel] qui :
  - lance une exception si un problème survient (erreur de connexion à la base de données...) ;
  - rend un pointeur *null* si l'utilisateur n'a pas pu être identifié ;
  - rend sinon, l'utilisateur identifié par le login et le mot de passe passés en paramètres ;
- lorsque la connexion réussit, l'utilisateur connecté doit être mis en session dans [SessionModel.Utilisateur] ;
- la page [Bilans.cshtml] attend un modèle de type [BilansModel]. Elle va chercher le nom de l'utilisateur dans la session (ligne 6 ci-dessous) :

```
1. @model bilans_01.web.BilansModel
2. <table>
3.   <tbody>
4.     <tr>
5.       <td>@Html.Label("Etudiant : ")</td>
6.       <td>@Model.Session.Utilisateur.Nom</td>
7.     </tr>
8.   </tbody>
9. </table>
10.
11. <h3>
12.   <a id="lnkShowBilan" href="javascript:disConnect()">
13.     Disconnect
14.   </a>
15. </h3>
```

Pour afficher la page [Bilans.cshtml], vous procédez de la façon suivante dans la méthode [Connect] du contrôleur :

```
// on affiche la page [Bilans]
BilansModel bilansModel = new BilansModel() { Session = session };
return PartialView("Bilans", bilansModel);
```

### 3.7.2 Étape 2

La page [Bilans.cshtml] ci-dessus a pour modèle la classe [BilansModel] (ligne 1 ci-dessus) suivante :

```
► └ Infrastructure
  ┌ Models
    ┌─► ApplicationModel.cs
    ┌─► BilansModel.cs
    ┌─► ConnectModel.cs
    ┌─► SessionModel.cs
  └ Scripts
```

```
1. using System.ComponentModel.DataAnnotations;
2. using System.Web.Mvc;
3.
4. namespace bilans_01.web
5. {
6.     [Bind(Exclude = "Session,Messages,NoteFormation")]
7.     public class BilansModel
8.     {
9.         public SessionModel Session { get; set; }
10.
11.        // valeurs postées
12.        [Required]
13.        public long CoursId { get; set; }
14.        [Range(0, 10)]
15.        public int Note { get; set; }
16.        public string Plus { get; set; }
17.        public string Moins { get; set; }
18.        public string Messages { get; set; }
19.        public double NoteFormation { get; set; }
20.    }
21. }
```

- ligne 9 : la session. Nous venons de l'utiliser dans la question précédente ;
- ligne 13 : le n° du cours actuellement sélectionné dans la liste de cours de la page [Bilans.cshtml] ;
- lignes 15-17 : les champs [Note, Plus, Moins] affichés par le bilan dans la page [Bilans.cshtml]. On rappelle que la classe [Bilan] a été présentée au paragraphe 3.5.2, page 352 ;
- ligne 18 : un champ [Messages] pour afficher d'éventuels messages d'erreur ;
- ligne 19 : la note de la formation également affichée par la page [Bilans.cshtml] ;

**Question 2 :** modifiez l'application (page [Bilans.cshtml], méthode [Connect]) pour obtenir la page suivante après la connexion.

### Gestion de bilans de formation

Login : Form0-Etudiant  
 Mot de passe :   
 Messages :

Connect

EI5 AGI - Février 2017 - ASP.NET MVC

### Gestion de bilans de formation

Etudiant : Form0-Etudiant  
 Formation: Form0 1

Disconnect

EI5 AGI - Février 2017 - ASP.NET MVC

- en [1], la formation de l'étudiant connecté ;

#### Conseils et contraintes :

- l'utilisateur mis en session a été obtenu par la méthode [GetLongUtilisateurByCredentials(login, password)]. Il contient donc la formation ;

### 3.7.3 Étape 3

Question 3 : modifiez l'application (page [bilans.xhtml], méthode [Connect]) pour obtenir la page suivante après la connexion.

#### Gestion de bilans de formation

Etudiant : Form0-Etudiant

Formation:Form0

Cours: Form0-Module0-Cours0 ▾

Disconnect

EI5 AGI - Février 2017 - ASP.NET MVC

#### Gestion de bilans de formation

Etudiant : Form0-Etudiant

Formation:Form0

Cours: Form0-Module0-Cours0 ▾  
Form0-Module0-Cours0  
Form0-Module0-Cours1  
Form0-Module1-Cours0  
Form0-Module1-Cours1  
Form0-Module2-Cours0  
Form0-Module2-Cours1

Disconnect

1

EI5 AGI - Fe

- en [1], tous les cours de la formation. Ce sont ces cours que l'étudiant va évaluer ;

#### Conseils et contraintes :

- utilisez la méthode [application.GetShortCoursByFormationId] pour obtenir tous les cours de la formation de l'étudiant. Le paramètre de la méthode est le n° [Id] de la formation ;
- la liste de cours sera mise dans la session et c'est là qu'ira la chercher la page [Bilans.cshtml] ;
- la liste déroulante sera une liste HTML où la valeur des options de la liste sera le n° [Id] des cours et le libellé des options sera le nom du cours. La valeur de la liste déroulante sera associée au champ [BilansModel.CoursId]. Cela signifie que lorsque la page sera postée par l'un des liens qu'on va y mettre, le n° [Id] du cours sélectionné sera affecté au champ [BilansModel.CoursId] ;

### 3.7.4 Étape 4

Question 4 : Modifiez la méthode [Connect] pour qu'à son issue, tous les bilans déjà faits (çà-d présents en base de données) par l'utilisateur connecté, sur les cours de sa formation, soient en session. Cette liste vous sera utile pour gérer les bilans de l'étudiant.

#### Conseils et contraintes :

- utilisez la méthode [application.GetShortBilansByEtudiantId] qui admet pour paramètre l'identifiant de l'étudiant dont on cherche les bilans ;
- cette liste de bilans sera mise en session ;

### 3.7.5 Étape 5

Question 5 : modifiez l'application (page [Bilans.cshtml], méthode [Connect]) pour obtenir la page suivante après la connexion.

# Gestion de bilans de formation

Etudiant : Form0-Etudiant

Formation:Form0

Cours: Form0-Module0-Cours0 ▾

1

Show Save Delete 3

Note : 0

Plus :

Moin 2

Messages:

Disconnect

EI5 AGI - Février 2017 - ASP.NET MVC

- en [2], on affiche le bilan du 1er cours [1]. Cela revient à afficher pour ce cours :
  - la note [Note] dans une boîte de saisie (HTML <input type='text' ...>) ;
  - les champ [Plus] et [Moins], [Messages] dans des boîtes de saisie multi-lignes (Html <textarea>) ;

Si le 1er cours n'a pas de bilan, on affichera un bilan vierge : 0 pour [Note] et des chaînes vides pour [Plus] et [Moins]. C'est ce qui a été fait sur la copie d'écran ci-dessus ;

- en [3], des liens pour afficher, sauvegarder ou supprimer le bilan actuellement affiché ;

## Conseils et contraintes :

- attention, le bilan du 1er cours affiché en [1] n'est pas nécessairement le 1er bilan de la liste des bilans mémorisée à l'étape précédente ;
- mémorisez dans la session le bilan affiché, avec toutes ses propriétés [note, plus, moins] mais aussi [cours, courseId, utilisateur, userId] (cf définition de la classe [Bilan] au paragraphe 3.5.2) ;
- utilisez une méthode séparée pour afficher le bilan car on en a besoin à d'autres moments ;

Pour vérifier votre travail, faites le test suivant :

- videz la base de données avec le script SQL [cleandatabase] ;
- exécutez l'application ;
- le bilan affiché pour le cours [Form0-Module0-Cours0] doit être vide et il ne doit pas y avoir d'exception ;

## 3.7.6 Étape 6

**Question 6 :** modifiez l'application (contrôleur [BilansController]) pour gérer le lien [Show] qui affiche le bilan du cours sélectionné par l'utilisateur en [1].

## Gestion de bilans de formation

Etudiant : Form0-Etudiant

Formation:Form0

Cours: Form0-Module2-Cours0 ▾

Show Save Delete

Note :

Plus :

Moins :

Messages:

Disconnect

EI5 AGI - Février 2017 - ASP.NET MVC

## Gestion de bilans de formation

Etudiant : Form0-Etudiant

Formation:Form0

Cours: Form0-Module2-Cours0 ▾

3

Show Save Delete

Note :

Plus :

Moins :

Messages:

2

Disconnect

EI5 AGI - Février 2017 - ASP.NET MVC

### Conseils et contraintes :

- lorsqu'on clique sur le lien [Show] un appel Ajax doit être fait au serveur. Pour cela suivez l'exemple de la page 372 et insérez votre nouvelle fonction dans le fichier [bilans.js]. Cette fonction fera un POST vers une nouvelle méthode du contrôleur [BilansController] que vous aurez également à écrire ;
- en [2], le nouveau bilan. En [3], on remarquera que la liste déroulante est positionnée sur le cours sélectionné en [1] ;
- s'il n'existe pas encore de bilan pour le cours sélectionné, on affichera un bilan vierge : 0 pour [note] et des chaînes vides pour [plus] et [moins] ;
- on exploitera la liste des bilans présente en session ;
- le nouveau bilan affiché sera mis en session ;
- le nouveau cours sélectionné sera mis en session ;

Pour vérifier votre travail, faites le test suivant :

- remplissez la base de données avec le script SQL [filldatabase] ;
- exécutez l'application ;
- avec le lien [Bilan], faites afficher les différents bilans et vérifiez qu'ils correspondent bien au cours affiché dans la liste déroulante. Les valeurs [xyz] des champs [Plus, Moins] reprennent les indices xyz du cours [Formx-Moduley-Coursz]. Il doit y avoir deux bilans [020, 021]. Les autres doivent être vides ;

### 3.7.7 Étape 7

**Question 7 :** modifiez l'application (contrôleur [BilansController]) pour gérer le lien [Save] qui sauvegarde en base le bilan affiché et éventuellement modifié par l'utilisateur.

## Gestion de bilans de formation

Etudiant : Form0-Etudiant

Formation:Form0

Cours: Form0-Module0-Cours0 ▾

Show Save Delete

Note :	6
Plus :	000 - moyen
Moins :	000 - trop de PPT et pas assez de pratique
Messages:	

Disconnect

EI5 AGI - Février 2017 - ASP.NET MVC

## Gestion de bilans de formation

Etudiant : Form0-Etudiant

Formation:Form0

Cours: Form0-Module0-Cours0 ▾

Show Save Delete

Note :	6
Plus :	000 - moyen
Moins :	000 - trop de PPT et pas assez de pratique
Messages:	Bilan sauvegardé

Disconnect

1

EI5 AGI - Février 2017 - ASP.NET MVC

- en [1], un message indique que la sauvegarde a réussi. Il faut aller dans la base de données vérifier l'ajout ou la modification dans la table [BILANS] :

ID	MOINS	NOTE	PLUS	VERSIONING	COURS_ID	ETUDIANT_ID
268	020	4	020	0	170	80
269	021	7	021	0	171	80
288	000 - trop de PPT et pas assez de pratique	6	000 - moyen	3	166	80

Maintenant, on arrête le SGBD et on essaie de refaire une seconde sauvegarde. On obtient alors le résultat suivant :

# Gestion de bilans de formation

Etudiant : Form0-Etudiant  
Formation:Form0  
Cours: Form0-Module0-Cours1 ▾

## Show Save Delete

Note :

Plus : 001 - trop bien

Moins : 001 - ras

System.Data.Entity.Core.EntityException: The underlying provider failed on Open. -->  
MySQL.Data.MySqlClient.MySqlException: Unable to connect to any of the specified MySQL hosts.  
    à MySql.Data.MySqlClient.NativeDriver.Open()  
    à MySql.Data.MySqlClient.Driver.Open()  
    à MySql.Data.MySqlClient.Driver.Create(MySqlConnectionStringBuilder settings)  
    à MySql.Data.MySqlClient.MySqlPool.GetPooledConnection()  
    à MySql.Data.MySqlClient.MySqlPool.TryToGetDriver()  
    à MySql.Data.MySqlClient.MySqlPool.GetConnection()  
    à MySql.Data.MySqlClient.MySqlConnection.Open()  
    à System.Data.Entity.Infrastructure.Interception.DbConnectionDispatcher.<Open>b\_\_36(DbConnection t,  
DbConnectionInterceptionContext c)  
    à System.Data.Entity.Infrastructure.Interception.InternalDisp

Messages:

Disconnect

EI5 AGI - Février 2017 - ASP.NET MVC

### Conseils et contraintes :

- lorsqu'on clique sur le lien [Save] un appel Ajax doit être fait au serveur. Pour cela suivez l'exemple de la page 372 et insérez votre nouvelle fonction dans le fichier [bilans.js]. Cette fonction fera un POST vers une nouvelle méthode du contrôleur [BilansController] que vous aurez également à écrire ;
- utilisez la méthode [application.SaveBilan] qui permet de sauvegarder en base le bilan passé en paramètre. Si celui-ci a un identifiant [id] égal à `null`, c'est une insertion qui sera faite, sinon ce sera une modification. Vous n'avez pas à vous préoccuper de ce point. La méthode rend le nouveau bilan, avec sa clé primaire si une insertion a eu lieu, avec sa nouvelle version si une modification a eu lieu ;
- dans les bilans mis en session, supprimez l'ancien bilan et ajoutez le nouveau ;
- mettez le nouveau bilan en session ;
- l'utilisateur peut choisir un cours dans la liste déroulante sans cliquer ensuite sur le lien [Show]. Ceci fait que le cours actuellement mémorisé dans la session n'est pas celui que l'utilisateur voit dans la liste déroulante. Idem pour le bilan qui est celui du cours courant actuellement en session. Si ensuite l'utilisateur modifie le bilan qu'il voit et clique sur le lien [Save], ce n'est pas le bilan du cours qu'il voit dans la liste déroulante qui va être sauvegardé mais le bilan du cours actuellement en session. Ce cas peut être traité mais on ne le fera pas. **On supposera toujours que lorsque l'utilisateur change de cours dans la liste déroulante, il clique ensuite sur le lien [Show] pour changer le bilan affiché.** Cela signifie que pour nous, le bilan en session qui va être sauvegardé ou supprimé est toujours celui du cours actuellement sélectionné dans la liste déroulante ;

### 3.7.8 Étape 8

Question 8 : modifiez l'application (contrôleur [BilansController]) pour gérer le lien [Delete] qui supprime en base le bilan affiché.

## Gestion de bilans de formation

Etudiant : Form0-Etudiant

Formation:Form0

Cours: Form0-Module0-Cours0 ▾

Show Save Delete

Note :	<input type="text" value="6"/>
Plus :	000 - moyen
Moins :	000 - trop de PPT et pas assez de pratique
Messages:	

Disconnect

EI5 AGI - Février 2017 - ASP.NET MVC

## Gestion de bilans de formation

Etudiant : Form0-Etudiant

Formation:Form0

Cours: Form0-Module0-Cours0 ▾

Show Save Delete

Note :	<input type="text" value="0"/>
Plus :	
Moins :	
Messages:	Bilan supprimé

1

Disconnect

EI5 AGI - Février 2017 - ASP.NET MVC

- en [1], un bilan vide a été affiché. Il faut aller dans la base de données pour vérifier la suppression du bilan de la table [BILANS] :

ID	MOINS	NOTE	PLUS	VERSIONING	COURS_ID	ETUDIANT_ID
268	020	4	020	0	170	80
269	021	7	021	0	171	80

### Conseils et contraintes :

- lorsqu'on clique sur le lien [Delete] un appel Ajax doit être fait au serveur. Pour cela suivez l'exemple de la page 372 et insérez votre nouvelle fonction dans le fichier [bilans.js]. Cette fonction fera un POST vers une nouvelle méthode du contrôleur [BilansController] que vous aurez également à écrire ;
- utiliser la méthode [application.DeleteBilan] pour supprimer un bilan dont on passe l'identifiant ;
- le bilan supprimé doit être retiré de la liste des bilans présente en session ;
- le bilan affiché doit être mis en session avec toutes ses caractéristiques (cf définition de la classe [Bilan] au paragraphe 3.5.2), notamment un identifiant *null* qui provoquera une insertion si l'utilisateur modifie ce bilan puis le sauvegarde ;

Avant de passer à l'étape suivante, faites le test qui suit :

- sélectionnez le cours [Form0-Module2-Cours1] ;
- cliquez sur le lien [Bilan]. Vous allez faire trois opérations successives sur le bilan affiché ;
- modifiez le bilan affiché et sauvegardez-le. Vérifiez ensuite qu'il est présent en base ;
- puis supprimez ce bilan qui vient d'être sauvegardé (sans cliquer sur le lien [bilan]). Vérifiez ensuite qu'il n'est plus présent en base ;
- puis remplissez ce bilan devenu vide et sauvegardez-le de nouveau (toujours sans cliquer sur le lien [bilan]). Vérifiez ensuite qu'il est de nouveau présent en base ;

Faites également le test suivant :

- lancez l'application ;
- supprimez le bilan du 1er cours [Form0-Module0-Cours0] ;
- relancez l'application ;
- le 1er bilan affiché, celui du cours [Form0-Module0-Cours0] doit être vide ;

Cette séquence peut faire apparaître des exceptions si vos algorithmes de gestion des bilans et des cours en session sont incorrects.

### 3.7.9 Étape 9

**Question 9 :** modifiez l'application (page [Bilans.cshtml], contrôleur [BilansController]) pour afficher la note de la formation. Celle-ci est obtenue de la façon suivante :  $note = (n_0 * ects_0 + \dots + n_p * ects_p) / (ects_0 + \dots + ects_p)$  où  $n_i$  est la note du bilan n° i et  $ects_i$  les ECTS du cours associé à ce bilan.

Voici des exemples de ce qui est attendu :

## Gestion de bilans de formation

Etudiant : Form0-Etudiant

Formation:Form0

Cours: Form0-Module0-Cours0 ▾

Show Save Delete

Note :

Plus :

Moins :

Messages:

Note : 6

1

Disconnect

EI5 AGI - Février 2017 - ASP.NET MVC

- la note de la formation est affichée dès le 1er affichage des bilans [1]. Ensuite, à chaque fois qu'on sauvegarde / supprime le bilan affiché, cette note est recalculée ;

## Gestion de bilans de formation

Etudiant : Form0-Etudiant

Formation:Form0

Cours: Form0-Module0-Cours0 ▾

Show Save Delete

Note :

Plus :

Moins :

Messages:

Note : 6

1

Disconnect

EI5 AGI - Février 2017 - ASP.NET MVC

## Gestion de bilans de formation

Etudiant : Form0-Etudiant

Formation:Form0

Cours: Form0-Module0-Cours0 ▾

Note : 6,75

2

Show Save Delete

Note :

Plus :

Moins :

Messages:

9

000 - excellent

000 - ras

Bilan sauvegardé

Disconnect

EI5 AGI - Février 2017 - ASP.NET MVC

- en [2], la note de la formation a changé ;

## Gestion de bilans de formation

Etudiant : Form0-Etudiant

Formation:Form0

Note : 6,75

Cours: Form0-Module0-Cours0 ▾

Show Save Delete

Note :	9
Plus :	000 - excellent
Moins :	000 - ras
Messages:	Bilan sauvegardé

3

Disconnect

EI5 AGI - Février 2017 - ASP.NET MVC

## Gestion de bilans de formation

Etudiant : Form0-Etudiant

Formation:Form0

Note : 6

Cours: Form0-Module0-Cours0 ▾

Show Save Delete

Note :	0
Plus :	
Moins :	
Messages:	Bilan supprimé

4

Disconnect

EI5 AGI - Février 2017 - ASP.NET MVC

- en [4], la note de la formation a changé ;

Pour vérifier votre travail, faites les deux tests suivants :

- remplissez la base avec le script [filddatabase] puis exécutez votre application. Lorsque la page [bilans] est affichée, la note de la formation doit être 6 ;
- remplissez la base avec le script [filddatabase] puis videz la table des bilans. Lorsque la page [bilans] est affichée, la note de la formation doit être -1 (valeur arbitraire pour dénoter une absence de bilans) et votre application ne doit pas planter :

## Gestion de bilans de formation

Etudiant : Form0-Etudiant

Formation:Form0

Note : -1,00

Cours: Form0-Module0-Cours0 ▾

Show Save

Note :	0
Plus :	
Moins :	
Messages:	



Disconnect

EI5 AGI - Février 2017 - ASP.NET MVC

### 3.7.10 Étape 10

Question 10 : modifiez l'application pour gérer la saisie de la note de bilan qui doit être un nombre entier dans l'intervalle [0,10].

Voici un exemple de ce qui est attendu :

## Gestion de bilans de formation

Etudiant : Form0-Etudiant

Formation:Form0

Note : 6

Cours: Form0-Module0-Cours0 ▾

Show Save Delete

Note :  1

Plus :

Moins :

Messages: Bilan supprimé

Disconnect

EI5 AGI - Février 2017 - ASP.NET MVC

## Gestion de bilans de formation

Etudiant : Form0-Etudiant

Formation:Form0

Note : 6

Cours: Form0-Module0-Cours0 ▾

Show Save Delete

Note : 0

Plus :

Moins :

Messages: Données invalides

Disconnect

EI5 AGI - Février 2017 - ASP.NET MVC

### 3.7.11 Étape 11

Question 11 : modifiez l'affichage de la note de la formation afin qu'elle présente deux chiffres après la virgule.

Voici un exemple de ce qui est attendu :

## Gestion de bilans de formation

Etudiant : Form0-Etudiant

Formation:Form0

Note : 6,00

Cours: Form0-Module0-Cours0 ▾

1

Show Save Delete

Note : 0

Plus :

Moins :

Messages:

Disconnect

EI5 AGI - Février 2017 - ASP.NET MVC

### 3.7.12 Étape 12

Question 12 : gérez l'affichage du bilan pour que lorsque le cours sélectionné n'a pas de bilan enregistré en base, le lien [Delete] soit caché.

Voici un exemple de ce qui est attendu :

## Gestion de bilans de formation

Etudiant : Form0-Etudiant

Formation:Form0

Note : 6,75

Cours: Form0-Module0-Cours0 ▾

Show Save Delete

Note :  

Plus : 000 - excellent

Moins : 000 - ras

Messages:

1

Disconnect

EI5 AGI - Février 2017 - ASP.NET MVC

## Gestion de bilans de formation

Etudiant : Form0-Etudiant

Formation:Form0

Note : 6,00

Cours: Form0-Module0-Cours0 ▾

Show Save

Note :  

Plus :

Moins :

Messages: Bilan supprimé

2

Disconnect

EI5 AGI - Février 2017 - ASP.NET MVC

- après la suppression du bilan en [1], le lien [Delete] devient caché [2] ;

### 3.7.13 Étape 13

Question 13 : gérez le lien [Disconnect] de la page [Bilans.cshtml].

Voici un exemple de ce qui est attendu :

## Gestion de bilans de formation

Etudiant : Form0-Etudiant

Formation:Form0

Note : 6,00

Cours: Form0-Module0-Cours0 ▾

1

Show Save

Note :  

Plus :

Moins :

Messages: Bilan supprimé

Disconnect

EI5 AGI - Février 2017 - ASP.NET MVC

## Gestion de bilans de formation

Login :

Mot de passe :

Messages :

Connect

2

EI5 AGI - Février 2017 - ASP.NET MVC

#### Conseils et contraintes :

- lorsqu'on clique sur le lien [Disconnect] un appel Ajax doit être fait au serveur. Pour cela suivez l'exemple de la page 372 et insérez votre nouvelle fonction dans le fichier [bilans.js]. Cette fonction fera un POST vers une nouvelle méthode du contrôleur [BilansController] que vous aurez également à écrire ;

### 3.7.14 Étape 14

Il est techniquement possible pour un hacker d'appeler directement les méthodes qui gèrent les liens de la page [bilans.xhtml] sans passer par la phase d'authentification.

---

**Question 14 :** gérez ce cas.

---

**Conseils et contraintes :**

- toute méthode liée à l'un des liens de la page [Bilans.cshtml] devra, avant de faire quoique ce soit, vérifier que dans la session, il y a bien un utilisateur enregistré et que celui-ci est un étudiant. Si ce n'est pas le cas, la page d'authentification devra être affichée.

## 4 étude de cas n° 3 : application e-commerce

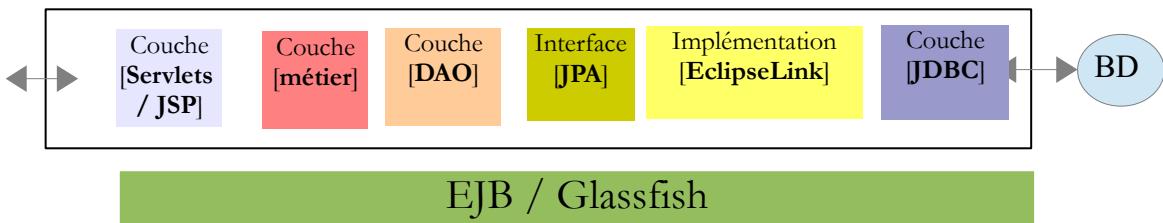
### 4.1 Les compétences mises en oeuvre

Les compétences mises en oeuvre par cet exercice sont les suivantes :

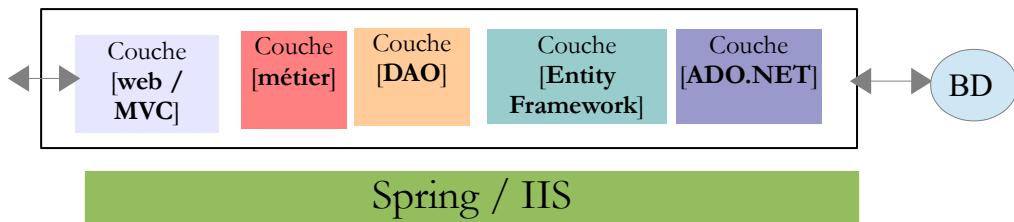
- maîtriser le paradigme MVC ;
- savoir écrire une vue **V** ASP.NET MVC ainsi que son contrôleur **C** et son modèle **M** ;
- connaître et savoir utiliser les différentes portées d'un modèle **M** de l'application web : application, session, request ;
- maîtriser la notion de [ModelBinder] ;
- savoir quels types de paramètres peuvent recevoir les différentes méthodes du contrôleur **C** : des modèles instanciés par des [ModelBinder], des éléments de l'URL d'un GET ou POST, des valeurs postées ;
- comprendre les paramètres de la syntaxe `[Html.BeginForm("méthode", "contrôleur", "méthode HTTP")]` ;
- maîtriser Visual Studio et son environnement de développement et notamment savoir utiliser son débogueur ;
- savoir utiliser WampServer, PhpMyAdmin, MySQL ;

### 4.2 Le problème

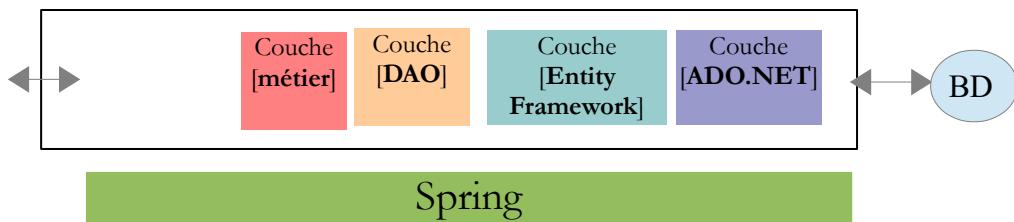
On trouve sur internet un tutoriel d'apprentissage de Java EE sous Netbeans. Il s'agit d'une application web de commerce électronique. Cette application utilise l'architecture à couches suivante :



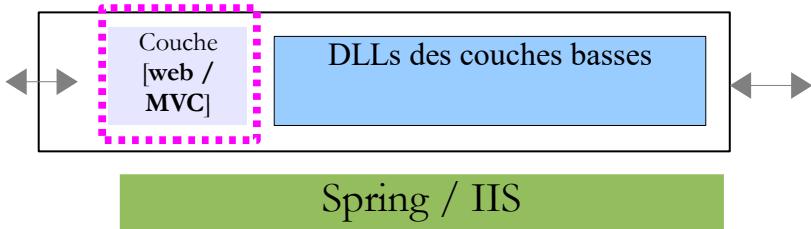
Nous allons porter cette application dans un environnement ASP.NET MVC / IIS :



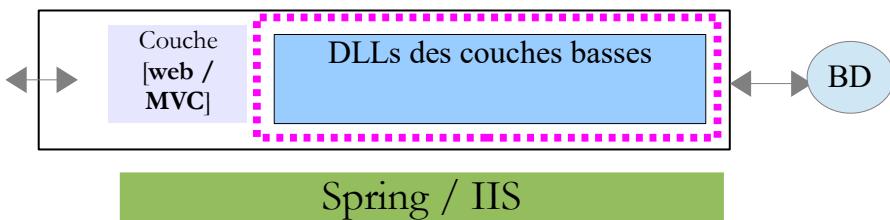
Nous allons diviser cette étude en deux parties. Nous allons d'abord présenter les couches basses de l'application, celles qui ne nécessitent pas la présence du serveur IIS :



puis nous étudierons la couche haute, la couche web / MVC implémentée avec le framework ASP.NET MVC :



## 4.3 Présentation des couches basses



Pour écrire la couche web / MVC, il nous faut connaître deux choses :

- l'interface présentée à la couche web par la DLL des couches basses ;
- la nature des entités échangées entre ces deux couches ;

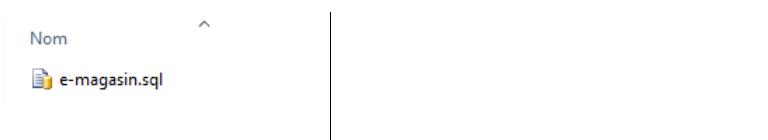
### 4.3.1 Mise en place de l'environnement de travail

Le dossier suivant vous est fourni :

.nuget
afb-webmvc-skel
database
packages
afb-skel-02.sln
afb-skel-02.suo
afb-skel-02.v12.suo

- [database] : contient le script SQL de génération de la base MySQL [e-magasin] ;
- [afb-webmvc-skel] : contient le projet Visual Studio de l'application web que vous avez à construire ;

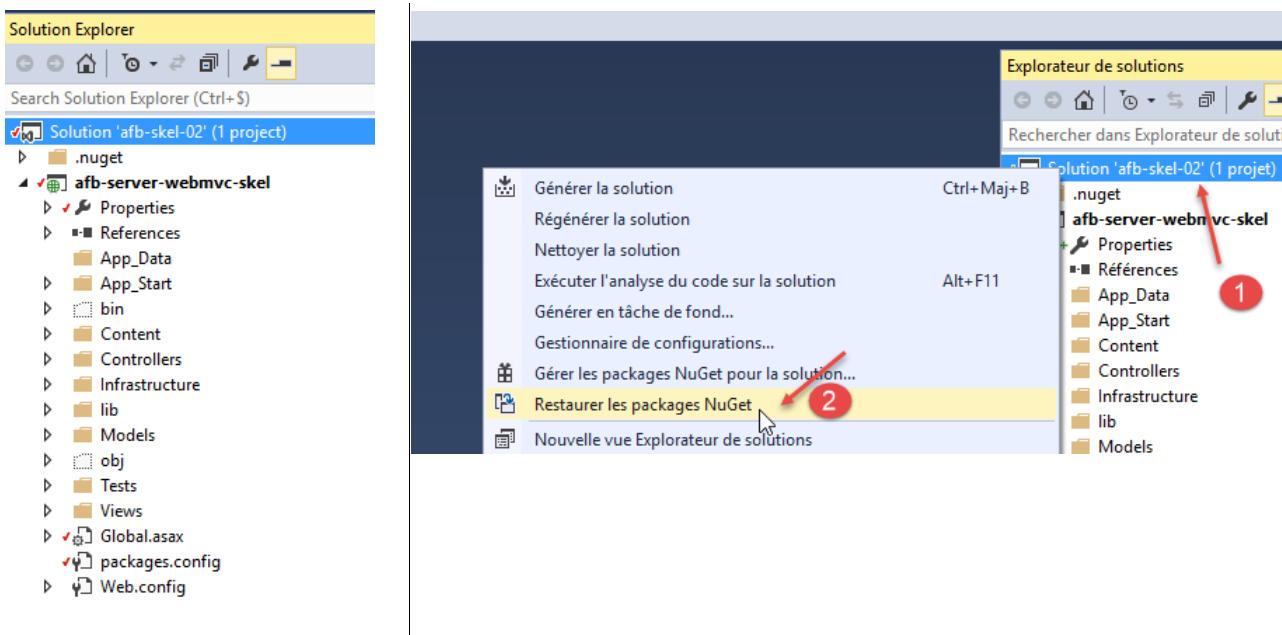
#### Installation de la base de données



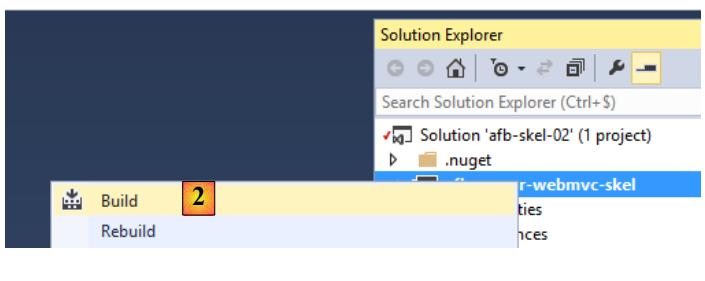
Avec WampServer / PhpMyAdmin, exécutez le script SQL trouvé dans le dossier [database]. Le script crée la base et la remplit avec des données. La base ne doit pas déjà exister.

#### La solution Visual Studio

Chargez dans Visual Studio, la solution [afb-skel-02.sln] du dossier qui vous est donné :

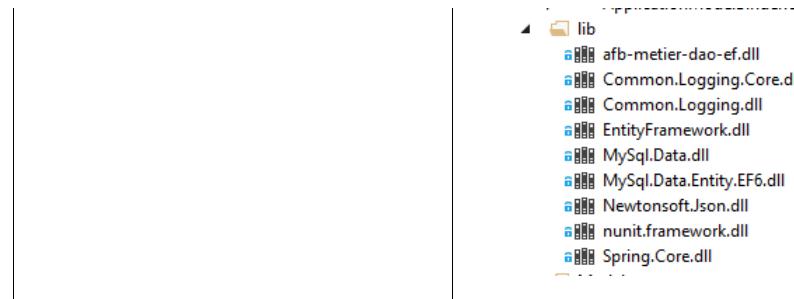


Le projet [afb-server-webmvc-skel] utilise un certain nombre de dépendances qu'on peut installer comme indiqué en [1-2].



Faites ensuite un [Build] sur la solution [2].

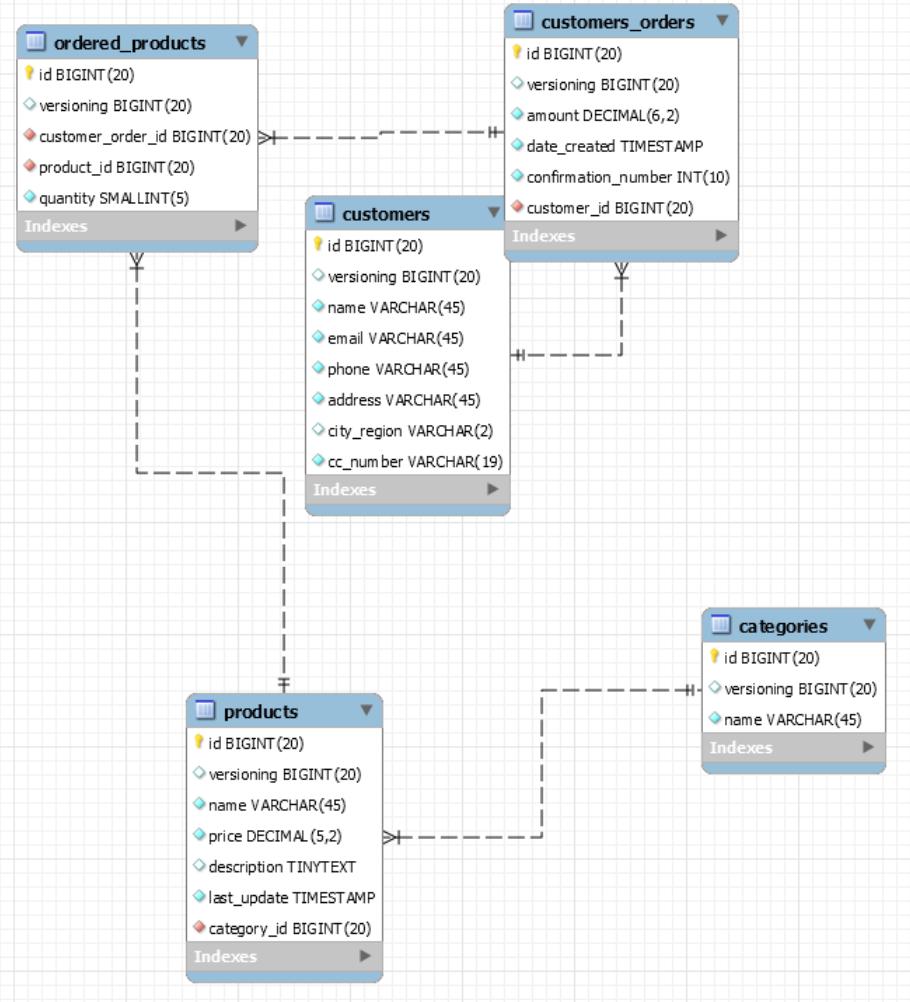
Si cette démarche échoue, alors utilisez le dossier [lib] du projet :



Il contient toutes les DLL nécessaires au projet. Ajoutez-les toutes comme références du projet.

#### 4.3.2 La base de données

La base de données utilisée est une base MySQL nommée "e-magasin". Sa structure est la suivante :



L'application est une application de commerce électronique.

- on y vend des produits enregistrés dans la table [product]. Un produit appartient à une catégorie enregistrée dans la table [category]. Le lien entre les deux tables se fait via la clé étrangère **[product].category\_id** ;
- les trois autres tables [customer, customer\_order, ordered\_product] sont au départ vides ;
- un client qui fait un achat le fait via une page web de confirmation où il entre des informations le concernant. Celles-ci seront enregistrées dans la table [customer]. Sa commande est enregistrée dans la table [customer\_order]. Celle-ci a un lien vers la table [customer] via la clé étrangère **[customer\_order].customer\_id**. Chaque produit acheté fait l'objet d'un enregistrement dans la table [ordered\_product]. La clé étrangère **[ordered\_product].product\_id** fait un lien vers le produit acheté, la clé étrangère **[ordered\_product].customer\_order\_id** faisant elle un lien vers la table [customer\_order] et donc vers la table [customer].

La définition des tables est la suivante :

```

1. SET FOREIGN_KEY_CHECKS=0;
2.
3. CREATE DATABASE `e-magasin`
4.   CHARACTER SET 'utf8'
5.   COLLATE 'utf8_unicode_ci';
6.
7. USE `e-magasin`;
8.
9. #
10. # Structure for the `categories` table :
11. #
12.
13. CREATE TABLE `categories` (
14.   `id` BIGINT(20) NOT NULL AUTO_INCREMENT,
15.   `versioning` BIGINT(20) DEFAULT 1,
16.   `name` VARCHAR(45) COLLATE utf8_unicode_ci NOT NULL DEFAULT '1',
17.   PRIMARY KEY (`id`) USING BTREE,
18.   UNIQUE KEY `name` (`name`) USING BTREE

```

```

19. ) ENGINE=InnoDB
20. AUTO_INCREMENT=5 CHARACTER SET 'utf8' COLLATE 'utf8_unicode_ci'
21. COMMENT='contains product categories, e.g., dairy, meats, etc.'
22. ;
23.
24. #
25. # Structure for the `customers` table :
26. #
27.
28. CREATE TABLE `customers` (
29.   `id` BIGINT(20) NOT NULL AUTO_INCREMENT,
30.   `versioning` BIGINT(20) DEFAULT 1,
31.   `name` VARCHAR(45) COLLATE utf8_unicode_ci NOT NULL,
32.   `email` VARCHAR(45) COLLATE utf8_unicode_ci NOT NULL,
33.   `phone` VARCHAR(45) COLLATE utf8_unicode_ci NOT NULL,
34.   `address` VARCHAR(45) COLLATE utf8_unicode_ci NOT NULL,
35.   `city_region` VARCHAR(2) COLLATE utf8_unicode_ci DEFAULT NULL,
36.   `cc_number` VARCHAR(19) COLLATE utf8_unicode_ci NOT NULL,
37.   PRIMARY KEY (`id`) USING BTREE,
38.   UNIQUE KEY `name`(`name`) USING BTREE,
39.   UNIQUE KEY `cc_number`(`cc_number`) USING BTREE
40. ) ENGINE=InnoDB
41. AUTO_INCREMENT=42 CHARACTER SET 'utf8' COLLATE 'utf8_unicode_ci'
42. COMMENT='maintains customer details'
43. ;
44.
45. #
46. # Structure for the `customers_orders` table :
47. #
48.
49. CREATE TABLE `customers_orders` (
50.   `id` BIGINT(20) NOT NULL AUTO_INCREMENT,
51.   `versioning` BIGINT(20) DEFAULT 1,
52.   `amount` DECIMAL(8,2) DEFAULT NULL,
53.   `date_created` TIMESTAMP NOT NULL DEFAULT CURRENT_TIMESTAMP,
54.   `confirmation_number` INTEGER(10) UNSIGNED DEFAULT NULL,
55.   `customer_id` BIGINT(20) NOT NULL,
56.   PRIMARY KEY (`id`) USING BTREE,
57.   KEY `fk_customer_order_customer`(`customer_id`) USING BTREE,
58.   CONSTRAINT `customers_orders_fk1` FOREIGN KEY (`customer_id`) REFERENCES `customers`(`id`) ON DELETE CASCADE ON UPDATE CASCADE
59. ) ENGINE=InnoDB
60. AUTO_INCREMENT=35 CHARACTER SET 'utf8' COLLATE 'utf8_unicode_ci'
61. COMMENT='maintains customer order details'
62. ;
63.
64. #
65. # Structure for the `products` table :
66. #
67.
68. CREATE TABLE `products` (
69.   `id` BIGINT(20) NOT NULL AUTO_INCREMENT,
70.   `versioning` BIGINT(20) DEFAULT 1,
71.   `name` VARCHAR(45) COLLATE utf8_unicode_ci NOT NULL,
72.   `price` DECIMAL(5,2) NOT NULL,
73.   `description` TINYTEXT COLLATE utf8_unicode_ci,
74.   `last_update` TIMESTAMP NOT NULL DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP,
75.   `category_id` BIGINT(20) NOT NULL,
76.   PRIMARY KEY (`id`) USING BTREE,
77.   UNIQUE KEY `name`(`name`) USING BTREE,
78.   KEY `fk_product_category`(`category_id`) USING BTREE,
79.   CONSTRAINT `products_fk1` FOREIGN KEY (`category_id`) REFERENCES `categories`(`id`) ON DELETE CASCADE ON UPDATE CASCADE
80. ) ENGINE=InnoDB
81. AUTO_INCREMENT=21 CHARACTER SET 'utf8' COLLATE 'utf8_unicode_ci'
82. COMMENT='contains product details'
83. ;
84.
85. #
86. # Structure for the `ordered_products` table :
87. #
88.
89. CREATE TABLE `ordered_products` (
90.   `id` BIGINT(20) NOT NULL AUTO_INCREMENT,
91.   `versioning` BIGINT(20) DEFAULT 1,
92.   `customer_order_id` BIGINT(20) NOT NULL,
93.   `product_id` BIGINT(20) NOT NULL,
94.   `quantity` SMALLINT(5) UNSIGNED NOT NULL DEFAULT 1,
95.   PRIMARY KEY (`id`) USING BTREE,
96.   UNIQUE KEY `customer_order_id`(`customer_order_id`, `product_id`) USING BTREE,
97.   KEY `fk_ordered_product_customer_order`(`customer_order_id`) USING BTREE,
98.   KEY `fk_ordered_product_product`(`product_id`) USING BTREE,
99.   CONSTRAINT `ordered_products_fk1` FOREIGN KEY (`customer_order_id`) REFERENCES `customers_orders`(`id`) ON DELETE CASCADE ON UPDATE CASCADE,
100.  CONSTRAINT `ordered_products_fk2` FOREIGN KEY (`product_id`) REFERENCES `products`(`id`) ON DELETE CASCADE ON UPDATE CASCADE
101. ) ENGINE=InnoDB
102. AUTO_INCREMENT=53 CHARACTER SET 'utf8' COLLATE 'utf8_unicode_ci'

```

```

103. COMMENT='matches products with customer orders and records their quantity'
104. ;
105.
106. DELIMITER $$ 
107.
108. CREATE DEFINER = 'root'@'localhost' TRIGGER `categories_versioning_insert` BEFORE INSERT ON `categories`
109.     FOR EACH ROW
110. BEGIN
111.     if NEW.VERSIONING IS NULL THEN
112.         SET NEW.VERSIONING=1;
113.     END IF;
114. END$$
115.
116. CREATE DEFINER = 'root'@'localhost' TRIGGER `categories_versioning_update` BEFORE UPDATE ON `categories`
117.     FOR EACH ROW
118. BEGIN
119.     SET NEW.VERSIONING=OLD.VERSIONING+1;
120. END$$
121.
122. DELIMITER ;
123.
124. ...

```

- on notera que toutes les clés primaires sont en mode [AUTO\_INCREMENT], c-à-d générées par le SGBD ;
- on notera lignes 58, 79, 99, 100 les attributs [ON DELETE CASCADE] et [ON UPDATE CASCADE] sur les clés étrangères ;
- ligne 58, l'attribut [ON DELETE CASCADE] sur la clé étrangère [CustomerOrders.customer\_id --> Customers.id] fait que lorsqu'on supprime un client [Customer], tous les ordres [CustomerOrder] qui dépendent de lui sont également supprimés ;
- ligne 99, l'attribut [ON DELETE CASCADE] sur la clé étrangère [OrderedProducts.customer\_order\_id --> CustomerOrders.id] fait que lorsqu'on supprime une commande [CustomerOrder], tous les produits [OrderedProduct] de cette commande sont également supprimés ;
- au final, les deux règles précédentes font que lorsqu'on supprime un client [Customer], tous les ordres [CustomerOrder] et les produits [OrderedProduct] qui dépendent de lui sont également supprimés. C'est important à comprendre car le code qui suit utilise cette particularité ;

La gestion du versioning des lignes est déléguée à des *triggers*. Les *triggers* sont des programmes embarqués par le SGBD. Ils ne sont pas normalisés, aussi diffèrent-ils d'un SGBD à un autre. Ils peuvent même ne pas exister dans certains SGBD.

- ligne 108 : on définit un code à exécuter avant chaque **insertion** dans la table [CATEGORIES] ;
- lignes 109-113 : le code dit que lorsqu'à l'insertion d'une nouvelle ligne la colonne [VERSIONING] est vide, alors on lui affecte la valeur 1 ;
- ligne 116 : on définit un code à exécuter avant chaque **mise à jour** dans la table [CATEGORIES] ;
- lignes 109-113 : le code dit qu'à la mise à jour d'une ligne la nouvelle valeur de la colonne [VERSIONING] est l'ancienne valeur augmentée de 1 ;

On retiendra de ces informations que vous n'avez à gérer :

- ni la gestion des clés primaires ;
- ni la gestion du versioning ;

Le SGBD s'en charge.

Les données de la base pourraient ressembler à ceci :

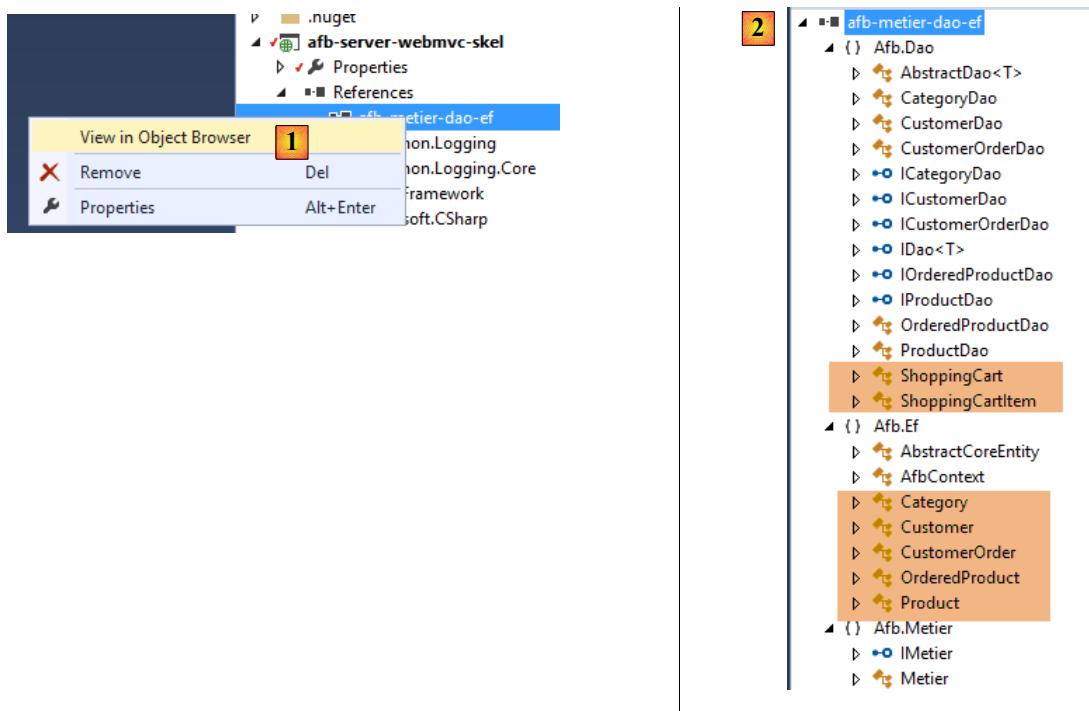
<b>[customer]</b>	<b>[customer_order]</b>	<b>[ordered_product]</b>
id   name   email   phone   address   city_region   cc_number	id   amount   date_created   confirmation_number   customer_id	ID   customer_order_id   product_id   quantity
40   1   2   3   4   5   6	40   10.57   2012-11-04 15:52:16   317891637   40	49   40   1   2
		50   40   2   3
		<b>[product]</b>

[category]

6 parma ham	3.49	matured, organic (70g)	2012-10-30 16:36:41	2
7 chicken leg	2.59	free range (250g)	2012-10-30 16:36:41	2
8 sausages	3.55	reduced fat, pork 3 sausages (350g)	2012-10-30 16:36:41	2
9 sunflower seed loaf	1.89	600g	2012-10-30 16:36:41	3
10 sesame seed bagel	1.19	4 bagels	2012-10-30 16:36:41	3
11 pumpkin seed bun	1.15	4 buns	2012-10-30 16:36:41	3
12 chocolate cookies	2.39	contain peanuts (3 cookies)	2012-10-30 16:36:41	3
13 corn on the cob	1.59	2 pieces	2012-10-30 16:36:41	4
14 red currants	2.49	150g	2012-10-30 16:36:41	4
15 broccoli	1.29	500g	2012-10-30 16:36:41	4
16 seedless watermelon	1.49	250g	2012-10-30 16:36:41	4

### 4.3.3 Les entités manipulées par les couches basses

Le projet web contient une référence sur la DLL des couches basses nommée [afb-metier-dao-ef]. Ouvrez-la pour examiner son contenu [1-2] :



Les entités dites "persistantes" sont les objets C# qui encapsulent les lignes des tables de la base de données.

La classe [AbstractCoreEntity] est la classe parent de toutes les entités persistantes :

```

1.  using Newtonsoft.Json;
2.  using System;
3.  using System.ComponentModel.DataAnnotations;
4.  using System.ComponentModel.DataAnnotations.Schema;
5.
6.  namespace Afb.Ef
7.  {
8.      [Serializable]
9.      public class AbstractCoreEntity
10.     {
11.
12.         [Key]
13.         [Column("ID")]
14.         public int? Id { get; set; }
15.
16.         [ConcurrencyCheck]
17.         [Column("VERSIONING")]
18.         public int? Version { get; set; }
19.
20.         // signature
21.         public override string ToString()
22.         {
23.             return JsonConvert.SerializeObject(this);
24.         }
25.     }
26. }

```

- lignes 12-14 : les entités ont toutes une clé primaire [Id] ;
- lignes 16-18 : les entités ont toutes une propriété [Version] utilisé par EF pour gérer la concurrence d'accès à une ligne ([ConcurrencyCheck], ligne 16) ;

La classe [Category] est l'entité représentant une ligne de la table [CATEGORY] :

```

1.  using System;
2.  using System.ComponentModel.DataAnnotations;
3.  using System.ComponentModel.DataAnnotations.Schema;
4.
5.  namespace Afb.Ef
6.  {
7.      [Table("CATEGORIES")]
8.      public class Category : AbstractCoreEntity
9.      {
10.          [Required]
11.          [MaxLength(30)]
12.          [Column("NAME")]
13.          public String Name { get; set; }
14.
15.      }
16. }

```

- ligne 13 : le nom de la catégorie ;

La classe [Product] est l'entité représentant une ligne de la table [PRODUCT] :

```

1.  using System;
2.  using System.ComponentModel.DataAnnotations;
3.  using System.ComponentModel.DataAnnotations.Schema;
4.
5.  namespace Afb.Ef
6.  {
7.      [Table("PRODUCTS")]
8.      public class Product : AbstractCoreEntity
9.      {
10.          [Required]
11.          [MaxLength(30)]
12.          [Column("NAME")]
13.          public string Name { get; set; }
14.
15.          [MaxLength(100)]
16.          [Column("DESCRIPTION")]
17.          public string Description { get; set; }
18. }

```

```

19.     [Required]
20.     [Column("PRICE")]
21.     public double Price { get; set; }
22.
23.     [Column("LAST_UPDATE")]
24.     public DateTime LastUpdate { get; set; }
25.
26.     [Column("CATEGORY_ID")]
27.     public int? CategoryId { get; set; }
28.
29.     [ForeignKey("CategoryId")]
30.     public Category Category { get; set; }
31.   }
32. }
```

- ligne 24 : la valeur du champ *LastUpdate* est générée par le SGBD. On peut ne pas l'initialiser lors d'une insertion ;
- ligne 27 : le n° de la catégorie à laquelle appartient le produit ; C'est une clé étrangère sur la table [CATEGORY]. Ce champ est toujours initialisé lors d'une lecture en base ;
- ligne 29 : la catégorie à laquelle appartient le produit. Ce champ n'est pas initialisé par défaut lors d'une lecture en base (lazy loading) ;

**A noter :**

- lors d'une lecture en base, le champ [CategoryId] est initialisé mais pas toujours le champ [Category] (lazy loading) ;
- lors d'une insertion en base d'un type [Product] : si [CategoryId==null] (ligne 27), alors il y aura persistance par cascade de l'objet [Category] de la ligne 30 (qui doit donc exister), sinon il n'y aura pas de persistance de cet objet [Category] (qui peut donc être *null*) ;

La classe [Customer] est l'entité représentant une ligne de la table [CUSTOMER] :

```

1.  using System;
2.  using System.ComponentModel.DataAnnotations;
3.  using System.ComponentModel.DataAnnotations.Schema;
4.
5.  namespace Afb.Ef
6.  {
7.    [Table("CUSTOMERS")]
8.    public class Customer : AbstractCoreEntity
9.    {
10.      [Required]
11.      [MaxLength(30)]
12.      [Column("NAME")]
13.      public String Name { get; set; }
14.
15.      [Required]
16.      [MaxLength(50)]
17.      [Column("EMAIL")]
18.      public String Email { get; set; }
19.
20.      [MaxLength(20)]
21.      [Column("PHONE")]
22.      public String Phone { get; set; }
23.
24.      [Required]
25.      [MaxLength(20)]
26.      [Column("ADDRESS")]
27.      public String Address { get; set; }
28.
29.      [MaxLength(2)]
30.      [MinLength(2)]
31.      [Column("CITY_REGION")]
32.      public String CityRegion { get; set; }
33.
34.      [Required]
35.      [MaxLength(19)]
36.      [Column("CC_NUMBER")]
37.      public String CcNumber { get; set; }
38.
39.    }
40. }
```

- lors d'une insertion d'un type [Customer], on pourra ne pas initialiser le champ [CityRegion] (ligne 32) car la colonne correspondante dans la base accepte les valeurs [NULL] ;

La classe [CustomerOrder] est l'entité représentant une ligne de la table [CUSTOMER\_ORDER] :

```

1.  using System;
2.  using System.ComponentModel.DataAnnotations;
3.  using System.ComponentModel.DataAnnotations.Schema;
4.
5.  namespace Afb.Ef
6.  {
7.      [Table("CUSTOMERS_ORDERS")]
8.      public class CustomerOrder : AbstractCoreEntity
9.      {
10.         [Required]
11.         [Column("AMOUNT")]
12.         public double Amount { get; set; }
13.
14.         [DatabaseGenerated(DatabaseGeneratedOption.Computed)]
15.         [Column("DATE_CREATED")]
16.         public DateTime? DateCreated { get; set; }
17.
18.         [Required]
19.         [Column("CONFIRMATION_NUMBER")]
20.         public int ConfirmationNumber { get; set; }
21.
22.         [Column("CUSTOMER_ID")]
23.         public int? CustomerId { get; set; }
24.
25.         [ForeignKey("CustomerId")]
26.         public Customer Customer { get; set; }
27.     }
28. }
```

- ligne 23 : le n° du client qui a fait la commande. La colonne correspondante dans la base est clé étrangère sur la colonne [CUSTOMERS.ID] ;
- ligne 26 : le client qui a fait la commande ;

#### A noter :

- lors d'une lecture en base, le champ [CustomerId] est initialisé mais pas toujours le champ [Customer] (lazy loading) ;
- lors d'une insertion en base d'un type [CustomerOrder] : si [CustomerId==null] (ligne 23), alors il y aura persistance par cascade de l'objet [Customer] de la ligne 26 (qui doit donc exister), sinon il n'y aura pas de persistance de cet objet [Customer] (qui peut donc être null) ;

La classe [OrderedProduct] est l'entité représentant une ligne de la table [ORDERED\_PRODUCT] :

```

1.  using System;
2.  using System.ComponentModel.DataAnnotations;
3.  using System.ComponentModel.DataAnnotations.Schema;
4.
5.  namespace Afb.Ef
6.  {
7.      [Table("ORDERED_PRODUCTS")]
8.      public class OrderedProduct : AbstractCoreEntity
9.      {
10.         [Required]
11.         [Column("QUANTITY")]
12.         public int Quantity { get; set; }
13.
14.         [Column("PRODUCT_ID")]
15.         public int? ProductId { get; set; }
16.
17.         [ForeignKey("ProductId")]
18.         public Product Product { get; set; }
19.
20.         [Column("CUSTOMER_ORDER_ID")]
21.         public int? CustomerOrderId { get; set; }
22.
23.         [ForeignKey("CustomerOrderId")]
24.         public CustomerOrder CustomerOrder { get; set; }
25. }
```

```
26.     }
27. }
```

- ligne 15 : le n° de l'article acheté. La colonne correspondante dans la base est clé étrangère sur la colonne [PRODUCTS.ID] ;
- ligne 18 : le produit acheté ;
- ligne 21 : le n° de la commande à laquelle appartient ce produit acheté. La colonne correspondante dans la base est clé étrangère sur la colonne [CUSTOMER\_ORDERS.ID] ;
- ligne 24 : l'ordre de commande ;

#### A noter :

- lors d'une lecture en base, le champ [ProductId] est initialisé mais pas toujours le champ [Product] (lazy loading). De même, le champ [CustomerOrderId] est initialisé mais pas toujours le champ [CustomerOrder] ;
- lors d'une insertion en base d'un type [OrderedProduct] :
  - si [ProductId==null] (ligne 15), alors il y aura persistance par cascade de l'objet [Product] de la ligne 18 (qui doit donc exister), sinon il n'y aura pas de persistance de cet objet [Product] (qui peut donc être null) ;
  - si [CustomerOrderId==null] (ligne 21), alors il y aura persistance par cascade de l'objet [CustomerOrder] de la ligne 24 (qui doit donc exister), sinon il n'y aura pas de persistance de cet objet [CustomerOrder] (qui peut donc être null) ;

Le type [ShoppingCart] représente le chariot du client :

```
1.  using Afb.Ef;
2.  using System;
3.  using System.Collections.Generic;
4.  using System.Linq;
5.
6.  namespace Afb.Dao
7.  {
8.      [Serializable]
9.      public class ShoppingCart {
10.
11.         // data
12.         public List<ShoppingCartItem> Items { get; private set; }
13.         private int numberofItems;
14.         public int NumberOfItems {
15.             get {
16.                 return Items.Sum(item => item.Quantity);
17.             }
18.             private set {
19.                 numberofItems = value;
20.             }
21.         }
22.         public double Total { get; set; }
23.
24.         // constructeur
25.         public ShoppingCart() {
26.             Items = new List<ShoppingCartItem>();
27.             NumberofItems = 0;
28.             Total = 0;
29.         }
30.
31.         // met à jour la qté achetée d'un produit
32.         // si quantity=0, le produit est enlevé du panier
33.         // si le produit n'existe pas, il est ajouté
34.         public void Update(Product product, int quantity)
35.         {
36.             ShoppingCartItem item = null;
37.             foreach (ShoppingCartItem scItem in Items)
38.             {
39.                 if (scItem.Product.Id == product.Id)
40.                 {
41.                     item = scItem;
42.                     break;
43.                 }
44.             }
45.
46.             if (item == null)
47.             {
48.                 // add to cart
49.                 ShoppingCartItem newItem = new ShoppingCartItem() { Product = product, Quantity = quantity };
50.                 Items.Add(newItem);
```

```

51.         }
52.     else
53.     {
54.         if (quantity == 0)
55.         {
56.             Items.Remove(item);
57.         }
58.     else
59.     {
60.         item.Quantity = quantity;
61.     }
62. }
63. }
64.
65. // coût du panier sans la taxe
66. public double Subtotal() {
67.     return Items.Sum(item => item.Product.Price * item.Quantity);
68. }
69.
70. // coût du panier avec la taxe
71. public void CalculateTotal(double surcharge) {
72.     Total = Subtotal() + surcharge;
73. }
74.
75. // raz liste d'articles
76. public void Clear() {
77.     Items.Clear();
78.     NumberOfItems = 0;
79.     Total = 0;
80. }
81. }
82. }
```

- ligne 12 : la liste des articles achetés par le client ;
- lignes 14-21 : la propriété [NumberOfItems] représente le nombre total d'articles présents dans le chariot ;
- ligne 16 : on utilise une méthode LINQ [Sum] qui permet de sommer les éléments d'une collection. Le paramètre de cette méthode, est une fonction lambda qui ici admet pour paramètre un élément [CartItem] de la collection et qui rend la propriété [CartItem.Quantity]. C'est donc au final, les propriétés [Quantity] des objets de la collection [Items] qui sont sommés ;
- ligne 22 : la propriété [Total] représente le prix à payer pour les articles achetés et la taxe de livraison ;
- lignes 34-63 : la méthode [Update] permet d'ajouter [quantity] articles de type [Product] au chariot ;
- lignes 66-68 : la méthode [SubTotal] calcule le prix à payer pour les articles achetés sans la taxe de livraison ;
- ligne 67 : on retrouve la méthode LINQ [Sum] que nous avons expliquée plus haut ;
- lignes 71-73 : la méthode [CalculateTotal] calcule la propriété [Total] de la ligne 22 ;
- lignes 76-80 : la méthode [Clear] vide le chariot ;

Le type [ShoppingCartItem] des articles du chariot est le suivant :

```

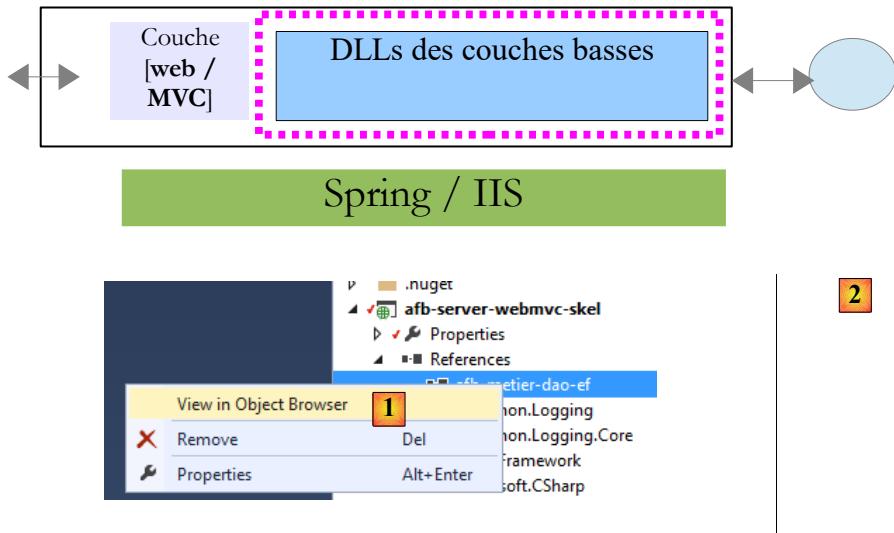
1. using Afb.Ef;
2. using System;
3.
4. namespace Afb.Dao
5. {
6.     [Serializable]
7.     public class ShoppingCartItem
8.     {
9.         // data
10.        public Product Product { get; set; }
11.        public int Quantity { get; set; }
12.        public double Total
13.        {
14.            get
15.            {
16.                return (Quantity * Product.Price);
17.            }
18.        }
19.
20.        // constructeurs
21.        public ShoppingCartItem()
22.        {
23.        }
24. }
```

```

25.     public ShoppingCartItem(Product product)
26.     {
27.         this.Product = product;
28.         Quantity = 1;
29.     }
30. }
31. }
```

- ligne 10 : le produit acheté ;
- ligne 11 : la quantité achetée ;
- lignes 12-18 : la propriété [Total] est le coût des [Quantity] articles achetés ;

#### 4.3.4 L'interface [IMetier] des couches basses

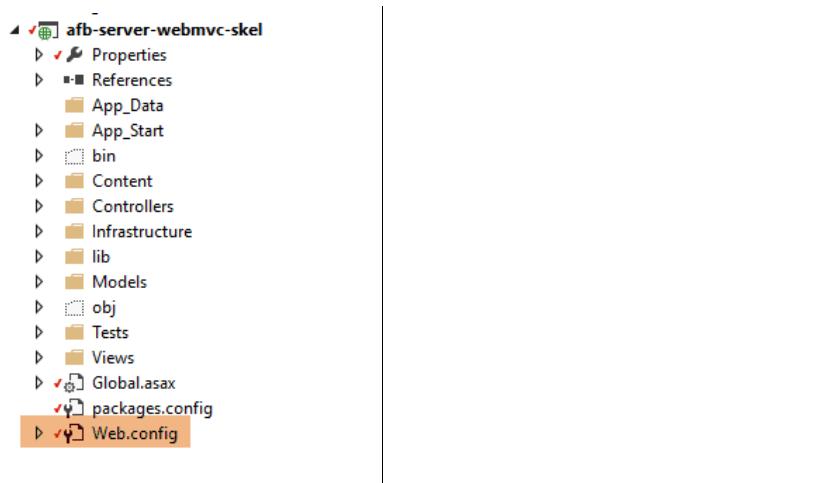


La DLL des couches basses présente l'interface [IMetier] suivante :

```

1.  using Afb.Dao;
2.  using Afb.Ef;
3.  using System.Collections.Generic;
4.
5.  namespace Afb.Metier
6.  {
7.
8.      public interface IMetier
9.      {
10.
11.         // détails d'une commande identifiée par sa clé primaire
12.         List<OrderedProduct> OrderDetails(int orderId);
13.
14.         // persister une commande
15.         // customer : l'acheteur
16.         // cart : son panier d'achats
17.         CustomerOrder PlaceOrder(Customer customer, ShoppingCart cart);
18.
19.         // liste des catégories de produits
20.         List<Category> Categories();
21.
22.         // liste des produits d'une catégorie identifiée par sa clé primaire
23.         List<Product> ProductsByCategory(int categoryId);
24.
25.         // Reset des tables [Customers, CustomerOrders, OrderedProducts]
26.         void ClearCustomers();
27.     }
28. }
```

### 4.3.5 La configuration des couches basses



Le projet [afb-server-webmvc-skel] est configuré par le fichier [Web.config] suivant :

```
1. <?xml version="1.0"?>
2. <configuration>
3.   <configSections>
4.     <!-- For more information on Entity Framework configuration, visit http://go.microsoft.com/fwlink/?LinkID=237468 -->
5.     <section name="entityFramework" type="System.Data.Entity.Internal.ConfigFile.EntityFrameworkSection,
EntityFramework, Version=6.0.0.0, Culture=neutral, PublicKeyToken=b77a5c561934e089"
requirePermission="false"/>
6.     <!-- spring -->
7.     <sectionGroup name="spring">
8.       <section name="context" type="Spring.Context.Support.ContextHandler, Spring.Core"/>
9.       <section name="objects" type="Spring.Context.Support.DefaultSectionHandler, Spring.Core"/>
10.    </sectionGroup>
11.  </configSections>
12.  <!-- validations client en JS-->
13.  <appSettings>
14.    <add key="ClientValidationEnabled" value="true"/>
15.    <add key="UnobtrusiveJavaScriptEnabled" value="true"/>
16.  </appSettings>
17.  <!-- configuration Spring -->
18.  <spring>
19.    <context>
20.      <resource uri="config://spring/objects"/>
21.    </context>
22.    <objects xmlns="http://www.springframework.net">
23.      <object id="categoryDao" type="Afb.Dao.CategoryDao,afb-metier-dao-ef"/>
24.      <object id="customerDao" type="Afb.Dao.CustomerDao,afb-metier-dao-ef"/>
25.      <object id="customerOrderDao" type="Afb.Dao.CustomerOrderDao,afb-metier-dao-ef"/>
26.      <object id="orderedProductDao" type="Afb.Dao.OrderedProductDao,afb-metier-dao-ef"/>
27.      <object id="metier" type="Afb.Metier.Metier,afb-metier-dao-ef">
28.        <property name="OrderedProductDao" ref="orderedProductDao"/>
29.        <property name="CategoryDao" ref="categoryDao"/>
30.        <property name="CustomerOrderDao" ref="customerOrderDao"/>
31.        <property name="CustomerDao" ref="customerDao"/>
32.      </object>
33.      <object id="config" type="Afb.Web.Models.Config,afb-webmvc">
34.        <property name="DeliverySurcharge" value="2.8"/>
35.      </object>
36.      <object id="application" type="Afb.Web.Models.ApplicationModel,afb-webmvc">
37.        <property name="Metier" ref="metier"/>
38.        <property name="Config" ref="config"/>
39.      </object>
40.    </objects>
41.  </spring>
42.  <startup>
43.    <supportedRuntime version="v4.0" sku=".NETFramework,Version=v4.5"/>
44.  </startup>
```

```

45.      <entityFramework>
46.          <providers>
47.              <provider invariantName="MySql.Data.MySqlClient"
48.                  type="MySql.Data.MySqlClient.MySqlProviderServices, MySql.Data.Entity.EF6, Version=6.9.7.0,
49.                  Culture=neutral, PublicKeyToken=c5687fc88969c44d"/>
50.          </providers>
51.      </entityFramework>
52.      <system.data>
53.          <DbProviderFactories>
54.              <remove invariant="MySql.Data.MySqlClient"/>
55.              <add name="MySQL Data Provider" invariant="MySql.Data.MySqlClient" description=".Net Framework
56. Data Provider for MySQL" type="MySql.Data.MySqlClient.MySqlClientFactory, MySql.Data, Version=6.9.7.0,
57. Culture=neutral, PublicKeyToken=c5687fc88969c44d"/>
58.          </DbProviderFactories>
59.      </system.data>
60.      <connectionStrings>
61.          <add name="AfbContext" connectionString="Server=localhost;Database=e-magasin;Uid=root;Pwd=;" providerName="MySql.Data.MySqlClient"/>
62.      </connectionStrings>
63.  </system.web>
64.  <compilation debug="true"/>
65. </system.web>
66. </configuration>

```

- ligne 57 : la chaîne de connexion [ADO.NET] pour se connecter à la base de données MySQL [e-magasin] ;
- ligne 57 : l'attribut [providerName="MySql.Data.MySqlClient"] sert à indiquer le fournisseur ADO.NET à utiliser, c-à-d le pilote du SGBD qui gère la base de données, ici MySQL. Ce nom intervient à divers endroits : lignes 47, 52, 57 ;
- ligne 52 : le fournisseur ADO.NET [MySql.Data.MySqlClient] peut être déjà présent sur la machine, auquel cas la ligne 53 provoque un plantage. La ligne 52 enlève donc auparavant le fournisseur ADO.NET [MySql.Data.MySqlClient] de la machine ;
- ligne 57 : l'attribut [connectionString="Server=localhost;Database=e-magasin;Uid=root;Pwd;"] est la chaîne de connexion ADO.NET à la base de données MySQL [e-magasin]. Cette chaîne est propriétaire. Elle change de forme avec chaque SGBD ;
- lignes 18-41 : définition des objets gérés par Spring ;
- lignes 27-32 : l'objet spring nommé [metier] est associé à la classe implémentant l'interface [IMetier] des couches basses ;

### 4.3.6 Vérification du fonctionnement des couches basses

Pour vérifier le bon fonctionnement des couches basses, le projet web [afb-server-webmvc-skel] contient un test unitaire Nunit :

```

1.  using Afb.Dao;
2.  using Afb.Ef;
3.  using Afb.Metier;
4.  using NUnit.Framework;
5.  using Spring.Context.Support;
6.  using System.Collections.Generic;
7.
8.  namespace Istia.St.Afb.Web.Tests
9.  {
10.
11.     [TestFixture]
12.     class NUnitTestMetier
13.     {
14.         // la couche [métier]
15.         private IMetier metier;
16.
17.         [TestFixtureSetUp]
18.         public void Init()
19.         {
20.             // instanciation couche [metier] via Spring
21.             metier = ContextRegistry.GetContext().GetObject("metier") as IMetier;
22.         }
23.
24.         [TestFixtureTearDown]
25.         public void Dispose()
26.         { /* ... */ }
27.
28.         [SetUp]
29.         public void BeforeTest()
30.         {
31.             metier.ClearCustomers();
32.         }

```

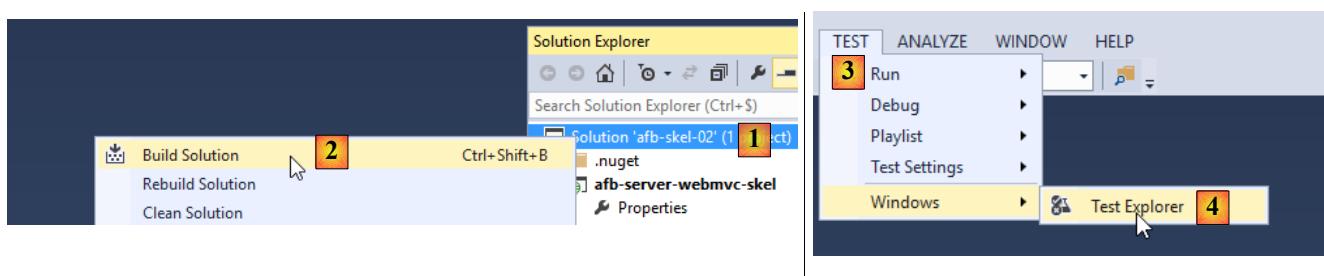
```

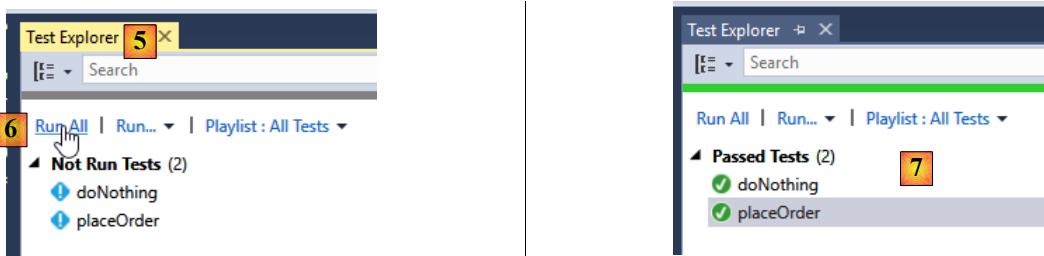
33.
34.     [TearDown]
35.     public void endTest()
36.     {
37.     }
38.
39.     [Test]
40.     public void doNothing()
41.     {
42.     }
43.
44.     [Test]
45.     public void placeOrder()
46.     {
47.         // liste des catégories
48.         List<Category> categories = metier.Categories();
49.         Assert.AreEqual(4, categories.Count);
50.         // liste des produits catégorie 0
51.         long? categoryId = categories[0].Id;
52.         List<Product> products = metier.ProductsByCategory((int)categoryId);
53.         // remplissage chariot
54.         ShoppingCart chariot = new ShoppingCart();
55.         // 2 produits 0
56.         chariot.Update(products[0], 2);
57.         // 3 produits 1
58.         chariot.Update(products[1], 3);
59.         // on force le calcul du prix à payer
60.         chariot.CalculateTotal();
61.         // persistance client
62.         Customer customer = new Customer() { Name = "1", Email = "2", Phone = "3", Address = "4",
CityRegion = "55", CcNumber = "6" };
63.         CustomerOrder customerOrder = metier.PlaceOrder(customer, chariot);
64.
65.         // vérifications
66.         List<OrderedProduct> orderedProducts = metier.OrderDetails((int)customerOrder.Id);
67.         Assert.AreEqual(products[0].Name, orderedProducts[0].Product.Name);
68.         Assert.AreEqual(products[1].Name, orderedProducts[1].Product.Name);
69.         Assert.AreEqual(2, orderedProducts[0].Quantity);
70.         Assert.AreEqual(3, orderedProducts[1].Quantity);
71.         CustomerOrder customerOrder2 = orderedProducts[0].CustomerOrder;
72.         Assert.AreEqual(10.57, (double)customerOrder2.Amount, 1e-6);
73.         Customer customer2 = customerOrder2.Customer;
74.         Assert.AreEqual("1", customer2.Name);
75.         Assert.AreEqual("2", customer2.Email);
76.         Assert.AreEqual("3", customer2.Phone);
77.         Assert.AreEqual("4", customer2.Address);
78.         Assert.AreEqual("55", customer2.CityRegion);
79.         Assert.AreEqual("6", customer2.CcNumber);
80.     }
81. }
82. }

```

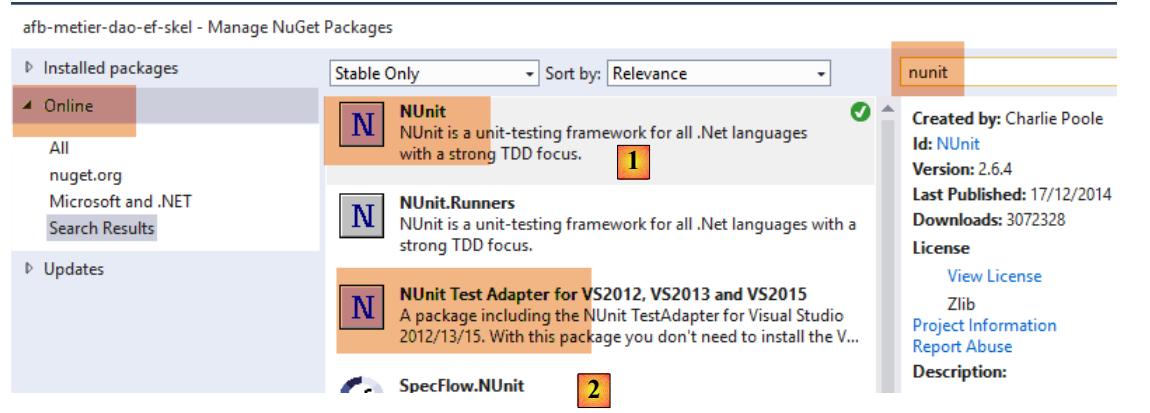
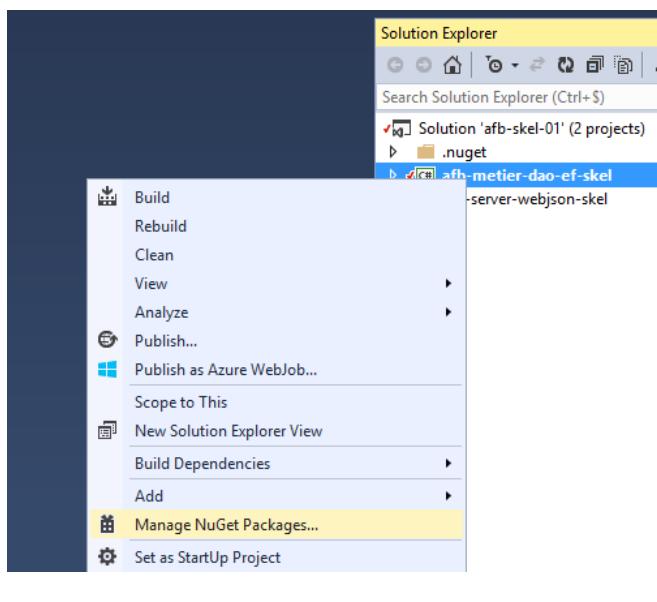
Comprenez ce code. Il vous dit comment utiliser l'interface [IMetier].

Il est possible d'exécuter un test NUnit à l'intérieur de Visual Studio. Pour cela, procédez comme suit [1-7] :

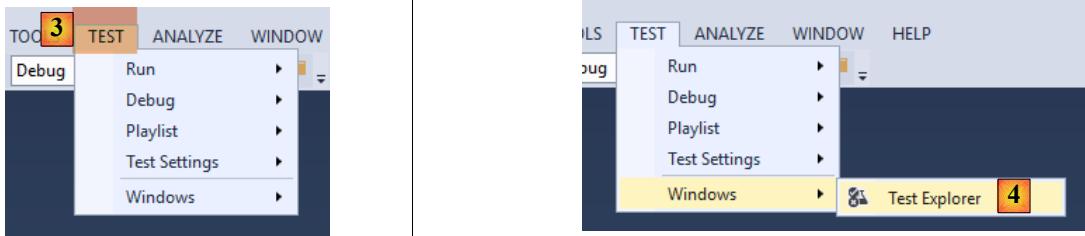




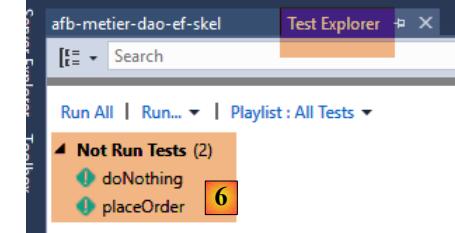
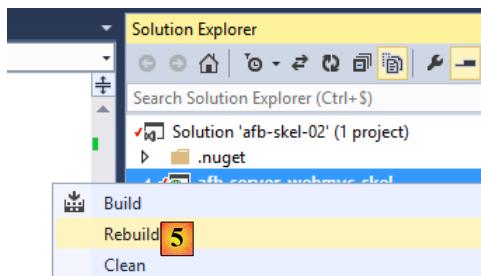
En [7], les tests doivent avoir été réussis. Le test précédent nécessite des packages Nuget qui ne sont peut-être pas installés sur votre machine. Dans ce cas, procédez comme suit, sinon passez au paragraphe suivant :



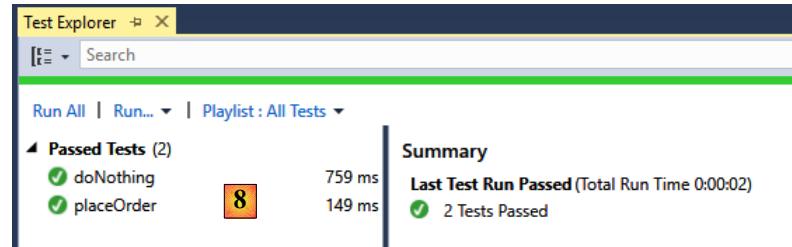
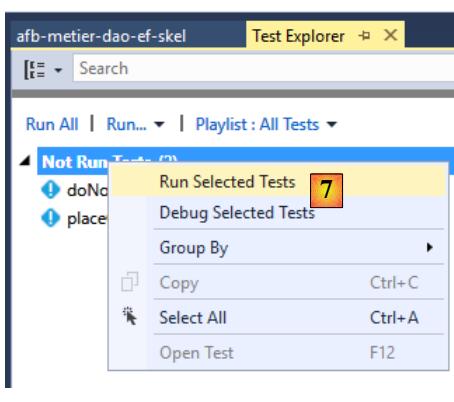
- installez les packages [1] et [2] ;
- fermez puis réouvrez Visual Studio ;



- l'option [Test] [3] doit être présente dans le menu ;
- ouvrez la fenêtre de tests [4] ;



- en [5], reconstruisez votre projet ;
- en [6], les méthodes de test NUnit doivent apparaître dans la fenêtre de tests ;

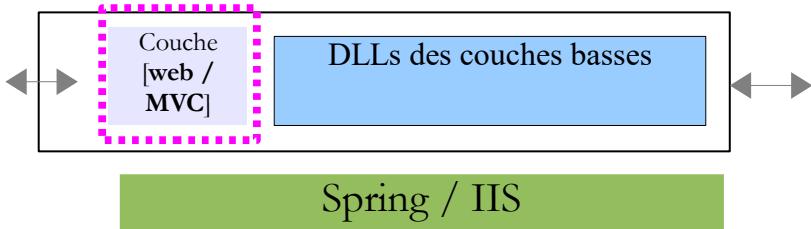


- en [7], exéutez les tests ;
- en [8], ils doivent réussir ;

**Note :** si le test NUnit ne réussit pas, vérifiez bien toutes les étapes précédentes.

## 4.4 Ecriture de la couche web / MVC

Nous allons maintenant construire la couche web de l'application :



#### 4.4.1 Les vues de l'application

L'application est une application de commerce électronique. La page d'accueil est la suivante :



Les images ci-dessus sont des liens. L'utilisateur choisit l'une des catégories de produits en cliquant dessus :



Il ajoute différents articles à son panier :

The screenshot shows the homepage of L'épicerie verte. At the top right, there's a red-bordered box containing "2 articles" with a shopping cart icon, a link to "voir le panier", and a link to "passer au paiement". Below this is the website's logo, "L'épicerie verte", with a stylized green vine and leaf graphic. To the left of the main content area, there are four category buttons: "dairy", "meats", "bakery", and "fruit & veg". To the right is a grid of four products:

	sunflower seed loaf	1,89 €	<a href="#">Ajouter au panier</a>
	sesame seed bagel	1,19 €	<a href="#">Ajouter au panier</a>
	pumpkin seed bun	1,15 €	<a href="#">Ajouter au panier</a>
	chocolate cookies	2,39 €	<a href="#">Ajouter au panier</a>

At the bottom left, there are links to "Politique intime", "Contact", and the year "2012 L'épicerie verte".

Il peut visualiser son panier :

The screenshot shows the shopping cart page. At the top right, there's a red-bordered box containing "2 articles" with a shopping cart icon, a link to "voir le panier", and a link to "passer au paiement". Below this is the website's logo, "L'épicerie verte", with a stylized green vine and leaf graphic. A message "Votre chariot contient 2 articles." is displayed. There are two buttons at the top: "vider le panier" and "continuer les achats". Below these are two rows of product information:

article	nom	prix	quantité	
	sunflower seed loaf	1,89 € (1,89 € l'unité)	<input type="text" value="1"/> <a href="#">Modifier</a>	
	sesame seed bagel	1,19 € (1,19 € l'unité)	<input type="text" value="1"/> <a href="#">Modifier</a>	

At the bottom left, there are links to "Politique intime", "Contact", and the year "2012 L'épicerie verte".

Lorsqu'il a fini, il passe au paiement :

The screenshot shows the payment page. At the top right, there's a red-bordered box containing "2 articles" with a shopping cart icon, a link to "voir le panier", and a link to "passer au paiement". Below this is the website's logo, "L'épicerie verte", with a stylized green vine and leaf graphic. The word "paiement" is centered above a form. A message "Pour le paiement de vos achats, veuillez nous donner les informations suivantes :" is displayed. The form contains fields for "nom", "adresse électronique", "téléphone", "adresse", and "carte de crédit", each with an "x" placeholder. A blue "valider" button is at the bottom left of the form. To the right, there's a summary box with the following text:

- Livraison J+1 garantie
- A 2,80 € le surcoût de livraison est appliquée à tous les articles

sous-total : 3,08 €  
coût livraison : 2,80 €  
total : 5,88 €

At the bottom left, there are links to "Politique intime", "Contact", and the year "2012 L'épicerie verte".

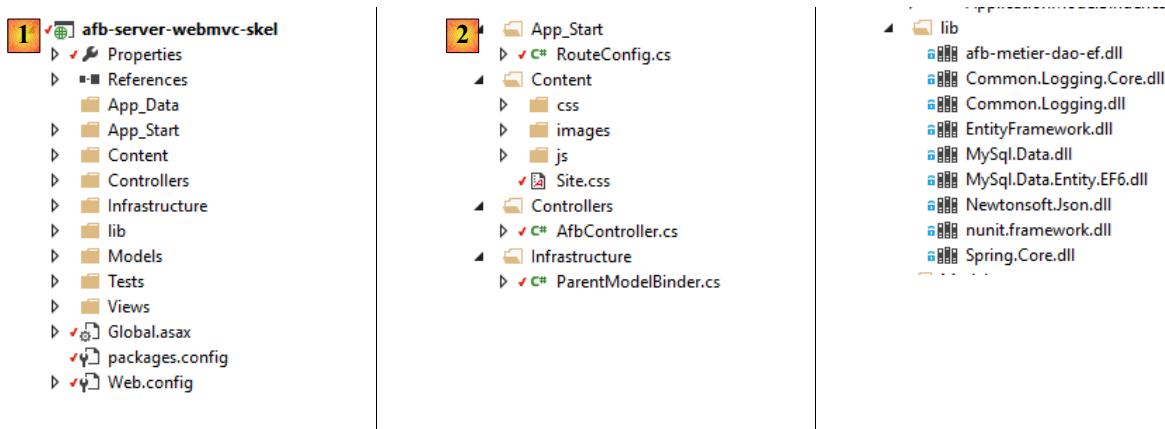
Lorsqu'il valide sa commande, il obtient une page de confirmation :



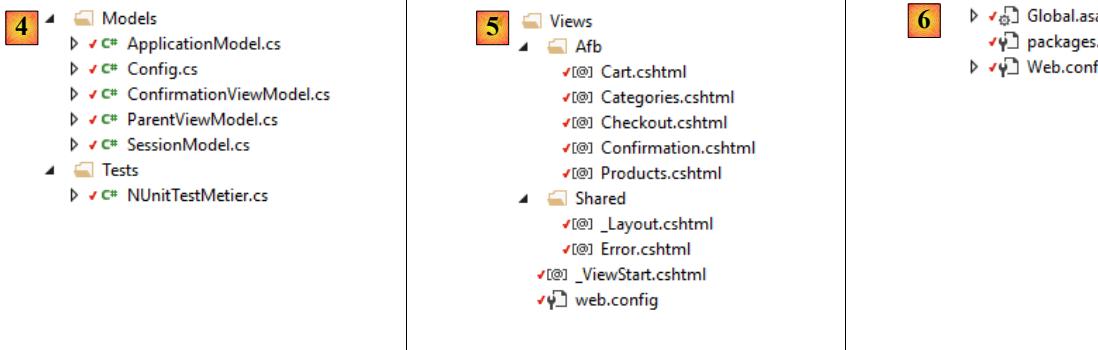
Il peut revenir aux achats en cliquant sur [1] qui est une image cliquable.

#### 4.4.2 Le projet Visual Studio

Le projet Visual Studio du serveur web / MVC est le suivant :

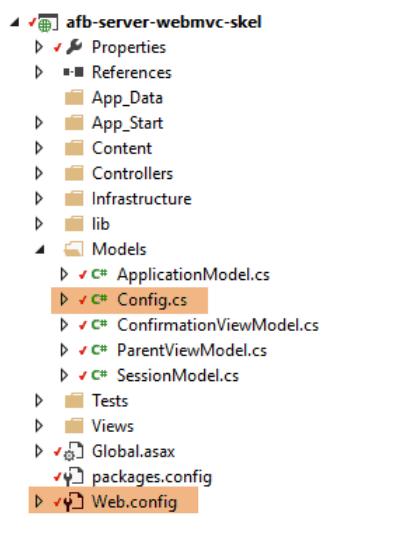


- en [1], la structure générale du projet ;
  - en [2] :
    - [RouteConfig] est la classe de configuration des routes du service web / JSON ;
    - [AfbController] est la classe qui assure le traitement des requêtes des clients ;
    - [ParentModelBinder] est la classe qui permet d'obtenir une référence sur la classe [ParentViewModel] présente en [4] ;
    - [Content] est le dossier qui contient les fichiers statiques de l'application (CSS, images et Javascript) ;
  - en [3], les dépendances utilisées par le projet. Si vous avez suivi la démarche expliquée au paragraphe 4.3.1, page 389, ce dossier est inutile. Si la démarche précédente a échoué, alors mettez toutes les DLL de ce dossier dans les références du projet ;



- en [4] le dossier [Models] contient différents modèles : ceux des actions (ParentViewModel, ApplicationModel, SessionModel), celle de la vue de confirmation (ConfirmationViewModel) ainsi que la classe [Config] qui encapsule des données de configuration ;
- en [5], les différentes vues de l'application :
  - \_Layout : la page template des autres vues - factorise les éléments communs à toutes les vues ;
  - Cart : affiche le panier du client ;
  - Categories : affiche les catégories de produits du e-magasin ;
  - Products : affiche les produits d'une catégorie particulière ;
  - Checkout : page de validation des achats ;
  - Confirmation : page de confirmation d'une commande validée ;
- en [6] :
  - [Web.config] est le fichier global de configuration de l'application web ;
  - [Global.asax] contient la classe qui démarre l'application en exploitant le fichier [Web.config] précédent ;

#### 4.4.3 Configuration



Le fichier [Web.config] est le fichier standard qui sert à configurer une application ASP.NET MVC. Nous lui avons ajouté la classe [Config] pour loger les constantes de l'application. Celles-ci seront prises dans le fichier [Web.config] et encapsulées dans la classe [Config] par des injections Spring. Cela permet d'encapsuler toutes les constantes de configuration de l'application à un seul endroit.

La classe [Config] est la suivante :

```

1. namespace Afb.Web.Models
2. {
3.     public class Config
4.     {
5.         // la taxe de livraison
6.         public double DeliverySurcharge { get; set; }
7.     }

```

Il n'y a ici qu'une unique constante de configuration, ligne 6 : la taxe de livraison de la commande au client. Sa valeur sera trouvée dans le fichier [Web.config].

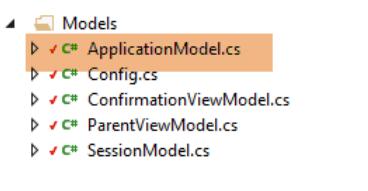
Le fichier [Web.config] a déjà été partiellement présenté au paragraphe 4.3.5, page 401 où on en a expliqué les parties liées à la configuration des couches basses. Nous allons maintenant expliquer les parties du fichier qui configurent la couche [web / MVC] :

```

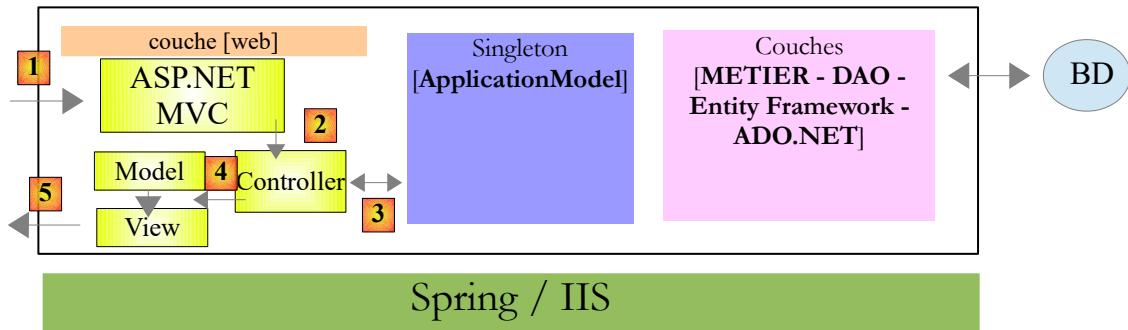
1. <?xml version="1.0"?>
2. <configuration>
3.   <configSections>
4.     <!-- For more information on Entity Framework configuration, visit http://go.microsoft.com/fwlink/?LinkID=237468 -->
5.     <section name="entityFramework" type="System.Data.Entity.Internal.ConfigFile.EntityFrameworkSection,
EntityFramework, Version=6.0.0.0, Culture=neutral, PublicKeyToken=b77a5c561934e089"
requirePermission="false"/>
6.     <!-- spring -->
7.     <sectionGroup name="spring">
8.       <section name="context" type="Spring.Context.Support.ContextHandler, Spring.Core"/>
9.       <section name="objects" type="Spring.Context.Support.DefaultSectionHandler, Spring.Core"/>
10.    </sectionGroup>
11.   </configSections>
12.   <!-- validations client en JS-->
13.   <appSettings>
14.     <add key="ClientValidationEnabled" value="true"/>
15.     <add key="UnobtrusiveJavaScriptEnabled" value="true"/>
16.   </appSettings>
17.   <!-- configuration Spring -->
18.   <spring>
19.     <context>
20.       <resource uri="config://spring/objects"/>
21.     </context>
22.     <objects xmlns="http://www.springframework.net">
23.       ...
24.       <object id="metier" type="Afb.Metier.Metier,afb-metier-dao-ef">
25.         ...
26.       </object>
27.       <object id="config" type="Afb.Web.Models.Config,afb-webmvc">
28.         <property name="DeliverySurcharge" value="2.8"/>
29.       </object>
30.       <object id="application" type="Afb.Web.Models.ApplicationModel,afb-webmvc">
31.         <property name="Metier" ref="metier"/>
32.         <property name="Config" ref="config"/>
33.       </object>
34.     </objects>
35.   </spring>
36. ...
37. </configuration>
```

- lignes 24-26 : définition de l'objet Spring nommé [metier] et associé à la DLL des couches basses ;
- lignes 27-20 : définition de l'objet Spring nommé [config] et associé à la classe [Config] que nous venons de présenter. En ligne 28, la valeur de la taxe de livraison ;
- lignes 30-33 : définition de l'objet Spring nommé [application] et associé à une classe de type [ApplicationModel] qui encapsule les deux objets [metier] et [config] définis précédemment. Le paragraphe suivant présente cette classe.

#### 4.4.4 La classe [ApplicationModel]



La classe [ApplicationModel] encapsule les données de portée [Application], celles disponibles en lecture seule à toutes les requêtes de tous les utilisateurs. On peut également lui faire implémenter l'interface [IMetier] qui gère l'accès aux couches basses. Ce sera le cas ici. On aura alors l'architecture MVC suivante :



- en [1], une requête client arrive. Elle est dispatchée [2] sur l'un des contrôleurs de l'application ;
- en [3], ce contrôleur peut avoir besoin d'accéder aux données de la base de données. Cela se fera via les méthodes de la classe [ApplicationModel] qui implémente l'interface [IMetier] ;
- en [4], le contrôleur génère le modèle M que la vue V qu'il désigne doit afficher ;
- en [5], la vue génère le flux HTML destiné au client ;

Les méthodes du contrôleur utilisent les services offerts par le singleton [ApplicationModel]. Le code de celui-ci est le suivant :

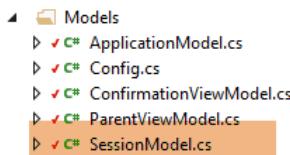
```

1.  using Afb.Dao;
2.  using Afb.Ef;
3.  using Afb.Metier;
4.  using System;
5.  using System.Collections.Generic;
6.
7.  namespace Afb.Web.Models
8.  {
9.      public class ApplicationModel : IMetier
10.     {
11.         // --- début injections Spring
12.         // la couche [métier]
13.         public IMetier Metier { get; set; }
14.         // configuration
15.         public Config Config { get; set; }
16.         // --- fin injections Spring
17.
18.         // ----- interface IMetier
19.         // détails d'une commande
20.         public List<OrderedProduct> OrderDetails(int orderId)
21.         {
22.             return Metier.OrderDetails(orderId);
23.         }
24.
25.         // faire une commande
26.         public CustomerOrder PlaceOrder(Customer customer, ShoppingCart cart)
27.         {
28.             return Metier.PlaceOrder(customer, cart);
29.         }
30.
31.         // liste des catégories de produits
32.         public List<Category> Categories()
33.         {
34.             return Metier.Categories();
35.         }
36.
37.         // liste des produits d'une catégorie
38.         public List<Product> ProductsByCategory(int categoryId)
39.         {
40.             return Metier.ProductsByCategory(categoryId);
41.         }
42.
43.         // reset de la base
44.         public void ClearCustomers()
45.         {
46.             Metier.ClearCustomers();
47.         }
48.     }
49. }
```

- ligne 13 : une référence sur la couche [métier] qui permet d'implémenter l'interface [IMetier] (lignes 20-47) ;
- ligne 15 : une référence sur l'instance qui contient les données de configuration de l'application ;

Note : la classe [ApplicationModel] ne devra pas être modifiée.

#### 4.4.5 La classe [SessionModel]



La classe [SessionModel] encapsule les données de portée [Session], celles disponibles en lecture et écriture à toutes les requêtes d'un même utilisateur. C'est la mémoire de celui-ci. Son code est le suivant :

```

1. namespace Afb.Web.Models {
2.   public class SessionModel {
3.     ...
4.   }
5. }
  
```

Note : ce sera à vous d'ajouter des propriétés à cette classe.

#### 4.4.6 Les classes [ParentViewModel] et [ParentModelBinder]



La classe [ParentViewModel] encapsule les données de portées [Application] et [Session] :

```

1. namespace Afb.Web.Models {
2.   public class ParentViewModel {
3.     // l'application
4.     public ApplicationModel Application { get; set; }
5.     // la session
6.     public SessionModel Session { get; set; }
7.   }
8. }
  
```

Beaucoup de méthodes du contrôleur [AfbController] ont une signature analogue à la suivante :

```

1.     [HttpGet]
2.     public ViewResult ProceedCheckout(ParentViewModel modèle)
3.   {
  
```

Le paramètre de type [ParentViewModel] sera automatiquement instancié par ASP.NET MVC grâce au [ModelBinder] suivant :

```

1. using Afb.Web.Models;
2. using System.Web.Mvc;
3.
4. namespace Afb.Web.Infrastructure {
5.   public class ParentModelBinder : IModelBinder {
6.     public object BindModel(ControllerContext controllerContext, ModelBindingContext bindingContext) {
  
```

```

7.     return new ParentViewModel() {
8.         Application = (ApplicationModel)controllerContext.HttpContext.Application["data"],
9.         Session = (SessionModel)controllerContext.HttpContext.Session["data"]
10.    };
11. }
12. }
13. }

```

- ligne 7 la classe [ParentModelBinder] rend une instance de type [ParentViewModel] qui encapsule :
  - ligne 8 : le type [ApplicationModel] qui se trouve dans les données de portée [Application] associé à la clé [data] ;
  - ligne 9 : le type [SessionModel] qui se trouve dans les données de portée [Session] associé à la clé [data] ;

C'est la classe d'initialisation se trouvant dans [Global.asax] qui va se charger d'instancier les deux modèles précédents ainsi que la classe [ParentModelBinder].

#### Important :

Toutes les vues de l'application ont comme modèle la classe [ParentViewModel] ou une classe dérivée. Revenons à la définition de la classe [ParentViewModel] :

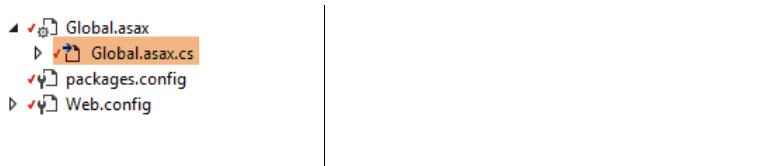
```

1. namespace Afb.Web.Models {
2.     public class ParentViewModel {
3.         // l'application
4.         public ApplicationModel Application { get; set; }
5.         // la session
6.         public SessionModel Session { get; set; }
7.     }
8. }

```

- ligne 4 : la classe [ApplicationModel] est fixée. Vous ne devez pas y toucher ;
- ligne 6 : la classe [SessionModel] est à votre disposition pour y mettre des données de portée session ou des données nécessaires aux vues. La bonne pratique veut que la classe de type [SessionModel] n'ait que des données de portée session. Lorsqu'une vue a besoin de données qui ne sont pas de portée session, vous pouvez mettre ces données dans une classe dérivée de [ParentViewModel]. Dans la suite ce dernier cas a été explicitement prévu. Donc si dans le texte, une vue a pour modèle la classe [ParentViewModel], cela signifie que ses données sont accessibles de deux façons :
  - par l'usage d'une des méthodes de la propriété [Application] de son modèle ;
  - par l'utilisation d'une des propriétés de la propriété [Session] de son modèle ;

#### 4.4.7 Initialisation de l'application



Le serveur web / MVC est initialisé par la classe suivante du fichier [Global.asax.cs] :

```

1. using Afb.Dao;
2. using Afb.Web.Infrastructure;
3. using Afb.Web.Models;
4. using Afb.Web.Routes;
5. using Spring.Context.Support;
6. using System;
7. using System.Web.Mvc;
8. using System.Web.Routing;
9.
10. namespace Afb.Web {
11.     public class Global : System.Web.HttpApplication {
12.         protected void Application_Start() {
13.             // configuration MVC
14.             AreaRegistration.RegisterAllAreas();
15.             RouteConfig.RegisterRoutes(RouteTable.Routes);
16.             // instanciation modèle de l'application via Spring
17.             ApplicationModel application = ContextRegistry.GetContext().GetObject("application") as ApplicationModel;
18.             // il est mis dans les données de portée [Application]
19.             Application["data"] = application;
20.             // ModelBinder de la classe [ParentViewModel]
21.             ModelBinders.Binders.Add(typeof(ParentViewModel), new ParentModelBinder());

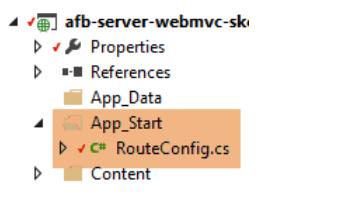
```

```

22.     }
23.
24.     // session
25.     protected void Session_Start(object sender, EventArgs e) {
26.         // le panier
27.         SessionModel session = new SessionModel() {...};
28.         Session["data"] = session;
29.     }
30. }
31. }
```

Ce code est laissé à votre compréhension. Du code analogue a été vu en TD. Selon votre définition de la classe [SessionModel], vous aurez peut-être à mettre du code ligne 27 pour initialiser l'instance [SessionModel].

#### 4.4.8 Routage de l'application



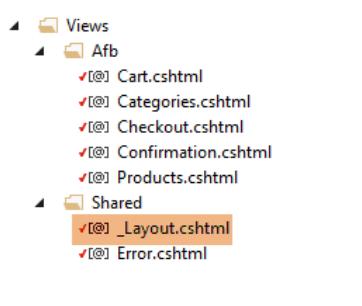
La classe [RouteConfig] qui fait le routage des URL entrantes vers les méthodes des contrôleurs chargées de les traiter est la suivante :

```

1. using System.Web.Mvc;
2. using System.Web.Routing;
3.
4. namespace Afb.Web.Routes {
5.     public class RouteConfig {
6.         public static void RegisterRoutes(RouteCollection routes) {
7.             routes.IgnoreRoute("{resource}.axd/{*pathInfo}");
8.
9.             routes.MapRoute(
10.                 name: "Default",
11.                 url: "{controller}/{action}/{id}",
12.                 defaults: new { controller = "Afb", action = "Categories", id = UrlParameter.Optional }
13. );
14.             routes.MapRoute(
15.                 name: "R1",
16.                 url: "Afb/UpdateQuantity/{productId}"
17. );
18.             routes.MapRoute(
19.                 name: "R2",
20.                 url: "Afb/Checkout"
21. );
22.     }
23. }
24. }
```

Ce code est laissé à votre compréhension. Il définit trois routes associées à différentes URL.

#### 4.4.9 Le modèle des vues



Les différentes vues de l'application sont construites à partir du modèle [\_Layout.cshtml] suivant :

```
1. @model Afb.Web.Models.ParentViewModel
2. @{
3.     var cart = new ShoppingCart();
4. }
5. <!DOCTYPE html>
6. <html>
7. <head>
8.     <meta charset="utf-8" />
9.     <meta name="viewport" content="width=device-width" />
10.    <title>L'épicerie verte</title>
11.    <!-- feuille de style -->
12.    <link rel="stylesheet" href("~/Content/css/affablebean.css" />
13.    <!-- favicon -->
14.    <link rel="shortcut icon" href("~/Content/images/favicon.ico" />
15.    <!-- script Js -->
16.    <script src "~/Content/js/jquery-1.4.2.js" type="text/javascript"></script>
17.
18.    <!-- scripts locaux -->
19.    <script type="text/javascript">
20.        $(document).ready(function () {
21.            $('#a.categoryButton').hover(
22.                function () { $(this).animate({ backgroundColor: '#b2d2d2' }) },
23.                function () { $(this).animate({ backgroundColor: '#d3ede8' }) }
24.            );
25.
26.            $('div.categoryBox').hover(over, out);
27.
28.            function over() {
29.                var span = this.getElementsByTagName('span');
30.                $(span[0]).animate({ opacity: 0.3 });
31.                $(span[1]).animate({ color: 'white' });
32.
33.            }
34.
35.            function out() {
36.                var span = this.getElementsByTagName('span');
37.                $(span[0]).animate({ opacity: 0.7 });
38.                $(span[1]).animate({ color: '#444' });
39.            }
40.        });
41.    </script>
42.
43. </head>
44. <body>
45.     <!-- header-->
46.     <div id="main">
47.         <div id="header">
48.             <div id="widgetBar">
49.                 <!-- validation commande -->
50.                 @if (cart.NumberOfItems > 0)
51.                 {
52.                     <div class="headerWidget">
53.                         @Html.ActionLink("passer au paiement", "ProceedCheckout", "Afb", new { @class = "bubble" })
54.                     </div>
55.                 }
56.                 <!-- panier -->
57.                 <div class="headerWidget" id="viewCart">
58.                     <!-- image -->
59.                     <img src "~/Content/images/cart.gif" alt="shopping cart icon" id="cart" />
60.                     @if (cart.NumberOfItems == 1)
61.                     {
62.                         <!-- 1 article dans le panier -->
63.                         <div id="viewCart2">
64.                             <span class="horizontalMargin">
65.                                 1 article
66.                             </span>
67.                             @Html.ActionLink("voir le panier", "ShowCart", "Afb", new { @class = "bubble" })
68.                         </div>
69.                     }
70.                     @if (cart.NumberOfItems > 1)
71.                     {
72.                         <!-- plusieurs articles dans le panier -->
```

```

73.      <div id="viewCart3">
74.          <label class="horizontalMargin" id="labelNumberOfItems">
75.              @(cart.NumberOfItems)
76.          </label>
77.          articles
78.          @Html.ActionLink("voir le panier", "ShowCart", "Afb", new { @class = "bubble" })
79.      </div>
80.  }
81. </div>
82. </div>
83. <!-- les images -->
84. <a href="/Afb/Categories">
85.     <img src "~/Content/images/logo.png" id="logo" alt="Affable Bean logo" />
86. </a>
87.     <img src "~/Content/images/logoText.png" id="logoText" alt="the affable bean" />
88. </div>
89. </div>
90. <!-- contenu -->
91. @RenderBody()
92. <!-- footer -->
93. <div id="footer">
94.     <br />
95.     <hr id="footerDivider" />
96.     <p id="footerText" class="reallySmallText">
97.         <a href="#">Politique interne</a>
98.         &nbsp;&nbsp;::&nbsp;
99.         <a href="#">Contact</a>
100.        &nbsp;&nbsp;&copy;&nbsp;&nbsp;
101.        2014 L'épicerie verte
102.    </p>
103. </div>
104.</body>
105.</html>

```

- ligne 1 : la page template a pour modèle la classe [ParentViewModel] définie au paragraphe 4.4.6, page 412. **Comme cette page est la page parent de toutes les vues, cela nous oblige à utiliser pour celles-ci, le modèle [ParentViewModel] ou une classe dérivée ;**
- lignes 2-4 : la variable [cart] désigne le chariot de l'utilisateur. Pour l'instant, cette variable est initialisée avec un chariot vide. **Vous aurez à changer cette initialisation** pour que le chariot utilisé soit celui de l'utilisateur ;
- ligne 12 : l'application web utilise une feuille de style complexe que nous utiliserons sans l'expliquer ;
- ligne 14 : l'image [favicon] qui sera affichée par le navigateur client ;
- ligne 16 : certaines animations de l'application nécessitent du Javascript (lignes 18-41) ;
- lignes 45-89 : l'en-tête présent sur toutes les vues ;
- lignes 84-86 : on notera que l'image est un lien. Celui-ci ramène dans la vue des catégories de produits en vente ;
- le modèle inclut un en-tête prédéfini (ligne 6), un corps qui va changer avec chaque vue (ligne 9), un bas de page prédéfini (ligne 11) ;
- ligne 91 : la vue particulière affichée dans le patron des vues ;
- lignes 93-103 : le bas de page ;

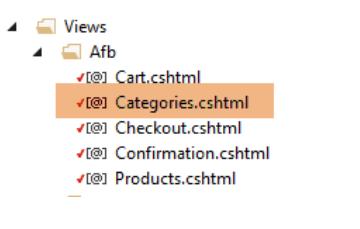
La vue [Categories.cshtml] affichée au démarrage de l'application est par exemple la suivante :



- en [1], l'entête, en [2] le corps, en [3] le bas de page.

Votre travail sur les vues va consister à compléter celles qui vous sont données et à écrire les méthodes du contrôleur qui leur fournissent leurs modèles.

#### 4.4.10 La page d'accueil [Categories.cshtml]



Lorsqu'on lance l'application (F5 ou Ctrl-F5), on obtient la vue suivante :



Bienvenue à l'Epicerie Verte

Notre service de livraison à domicile vous livre des produits frais, des produits laitiers, des viandes, des pains et autres produits délicieux et sains.

[Politique interne](#) :: [Contact](#) © 2014 L'épicerie verte

Nous avons présenté au paragraphe 4.4.8, page 414, le routage des URL vers leurs méthodes de traitement. Il apparaît que lorsque l'utilisateur demande l'URL /, c'est la méthode [Categories] du contrôleur [AfbController] qui est exécutée. Celle-ci est la suivante :

```
1.      [HttpGet]
2.      public ViewResult Categories(ParentViewModel modèle)
3.      {
4.          // affichage vue des catégories
5.          return View("Categories", modèle);
6.      }
```

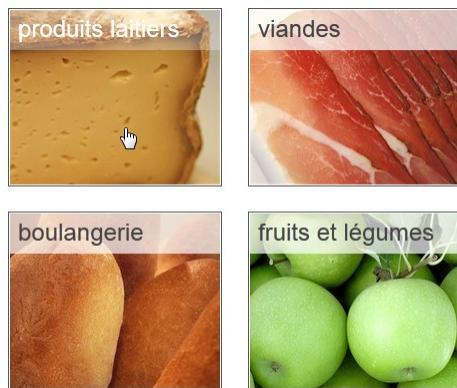
La vue [Categories.cshtml] est pour l'instant la suivante :

```
1. @model Afb.Web.Models.ParentViewModel
2. @using Afb.EF;
3.
4. <!-- les catégories d'articles - vue n° 0-->
5. <!-- colonne gauche -->
6. <div id="indexLeftColumn">
7.     <div id="welcomeText">
8.         <p style="font-size: larger">Bienvenue à l'Epicerie Verte</p>
9.         <p>Notre service de livraison à domicile vous livre des produits frais, des produits laitiers, des
viandes, des pains et autres produits délicieux et sains.</p>
10.    </div>
11. </div>
12. <!-- colonne droite -->
13. <div id="indexRightColumn">
14.     @foreach (var category in new List<Category>())
15.     {
16.         <div class="categoryBox">
17.             <a href="/Afb/Products/@category.Id">
18.                 <span class="categoryLabel">
19.                     <span class="categoryLabelText">
20.                         @category.Name
21.                     </span>
22.                 </span>
23.                 <img src("~/Content/images/categories/@(category.Name).jpg"
24.                      alt="@category.Name" class="categoryImage" />
25.             </a>
26.         </div>
```

```
27. }
28. </div>
```

Les pages [cshtml] de cette application sont essentiellement constituées de mises en forme à l'aide d'une feuille de style. On supposera que celle-ci a été créée par un designer. Il a laissé des zones blanches pour les développeurs (ligne 14). Notre travail est de remplir celles-ci.

Il faut se repérer dans le code. Les lignes 14-27 affichent les images des catégories de produits en vente.



Chaque image est un lien. Elles sont générées par le code suivant :

```
1.     <a href="/Afb/Products/@category.Id">
2.         <span class="categoryLabel">
3.             <span class="categoryLabelText">
4.                 @category.Name
5.             </span>
6.         </span>
7.         <img src("~/Content/images/categories/(@category.Name).jpg"
8.               alt="@category.Name" class="categoryImage" />
9.     </a>
```

A noter :

- ligne 1 : lorsque l'une des images est cliquée, l'URL [/Afb/Products/@category.Id] est demandée ;

---

**Question 1 :** la ligne 14 parcourt pour l'instant une liste vide. Modifiez-la pour qu'elle parcourt la liste des catégories de produits en vente.

---

**Conseils :**

- aidez-vous du modèle de la vue, ligne 1 ;
- on supposera dans tout l'exercice qu'il n'y a pas d'exceptions. Ce n'est pas réaliste mais cela va simplifier l'exercice ;

Vérifiez que maintenant, vous obtenez la page suivante lorsque vous lancez l'application :



#### 4.4.11 La page des produits vendus [Products.cshtml]

Lorsque l'une des images de la vue [Categories.cshtml] est cliquée, l'URL [\[/Afb/Products/@category.Id"\]](#) est demandée. Le routage du paragraphe 4.4.8, page 414, indique que cette URL est traitée par la méthode [Products] du contrôleur :

```

1.      [HttpGet]
2.      public ViewResult Products(ParentViewModel modèle, int id)
3.      {
4.          // on affiche la vue des produits
5.          throw new NotImplementedException("méthode [Products] pas encore implémentée");
6.      }

```

**Question 2 :** écrire la méthode précédente.

##### Spécifications et conseils :

- la vue à afficher est [Products.cshtml]. Son modèle sera [ParentViewModel] ;
- la vue [Products.cshtml] affichera les produit de la catégorie n° id (ligne 2) ;

Lorsque l'utilisateur a choisi une catégorie, les produits de celle-ci sont affichés :

	Nom	Prix	
	milk	1,70 €	<a href="#">ajouter au panier</a>
	cheese	2,39 €	<a href="#">ajouter au panier</a>
	butter	1,09 €	<a href="#">ajouter au panier</a>
	free range eggs	1,76 €	<a href="#">ajouter au panier</a>

La zone [1] est produite par la page [Products.cshtml] suivante :

```

Afb
  ✓[@] Cart.cshtml
  ✓[@] Categories.cshtml
  ✓[@] Checkout.cshtml
  ✓[@] Confirmation.cshtml
  ✓[@] Products.cshtml

```

```

1. @model Afb.Web.Models.ParentViewModel
2. @using Afb.EF;
3.
4. <!-- les articles - vue n° 1 -->
5. <!-- colonne gauche -->
6. <div id="categoryLeftColumn">
7.     @{
8.         List<Category> categories = new List<Category>();
9.         foreach (var category in categories)
10.        {
11.            if (category.Id == 0)
12.            {
13.                <div class="categoryButton" id="selectedCategory">
14.                    <span class="categoryText">
15.                        @category.Name
16.                    </span>
17.                </div>
18.            }
19.            else
20.            {
21.                <a class="categoryButton" href="">
22.                    <span class="categoryText">
23.                        @category.Name
24.                    </span>
25.                </a>
26.            }
27.        }
28.    }
29. </div>
30. <!-- colonne droite -->
31. <div id="categoryRightColumn">
```

```

32.      <table>
33.        <thead>
34.          <tr>
35.            <th></th>
36.            <th>Nom</th>
37.            <th>Prix</th>
38.            <th></th>
39.          </tr>
40.        </thead>
41.        <tbody>
42.          @{
43.            List<Product> products = new List<Product>();
44.            for (int i = 0; i < products.Count; i++)
45.            {
46.              var classe = i % 2 == 0 ? "lightBlue" : "white";
47.              var product = products[i];
48.              <tr class="@classe">
49.                <td><img src("~/Content/images/products/@(product.Name).png" /></td>
50.                <td style="width:200px">@product.Name</td>
51.                <td style="width:100px">???</td>
52.                <td style="width:200px">@Html.ActionLink("ajouter au panier", "AddToCart", new { id =
product.Id })</td>
53.              </tr>
54.            }
55.          }
56.        </tbody>
57.      </table>
58.    </div>

```

- lignes 5-29 : reproduisent l'affichage des catégories avec un style différent. En termes de fonctionnalités, il n'y a rien de nouveau dans ce code ;
- lignes 11-28 : traitent différemment la catégorie dont la vue affiche les produits et les autres.
  - lignes 11-18 : traitent le cas de la catégorie dont la vue affiche les produits. On affiche alors un texte non cliquable. Ligne 11, il vous faudra remplacer le 0 par le n° de la catégorie dont la vue affiche actuellement les produits ;
  - lignes 21-28 : affichent un texte cliquable pour les autres catégories. Ligne 21, il vous faudra mettre l'URL demandée à la suite d'un clic sur le lien ;
- lignes 31-57 : affichent les produits de la catégorie sélectionnée par l'utilisateur dans une étape précédente ;

**Question 3 :** Mettez le code attendu aux lignes 8, 11, 21, 43 et 51.

---

**Vérification :** vérifiez que vous êtes maintenant capable d'obtenir les produits des catégories lorsque vous cliquez sur l'image de l'une d'elles ou que vous cliquez sur les textes cliquables de la colonne de gauche.

#### 4.4.12 La méthode [AddToCart] du contrôleur



	Nom	Prix	
	milk	1,70 €	<a href="#">ajouter au panier</a>
	cheese	2,39 €	<a href="#">ajouter au panier</a>
	butter	1,09 €	<a href="#">ajouter au panier</a>
	free range eggs	1,76 €	<a href="#">ajouter au panier</a>

[Politique Interne](#) :: [Contact](#) © 2014 L'épicerie verte



	Nom	Prix	
	milk	1,70 €	<a href="#">ajouter au panier</a>
	cheese	2,39 €	<a href="#">ajouter au panier</a>
	butter	1,09 €	<a href="#">ajouter au panier</a>
	free range eggs	1,76 €	<a href="#">ajouter au panier</a>

[Politique Interne](#) :: [Contact](#) © 2014 L'épicerie verte

Dès que le chariot a au moins un article, le lien [1] apparaît.

Dans la page [Products.cshtml], l'utilisateur peut cliquer sur un produit pour l'ajouter à son panier. Cette action est gérée par le code suivant de la vue :

```

1.      @{
2.          List<Product> products = new List<Product>();
3.          for (int i = 0; i < products.Count; i++)
4.          {
5.              var classe = i % 2 == 0 ? "lightBlue" : "white";
6.              var product = Model.Session.Products[i];
7.              <tr class="@classe">

```

```

8.          <td></td>
9.          <td style="width:200px">@product.Name</td>
10.         <td style="width:100px">???</td>
11.         <td style="width:200px">@Html.ActionLink("ajouter au panier", "AddToCart", new { id =
    product.Id })</td>
12.      </tr>
13.    }
14. }

```

Ligne 11 [products.xhtml], on voit qu'un clic sur le lien [ajouter au panier] provoque l'exécution de la méthode [AddToCart] du contrôleur. Celle-ci est la suivante :

```

1.      [HttpGet]
2.      public ViewResult AddToCart(...)
3.      {
4.          // ...
5.
6.          // on affiche la vue des produits
7.          throw new NotImplementedException("méthode [AddToCart] pas encore implémentée");
8.      }

```

**Question 4** : écrire la méthode *AddToCart*.

**Conseils :**

- il vous faut déterminer les paramètres de la méthode, ligne 2. Ceux-ci peuvent être des modèles instanciés par des [ModelBinder], des éléments de l'URL d'un GET ou POST, des valeurs postées ;
- regardez le code du lien qui amène à cette méthode pour savoir ce qu'elle va recevoir comme paramètres (ligne 2) ;
- la méthode doit ajouter un article dans le chariot de l'acheteur ;
- ligne 6 : la vue à afficher est toujours [Products.cshtml] car l'ajout au panier d'un produit ne change pas la page affichée ;

Revenons à ce qui a été dit plus haut :

Nom	Prix	
milk	1,70 €	<a href="#">ajouter au panier</a>
cheese	2,39 €	<a href="#">ajouter au panier</a>
butter	1,09 €	<a href="#">ajouter au panier</a>
free range eggs	1,76 €	<a href="#">ajouter au panier</a>

Dès que le chariot a au moins un article, le lien [1] apparaît. Ce lien est gouverné par le code suivant du template [\_Layout.cshtml] :

```

1. @model Afb.Web.Models.ParentViewModel
2. @{
3.     var cart = new Afb.Dao.ShoppingCart();
4. }
5. <!DOCTYPE html>
6. <html>
7. <head>

```

```

8. ...
9.
10.
11. <div id="header">
12.   <div id="widgetBar">
13.     <!-- validation commande -->
14.     @if (cart.NumberOfItems > 0)
15.     {
16.       <div class="headerWidget">
17.         @Html.ActionLink("passer au paiement", "ProceedCheckout", "Afb", new { @class = "bubble" })
18.       </div>
19.     }
20.     <!-- panier -->
21.     <div class="headerWidget" id="viewCart">
22.       <!-- image -->
23.       <img src("~/Content/images/cart.gif" alt="shopping cart icon" id="cart" />
24.       @if (cart.NumberOfItems == 1)
25.       {
26.         <!-- 1 article dans le panier -->
27.         <div id="viewCart2">
28.           <span class="horizontalMargin">
29.             1 article
30.           </span>
31.           @Html.ActionLink("voir le panier", "ShowCart", "Afb", new { @class = "bubble" })
32.         </div>
33.       }
34.       @if (cart.NumberOfItems > 1)
35.       {
36.         <!-- plusieurs articles dans le panier -->
37.         <div id="viewCart3">
38.           <label class="horizontalMargin" id="labelNumberOfItems">
39.             @(cart.NumberOfItems)
40.           </label>
41.           articles
42.           @Html.ActionLink("voir le panier", "ShowCart", "Afb", new { @class = "bubble" })
43.         </div>
44.       }
45.     </div>
46.   </div>
47. ...

```

Lignes 14, 24, 34, le template [\_Layout.cshtml] affiche des informations dépendant du nombre d'articles dans le chariot. Ligne 3, ce chariot est actuellement initialisé avec un chariot vide. Donc pour l'instant, aucun lien n'est affiché.

---

**Question 5 :** mettre ce qui est nécessaire ligne 3, pour que le chariot utilisé soit celui de l'acheteur.

**Vérification :** vérifiez que le lien [voir le panier] apparaît lorsque vous ajoutez au chariot votre premier article.

#### 4.4.13 La page du panier [Cart.cshtml]

Lorsque l'utilisateur a au moins un article dans son panier, celui-ci peut alors être visualisé :



Le code qui affiche le lien [1] est le suivant dans [\_Layout.cshtml] :

```
1. @Html.ActionLink("voir le panier", "ShowCart", "Afb", new { @class = "bubble" })
```

C'est donc la méthode [ShowCart] du contrôleur [Afb] qui est exécutée lors du clic sur le lien. Cette méthode est la suivante :

```

1.     [HttpGet]
2.     public ViewResult ShowCart(...)
3.     {
4.         // on affiche la vue du chariot
5.         throw new NotImplementedException("méthode [ShowCart] pas encore implémentée");
6.     }

```

**Question 6 :** écrire la méthode *ShowCart*.

Le panier est affiché de la façon suivante :

article	nom	prix	quantité
	lait	1,70 € (1,70 € unité)	<input type="button" value="valider"/> 1 <input type="button" value="supprimer"/>
	fromage	2,39 € (2,39 € unité)	<input type="button" value="valider"/> 1 <input type="button" value="supprimer"/>

La zone [1] est générée par le code [Cart.cshtml] suivant :

```

1. @model Afb.Web.Models.ParentViewModel
2. @using Afb.Ef;
3. @using Afb.Dao;
4.
5. @{
6.     var cart = new ShoppingCart();
7. }
8. <!-- le panier - vue n° 2 -->
9. <div id="singleColumn">
10.    @if (cart.NumberOfItems > 1)
11.    {
12.        <p>Votre chariot contient @(cart.NumberOfItems) articles.</p>
13.    }
14.    @if (cart.NumberOfItems == 1)
15.    {
16.        <p>Votre chariot contient 1 article.</p>
17.    }
18.    @if (cart.NumberOfItems == 0)
19.    {
20.        <p>Votre chariot est vide.</p>
21.    }
22.    <div id="ActionBar">
23.        <!-- clear cart widget -->
24.        @Html.ActionLink("vider le panier", "ClearCart", "Afb", new { @class = "bubble hMargin" })
25.        <!-- continue shopping widget -->
26.        @Html.ActionLink("continuer les achats", "GoOnShopping", "Afb", new { @class = "bubble hMargin" })
27.    </div>
28.    @if (cart.NumberOfItems != 0)
29.    {
30.        <h4 id="subtotal">

```

```

31.      sous-total
32.      <label>??</label>
33.    </h4>
34.    <table style="width:100%">
35.      <thead>
36.        <tr>
37.          <th>Article</th>
38.          <th>Nom</th>
39.          <th>Prix</th>
40.          <th>Quantité</th>
41.          <th></th>
42.        </tr>
43.      </thead>
44.      <tbody>
45.        @{
46.          List<ShoppingCartItem> items = new List<ShoppingCartItem>();
47.          var i=0;
48.          foreach (var item in items)
49.          {
50.            var classe = i % 2 == 0 ? "lightBlue" : "white";
51.            <tr class="@classe">
52.              <td></td>
53.              <td>@(item.Product.Name)</td>
54.              <td>??</td>
55.              @using (@Html.BeginForm("UpdateQuantity", "Afb", new { productId = @item.Product.Id }))
56.              {
57.                <td>
58.                  <input type="text" name="quantity" size="3" value="@item.Quantity" />
59.                </td>
60.                <td>
61.                  <input type="submit" value="modifier" />
62.                </td>
63.              }
64.            </tr>
65.            i++;
66.          }
67.        }
68.      </tbody>
69.    </table>
70.  }
71. </div>

```

Faites le lien entre ce code et l'affichage produit.

---

**Question 7 :** mettez le code correct aux lignes 6, 32, 46 et 54.

---

**Vérification :**

- vérifiez que vous pouvez maintenant afficher le contenu du chariot ;

#### 4.4.14 Le lien [continuer les achats]

De la vue du chariot, on peut revenir à la vue des produits pour continuer les achats :



Votre chariot contient 1 article.

[vider le panier](#) [continuer les achats](#)

sous-total 1,70 €

Article	Nom	Prix	Quantité	
	milk	1,70 € (1,70 € l'unité)	1	<a href="#">modifier</a>

[Politique interne](#) :: [Contact](#) © 2014 L'épicerie verte

Dans la vue [Cart.cshtml], le lien est généré de la façon suivante :

```
1. <!-- continue shopping widget -->
2. @Html.ActionLink("continuer les achats", "GoOnShopping", "Afb", new { @class = "bubble hMargin" })
```

---

**Question 8** : écrire la méthode du contrôleur qui va traiter le clic sur ce lien.

---

**Vérification** : vérifiez que vous pouvez continuer les achats.

#### 4.4.15 La méthode [UpdateQuantity] du contrôleur

Lorsque le panier est affiché, l'utilisateur peut modifier les quantités achetées :



Votre chariot contient 1 article.

[vider le panier](#) [continuer les achats](#)

sous-total 1,70 €

Article	Nom	Prix	Quantité	
	milk	1,70 € (1,70 € l'unité)	1	<a href="#">modifier</a>

[Politique interne](#) :: [Contact](#) © 2014 L'épicerie verte

Votre chariot contient 2 articles.

Article	Nom	Prix	Quantité
	milk	3,40 € (1,70 € l'unité)	2 <input type="button" value="modifie"/>

Politique intime :: Contact © 2014 L'épicerie verte

Le code concerné dans la vue [Cart.cshtml] est le suivant :

```

1.     foreach (var item in items)
2.     {
3.         var classe = i % 2 == 0 ? "lightBlue" : "white";
4.         <tr class="@classe">
5.             <td><img src "~/Content/images/products/@(item.Product.Name).png" /></td>
6.             <td>@(item.Product.Name)</td>
7.             <td>??</td>
8.             @using (@Html.BeginForm("UpdateQuantity", "Afb", new { productId = @item.Product.Id }))
9.             {
10.                 <td>
11.                     <input type="text" name="quantity" size="3" value="@item.Quantity" />
12.                 </td>
13.                 <td>
14.                     <input type="submit" value="modifier" />
15.                 </td>
16.             }
17.         </tr>
18.         i++;
19.     }

```

- lignes 8-16 : pour chaque article, on a un formulaire autour de la zone qui affiche la quantité achetée et qui est également une zone de saisie afin que cette quantité puisse être modifiée ;
- ligne 8 : la méthode du contrôleur appelée lorsque l'utilisateur clique sur le bouton [modifier] est la méthode [UpdateQuantity] suivante :

```

1.     [HttpPost]
2.     public ViewResult UpdateQuantity(...)
3.     {
4.         // on affiche la vue du chariot
5.         throw new NotImplementedException("méthode [UpdateQuantity] pas encore implémentée");
6.     }

```

---

**Question 9** : écrire la méthode [UpdateQuantity].

---

**Conseils :**

- il vous faut déterminer les paramètres de la méthode, ligne 2. Ceux-ci peuvent être des modèles instanciés par des [ModelBinder], des éléments de l'URL d'un GET ou POST, des valeurs postées ;
- la méthode change la quantité d'un article particulier du chariot ;
- après l'action [UpdateQuantity], la vue affichée est de nouveau [Cart.cshtml].

**Vérification :**

- vérifiez que vous êtes maintenant capable de changer la quantité achetée d'un produit grâce au bouton [modifier] :

#### 4.4.16 La page de paiement [Checkout.cshtml]

Lorsqu'il a au moins un article dans son panier, l'utilisateur peut passer au paiement de ses achats :



Le code du lien [1] dans [\_Layout.cshtml] est le suivant :

```
1. <div class="headerWidget">
2.     @Html.ActionLink("passer au paiement", "ProceedCheckout", "Afb", new { @class = "bubble" })
3. </div>
```

---

Question 10 : écrire la méthode qui traite le clic sur le lien ci-dessous.

---

Conseils :

- la vue à afficher est [Checkout.cshtml] ;

La page [Checkout.cshtml] affichée est la suivante :

1

**paiement**

Pour le paiement de vos achats, veuillez nous donner les informations suivantes :

nom	<input type="text"/>
adresse électronique	<input type="text"/>
téléphone	<input type="text"/>
adresse	<input type="text"/>
carte de crédit	<input type="text"/>
<input type="button" value="valider"/>	

• Livraison J+1 garantie  
• A 2,80 € le surcoût de livraison est appliqué à tous les articles

sous-total : 4,08 €
coût livraison : 2,80 €
total : 6,88 €

La zone [1] ci-dessus est générée par le code [Checkout.cshtml] suivant :

```
1. @model Afb.Web.Models.ParentViewModel
2. @using Afb.Web;
3. @using Afb.Ef;
4.
5. <div id="Div1">
6.     <h2>paiement</h2>
7.     <p>Pour le paiement de vos achats, veuillez nous donner les informations suivantes :</p>
8.     @using(Html.BeginForm("Checkout", "Afb", "post")){
9.         <table>
10.            <tr>
11.                <td>
12.                    <table style="width: 405px; padding: 10px; background-color: #f5eabe; float: left; border-radius: 4px;">
13.                        <tr>
14.                            <td>
15.                                nom
```

```

16.          </td>
17.          <td>
18.              <input type="text" name="Name" size="31" maxlength="45" class="inputField" />
19.          </td>
20.      </tr>
21.      <tr>
22.          <td />
23.          <td>
24.              @Html.ValidationMessage("Name", new { style = "color:red" })
25.          </td>
26.      </tr>
27.      <tr>
28.          <td>
29.              adresse électronique
30.          </td>
31.          <td>
32.              <input type="text" name="Email" size="31" maxlength="45" class="inputField" />
33.          </td>
34.      </tr>
35.      <tr>
36.          <td />
37.          <td>
38.              @Html.ValidationMessage("Email", new { style = "color:red" })
39.          </td>
40.      </tr>
41.      <tr>
42.          <td>
43.              téléphone
44.          </td>
45.          <td>
46.              <input type="text" name="Phone" size="31" maxlength="45" class="inputField" />
47.          </td>
48.      </tr>
49.      <tr>
50.          <td />
51.          <td>
52.              @Html.ValidationMessage("Phone", new { style = "color:red" })
53.          </td>
54.      </tr>
55.      <tr>
56.          <td>
57.              adresse
58.          </td>
59.          <td>
60.              <input type="text" name="Address" size="31" maxlength="45" class="inputField" />
61.          </td>
62.      </tr>
63.      <tr>
64.          <td />
65.          <td>
66.              @Html.ValidationMessage("Address", new { style = "color:red" })
67.          </td>
68.      </tr>
69.      <tr>
70.          <td>
71.              carte de crédit
72.          </td>
73.          <td>
74.              <input type="text" name="CcNumber" size="31" maxlength="19" class="inputField" />
75.          </td>
76.      </tr>
77.      <tr>
78.          <td />
79.          <td>
80.              @Html.ValidationMessage("CreditCart", new { style = "color:red" })
81.          </td>
82.      </tr>
83.      <tr>
84.          <td>
85.              <input type="submit" id="valider" value="valider" />
86.          </td>
87.      </tr>
88.  </table>
89. </td>
90. <td>
```

```

91.         <ul style="text-align: left">
92.             <li>Livraison J+1 garantie</li>
93.             <li>
94.                 A
95.                 <label class="checkoutPriceColumn">??</label>
96.                 le surcoût de livraison est appliquée à tous les articles
97.             </li>
98.         </ul>
99.         <table style="padding: 10px; border: 1px solid #c3e3e0; background-color: #c3e3e0; border-
radius: 4px;">
100.            <tr>
101.                <td>
102.                    sous-total :
103.                </td>
104.                <td>
105.                    <label class="checkoutPriceColumn">??</label>
106.                </td>
107.            </tr>
108.            <tr>
109.                <td>
110.                    coût livraison :
111.                </td>
112.                <td>
113.                    <label class="checkoutPriceColumn">??</label>
114.                </td>
115.            </tr>
116.            <tr>
117.                <td>
118.                    total :
119.                </td>
120.                <td>
121.                    <label class="checkoutPriceColumn">??</label>
122.                </td>
123.            </tr>
124.        </table>
125.    </td>
126.    </tr>
127. </table>
128. }
129. </div>

```

Faites le lien entre ce code et ce qu'il affiche.

---

**Question 11 :** complétez les lignes 95, 105, 113, 121.

---

**Vérification :**

- vérifiez qu'après avoir acheté quelques produits, vous êtes capable d'afficher la page de paiement ;

#### 4.4.17 La méthode [Checkout] du contrôleur

La vue [Checkout.cshtml] présente le formulaire suivant :

```

1. @using(Html.BeginForm("Checkout","Afb","post")){
2.     <table>
3.         <tr>
4.             <td>
5.                 <table style="width: 405px; padding: 10px; background-color: #f5eabe; float: left; border-
radius: 4px;">
6.                     ...
7.                     <tr>
8.                         <td>
9.                             <input type="submit" id="valider" value="valider" />
10.                        </td>
11.                    </tr>
12.                </table>
13.            </td>

```

Un clic sur le bouton [valider] de la ligne 9 provoque une requête vers l'URL définie par les éléments de la ligne 1.

---

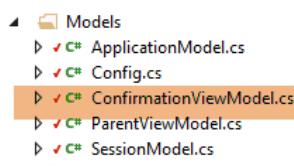
**Question 12 :** écrire la méthode qui va traiter les valeurs postées par le formulaire.

---

## Conseils :

- la méthode doit enregistrer en base la commande de l'acheteur. Si cet enregistrement se passe bien, le chariot est vidé et la vue [Confirmation] affichée. Si elle ne se passe pas bien, un message d'erreur doit être affiché sur la vue [Checkout] ;
- lors de l'enregistrement de la commande, les couches basses lancent une exception si l'acheteur existe déjà en base ;

La vue [Confirmation.cshtml] attend un modèle de type suivant :



```
1. using Afb.Dao;
2. using Afb.Ef;
3. using System.Collections.Generic;
4.
5. namespace Afb.Web.Models {
6.     public class ConfirmationViewModel : ParentViewModel {
7.         // le client
8.         public CustomerOrder CustomerOrder { get; set; }
9.         // les articles achetés
10.        public List<ShoppingCartItem> Items { get; set; }
11.    }
12. }
```

### 4.4.18 La page de confirmation [Confirmation.cshtml]

Lorsque l'utilisateur a validé le paiement de son panier, une page de confirmation est affichée :



La zone [1] est générée par le code [Confirmation.cshtml] suivant :

```
1. @model Afb.Web.Models.ConfirmationViewModel
2.
3. @using Afb.Ef;
4. @using Afb.Dao;
5.
6. <p id="confirmationText">
```

```

7.      <strong>Votre commande a été traitée et sera livrée dans les 24 h.</strong>
8.      <br />
9.      <br />
10.     Veuillez conserver votre n° de confirmation :
11.     <strong>
12.         ???
13.     </strong>
14.     <br />
15.     Si vous avez une question concernant votre commande, veuillez nous <a href="#">contacter</a>.
16.     <br />
17.     <br />
18.     Merci de votre visite à l'Epicerie Verte.
19. </p>
20.
21. <div class="summaryColumn">
22.     <!-- tableau des articles achetés --&gt;
23.     &lt;table id="orderSummaryTable" class="detailsTable"&gt;
24.         &lt;thead&gt;
25.             &lt;tr class="header"&gt;
26.                 &lt;th colspan="3"&gt;résumé de la commande&lt;/th&gt;
27.             &lt;/tr&gt;
28.             &lt;tr class="tableHeading"&gt;
29.                 &lt;td&gt;article&lt;/td&gt;
30.                 &lt;td&gt;quantité&lt;/td&gt;
31.                 &lt;td&gt;prix&lt;/td&gt;
32.             &lt;/tr&gt;
33.         &lt;/thead&gt;
34.         @{
35.             List&lt;ShoppingCartItem&gt; items=new List&lt;ShoppingCartItem&gt;();
36.             foreach (var item in items)
37.             {
38.                 &lt;tr class="lightBlue"&gt;
39.                     &lt;td&gt;
40.                         @(item.Product.Name)
41.                     &lt;/td&gt;
42.                     &lt;td class="quantityColumn"&gt;
43.                         @(item.Quantity)
44.                     &lt;/td&gt;
45.                     &lt;td class="confirmationPriceColumn"&gt;
46.                         ???
47.                     &lt;/td&gt;
48.                 &lt;/tr&gt;
49.             }
50.         }
51.         &lt;tr class="lightBlue"&gt;
52.             &lt;td colspan="3" style="padding: 0 20px"&gt;
53.                 &lt;hr /&gt;
54.             &lt;/td&gt;
55.         &lt;/tr&gt;
56.
57.         &lt;tr class="lightBlue"&gt;
58.             &lt;td colspan="2" id="deliverySurchargeCellLeft"&gt;&lt;strong&gt;coût de livraison :&lt;/strong&gt;&lt;/td&gt;
59.             &lt;td id="deliverySurchargeCellRight"&gt;
60.                 ???
61.             &lt;/td&gt;
62.         &lt;/tr&gt;
63.
64.         &lt;tr class="lightBlue"&gt;
65.             &lt;td colspan="2" id="totalCellLeft"&gt;&lt;strong&gt;Total :&lt;/strong&gt;&lt;/td&gt;
66.             &lt;td id="totalCellRight"&gt;
67.                 ???
68.             &lt;/td&gt;
69.         &lt;/tr&gt;
70.
71.         &lt;tr class="lightBlue"&gt;
72.             &lt;td colspan="3" style="padding: 0 20px"&gt;
73.                 &lt;hr /&gt;
74.             &lt;/td&gt;
75.         &lt;/tr&gt;
76.
77.         &lt;tr class="lightBlue"&gt;
78.             &lt;td colspan="3" id="dateProcessedRow"&gt;
79.                 &lt;strong&gt;date de traitement :&lt;/strong&gt;
80.                 @(Model.CustomerOrder.DateCreated)
81.             &lt;/td&gt;
</pre>

```

```

82.      </tr>
83.    </table>
84.  </div>
85. <div class="summaryColumn">
86.   <table class="detailsTable">
87.     <thead class="header">
88.       <tr class="header">
89.         <th><strong>Adresse de livraison</strong></th>
90.       </tr>
91.     </thead>
92.     <tbody>
93.       <tr>
94.         <td class="lightBlue">
95.           ???<br />???
96.           ???<br />???
97.           <br />
98.           <hr />
99.           <strong>Adresse mail :</strong>
100.          ???<br />???
101.          <strong>Téléphone :</strong>
102.          ???<br />???
103.        </td>
104.      </tr>
105.    </tbody>
106.  </table>
107. </div>
108. </div>
109.</div>

```

Faites le rapprochement entre ce code et ce qu'il affiche.

---

**Question 13 :** remplacer les codes ??? par le code attendu ;

---

**Conseils :**

- les informations affichées sur l'acheteur sont son nom et son adresse ;

**Vérification :**

- vérifiez que vous obtenez désormais une page de confirmation correcte ;

#### 4.4.19 Le lien [vider le chariot]




---

**Question 14 :** gérez le lien [1] ci-dessus.

---

## 5 Étude de cas n° 4 : gestion de rendez-vous

Références :

- [ref1] : "Introduction à l'ORM Entity Framework 5 Code First" [<http://tahe.developpez.com/dotnet/ef5cf/>];
- [ref2] : "Etude de cas avec ASP.NET 2.0, C#, Spring.NET et NHibernate" [<http://tahe.developpez.com/dotnet/pam-aspnet/>];

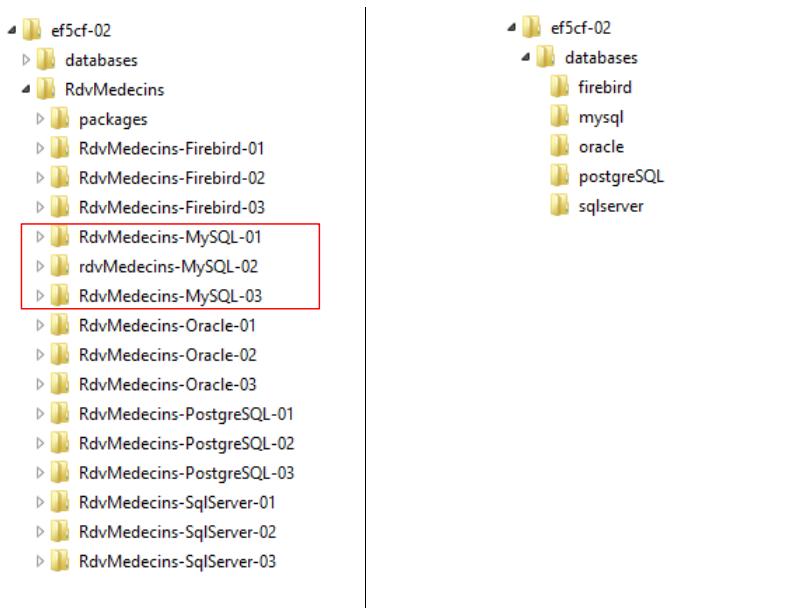
### 5.1 Introduction

Cette étude de cas vous laisse davantage de liberté que les précédentes de ce document.

#### 5.1.1 Le projet existant

Dans le document [ref1], une application web de gestion de rendez-vous de médecins est présentée. Celle-ci a été écrite avec le framework ASP.NET classique. Nous nous proposons de la réécrire avec le framework ASP.NET MVC.

Téléchargez les exemples du document "Introduction à l'ORM Entity Framework 5 Code First" [<http://1drv.ms/1ODTCda>]. Vous allez obtenir l'arborescence suivante :



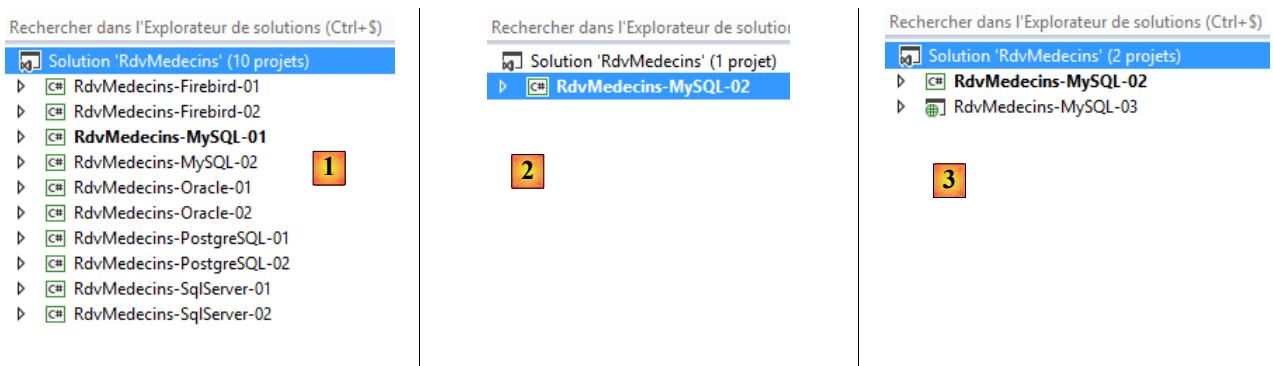
L'application web est déclinée pour cinq SGBD :

- SQL Server 2012 ;
- SGBD MySQL 5.5.28 ;
- SGBD Oracle Database Express Edition 11g Release 2 ;
- SGBD PostgreSQL 9.2.1 ;
- SGBD Firebird 2.1.

Nous allons nous intéresser à la version pour MySQL. Vous avez besoin des projets Visual Studio suivants :

- RdvMedecins-MySQL-02 ;
- RdvMedecins-MySQL-03.

Chargez la solution [RdvMedecins] du dossier [ef5cf-02/RdvMedecins] dans Visual Studio (VS) :



Cette solution [1] a été construite pour VS Express 2012 pour le bureau. Les projets web n'y sont pas. Supprimez tous les projets de cette solution et ne gardez que le projet [RdvMedecins-MySQL-02] [2]. Ajoutez à la solution le projet web [RdvMedecins-MySQL-03] [3].

La base de données est configurée dans le fichier [web.config] du projet [RdvMedecins-MySQL-03] par les lignes suivantes :

```

1.    <!-- chaîne de connexion-->
2.    <connectionStrings>
3.      <add name="monContexte"
4.        connectionString="Server=localhost;Database=rdvmedecins-ef;Uid=root;Pwd=root;" 
5.        providerName=" MySql.Data.MySqlClient " />
6.    </connectionStrings>

```

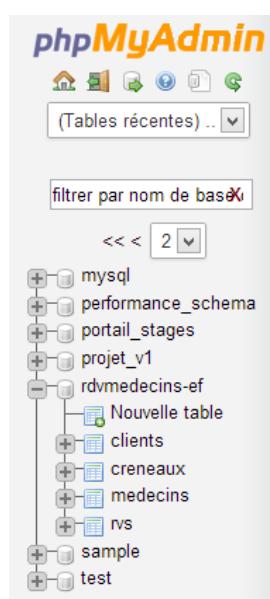
La ligne 4 définit la façon de se connecter à la base de données MySQL :

- **Server=localhost** : le SGBD est sur la même machine que le projet web ;
- **Database=rdvmedecins-ef** : elle s'appelle [rdvmedecins-ef] ;
- **Uid=root** : on se connecte avec le login [root] ;
- **Pwd=root** : et le mot de passe [root].

Modifiez la ligne 4 pour qu'il n'y ait pas de mot de passe :

```
connectionString="Server=localhost;Database=rdvmedecins-ef;Uid=root;Pwd="
```

Créez la base MySQL [rdvmedecins-ef] puis jouez le script SQL que vous trouverez dans le dossier [ef5cf-02/databases/mysql]. Prenez la version [with-data]. Vous allez obtenir la base suivante :

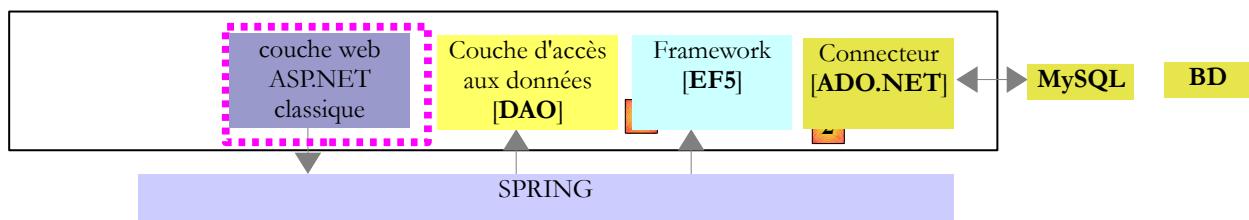


Exécutez le projet web [RdvMedecins-MySQL-03]. Vous devez obtenir la page suivante :

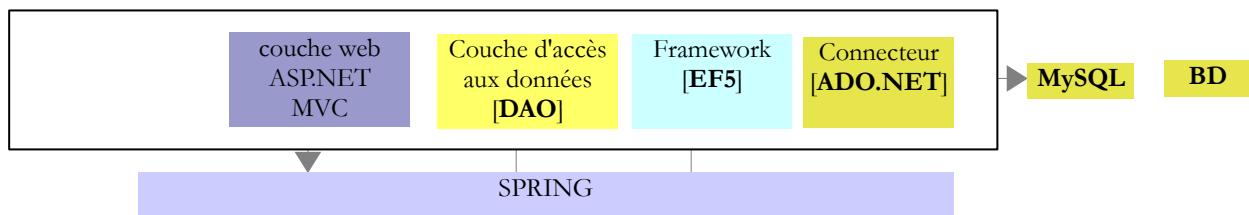
Jouez avec l'application pour comprendre son fonctionnement. Vous devrez le reproduire dans l'application web à construire. Pensez à consulter la table [rv] des rendez-vous. Elle enregistre les rendez-vous que vous prenez.

### 5.1.2 Méthodes de travail

Vous disposez d'une application web [RdvMedecins-MySQL-03] qui a l'architecture suivante :



Vous allez construire la nouvelle application [RdvMedecins-MySQL-04] qui aura l'architecture suivante :



Donc seule la couche [web] change. Vous pouvez ignorer le fonctionnement des couches [DAO] et [EF5]. Si vous souhaitez en savoir plus sur ces couches, il vous faut lire les quatre premiers chapitres de [ref1]. Si vous souhaitez comprendre le code de l'application web [RdvMedecins-MySQL-03], il vous faut lire le chapitre 1 de [ref2]. Cette étude de cas suit la démarche de celle du paragraphe 2, page 220.

## 5.2 Les vues de l'application web

L'application web réalisée avec ASP.NET MVC. Nous présentons ci-dessous des copies d'écran de son fonctionnement. La page d'accueil de l'application est la suivante :

RdvMedecins

localhost:53125

Applications Favoris Gestionnaire de favo...

## Cabinet Médical - Les Médecins Associés [Afficher l'agenda](#)

---

### Prise de rendez-vous

Médecin	Date
Mme Marie Pelissier	jj / mm / aaaa

A partir de cette première page, l'utilisateur (Secrétariat, Médecin) va engager un certain nombre d'actions. Nous les présentons ci-dessous. La vue de gauche présente la vue à partir de laquelle l'utilisateur fait une **demande**, la vue de droite la **réponse** envoyée par le serveur.

RdvMedecins

localhost:53125

Applications Favoris Gestionnaire de favo...

## Cabinet Médical - Les Médecins Associés [Afficher l'agenda](#)

---

### Prise de rendez-vous

Médecin	Date
Mme Marie Pelissier	21/11/2013

L'utilisateur a sélectionné un médecin et a saisi un jour de RV

RdvMedecins

localhost:53125

Applications Favoris Gestionnaire de favo...

## Cabinet Médical - Les Médecins Associés [Accueil](#)

---

### Rendez-vous de Mme Marie Pelissier le 21/11/2013

Créneau horaire	Client	Action
08h00-08h20		<a href="#">Réserver</a>
08h20-08h40		<a href="#">Réserver</a>
08h40-09h00		<a href="#">Réserver</a>
09h00-09h20		<a href="#">Réserver</a>
09h20-09h40		<a href="#">Réserver</a>
09h40-10h00		<a href="#">Réserver</a>
10h00-10h20		<a href="#">Réserver</a>
10h20-10h40		<a href="#">Réserver</a>

On obtient la liste (vue partielle ici) des RV du médecin sélectionné pour le jour indiqué.

**Cabinet Médical - Les Médecins**

Rendez-vous de Mme Marie Pelissier le 21/11/2013

Créneau horaire	Client	Action
08h00-08h20		<a href="#">Réserver</a>
08h20-08h40		<a href="#">Réserver</a>
08h40-09h00		<a href="#">Réserver</a>

On réserve

**Cabinet Médical - Les Médecins Associés**

Rendez-vous : 08h20-08h40, Jour : 21/11/2013, Médecin : Mme Marie Pelissier

Client : Mr Jules Martin

On obtient une page à renseigner

**Cabinet Médical - Les Médecins Associés**

Rendez-vous : 08h20-08h40, Jour : 21/11/2013, Médecin : Mme Marie Pelissier

Client : Melle Brigitte Bistrou

On la renseigne

Créneau horaire	Client	Action
08h00-08h20		<a href="#">Réserver</a>
08h20-08h40	Melle Brigitte Bistrou	<a href="#">Supprimer</a>
08h40-09h00		<a href="#">Réserver</a>
09h00-09h20		<a href="#">Réserver</a>
09h20-09h40		<a href="#">Réserver</a>

Le nouveau RV apparaît dans la liste

**Cabinet Médical - Les Médecins As**

Rendez-vous de Mme Marie Pelissier le 21/1

Créneau horaire	Client	Action
08h00-08h20		<a href="#">Réserver</a>
08h20-08h40	Melle Brigitte Bistrou	<a href="#">Supprimer</a>
08h40-09h00		<a href="#">Réserver</a>
09h00-09h20		<a href="#">Réserver</a>

L'utilisateur peut supprimer un RV

**Cabinet Médical - Les Médecins**

Rendez-vous de Mme Marie Pelissier le 2

Créneau horaire	Client	Action
08h00-08h20		<a href="#">Réserver</a>
08h20-08h40		<a href="#">Réserver</a>
08h40-09h00		<a href="#">Réserver</a>
09h00-09h20		<a href="#">Réserver</a>

Le RV supprimé a disparu de la liste des RV

Les erreurs avec la base de données sont gérées :

### Cabinet Médical - Les Médecins associés

1

#### Les erreurs suivantes se sont produites à l'initialisation de l'application :

- GetAllMedecins
- Une erreur s'est produite lors de la récupération d'informations du fournisseur depuis la base de données. Cela peut être causé par Entity Framework qui utilise une chaîne de connexion incorrecte. Vérifiez les exceptions internes pour plus d'informations et vérifiez que la chaîne de connexion est correcte.
- Le fournisseur n'a pas retourné de chaîne ProviderManifestToken.
- Unable to connect to any of the specified MySQL hosts.

### Cabinet Médical - Les Médecins Associés

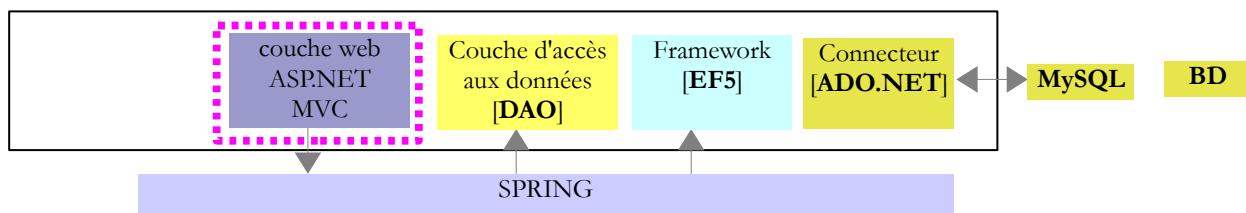
[Accueil](#)

#### Les erreurs suivantes se sont produites

- 2
- AjouterRv
  - Échec du fournisseur sous-jacent sur Open.
  - Unable to connect to any of the specified MySQL hosts.

- en [1], la page affichée si la base n'est pas disponible au lancement de l'application ;
- en [2], la page affichée si la base n'est pas disponible en cours d'utilisation de l'application.

## 5.3 Travail à faire



Il vous est demandé de reproduire les fonctionnalités attendues avec une application web ASP.NET MVC suivant le modèle APU (Application à Page Unique). Vous êtes libres d'adopter l'interface web de votre choix. Pour réaliser cette application suivez la démarche de l'étude de cas du paragraphe 2, page 220. Tout y est. Par ailleurs, lisez le code du projet [RdvMedecins-MySQL-03]. Vous y trouverez des classes et du code que vous pourrez réutiliser dans la nouvelle couche [web].

## 6 Étude de cas n° 5 : gestion basique de notes d'élèves

### 6.1 Les compétences mises en oeuvre

Les compétences mises en oeuvre par cet exercice sont les suivantes :

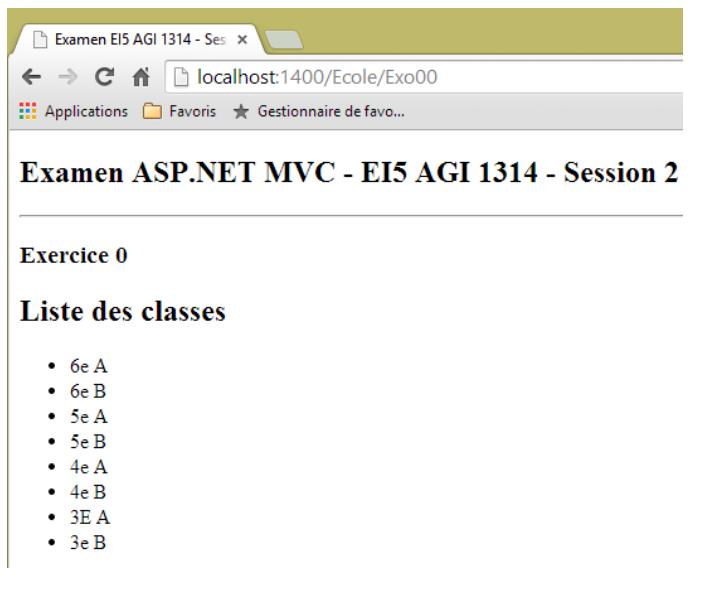
- savoir écrire une page ASP.NET MVC ;
- savoir gérer le modèle et les événements d'une page ASP.NET MVC ;
- savoir faire des appels AJAX avec JQuery ;
- connaître et savoir utiliser les différentes portées des modèles d'une application web : application, session, request ;
- maîtriser Visual Studio et son environnement de développement ;

### 6.2 Le problème

Nous nous proposons d'écrire une application ASP.NET MVC permettant l'affichage des notes d'une classe de collège. L'application présentera trois vues :



La vue [Exo00.cshtml] est une vue de démonstration. Elle présente la liste des classes du collège :



La vue [Exo01.cshtml] permet d'avoir les notes d'un élève :

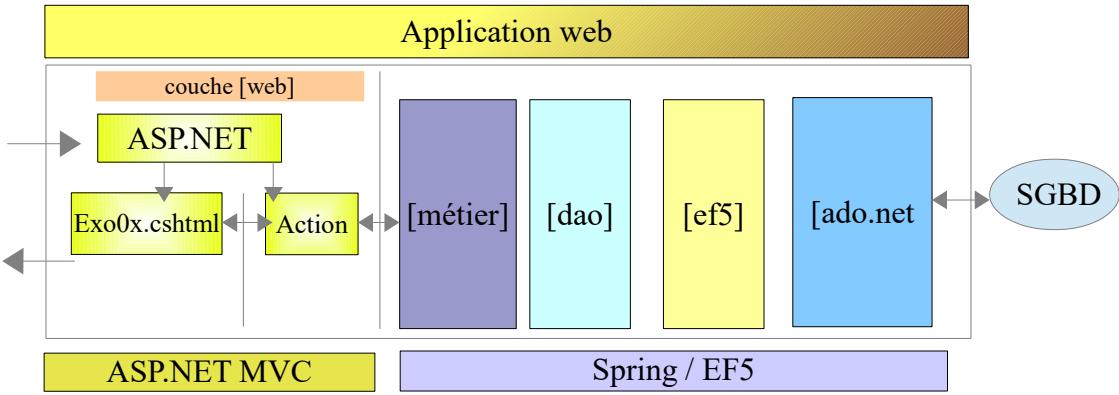
The screenshot shows a web browser window with the title "Examen EI5 AGI 1314 - Session 2". The URL in the address bar is "localhost:1400/Ecole/Exo01". The page content is titled "Examen ASP.NET MVC - EI5 AGI 1314 - Session 2". Under "Exercice 1", there is a dropdown menu for "Classes" set to "6e A". Below it is a dropdown menu for "Afficher les élèves" showing "Valérie Charlet". A link "Afficher les notes" leads to a modal dialog box displaying student grades: Anglais : 14, Espagnol : 10, Français : 12, and Mathématiques : 14. At the bottom, the average is shown as "Moyenne : 12,75".

La vue [Exo02.cshtml] permet d'avoir les notes d'une classe :

The screenshot shows a web browser window with the title "Examen EI5 AGI 1314 - Session 2". The URL in the address bar is "localhost:1400/Ecole/Exo02". The page content is titled "Examen ASP.NET MVC - EI5 AGI 1314 - Session 2". Under "Exercice 2", there is a dropdown menu for "Classes" set to "6e A". Below it is a dropdown menu for "Matières" set to "Anglais". A link "Afficher les notes" leads to a modal dialog box displaying student grades: Julien Boisseau : 10, Valérie Charlet : 14, Gabriel Monfort : 8, and Emilie Parton : 12. At the bottom, the average is shown as "Moyenne : 11", the standard deviation as "Ecart-type : 2,23606797749979", the maximum as "Max : 14", and the minimum as "Min : 8".

## 6.3 L'architecture de l'application web

L'architecture proposée pour l'application web est la suivante :



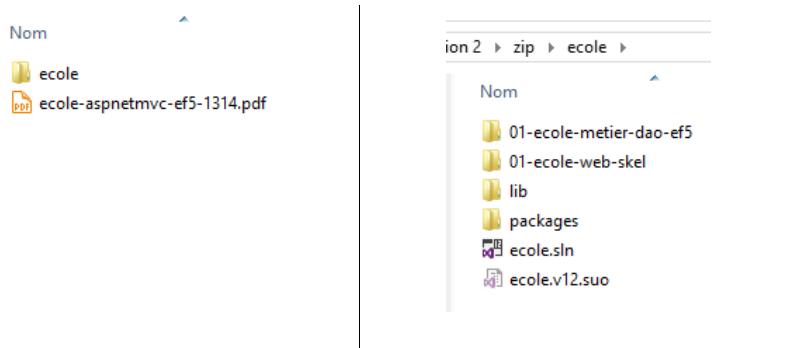
L'application est constituée de plusieurs couches. De droite à gauche :

- la couche [ado.net] constituée du pilote ADO.NET du SGBD utilisé ;
- la couche [ef5] (Entity Framework 5) qui assure les échanges avec le SGBD via la couche [ado.net] ;
- la couche [dao] ;
- la couche [métier] ;
- la couche [web] implémentée avec le framework ASP.NET MVC.

La couche [métier] vous est donnée. Seule la couche web est à construire.

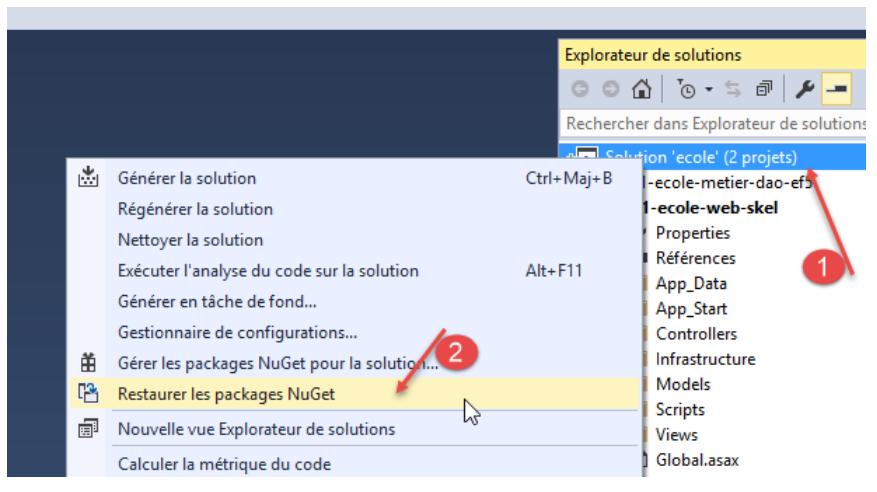
## 6.4 Les projets Visual Studio de l'application

Votre travail va consister à compléter des projets Visual Studio que vous trouverez dans une archive qui vous sera donnée. Le contenu de celle-ci est le suivant :



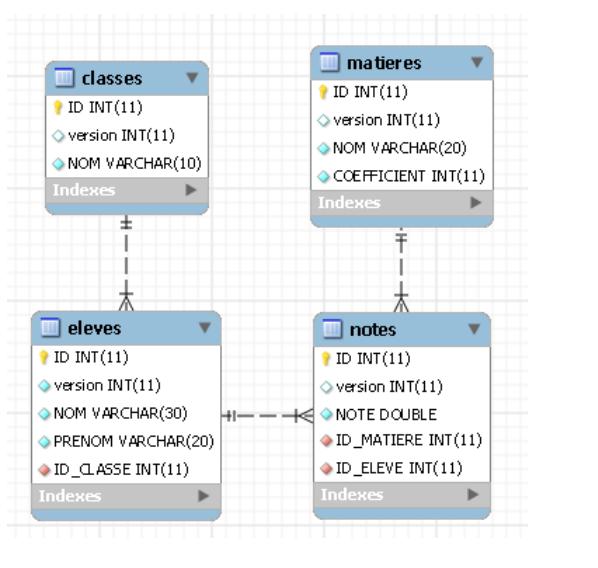
- [01-ecole-métier-dao-ef5] est un projet de test de la couche [métier] ;
- [01-ecole-web-skel] est le projet Visual Studio de la couche [web]. Il est incomplet et votre travail consiste à le compléter pour obtenir les résultats présentés précédemment ;
- [lib] contient les DLL nécessaires aux deux projets précédents ;

La solution utilise des packages NuGet qu'il faut régénérer. Pour cela suivez la procédure suivante :



## 6.5 La base de données

La base de données est une base MySQL5 ayant la structure suivante :



La table [CLASSES] mémorise les différentes classes du collège :

classes		
ID	INT(11)	
version	INT(11)	
NOM	VARCHAR(10)	
Indexes		

ID	version	NOM
1	1	6e A
2	1	6e B
3	1	5e A
4	1	5e B
5	1	4e A
6	1	4e B
7	1	3E A
8	1	3e B

- ID : N° de la classe, clé primaire
- VERSION : n° de version de l'enregistrement
- NOM : nom de la classe

La table [MATIERES] mémorise les différentes matières enseignées :

matieres		
ID	INT(11)	
version	INT(11)	
NOM	VARCHAR(20)	
COEFFICIENT	INT(11)	
Indexes		

ID	version	NOM	COEFFICIENT
1	1	Anglais	1
2	1	Espagnol	1
3	1	Français	2
4	1	Mathématiques	2
5	1	Histoire Géographie	1
6	1	Sciences Naturelles	1

- ID : N° de la matière, clé primaire
- VERSION : n° de version de l'enregistrement
- NOM : nom de la matière
- COEFFICIENT : coefficient de la matière dans le calcul de la moyenne d'un élève

La table [ELEVES] mémorise les différents élèves du collège :

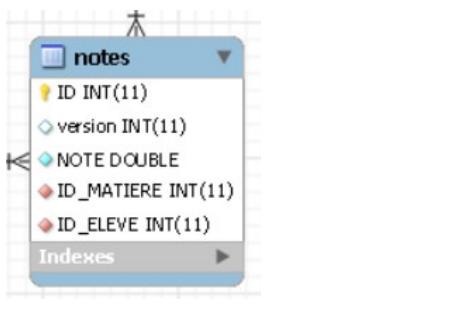
ID	version	NOM	PRENOM	ID_CLASSE
1	1	Boisseau	Julien	1
2	1	Charlet	Valérie	1
3	1	Monfort	Gabriel	1
4	1	Parton	Emilie	1
5	1	Galopin	Félix	2
6	1	Germain	Gabrielle	2

- ID : N° de l'élève, clé primaire
- VERSION : n° de version de l'enregistrement
- NOM : nom de l'élève
- PRENOM : prénom de l'élève
- ID\_CLASSE : n° de sa classe, clé étrangère sur la colonne ID de la table CLASSES

La table [NOTES] mémorise les notes des élèves dans les différentes matières :

ID	version	NOTE	ID_MATIERE	ID_ELEVE
1	1	10	1	1
2	1	14	1	2
3	1	8	1	3
4	1	12	1	4
5	1	10	2	2
6	1	12	3	2
7	1	14	4	2
8	1	15	5	2
9	1	11	6	2

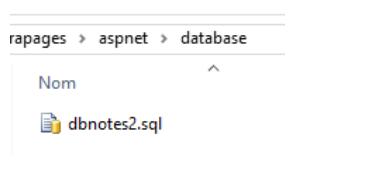
- ID : n° de la note, clé primaire
- VERSION : n° de version de l'enregistrement
- ID\_ELEVE : n° de l'élève, clé étrangère sur la colonne ID de la table ELEVES
- ID\_MATIERE : n° de la matière, clé étrangère sur la colonne ID de la table MATIERES
- NOTE : note de l'élève dans la matière



ID	version	NOTE	ID_MATIERE	ID_ELEVE
1	1	10	1	1
2	1	14	1	2
3	1	8	1	3
4	1	12	1	4
5	1	10	2	2
6	1	12	3	2
7	1	14	4	2
8	1	15	5	2
9	1	11	6	2

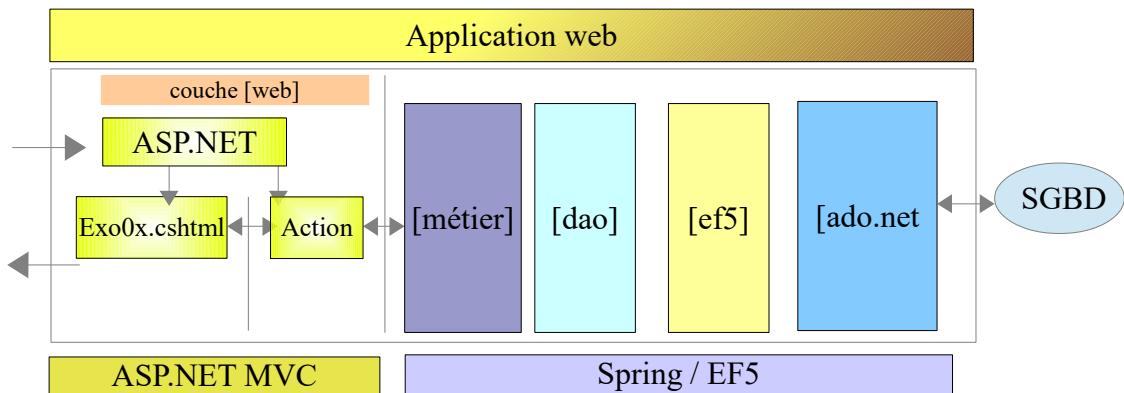
- ID : n° de la note, clé primaire
- VERSION : n° de version de l'enregistrement
- ID\_ELEVE : n° de l'élève, clé étrangère sur la colonne ID de la table ELEVES
- ID\_MATIERE : n° de la matière, clé étrangère sur la colonne ID de la table MATIERES
- NOTE : note de l'élève dans la matière

Le script de génération de la base de données vous est fourni :



Travail n° 1 : Créez la base [dbnotes2] en utilisant WampServer et PhpMyAdmin.

## 6.6 La couche [métier] et les entités manipulées par celle-ci



Vous allez écrire la couche [web] de l'application. Celle-ci va communiquer avec la couche [métier] qui se trouve dans une DLL qui vous sera fournie. L'interface [IMetier] est la suivante :

```

1. using System.Collections.Generic;
2. using EcoleMetierDaoEf.Models;
3.
4. namespace EcoleMetierDaoEf.Metier {
5.     public interface IMetier {
6.

```

```

7.     List<Classe> GetClasses();
8.
9.     List<Matiere> GetMatières();
10.
11.    List<Eleve> GetElevesForClasse(long idClasse);
12.
13.    StatsForEleve GetStatsForEleve(long idEleve);
14.
15.    List<Note> GetNotesForEleve(long idEleve);
16.
17.    List<Note> GetNotesForMatiereAndClasse(long idMatiere, long idClasse);
18.
19. }

```

- ligne 7 : pour avoir la liste des classes ;
- ligne 9 : pour avoir la liste des matières ;
- ligne 11 : pour avoir la liste des élèves d'une classe ;
- ligne 13 : pour avoir les notes et la moyenne d'un élève ;
- ligne 15 : liste des notes d'un élève donné ;
- ligne 17 : pour avoir les notes des élèves d'une classe dans une matière donnée ;

L'entité [Classe] est la suivante :

```

1.  [Table("CLASSES")]
2.  public class Classe {
3.
4.      [Key]
5.      [Column("ID")]
6.      public long Id { get; set; }
7.
8.      [Column("NOM")]
9.      public string Nom { get; set; }
10.
11.     // signature
12.     public override string ToString() {
13.         return String.Format("Classe [nom={0}]", Nom);
14.     }
15. }

```

- une instance de type [Classe] représente une ligne de la table [CLASSES] de la base de données ;
- ligne 6 : l'identifiant de la classe ;
- ligne 9 : le nom de la classe ;

L'entité [Matiere] est la suivante :

```

1.  [Table("MATIERES")]
2.  public class Matiere {
3.
4.      [Key]
5.      [Column("ID")]
6.      public long Id { get; set; }
7.
8.      [Column("NOM")]
9.      public string Nom { get; set; }
10.
11.     [Column("COEFFICIENT")]
12.     public int Coefficient { get; set; }
13.
14.     // signature
15.     public override string ToString() {
16.         return String.Format("Matière [nom={0}, coeff={1}]", Nom, Coefficient);
17.     }
18. }

```

- une instance de type [Matiere] représente une ligne de la table [MATIERES] de la base de données ;
- ligne 6 : l'identifiant de la classe ;
- ligne 9 : le nom de la matière ;
- ligne 12 : son coefficient ;

L'entité [Eleve] est la suivante :

```

1.  [Table("ELEVES")]
2.  public class Eleve {
3.
4.      [Key]
5.      [Column("ID")]
6.      public long Id { get; set; }
7.

```

```

8.     [Column("PRENOM")]
9.     public string Prénom { get; set; }
10.
11.    [Column("NOM")]
12.    public string Nom { get; set; }
13.
14.    [Column("ID_CLASSE")]
15.    public long IdClasse { get; set; }
16.
17.    [Required]
18.    [ForeignKey("IdClasse")]
19.    public virtual Classe Classe { get; set; }
20.
21.    // signature
22.    public override string ToString() {
23.        return String.Format("Elève [prénom={0}, nom= {1}, idClasse={2}]", Prénom, Nom, IdClasse);
24.    }
25. }
```

- une instance de type [Eleve] représente une ligne de la table [ELEVES] de la base de données ;
- ligne 5 : l'identifiant de l'élève ;
- ligne 12 : le nom de l'élève ;
- ligne 9 : son prénom ;
- ligne 15 : l'identifiant de sa classe ;
- ligne 19 : sa classe ;

Enfin, l'entité [Note] est la suivante :

```

1.     [Table("NOTES")]
2.     public class Note {
3.
4.         [Key]
5.         [Column("ID")]
6.         public long Id { get; set; }
7.
8.         [Column("NOTE")]
9.         public double Valeur { get; set; }
10.
11.        [Column("ID_MATIERE")]
12.        public long IdMatiere { get; set; }
13.
14.        [Required]
15.        [ForeignKey("IdMatiere")]
16.        public virtual Matiere Matiere { get; set; }
17.
18.        [Column("ID_ELEVE")]
19.        public long IdEleve { get; set; }
20.
21.        [Required]
22.        [ForeignKey("IdEleve")]
23.        public virtual Eleve Eleve { get; set; }
24.
25.        // signature
26.        public override string ToString() {
27.            return String.Format("Note [valeur={0}, idEleve={1}, idMatiere={2}]", Valeur, IdEleve, IdMatiere);
28.        }
29.    }
30. }
```

- une instance de type [Note] représente une ligne de la table [NOTES] de la base de données. C'est une note pour un élève et une matière donnés ;
- ligne 6 : l'identifiant de la note ;
- ligne 9 : la note elle-même ;
- ligne 12 : l'identifiant de la matière qui est ici notée ;
- ligne 16 : la matière ici notée ;
- ligne 19 : l'identifiant de l'élève ici noté ;
- ligne 23 : l'élève noté ;

L'entité [StatsForEleve] encapsule des informations sur les notes d'un élève donné. Son code est le suivant :

```

1.     using System;
2.     using System.Collections.Generic;
3.     using EcoleMetierDaoEf.Models;
4.
5.     namespace EcoleMetierDaoEf.Metier {
6.         public class StatsForEleve {
7.             // élève
```

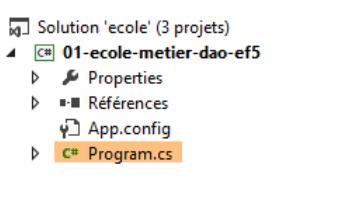
```

8.     public Eleve Eleve { get; set; }
9.     // notes de l'élève
10.    public List<Note> Notes { get; set; }
11.    // indicateurs sur ces notes
12.    public double MoyennePonderee { get; set; }
13.    public double Max { get; set; }
14.    public double Min { get; set; }
15.
16.    // constructeurs
17.    public StatsForEleve() {
18.    }
19.
20.    public StatsForEleve(Eleve eleve, List<Note> notes, List<Matiere> matieres) {
21.        ...
22.    }
23.
24.    // signature
25.    override
26.    public String ToString() {
27.        if (Notes.Count == 0) {
28.            return String.Format("Eleve={0}, notes=[]", Eleve);
29.        }
30.        // passage des notes en String
31.        String strNotes = "";
32.        for (int i = 0; i < Notes.Count; i++) {
33.            strNotes += String.Format("{0} ", Notes[i].Valeur);
34.        }
35.        return String.Format("{0}, [{1}], {2}, {3}, {4}", Eleve, strNotes, MoyennePonderee, Max, Min);
36.    }
37.
38. }
39.

```

- ligne 8 : l'élève concerné ;
- ligne 10 : ses notes ;
- ligne 12 : sa moyenne ;
- ligne 13 : sa note la plus haute ;
- ligne 14 : sa note la plus basse ;

Un programme de test de la couche [métier] vous est donné dans le projet [01-ecole-metier-dao-ef5] :



Le code de la classe [Program.cs] est le suivant :

```

1.  using System;
2.  using EcoleMetierDaoEf.Models;
3.  using Spring.Context.Support;
4.
5.  namespace EcoleMetierDaoEf.Tests {
6.      class ProgramMetier {
7.          static void Main(string[] args) {
8.              // instanciation de la couche [métier]
9.              Metier.IMetier métier = ContextRegistry.GetContext().GetObject("metier") as Metier.IMetier;
10.             // les classes
11.             Console.WriteLine("Classes-----");
12.             foreach (Classe classe in métier.GetClasses()) {
13.                 Console.WriteLine(classe);
14.             }
15.             // les matières
16.             Console.WriteLine("Matières-----");
17.             foreach (Matiere matière in métier.GetMatieres()) {
18.                 Console.WriteLine(matière);
19.             }
20.             // les élèves
21.             Console.WriteLine("Eleves de la classe n° 1 -----");
22.             foreach (Eleve élève in métier.GetElevesForClasse(1L)) {
23.                 Console.WriteLine(élève);
24.             }
25.             // les notes
26.             Console.WriteLine("Liste des notes de la classe 1 dans la matière 2-----");
27.             foreach (Note note in métier.GetNotesForMatiereAndClasse(2L,1L)) {
28.                 Console.WriteLine(note);

```

```

29.      }
30.      // stats de l'élève n° 2
31.      Console.WriteLine("Statistiques de l'élève n° 2 : {0}", métier.GetStatsForEleve(2));
32.  }
33. }
34. }
```

L'interface [IMetier] est testée aux lignes 12, 17, 22, 27 et 31. Les résultats obtenus sont les suivants :

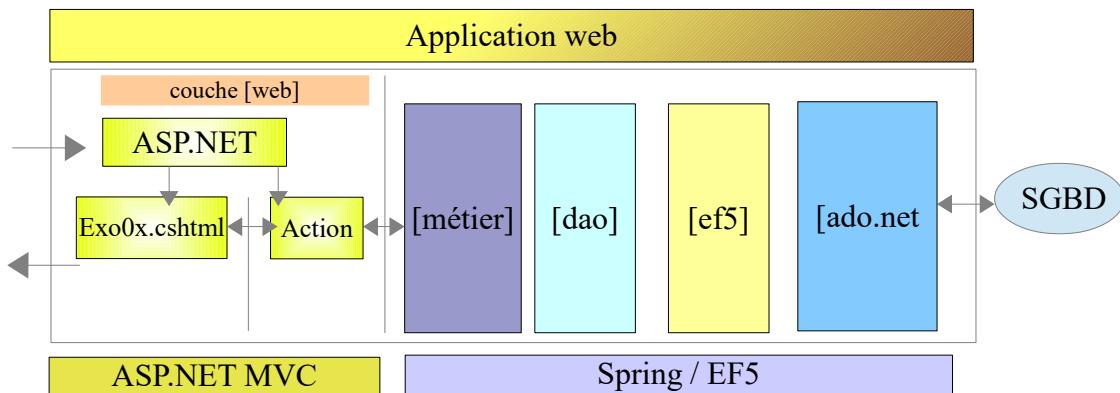
```

1. 2014/05/31 15:36:08:169 [INFO] Spring.Context.Support.XmlApplicationContext - ApplicationContext Refresh: Completed
2. Classes-----
3. Classe [nom=6e A]
4. Classe [nom=6e B]
5. Classe [nom=5e A]
6. Classe [nom=5e B]
7. Classe [nom=4e A]
8. Classe [nom=4e B]
9. Classe [nom=3E A]
10. Classe [nom=3e B]
11. Matières-----
12. Matière [nom=Anglais, coeff=1]
13. Matière [nom=Espagnol, coeff=1]
14. Matière [nom=Français, coeff=2]
15. Matière [nom=Mathématiques, coeff=2]
16. Matière [nom=Histoire Géographie, coeff=1]
17. Matière [nom=Sciences Naturelles, coeff=1]
18. Eleves de la classe n° 1 -----
19. Elève [prénom=Julien, nom= Boisseau, idClasse=1]
20. Elève [prénom=Valérie, nom= Charlet, idClasse=1]
21. Elève [prénom=Gabriel, nom= Monfort, idClasse=1]
22. Elève [prénom=Emilie, nom= Parton, idClasse=1]
23. Liste des notes de la classe 1 dans la matière 2-----
24. Note [valeur=10, idElève=2, idMatiere=2]
25. Statistiques de l'élève n° 2 : Stats=[Elève [prénom=Valérie, nom= Charlet, idClasse=1], notes=[14 10 12 14 15 11 ], moyenne=12,75, =max=15, min=10]
```

**Travail n° 2 :** exécutez le programme console [Program.cs] et vérifiez que vous obtenez les résultats ci-dessus. A partir de maintenant, vous utiliserez cette couche [métier] sans la modifier. Pour savoir comment votre couche [web] doit appeler la couche [métier], reportez-vous au programme console [Program.cs] ci-dessus. La couche [métier] peut lancer des exceptions. Pour simplifier les choses, dans tout ce qui suit, **on se placera dans le cas où il n'y a pas d'exceptions.**

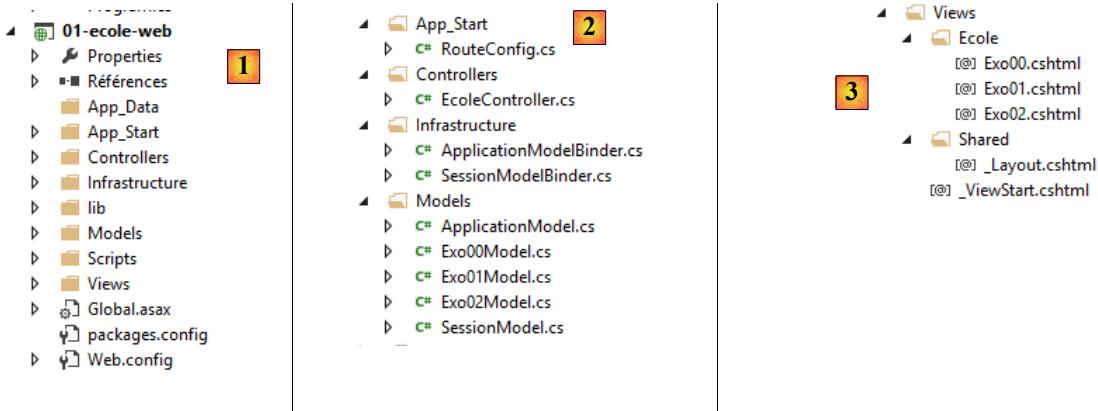
## 6.7 Implémentation de la couche [web]

L'architecture proposée pour l'application web est la suivante :

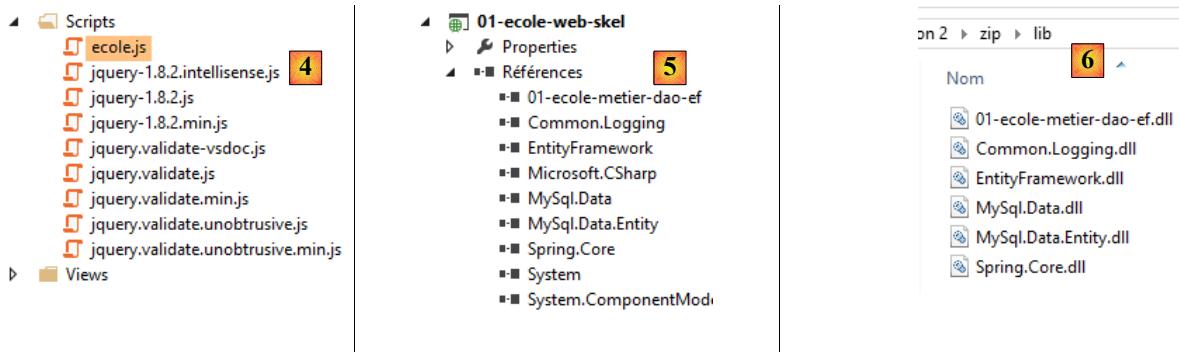


### 6.7.1 Le projet Visual Studio

Le projet Visual Studio de l'application web est le suivant :



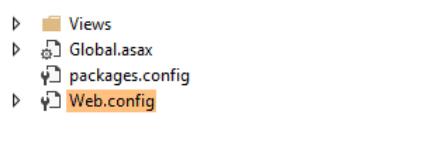
- en [1], le projet dans son ensemble ;
- en [2] :
  - [RouteConfig.cs] définit les routes gérées par l'application,
  - [EcoleController.cs] : contient les actions qui gèrent les différentes routes de l'application,
  - [ApplicationModelBinder.cs] : est le binder qui permet d'instancier le modèle de l'application,
  - [SessionModelBinder.cs] : est le binder qui permet d'instancier le modèle de la session,
  - [ApplicationModel.cs] : est le modèle de portée Application,
  - [SessionModel.cs] : est le modèle de portée Session,
  - [Exo00Model.cs] : est le modèle de la vue [Exo00.cshtml],
  - [Exo01Model.cs] : est le modèle de la vue [Exo01.cshtml],
  - [Exo02Model.cs] : est le modèle de la vue [Exo02.cshtml] ;
- en [3] :
  - [Exo00.cshtml] est la vue associée au modèle [Exo00Model],
  - [Exo01.cshtml] est la vue associée au modèle [Exo01Model],
  - [Exo02.cshtml] est la vue associée au modèle [Exo02Model],
  - [\_Layout.cshtml] est la page maître des vues précédentes ;



- en [4], le script [ecole.js] contient les codes Javascript réalisant les appels AJAX de l'application ;
- en [5], les références nécessaires au projet. Elles sont prises dans le dossier [lib] [6] de l'archive qui vous a été donnée ;

Le travail à faire consiste à écrire les vues [Exo01.cshtml], [Exo02.cshtml], leurs modèles [Exo01Model.cs] et [Exo02Model.cs] et le contrôleur [EcoleController.cs]. Le reste vous est donné et est expliqué dans ce qui suit. Attendez les questions avant de faire quoique ce soit.

### 6.7.2 Configuration du projet web



Le fichier [Web.config] est le suivant :

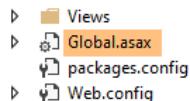
```
1.  <?xml version="1.0" encoding="utf-8"?>
2.  <configuration>
3.    <!-- spring -->
4.    <configSections>
5.      <sectionGroup name="spring">
6.        <section name="context" type="Spring.Context.Support.ContextHandler, Spring.Core" />
7.        <section name="objects" type="Spring.Context.Support.DefaultSectionHandler, Spring.Core" />
8.      </sectionGroup>
9.      <!-- common logging-->
10.     <sectionGroup name="common">
11.       <section name="logging" type="Common.Logging.ConfigurationSectionHandler, Common.Logging" />
12.     </sectionGroup>
13.   </configSections>
14.   <startup>
15.     <supportedRuntime version="v4.0" sku=".NETFramework,Version=v4.5" />
16.   </startup>
17.   <!-- chaîne de connexion-->
18.   <connectionStrings>
19.     <add name="EcoleContext" connectionString="Server=localhost;Database=dbnotes2;Uid=root;Pwd=" providerName="ecoleProvider" />
20.   </connectionStrings>
21.   <!-- le factory provider -->
22.   <system.data>
23.     <DbProviderFactories>
24.       <add name="MySQL Data Provider" invariant="ecoleProvider" description=".Net Framework Data Provider for MySQL" type="MySQL.Data.MySqlClient.MySqlClientFactory, MySql.Data" />
25.     </DbProviderFactories>
26.   </system.data>
27.   <!-- configuration Spring -->
28.   <spring>
29.     <context>
30.       <resource uri="config://spring/objects" />
31.     </context>
32.     <objects xmlns="http://www.springframework.net">
33.       <object id="dao" type="EcoleMetierDaoEf.Dao.Dao,01-ecole-metier-dao-ef" />
34.       <object id="metier" type="EcoleMetierDaoEf.Metier.Metier,01-ecole-metier-dao-ef">
35.         <property name="Dao" ref="dao" />
36.       </object>
37.     </objects>
38.   </spring>
39.   <!-- configuration common.logging -->
40.   <common>
41.     <logging>
42.       <factoryAdapter type="Common.Logging.Simple.ConsoleOutLoggerFactoryAdapter, Common.Logging">
43.         <arg key="showLogName" value="true" />
44.         <arg key="showDateTime" value="true" />
45.         <arg key="level" value="INFO" />
46.         <arg key="dateTimeFormat" value="yyyy/MM/dd HH:mm:ss:fff" />
47.       </factoryAdapter>
48.     </logging>
49.   </common>
50.
51.   <appSettings>
52.     <add key="webpages:Version" value="3.0.0.0" />
53.     <add key="webpages:Enabled" value="false" />
54.     <add key="ClientValidationEnabled" value="true" />
55.     <add key="UnobtrusiveJavaScriptEnabled" value="true" />
56.   </appSettings>
57.   <system.web>
58.     <compilation debug="true" targetFramework="4.5" />
59.     <httpRuntime targetFramework="4.5" />
60.   </system.web>
61. </configuration>
```

De cette configuration, il faut retenir les points suivants :

- la ligne 19 définit la base de données utilisée [dbnotes2]. Le pilote ADO.NET utilisé [ecoleProvider] est défini ligne 24. C'est le provider du SGBD MySQL5 ;
- lignes 28-38 : définissent les objets gérés par Spring. On retiendra l'id [metier] (ligne 34) de l'objet de la couche [métier] ;

### 6.7.3 Initialisation de l'application

L'application web est initialisée par la classe contenue dans [Global.asax] :



Son code est le suivant :

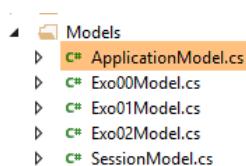
```

1.  using System;
2.  using System.Collections.Generic;
3.  using System.Linq;
4.  using System.Web;
5.  using System.Web.Mvc;
6.  using System.Web.Routing;
7.  using EcoleMetierDaoEf.Dao;
8.  using EcoleMetierDaoEf.Metier;
9.  using EcoleMetierDaoEf.Models;
10. using EcoleWeb;
11. using EcoleWeb.Models;
12. using RdvMedecins.Web.Infrastructure;
13. using Spring.Context.Support;
14.
15. namespace EcoleWeb {
16.     public class MvcApplication : System.Web.HttpApplication {
17.         protected void Application_Start() {
18.             // -----
19.             // ----- configuration générique
20.             //
21.             RouteConfig.RegisterRoutes(RouteTable.Routes);
22.             //
23.             // ----- configuration spécifique
24.             //
25.             // données de portée application
26.             ApplicationModel application = new ApplicationModel();
27.             Application["data"] = application;
28.             // on met en cache certaines données de la base de données
29.             // instanciation couche [métier]
30.             application.Metier = ContextRegistry.GetContext().GetObject("metier") as IMetier;
31.             // on mémorise les classes
32.             List<Classe> classes = application.Metier.GetClasses();
33.             // on génère les éléments du combo des classes
34.             application.ClassesItems = new SelectListItem[classes.Count];
35.             int i = 0;
36.             foreach (Classe classe in classes) {
37.                 application.ClassesItems[i] = new SelectListItem() { Text = classe.Nom, Value = classe.Id.ToString() };
38.                 i++;
39.             }
40.             // on récupère les matières
41.             List<Matiere> matières = application.Metier.GetMatieres();
42.             // on génère les éléments du combo des matières
43.             application.MatieresItems = new SelectListItem[matières.Count];
44.             i = 0;
45.             foreach (Matiere matière in matières) {
46.                 application.MatieresItems[i] = new SelectListItem() { Text = matière.Nom, Value = matière.Id.ToString() };
47.                 i++;
48.             }
49.             // model binders
50.             ModelBinders.Binders.Add(typeof(ApplicationModel), new ApplicationModelBinder());
51.             ModelBinders.Binders.Add(typeof(SessionModel), new SessionModelBinder());
52.         }
53.
54.         // session
55.         protected void Session_Start() {
56.             // le modèle de la session
57.             SessionModel session = new SessionModel();
58.             Session["data"] = session;
59.             // pas d'élèves pour l'instant
60.             session.ElevesItems = new SelectListItem[0];
61.         }
62.
63.     }
64. }
```

- ligne 26 : un modèle de portée [Application] est créé et associé à la clé [data] ;
- ligne 30 : le fichier [Web.config] est exploité pour instancier la couche [métier]. Une référence de la couche est stockée dans le modèle de portée [Application] ;
- ligne 32 : la liste des classes est demandée à la couche [métier] ;
- lignes 34-39 : les éléments nécessaires pour alimenter la liste déroulante des classes sont créés et mis dans le modèle de portée [Application] ;
- lignes 41-48 : on fait la même chose avec la liste des matières ;

- lignes 50-51 : définissent les deux [Binders] qui vont instancier les modèles de portée [Application] et [Session] ;
- lignes 57-58 : le modèle de portée [Session] est créé et associé à la clé [data] ;
- ligne 60 : les éléments nécessaires pour alimenter la liste déroulante des élèves sont initialisés avec un tableau vide ;

#### 6.7.4 Le modèle de portée [Application]



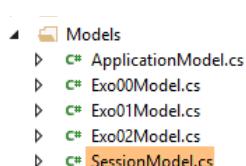
Le modèle de portée [Application] est le suivant :

```

1.  using System.Collections.Generic;
2.  using System.Web.Mvc;
3.  using EcoleMetierDaoEf.Metier;
4.  using EcoleMetierDaoEf.Models;
5.
6.  namespace EcoleWeb.Models {
7.      public class ApplicationModel {
8.
9.          // couche [métier]
10.         public IMetier Metier { get; set; }
11.         // les éléments du combo des classes
12.         public SelectListItem[] ClassesItems { get; set; }
13.         // les éléments du combo des matières
14.         public SelectListItem[] MatieresItems { get; set; }
15.     }
16. }
```

- ligne 10 : référence sur la couche [métier] ;
- ligne 12 : les éléments du combo des classes ;
- ligne 14 : les éléments du combo des matières ;

#### 6.7.5 Le modèle de portée [Session]



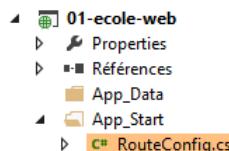
Le modèle de portée [Session] est le suivant :

```

1.  using System.Collections.Generic;
2.  using System.Web.Mvc;
3.  using EcoleMetierDaoEf.Models;
4.
5.  namespace EcoleWeb.Models {
6.      public class SessionModel {
7.          // l'application
8.          public ApplicationModel Application { get; set; }
9.          // le contenu du combo des élèves
10.         public SelectListItem[] ElevesItems { get; set; }
11.     }
12. }
```

- ligne 8 : référence sur le modèle de portée [Application] ;
- ligne 10 : les éléments du combo des élèves ;

## 6.7.6 Le routage de l'application



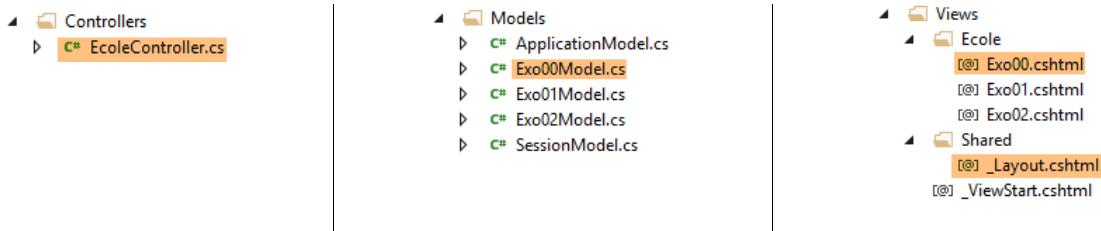
Les routes définies dans [RouteConfig.cs] sont les suivantes :

```
1. using System.Web.Mvc;
2. using System.Web.Routing;
3.
4. namespace EcoleWeb {
5.     public class RouteConfig {
6.         public static void RegisterRoutes(RouteCollection routes) {
7.             routes.IgnoreRoute("{resource}.axd/{*pathInfo}");
8.
9.             routes.MapRoute(
10.                 name: "route",
11.                 url: "{controller}/{action}",
12.                 defaults: new { controller = "Ecole", action = "Exo00" }
13.             );
14.         }
15.     }
16. }
```

- lignes 9-12 : indiquent que pour l'URL /, ce sera l'URL [/Ecole/Exo00] qui sera traitée ;

## 6.8 L'URL [/Ecole/Exo00]

L'URL [/Ecole/Exo00] est traitée par la méthode [Exo00] de la classe [EcoleController].



Son code est le suivant :

```
1.     [HttpGet]
2.     public ViewResult Exo00(ApplicationModel application) {
3.         // on affiche la vue
4.         return View("Exo00", new Exo00Model() { Application = application });
5.     }
```

- ligne 2 : la méthode [Exo00] a un paramètre : le modèle de l'application [ApplicationModel]. Grâce aux [binders] définis dans [Global.asax], ce paramètre est instancié par le framework ASP.NET ;
- ligne 4 : la vue [Exo00.cshtml] est affichée avec une instance du modèle [Exo00Model]. Ce modèle est le suivant :

```
1. namespace EcoleWeb.Models {
2.     public class Exo00Model {
3.
4.         // l'application
5.         public ApplicationModel Application { get; set; }
6.     }
7. }
```

- ligne 5 : une référence sur le modèle de portée [Application]

La vue [Exo00.cshtml] est la suivante :

```

1. @model EcoleWeb.Models.Exo00Model
2. @using EcoleMetierDaoEf.Models;
3.
4. <h3>Exercice 0</h3>
5. <html>
6. <body>
7.   <h2>Liste des classes</h2>
8.   <ul>
9.     @foreach (SelectListItem classe in Model.Application.ClassesItems) {
10.       <li>@classe.Text</li>
11.     }
12.   </ul>
13. </body>
14. </html>

```

- ligne 1 : le modèle de la vue est de type [Exo00Model] que nous venons de décrire ;
- lignes 9-11 : exploitent [Model.Application.ClassesItems] pour afficher les noms des classes dans une liste.

Lorsqu'on demande l'URL [/Ecole/Exo00], on obtient la vue suivante :

Le code HTML de la vue ci-dessus est le suivant :

```

1.  <!DOCTYPE html>
2.  <html>
3.  <head>
4.    <meta charset="utf-8" />
5.    <meta name="viewport" content="width=device-width, initial-scale=1.0">
6.    <title>Examen EI5 AGI 1314 - Session 2</title>
7.    <link href="/Content/Site.css" rel="stylesheet" type="text/css" />
8.    <link href="/Content/bootstrap.min.css" rel="stylesheet" type="text/css" />
9.    <script src="/Scripts/modernizr-2.6.2.js"></script>
10. </head>
11. <body>
12.   <div class="container body-content">
13.     <h2>Examen ASP.NET MVC - EI5 AGI 1314 - Session 2</h2>
14.     <hr />
15.     <div id="content">
16.
17.
18.     <h3>Exercice 0</h3>
19.     <html>
20.     <body>
21.       <h2>Liste des classes</h2>
22.       <ul>
23.         <li>6e A</li>
24.         <li>6e B</li>
25.         <li>5e A</li>
26.         <li>5e B</li>
27.         <li>4e A</li>
28.         <li>4e B</li>

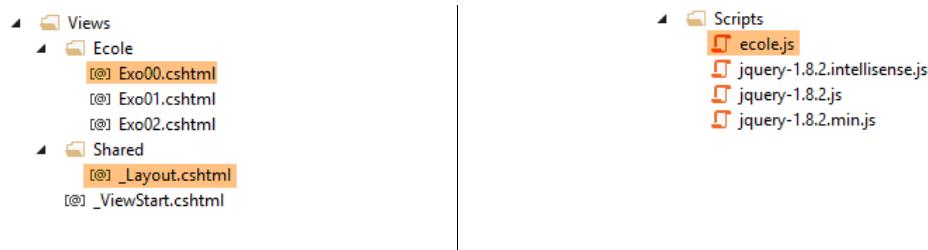
```

```

29.      <li>3E A</li>
30.      <li>3e B</li>
31.    </ul>
32.
33.    <!-- Visual Studio Browser Link -->
34.    <script type="application/json" id="__browserLink_initializationData">
35.      {"appName":"Chrome","requestId":"bf2975405d7d4deea28acb7c95246f05"}
36.    </script>
37.    <script type="text/javascript" src="http://localhost:39363/aef12727e8a44727b9fbb856547932c9/browserLink"
38.      async="async"></script>
39.    <!-- End Browser Link -->
40.  </body>
41. </html>
42.   </div>
43.   </div>
44.   <script src="/Scripts/jquery-1.8.2.min.js"></script>
45.   <script src="/Scripts/bootstrap.min.js"></script>
46.   <script src="/Scripts/ecolet.js"></script>
47. </body>
48. </html>

```

Tout n'a pas été généré par la vue [Exo00.cshtml]. Celle-ci est venue s'inscrire dans la page maître [\_Layout.cshtml] :



La vue [\_Layout.cshtml] est la suivante :

```

1.  <!DOCTYPE html>
2.  <html>
3.  <head>
4.    <meta charset="utf-8" />
5.    <meta name="viewport" content="width=device-width, initial-scale=1.0">
6.    <title>Examen EIS AGI 1516 - Session 2</title>
7.    <link href("~/Content/Site.css" rel="stylesheet" type="text/css" />
8.    <link href("~/Content/bootstrap.min.css" rel="stylesheet" type="text/css" />
9.    <script src "~/Scripts/modernizr-2.6.2.js"></script>
10.   </head>
11.   <body>
12.     <div class="container body-content">
13.       <h2>Examen ASP.NET MVC - EIS AGI 1516 - Session 2</h2>
14.       <hr />
15.       <div id="content">
16.         @RenderBody()
17.       </div>
18.     </div>
19.     <script src "~/Scripts/jquery-1.8.2.min.js"></script>
20.     <script src "~/Scripts/bootstrap.min.js"></script>
21.     <script src "~/Scripts/ecolet.js"></script>
22.   </body>
23. </html>

```

- les différentes vues viennent s'insérer ligne 16. Elles s'insèrent dans une zone délimitée par une balise <div> identifiée par [id=content]. Il est important de s'en souvenir car le code Javascript va utiliser cette zone ;
- ligne 13 : le titre commun à toutes les vues ;
- ligne 21 : la référence sur le fichier Javascript qui contient les fonctions AJAX qui vont être utilisées dans la suite ;

---

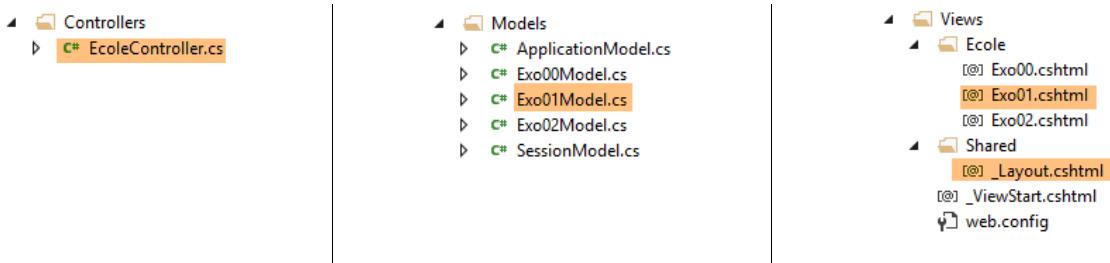
**Travail n° 3 :** lancez l'application web pour afficher l'URL [/Ecole/Exo00].

---

## 6.9 L'URL [/Ecole/Exo01]

### 6.9.1 Introduction

L'URL [/Ecole/Exo01] est traitée par la méthode [Exo01] de la classe [EcoleController].



Son code est le suivant :

```

1.     [HttpGet]
2.     public ViewResult Exo01(ApplicationModel application, SessionModel session) {
3.         // on affiche la vue
4.         return View("Exo01", new Exo01Model() { Application = application, Session = session });
5.     }

```

Il est très analogue à celui de la méthode [Exo00], si ce n'est que le modèle [Exo01Model] de la vue [Exo01.cshtml] intègre non seulement le modèle de portée [Application] mais également celui de portée [Session].

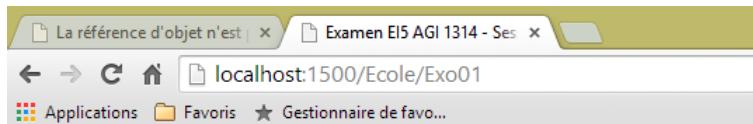
Le modèle [Exo01Model] est le suivant :

```

1.  using System;
2.  using System.Collections.Generic;
3.  using System.Linq;
4.  using System.Web;
5.  using System.Web.Mvc;
6.  using EcoleMetier.DaoEf.Metier;
7.
8.  namespace EcoleWeb.Models {
9.      public class Exo01Model {
10.
11.        // l'application
12.        public ApplicationModel Application { get; set; }
13.        // la session
14.        public SessionModel Session { get; set; }
15.        // la classe choisie
16.        public long IdClasse { get; set; }
17.        // l'élève choisi
18.        public long IdEleve { get; set; }
19.        // les notes de l'élève
20.        public SelectListItem[] NotesItems;
21.        // la note sélectionnée
22.        public long IdNote { get; set; }
23.        // la moyenne pondérée de l'élève
24.        public double MoyennePondérée { get; set; }
25.
26.        // constructeur
27.        public Exo01Model() {
28.            NotesItems = new SelectListItem[0];
29.            MoyennePondérée = -1;
30.        }
31.    }
32. }

```

C'est le modèle de la vue [Exo01.cshtml] suivante :



## Examen ASP.NET MVC - EI5 AGI 1314 - Session 2

### Exercice 1

Classes : 6e A

[Afficher les élèves](#) Valérie Charlet

[Afficher les notes](#) 
Anglais : 14
Espagnol : 10
Français : 12
Mathématiques : 14

Moyenne : 12,75

- ligne 12 : une référence sur le modèle de portée [Application] ;
- ligne 14 : une référence sur le modèle de portée [Session] ;
- ligne 16 : l'identifiant de la classe sélectionnée dans le combo des classes ;
- ligne 18 : l'identifiant de l'élève sélectionné dans le combo des élèves ;
- ligne 20 : les éléments de la liste des notes ;
- ligne 22 : l'identifiant de la note sélectionnée dans la liste des notes (n'a pas d'utilité par la suite) ;
- ligne 24 : la moyenne de l'élève sélectionné dans le combo des élèves ;

La vue [Exo01.cshtml] est la suivante :

```

1. @model EcoleWeb.Models.Exo01Model
2. @using EcoleMetierDaoEf.Models;
3.
4. <h3>Exercice 1</h3>
5. <html>
6. <body>
7.   @using (Html.BeginForm("Exo01", "Ecole", FormMethod.Post, new { id = "formulaire" })) {
8.     <table>
9.       <tr>
10.         <td>Classes :</td>
11.         <td>
12.           @Html.DropDownListFor(m => m.IdClasse, Model.Application.ClassesItems)
13.         </td>
14.       </tr>
15.       <tr>
16.         <td>
17.           <a id="lnkAfficherEleves" href="javascript:afficherEleves()">
18.             Afficher les élèves<br />
19.           </a>
20.         </td>
21.         @Html.DropDownListFor(m => m.IdEleve, Model.Session.ElevesItems)
22.       </td>
23.     </tr>
24.   ...
25. }
26. </body>
27. </html>
```

- ligne 1 : la vue [Exo01.cshtml] a pour modèle le type [Exo01Model]

Lorsqu'on demande l'URL [/Ecole/Exo01], on obtient la vue suivante :

Examen ASP.NET MVC - EI5 AGI 1314 - Session 2

Exercice 1

Classes : 6e A

Afficher les élèves

Elle est incomplète et il va vous être demandé de la compléter.

### 6.9.2 Gestion du lien [Afficher les élèves]

Ci-dessus, lorsqu'on clique sur le lien [Afficher les élèves], le combo des élèves doit se remplir avec les élèves de la classe 6e A. Voyons comment ce résultat peut être obtenu. Examinons le code HTML généré par la vue [Exo01.cshtml] affichée :

```
1 <!DOCTYPE html>
2 <html>
3   <head>
4     <meta charset="utf-8"/>
5     <meta name="viewport" content="width=device-width, initial-scale=1.0">
6     <title>Examen EI5 AGI 1314 - Session 2</title>
7     <link href="/Content/Site.css" rel="stylesheet" type="text/css"/>
8     <link href="/Content/bootstrap.min.css" rel="stylesheet" type="text/css"/>
9     <script src="/Scripts/modernizr-2.6.2.js"></script>
10    </head>
11    <body>
12      <div class="container body-content">
13        <h2>Examen ASP.NET MVC - EI5 AGI 1314 - Session 2</h2>
14        <hr/>
15        <div id="content" ...>
16        </div>
17        <script src="/Scripts/jquery-1.8.2.min.js"></script>
18        <script src="/Scripts/bootstrap.min.js"></script>
19        <script src="/Scripts/ecoile.js"></script>
20      </body>
21    </html>
```

- ligne 15 : la zone identifiée par [id=content] est la zone dans laquelle viennent s'insérer les vues de l'application.

Dans le cas de la vue [Exo01.cshtml], cette zone a le contenu suivant :

```

15 | <div id="content">
16 |   <h3>Exercice 1</h3>
17 |
18 |   <form action="/Ecole/Exo01" id="formulaire" method="post">
19 |     <table>
20 |       <tr>
21 |         <td>Classes :</td>
22 |         <td>
23 |           <select data-val="true" data-val-number="Le champ IdClasse doit être un nombre."
24 |                         data-val-required="Le champ IdClasse est requis." id="IdClasse" name="IdClasse">
25 |             <option value="1">6e A</option>
26 |             <option value="2">6e B</option>
27 |             <option value="3">5e A</option>
28 |             <option value="4">5e B</option>
29 |             <option value="5">4e A</option>
30 |             <option value="6">4e B</option>
31 |             <option value="7">3E A</option>
32 |             <option value="8">3e B</option>
33 |           </select>
34 |         </td>
35 |       </tr>
36 |       <tr>
37 |         <td>
38 |           <a id="lnkAfficherEleves" href="javascript:afficherEleves()">
39 |             Afficher les élèves<br/>
40 |           </a>
41 |         </td>
42 |         <td>
43 |           <select data-val="true" data-val-number="Le champ IdEleve doit être un nombre."
44 |                         data-val-required="Le champ IdEleve est requis." id="IdEleve" name="IdEleve"></select>
45 |         </td>
46 |       </tr>
47 |     </table>
48 |   </form>
49 | </div>

```

- ligne 18 : le formulaire HTML est identifié par [id=formulaire] ;
- ligne 24 : la liste déroulante a l'attribut [name=IdClasse]. Cela signifie que la valeur sélectionnée dans la liste déroulante sera postée sous la forme [IdClasse=n] où n est l'attribut [value] de la classe sélectionnée. Ainsi si l'utilisateur sélectionne la classe [3E A], on aura [IdClasse=7] ;
- ligne 38 : le clic sur le lien [Afficher les élèves] appelle une fonction Javascript. Celle-ci comme les autres que vous définirez sont dans le fichier [ecole.js] (ligne 63) :



La fonction Javascript [afficherEleves] est la suivante :

```

1. function afficherEleves() {
2.   // on récupère des références
3.   var formulaire = $("#formulaire");
4.   var content = $("#content");
5.   // on fait un appel Ajax à la main
6.   $.ajax({
7.     url: '/Ecole/AfficherEleves',
8.     type: 'POST',
9.     data: formulaire.serialize(),
10.    dataType: 'html',
11.    success: function (data) {
12.      // affichage résultats
13.      content.html(data);
14.    },
15.    error: function (jqXHR) {
16.      // affichage erreur
17.      content.html(jqXHR.responseText);
18.    },
19.  });
20. }

```

- ligne 4 : une référence Javascript sur la zone identifiée par [id=content], donc la zone où sont insérées les vues ;
- ligne 3 : une référence Javascript sur la zone identifiée par [id=formulaire], donc le formulaire HTML ;
- lignes 7-9 : les valeurs du formulaire sont récupérées (ligne 9) et postées (ligne 8) à l'URL [/Ecole/AfficherEleves] (ligne 7). Nous avons vu que ce formulaire n'avait qu'une seule valeur à poster [IdClasse=n] où n est l'identifiant de la classe sélectionnée ;
- ligne 10 : la réponse attendue du serveur est de type HTML ;
- lignes 11-14 : la réponse HTML reçue est placée dans la zone identifiée par [id=content]. En clair, la réponse renvoyée par le serveur remplace la vue actuellement affichée dans la zone identifiée par [id=content].

La vue renvoyée par le serveur est celle renvoyée par l'action [/Ecole/AfficherEleves] définie dans la classe [EcoleController]. Son code est le suivant :

```

1.  [HttpPost]
2.  public PartialViewResult AfficherEleves(ApplicationModel application, SessionModel session, long IdClasse) {
3.      // on récupère les élèves de la classe
4.      List<Eleve> élèves = application.Metier.GetElevesForClasse(IdClasse);
5.      // calcul des éléments du combo
6.      session.ElevesItems = new SelectListItems[élèves.Count];
7.      int i = 0;
8.      foreach (Eleve élève in élèves) {
9.          session.ElevesItems[i] = new SelectListItem() { Text = string.Format("{0} {1}", élève.Prénom, élève.Nom), Value =
    élève.Id.ToString() };
10.         i++;
11.     }
12.     // on affiche la vue
13.     return PartialView("Exo01", new Exo01Model() { Application = application, Session = session });
14. }
```

- ligne 2 : les paramètres de la méthode sont le modèle de l'application, le modèle de la session, et l'identifiant de la classe sélectionnée par l'utilisateur. Le nom du paramètre [IdClasse] est le nom du paramètre posté par la fonction Javascript étudiée précédemment ;
- lignes 4-11 – servent à calculer les éléments de la liste déroulante des élèves. On notera que ceux-ci sont mis dans la session (ligne 6) car c'est là que la vue [Exo01.cshtml] les attend ;
- ligne 13 : on renvoie la vue [Exo01.cshtml] qui va remplacer la vue précédemment affichée ;

**Question 1:** testez le lien [Afficher les élèves] puis complétez l'application pour faire afficher les notes de l'élève sélectionné comme montré ci-dessous. Il faut compléter le contrôleur [EcoleController], le modèle [Exo01Model] et fichier Javascript [ecole.js]. Suivez pour cela la démarche expliquée pour le lien [Afficher les élèves].

## 6.10 L'URL [Ecole/Exo02]

L'action [Exo02] fait afficher les notes d'une classe dans une matière :

Examen EI5 AGI 1314 - Session 2

**Exercice 2**

Classes :

Matières :

[Afficher les notes](#)

Moyenne : -1  
Ecart-type : -1  
Max : -1  
Min : -1

Examen EI5 AGI 1314 - Session 2

**Exercice 2**

Classes :

Matières :

[Afficher les notes](#)

Julien Boisseau : 10
Valérie Charlet : 14
Gabriel Monfort : 8
Emile Parton : 12

Moyenne : 11  
Ecart-type : 2,23606797749979  
Max : 14  
Min : 8

La vue [Exo02.cshtml] vous est donnée incomplète :

Examen EI5 AGI 1314 - Session 2

**Exercice 2**

Classes :

---

**Question 2 :** Complétez la vue [Exo02.cshtml], le contrôleur [EcoleController], le modèle [Exo02Model] et fichier Javascript [ecole.js] pour obtenir le fonctionnement des vues 1 à 2 ci-dessus.

---

**Note :** L'écart type d'une série de N notes  $Note_i$  est donné par la formule : racine carrée de  $(S_2/N - S_1^2/N^2)$  avec :  
 $S_2 = Note_1^2 + Note_2^2 + \dots + Note_N^2$   
 $S_1 = Note_1 + Note_2 + \dots + Note_N$

## 7 Étude de cas n° 6 : client / serveur jSON

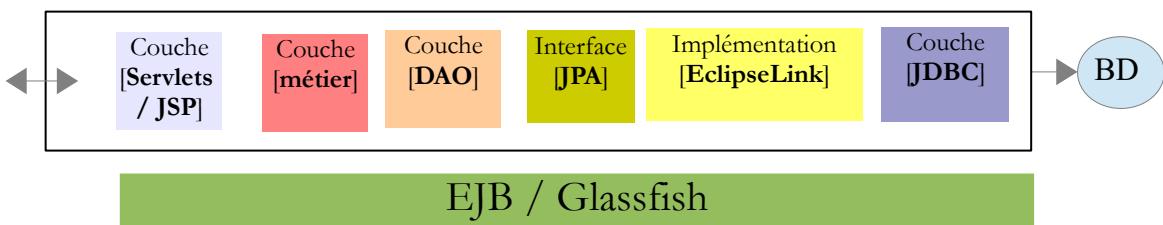
### 7.1 Les compétences mises en oeuvre

Les compétences mises en oeuvre par cet exercice sont les suivantes :

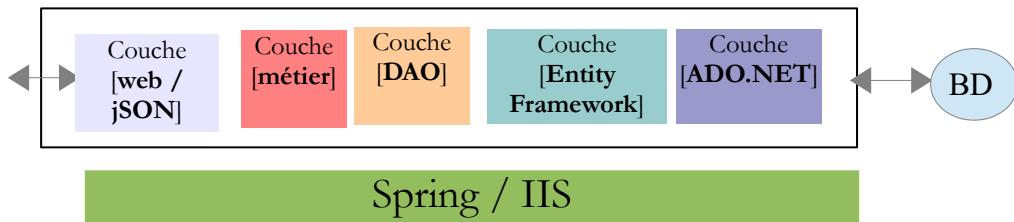
- maîtriser Visual Studio et son environnement de développement et notamment savoir utiliser le débogueur ;
- savoir utiliser WampServer, PhpMyAdmin, MySQL ;
- maîtriser l'injection de dépendances de Spring ;
- maîtriser Entity Framework ;

### 7.2 Le problème

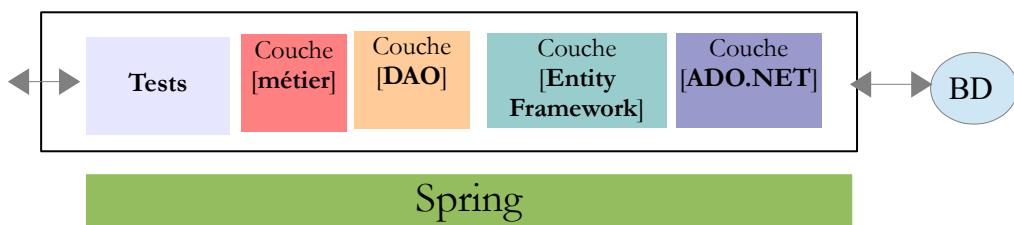
On trouve à l'adresse [https://netbeans.org/kb/docs/javaee/ecommerce/intro.html] un tutoriel d'apprentissage de Java EE sous Netbeans. Il s'agit d'une application web de commerce électronique. Cette application utilise l'architecture à couches suivante :



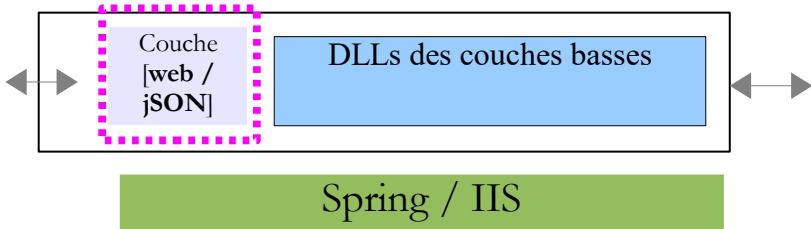
Nous allons porter cette application dans un environnement ASP.NET MVC / IIS :



Nous allons diviser cette étude en deux. Nous allons d'abord étudier les couches basses de l'application, celles qui ne nécessitent pas la présence du serveur IIS :

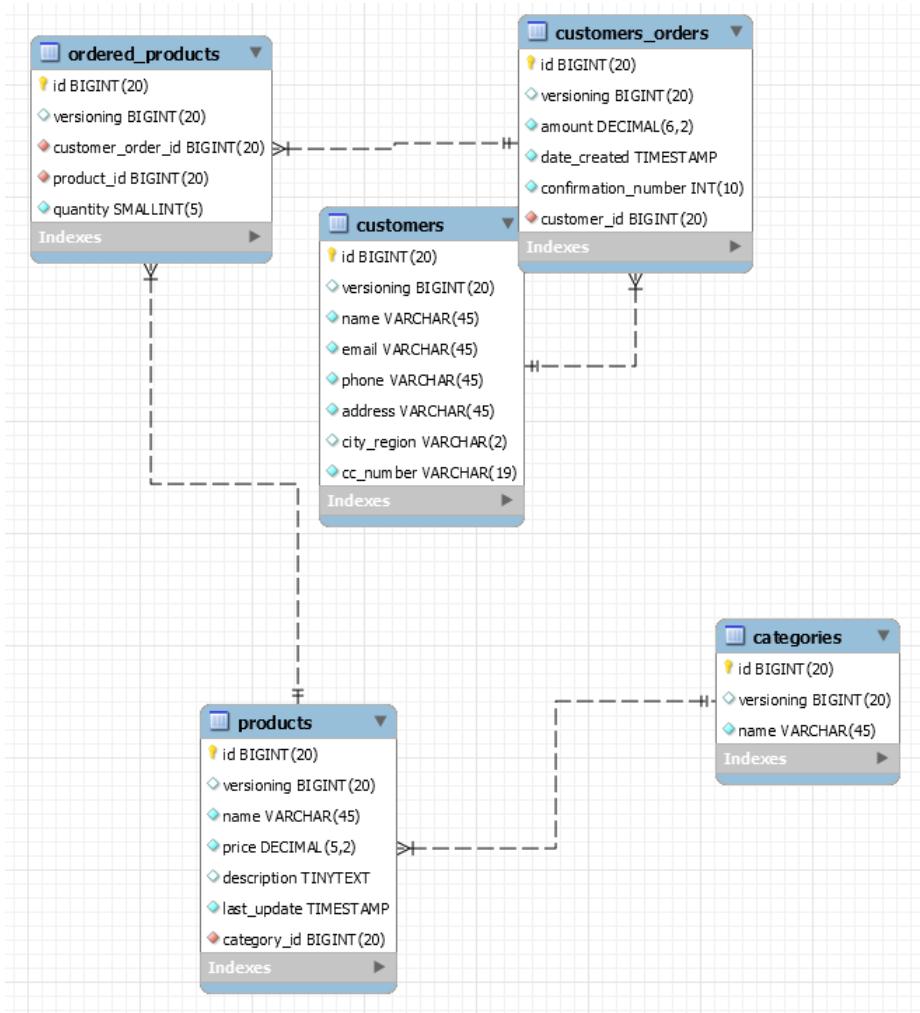


puis nous étudierons la couche haute, la couche web / jSON implémentée avec le framework ASP.NET MVC :



### 7.3 La base de données

La base de données utilisée est une base MySQL nommée "e-magasin-ef". Sa structure est la suivante :



L'application est une application de commerce électronique.

- on y vend des produits enregistrés dans la table [product]. Un produit appartient à une catégorie enregistrée dans la table [category]. Le lien entre les deux tables se fait via la clé étrangère **[product].category\_id** ;
- les trois autres tables [customer, customer\_order, ordered\_product] sont au départ vides ;
- un client qui fait un achat le fait via une page web de confirmation où il entre des informations le concernant. Celles-ci seront enregistrées dans la table [customer]. Sa commande est enregistrée dans la table [customer\_order]. Celle-ci a un lien vers la table [customer] via la clé étrangère **[customer\_order].customer\_id**. Chaque produit acheté fait l'objet d'un enregistrement dans la table [ordered\_product]. La clé étrangère **[ordered\_product].product\_id** fait un lien vers le produit acheté, la clé étrangère **[ordered\_product].customer\_order\_id** faisant elle un lien vers la table [customer\_order] et donc vers la table [customer].

La définition des tables est la suivante :

```
1. SET FOREIGN_KEY_CHECKS=0;
2.
3. CREATE DATABASE `e-magasin-ef`
4.   CHARACTER SET 'utf8'
5.   COLLATE 'utf8_unicode_ci';
6.
7. USE `e-magasin-ef`;
8.
9. #
10. # Structure for the `categories` table :
11. #
12.
13. CREATE TABLE `categories` (
14.   `id` BIGINT(20) NOT NULL AUTO_INCREMENT,
15.   `versioning` BIGINT(20) DEFAULT 1,
16.   `name` VARCHAR(45) COLLATE utf8_unicode_ci NOT NULL DEFAULT '1',
17.   PRIMARY KEY (`id`) USING BTREE,
18.   UNIQUE KEY `name`(`name`) USING BTREE
19. ) ENGINE=InnoDB
20. AUTO_INCREMENT=5 CHARACTER SET 'utf8' COLLATE 'utf8_unicode_ci'
21. COMMENT='contains product categories, e.g., dairy, meats, etc.'
22. ;
23.
24. #
25. # Structure for the `customers` table :
26. #
27.
28. CREATE TABLE `customers` (
29.   `id` BIGINT(20) NOT NULL AUTO_INCREMENT,
30.   `versioning` BIGINT(20) DEFAULT 1,
31.   `name` VARCHAR(45) COLLATE utf8_unicode_ci NOT NULL,
32.   `email` VARCHAR(45) COLLATE utf8_unicode_ci NOT NULL,
33.   `phone` VARCHAR(45) COLLATE utf8_unicode_ci NOT NULL,
34.   `address` VARCHAR(45) COLLATE utf8_unicode_ci NOT NULL,
35.   `city_region` VARCHAR(2) COLLATE utf8_unicode_ci DEFAULT NULL,
36.   `cc_number` VARCHAR(19) COLLATE utf8_unicode_ci NOT NULL,
37.   PRIMARY KEY (`id`) USING BTREE,
38.   UNIQUE KEY `name`(`name`) USING BTREE,
39.   UNIQUE KEY `cc_number`(`cc_number`) USING BTREE
40. ) ENGINE=InnoDB
41. AUTO_INCREMENT=42 CHARACTER SET 'utf8' COLLATE 'utf8_unicode_ci'
42. COMMENT='maintains customer details'
43. ;
44.
45. #
46. # Structure for the `customers_orders` table :
47. #
48.
49. CREATE TABLE `customers_orders` (
50.   `id` BIGINT(20) NOT NULL AUTO_INCREMENT,
51.   `versioning` BIGINT(20) DEFAULT 1,
52.   `amount` DECIMAL(8,2) DEFAULT NULL,
53.   `date_created` TIMESTAMP NOT NULL DEFAULT CURRENT_TIMESTAMP,
54.   `confirmation_number` INTEGER(10) UNSIGNED DEFAULT NULL,
55.   `customer_id` BIGINT(20) NOT NULL,
56.   PRIMARY KEY (`id`) USING BTREE,
57.   KEY `fk_customer_order_customer`(`customer_id`) USING BTREE,
58.   CONSTRAINT `customers_orders_fk1` FOREIGN KEY (`customer_id`) REFERENCES `customers`(`id`) ON DELETE CASCADE ON UPDATE CASCADE
59. ) ENGINE=InnoDB
60. AUTO_INCREMENT=35 CHARACTER SET 'utf8' COLLATE 'utf8_unicode_ci'
61. COMMENT='maintains customer order details'
62. ;
63.
64. #
65. # Structure for the `products` table :
66. #
67.
68. CREATE TABLE `products` (
69.   `id` BIGINT(20) NOT NULL AUTO_INCREMENT,
70.   `versioning` BIGINT(20) DEFAULT 1,
71.   `name` VARCHAR(45) COLLATE utf8_unicode_ci NOT NULL,
72.   `price` DECIMAL(5,2) NOT NULL,
73.   `description` TINYTEXT COLLATE utf8_unicode_ci,
74.   `last_update` TIMESTAMP NOT NULL DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP,
75.   `category_id` BIGINT(20) NOT NULL,
76.   PRIMARY KEY (`id`) USING BTREE,
77.   UNIQUE KEY `name`(`name`) USING BTREE,
78.   KEY `fk_product_category`(`category_id`) USING BTREE,
79.   CONSTRAINT `products_fk1` FOREIGN KEY (`category_id`) REFERENCES `categories`(`id`) ON DELETE CASCADE ON UPDATE CASCADE
80. ) ENGINE=InnoDB
81. AUTO_INCREMENT=21 CHARACTER SET 'utf8' COLLATE 'utf8_unicode_ci'
82. COMMENT='contains product details'
83. ;
```

```

84.
85. #
86. # Structure for the `ordered_products` table :
87. #
88.
89. CREATE TABLE `ordered_products` (
90.     `id` BIGINT(20) NOT NULL AUTO_INCREMENT,
91.     `versioning` BIGINT(20) DEFAULT 1,
92.     `customer_order_id` BIGINT(20) NOT NULL,
93.     `product_id` BIGINT(20) NOT NULL,
94.     `quantity` SMALLINT(5) UNSIGNED NOT NULL DEFAULT 1,
95.     PRIMARY KEY (`id`) USING BTREE,
96.     UNIQUE KEY `customer_order_id` (`customer_order_id`, `product_id`) USING BTREE,
97.     KEY `fk_ordered_product_customer_order` (`customer_order_id`) USING BTREE,
98.     KEY `fk_ordered_product_product` (`product_id`) USING BTREE,
99.     CONSTRAINT `ordered_products_fk1` FOREIGN KEY (`customer_order_id`) REFERENCES `customers_orders` (`id`) ON DELETE CASCADE ON UPDATE CASCADE,
100.    CONSTRAINT `ordered_products_fk2` FOREIGN KEY (`product_id`) REFERENCES `products` (`id`) ON DELETE CASCADE ON UPDATE CASCADE
101. ) ENGINE=InnoDB
102. AUTO_INCREMENT=53 CHARACTER SET 'utf8' COLLATE 'utf8_unicode_ci'
103. COMMENT='matches products with customer orders and records their quantity'
104. ;
105.
106. DELIMITER $$

107.

108. CREATE DEFINER = 'root'@'localhost' TRIGGER `categories_versioning_insert` BEFORE INSERT ON `categories`
109.     FOR EACH ROW
110. BEGIN
111.     if NEW.VERSIONING IS NULL THEN
112.         SET NEW.VERSIONING=1;
113.     END IF;
114. END$$
115.
116. CREATE DEFINER = 'root'@'localhost' TRIGGER `categories_versioning_update` BEFORE UPDATE ON `categories`
117.     FOR EACH ROW
118. BEGIN
119.     SET NEW.VERSIONING=OLD.VERSIONING+1;
120. END$$
121.
122. DELIMITER ;
123.
124. ...

```

- on notera que toutes les clés primaires sont en mode [AUTO\_INCREMENT], c-à-d générées par le SGBD ;
- on notera lignes 58, 79, 99, 100 les attributs [ON DELETE CASCADE] et [ON UPDATE CASCADE] sur les clés étrangères ;
- ligne 58, l'attribut [ON DELETE CASCADE] sur la clé étrangère [CustomerOrders.customer\_id --> Customers.id] fait que lorsqu'on supprime un client [Customer], tous les ordres [CustomerOrder] qui dépendent de lui sont également supprimés ;
- ligne 99, l'attribut [ON DELETE CASCADE] sur la clé étrangère [OrderedProducts.customer\_order\_id --> CustomerOrders.id] fait que lorsqu'on supprime une commande [CustomerOrder], tous les produits [OrderedProduct] de cette commande sont également supprimés ;
- au final, les deux règles précédentes font que lorsqu'on supprime un client [Customer], tous les ordres [CustomerOrder] et les produits [OrderedProduct] qui dépendent de lui sont également supprimés. C'est important à comprendre car le code qui suit utilise cette particularité ;

La gestion du versioning des lignes est déléguée à des *triggers*. Les *triggers* sont des programmes embarqués par le SGBD. Ils ne sont pas normalisés, aussi diffèrent-ils d'un SGBD à un autre. Ils peuvent même ne pas exister dans certains SGBD.

- ligne 108 : on définit un code à exécuter avant chaque **insertion** dans la table [CATEGORIES] ;
- lignes 109-113 : le code dit que lorsqu'à l'insertion d'une nouvelle ligne la colonne [VERSIONING] est vide, alors on lui affecte la valeur 1 ;
- ligne 116 : on définit un code à exécuter avant chaque **mise à jour** dans la table [CATEGORIES] ;
- lignes 109-113 : le code dit qu'à la mise à jour d'une ligne la nouvelle valeur de la colonne [VERSIONING] est l'ancienne valeur augmentée de 1 ;

On retiendra de ces informations que vous n'avez à gérer :

- ni la gestion des clés primaires ;
- ni la gestion du versioning ;

Le SGBD s'en charge.

Les données de la base pourraient ressembler à ceci :

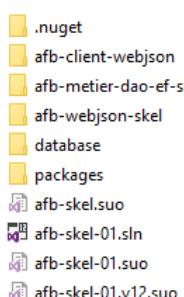
<b>id</b>	<b>name</b>	<b>id</b>	<b>name</b>	<b>price</b>	<b>description</b>	<b>last_update</b>	<b>category_id</b>
1	dairy	1	milk	1.70	semi skimmed (1L)	2012-10-30 16:36:41	1
2	meats	2	cheese	2.39	mild cheddar (330g)	2012-10-30 16:36:41	1
3	bakery	3	butter	1.09	unsalted (250g)	2012-10-30 16:36:41	1
4	fruit & veg	4	free range eggs	1.76	medium-sized (6 eggs)	2012-10-30 16:36:41	1
		5	organic meat patties	2.29	rolled in fresh herbs 2 patties (250g)	2012-10-30 16:36:41	2
		6	parma ham	3.49	matured, organic (70g)	2012-10-30 16:36:41	2
		7	chicken leg	2.59	free range (250g)	2012-10-30 16:36:41	2
		8	sausages	3.55	reduced fat, pork 3 sausages (350g)	2012-10-30 16:36:41	2
		9	sunflower seed loaf	1.89	600g	2012-10-30 16:36:41	3
		10	sesame seed bagel	1.19	4 bagels	2012-10-30 16:36:41	3
		11	pumpkin seed bun	1.15	4 buns	2012-10-30 16:36:41	3
		12	chocolate cookies	2.39	contain peanuts (3 cookies)	2012-10-30 16:36:41	3
		13	corn on the cob	1.59	2 pieces	2012-10-30 16:36:41	4
		14	red currants	2.49	150g	2012-10-30 16:36:41	4
		15	broccoli	1.29	500g	2012-10-30 16:36:41	4
		16	seedless watermelon	1.49	250g	2012-10-30 16:36:41	4

[product]

<b>id</b>	<b>name</b>	<b>email</b>	<b>phone</b>	<b>address</b>	<b>city_region</b>	<b>cc_number</b>	<b>id</b>	<b>amount</b>	<b>date_created</b>	<b>confirmation_number</b>	<b>customer_id</b>	<b>ID</b>	<b>customer_order_id</b>	<b>product_id</b>	<b>quantity</b>	
40	1	2	3	4	5	6	40	10.57	2012-11-04 15:52:16	317891637	40	49		1	2	
												50		40	2	3

## 7.4 Mise en place de l'environnement de travail

Le dossier suivant vous est fourni :



- [database] : contient le script SQL de génération de la base MySQL [e-magasin] ;
- [afb-metier-dao-ef-skel] : contient le projet Visual Studio des couches basses de l'application ;
- [afb-webjson-skel] : contient le projet Visual Studio d'un serveur web / JSON ;
- [afb-client-webjson] : contient le projet Visual Studio d'un client du serveur web / JSON précédent ;

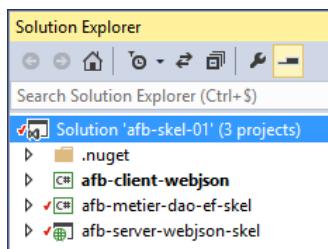
### Installation de la base de données



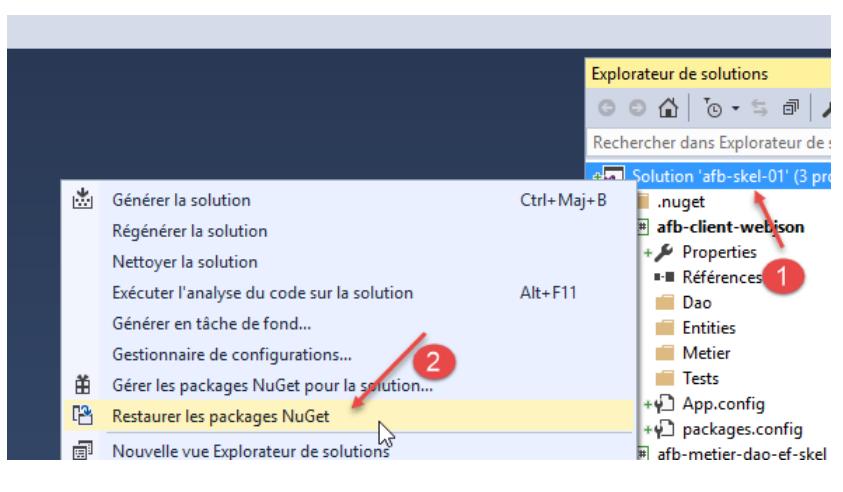
Avec WampServer / PhpMyAdmin, exécutez le script SQL trouvé dans le dossier [database]. Le script crée la base [e-magasin-ef] et la remplit avec des données. La base ne doit pas déjà exister.

### La solution Visual Studio

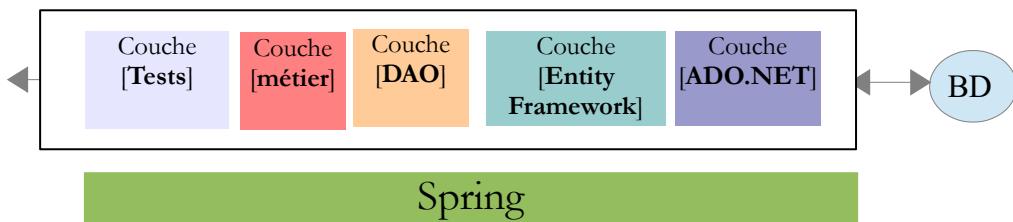
Chargez dans Visual Studio, la solution [afb-skel-01.sln] du dossier qui vous est donné :



Cette solution utilise des packages NuGet absents dans l'archive de la solution. Procédez comme suit pour les régénérer :

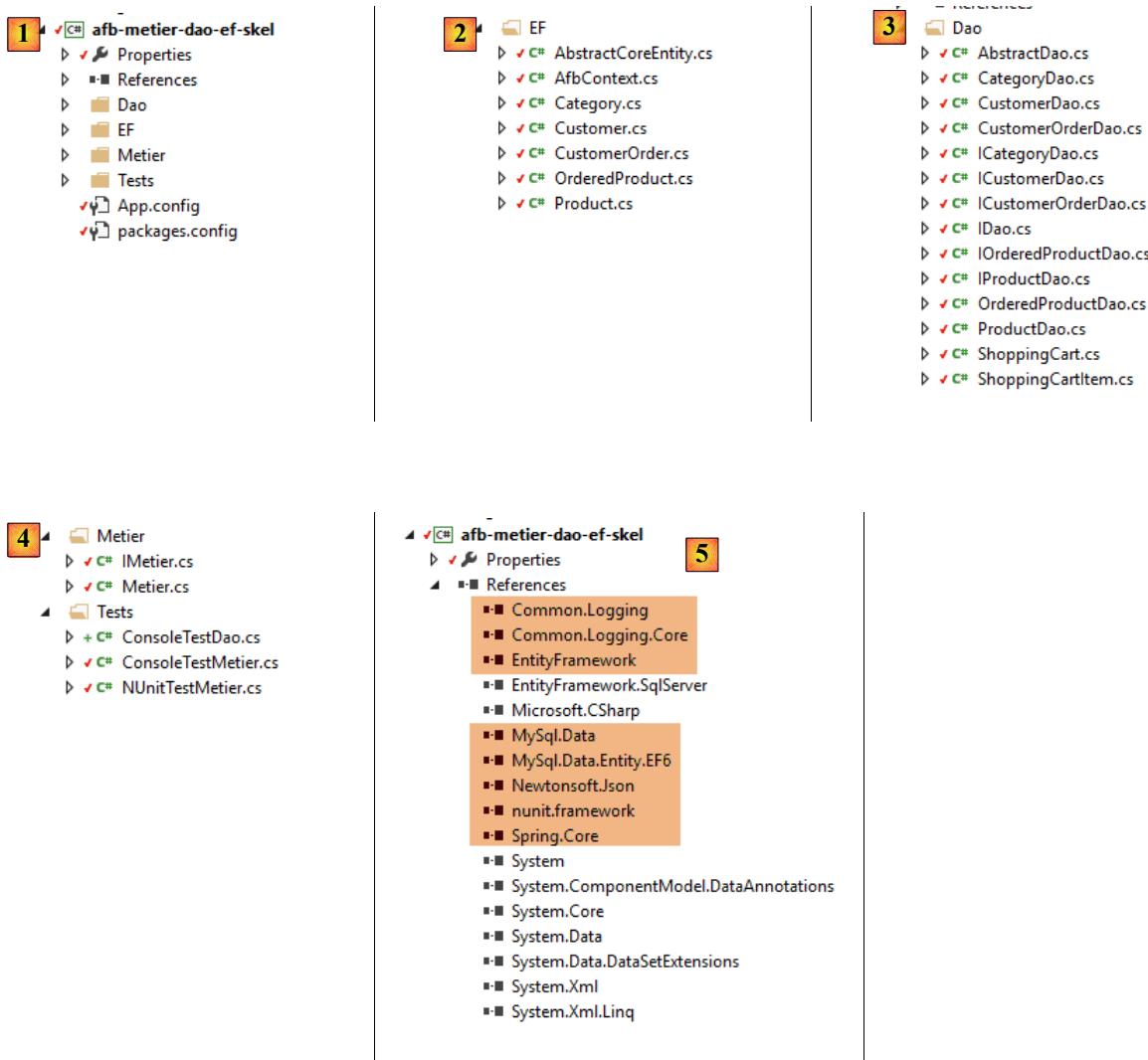


## 7.5 Ecriture des couches basses



## 7.5.1 Le projet Visual Studio

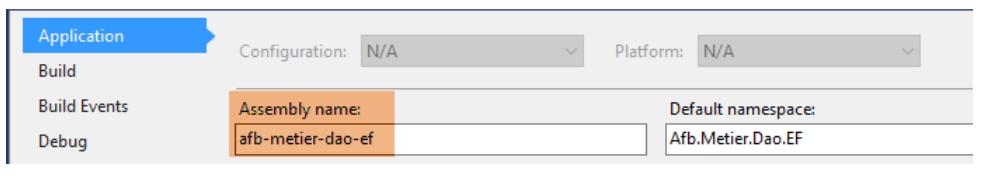
Le projet Visual Studio des couches basses est le suivant :



- en [1], les codes C# de l'application :
  - [EF] : la couche Entity Framework qui fait le lien entre la couche [DAO] et la base de données ;
  - [Dao] : la couche [DAO] ;
  - [Metier] : la couche [métier] ;
  - [Tests] : la couche de tests ;
- en [2], la couche [Entity Framework] :
  - [AbstractCoreEntity] : classe parent de toutes les entités gérées par EF ;
  - [Category] : classe image d'une ligne de la table [Categories] ;
  - [Product] : classe image d'une ligne de la table [Products] ;
  - [Customer] : classe image d'une ligne de la table [Customers] ;
  - [CustomerOrder] : classe image d'une ligne de la table [CustomersOrders] ;
  - [OrderedProduct] : classe image d'une ligne de la table [OrderedProducts] ;
- en [3], la couche [DAO] :
  - [IDao, ICategoryDao, ICustomerDao, IProductDao, ICustomerOrderDao, IOrderedProductDao] : les interfaces de la couche [DAO] ;
  - [AbstractDao, CategoryDao, CustomerDao, ProductDao, CustomerOrderDao, OrderedProductDao] : les classes implémentant ces interfaces ;
  - [ShoppingCart] : modélise un panier d'achats. L'application est une application de commerce électronique ;
  - [ShoppingCartItem] : modélise un article acheté ;
- en [4], la couche [metier] :
  - [IMetier] : l'interface de la couche [métier] ;

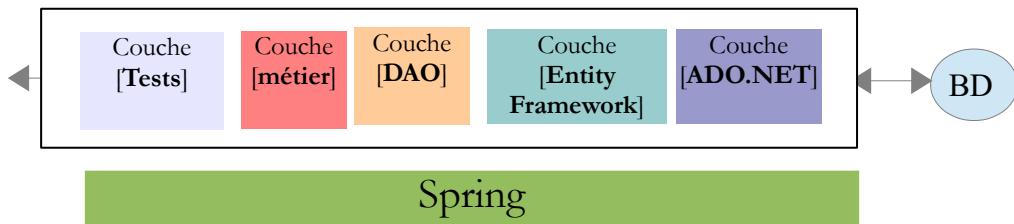
- [Metier] : l'implémentation de cette interface,
- en [4], la couche [Tests] :
  - [ConsoleTestDao] : un test console de la couche [DAO] ;
  - [ConsoleTestMetier] : un test console de la couche [métier] ;
  - [NUnitTestMetier] : le même test que précédemment mais transformé en test NUnit ;
- en [5], les dépendances du projet :
  - [EntityFramework] : le framework [Entity Framework] ;
  - [ MySql.Data.\* ] : le pilote ADO.NET du SGBD MySQL ainsi que l'adaptateur MySQL pour Entity Framework ;
  - [ Spring.Core, Common.Logging.\* ] : le framework Spring avec le framework [Common.Logging] de logs ;
  - [nunit.framework] : le framework de tests unitaires NUnit ;
  - [Newtonsoft.Json] : une bibliothèque de gestion du JSON ;

Vérifiez dans les propriétés du projet le nom de l'assembly généré par le projet :



Le nom de l'assembly généré doit être celui de la copie d'écran ci-dessus. Ce nom est utilisé dans le fichier de configuration [App.config] du projet ;

### 7.5.2 La couche [Entity Framework]



```

    ▾ EF
    └─ v C# AbstractCoreEntity.cs
    └─ v C# AfbContext.cs
    └─ v C# Category.cs
    └─ v C# Customer.cs
    └─ v C# CustomerOrder.cs
    └─ v C# OrderedProduct.cs
    └─ v C# Product.cs
  
```

#### 7.5.2.1 Le code

Les entités dites " persistantes " sont les objets C# qui encapsulent les lignes des tables de la base de données.

La classe [AbstractCoreEntity] est la classe parent de toutes les entités persistantes :

```

1.  using Newtonsoft.Json;
2.  using System;
3.  using System.ComponentModel.DataAnnotations;
4.  using System.ComponentModel.DataAnnotations.Schema;
5.
6.  namespace Afb.Ef
7.  {
8.      [Serializable]
9.      public class AbstractCoreEntity
10.     {
11.
  
```

```

12.     [Key]
13.     [Column("ID")]
14.     public int? Id { get; set; }
15.
16.     [ConcurrencyCheck]
17.     [Column("VERSIONING")]
18.     public int? Version { get; set; }
19.
20.     // signature
21.     public override string ToString()
22.     {
23.         return JsonConvert.SerializeObject(this);
24.     }
25. }
26.

```

- lignes 12-14 : les entités ont toutes une clé primaire [Id] ;
- lignes 16-18 : les entités ont toutes une propriété [Version] utilisée par EF pour gérer la concurrence d'accès à une ligne ([ConcurrencyCheck], ligne 16) ;

La classe [Category] est l'entité représentant une ligne de la table [CATEGORY] :

```

1. using System;
2.
3. namespace Afb.Ef
4. {
5.     public class Category : AbstractCoreEntity
6.     {
7.         public String Name { get; set; }
8.
9.     }
10. }

```

- ligne 7 : le nom de la catégorie ;

La classe [Product] est l'entité représentant une ligne de la table [PRODUCT] :

```

1. using System;
2.
3. namespace Afb.Ef
4. {
5.     public class Product : AbstractCoreEntity
6.     {
7.         public string Name { get; set; }
8.
9.         public string Description { get; set; }
10.
11.        public double Price { get; set; }
12.
13.        public DateTime LastUpdate { get; set; }
14.
15.        public int? CategoryId { get; set; }
16.
17.        public Category Category { get; set; }
18.
19.    }
20. }

```

- ligne 13 : la valeur du champ *LastUpdate* est générée par le SGBD. On peut ne pas l'initialiser lors d'une insertion ;
- ligne 15 : le n° de la catégorie à laquelle appartient le produit. C'est une clé étrangère sur la table [CATEGORY] ;
- ligne 17 : la catégorie à laquelle appartient le produit ;

#### A noter :

- lors d'une lecture en base, le champ [CustomerId] est initialisé mais pas toujours le champ [Customer] (lazy loading) ;
- lors d'une insertion en base d'un type [CustomerOrder] : si [CustomerId==null] (ligne 13), alors il y aura persistance par cascade de l'objet [Customer] de la ligne 15 (qui doit donc exister), sinon il n'y aura pas de persistance de cet objet [Customer] (qui peut donc être null) ;

La classe [Customer] est l'entité représentant une ligne de la table [CUSTOMER] :

```

1. using System;
2.
3. namespace Afb.Ef
4. {
5.     public class Customer : AbstractCoreEntity
6.     {
7.         public String Name { get; set; }
8.

```

```

9.     public String Email { get; set; }
10.    public String Phone { get; set; }
11.    public String Address { get; set; }
12.    public String CityRegion { get; set; }
13.    public String CcNumber { get; set; }
14.
15.    }
16.
17.    }
18.
19.    }
20. }
```

- lors d'une insertion d'un type [Customer], on pourra ne pas initialiser le champ [CityRegion] car la colonne correspondante dans la base accepte les valeurs [NULL] ;

La classe [CustomerOrder] est l'entité représentant une ligne de la table [CUSTOMER\_ORDER] :

```

1.  using System;
2.
3.  namespace Afb.Ef
4.  {
5.      public class CustomerOrder : AbstractCoreEntity
6.      {
7.          public double Amount { get; set; }
8.
9.          public DateTime? DateCreated { get; set; }
10.
11.         public int ConfirmationNumber { get; set; }
12.
13.         public int? CustomerId { get; set; }
14.
15.         public Customer Customer { get; set; }
16.     }
17. }
```

- ligne 13 : le n° du client qui a fait la commande. La colonne correspondante dans la base est clé étrangère sur la colonne [CUSTOMERS.ID] ;
- ligne 15 : le client qui a fait la commande ;

#### A noter :

- lors d'une lecture en base, le champ [CustomerId] est initialisé mais pas toujours le champ [Customer] (lazy loading) ;
- lors d'une insertion en base d'un type [CustomerOrder] : si [CustomerId==null] (ligne 13), alors il y aura persistance par cascade de l'objet [Customer] de la ligne 15 (qui doit donc exister), sinon il n'y aura pas de persistance de cet objet [Customer] (qui peut donc être null) ;

La classe [OrderedProduct] est l'entité représentant une ligne de la table [ORDERED\_PRODUCT] :

```

1.  namespace Afb.Ef
2.  {
3.      public class OrderedProduct : AbstractCoreEntity
4.      {
5.          public int Quantity { get; set; }
6.
7.          public int? ProductId { get; set; }
8.
9.          public Product Product { get; set; }
10.
11.         public int? CustomerOrderId { get; set; }
12.
13.         public CustomerOrder CustomerOrder { get; set; }
14.
15.     }
16. }
```

- ligne 7 : le n° de l'article acheté. La colonne correspondante dans la base est clé étrangère sur la colonne [PRODUCTS.ID] ;
- ligne 9 : le produit acheté ;
- ligne 11 : le n° de la commande à laquelle appartient ce produit acheté. La colonne correspondante dans la base est clé étrangère sur la colonne [CUSTOMER\_ORDERS.ID] ;
- ligne 13 : l'ordre de commande ;

#### A noter :

- lors d'une lecture en base, le champ [ProductId] est initialisé mais pas toujours le champ [Product] (lazy loading). De même, le champ [CustomerOrderId] est initialisé mais pas toujours le champ [CustomerOrder] ;

- lors d'une insertion en base d'un type [OrderedProduct] :
  - si [ProductId==null] (ligne 7), alors il y aura persistance par cascade de l'objet [Product] de la ligne 9 (qui doit donc exister), sinon il n'y aura pas de persistance de cet objet [Product] (qui peut donc être *null*) ;
  - si [CustomerOrderId==null] (ligne 11), alors il y aura persistance par cascade de l'objet [CustomerOrder] de la ligne 13 (qui doit donc exister), sinon il n'y aura pas de persistance de cet objet [CustomerOrder] (qui peut donc être *null*) ;

La classe [AfbContext] est le contexte d'Entity Framework qui rassemble les entités que l'on vient de décrire :

```

1.  using System.Data.Entity;
2.
3.  namespace Afb.EF
4.  {
5.      public class AfbContext : DbContext
6.      {
7.          // les entités
8.          public DbSet<Product> Products { get; set; }
9.          public DbSet<Category> Categories { get; set; }
10.         public DbSet<Customer> Customers { get; set; }
11.         public DbSet<CustomerOrder> CustomersOrders { get; set; }
12.         public DbSet<OrderedProduct> OrderedProducts { get; set; }
13.     }
14. }
```

**Question 1 :** compléter le code des entités persistantes afin qu'elles soient exploitable par Entity Framework et qu'elles vérifient les spécifications précédentes sur le mode [*lazy loading*] de certaines entités.

#### Conseils :

- il y a une difficulté avec la gestion des dates. Il faut leur ajouter un attribut :

#### Classe Product

```

1.  [DatabaseGenerated(DatabaseGeneratedOption.Computed)]
2.  [Column("LAST_UPDATE")]
3.  public DateTime? LastUpdate { get; set; }
```

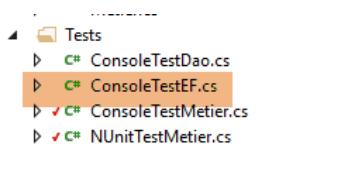
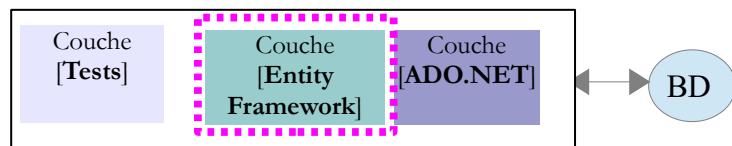
Il faut ajouter l'attribut de la ligne 1 pour indiquer à Entity Framework que la date est automatiquement générée par le SGBD. On fait de même pour la classe [CustomerOrder].

#### Classe CustomerOrder

```

1.  [DatabaseGenerated(DatabaseGeneratedOption.Computed)]
2.  [Column("DATE_CREATED")]
3.  public DateTime? DateCreated { get; set; }
```

#### 7.5.2.2 *Le test*



#### 7.5.2.2.1 *Le code*

Le test de la couche EF est le suivant :

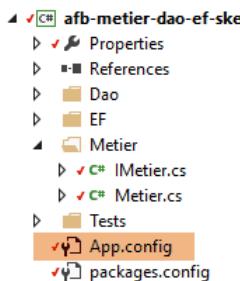
```
1.  using Afb.Dao;
2.  using Afb.Ef;
3.  using Newtonsoft.Json;
4.  using System;
5.  using System.Collections.Generic;
6.  using System.Linq;
7.
8.  namespace Afb.Tests
9.  {
10.     class ConsoleTestEf
11.     {
12.         static void Main(string[] args)
13.         {
14.             int orderedProductId;
15.             try
16.             {
17.                 // contexte EF
18.                 using (AfbContext afbContext = new AfbContext())
19.                 {
20.                     // catégories
21.                     Console.WriteLine("Catégories -----");
22.                     List<Category> categories = afbContext.Categories.ToList();
23.                     Console.WriteLine(JsonConvert.SerializeObject(categories));
24.                     // produits
25.                     Console.WriteLine("Produits de la catégorie n° 1 -----");
26.                     int categoryId = (int)categories[0].Id;
27.                     List<Product> products = afbContext.Products.Where(p => p.CategoryId == categoryId).ToList();
28.                     Console.WriteLine(JsonConvert.SerializeObject(products));
29.                     // reset base.
30.                     Console.WriteLine("ClearCustomers -----");
31.                     afbContext.Customers.RemoveRange(afbContext.Customers);
32.                     Console.WriteLine("PlaceOrder -----");
33.                     // remplissage chariot
34.                     ShoppingCart chariot = new ShoppingCart();
35.                     // 2 produits 0
36.                     chariot.Update(products[0], 2);
37.                     // 3 produits 1
38.                     chariot.Update(products[1], 3);
39.                     // on force le calcul du prix à payer
40.                     chariot.CalculateTotal();
41.                     // persistance commande
42.                     Customer customer = new Customer() { Name = "1", Email = "2", Phone = "3", Address = "4", CityRegion = "55",
43. CcNumber = "6" };
44.                     // l'ordre de commande
45.                     CustomerOrder order = new CustomerOrder() { Amount = chariot.Total, ConfirmationNumber = new
Random().Next(99999999), Customer = customer };
46.                     // persistance articles achetés et par cascade [CustomerOrder] puis [Customer]
47.                     foreach (ShoppingCartItem scItem in chariot.Items)
48.                     {
49.                         // persistance
50.                         // à noter : initialiser [ProductId] et pas [Product] afin qu'EF ne persiste pas le produit par cascade, ce qui
serait 1 erreur
51.                         afbContext.OrderedProducts.Add((new OrderedProduct() { Quantity = scItem.Quantity, CustomerOrder = order,
ProductId = scItem.Product.Id }));
52.                     }
53.                     // fin transaction
54.                     afbContext.SaveChanges();
55.                     Console.WriteLine(JsonConvert.SerializeObject(order));
56.                     // produits commandés
57.                     Console.WriteLine("OrderDetails -----");
58.                     // à noter : la navigation ["CustomerOrder.Customer"] pour forcer le chargement d'entités dépendantes
59.                     List<OrderedProduct> orderedProducts =
60.                     afbContext.OrderedProducts.Include("Product").Include("CustomerOrder").Include("CustomerOrder.Customer").Where(op =>
op.CustomerOrderId == order.Id).ToList<OrderedProduct>();
61.                     Console.WriteLine(JsonConvert.SerializeObject(orderedProducts));
62.                     // un produit acheté particulier
63.                     // ce produit a des dépendances facultatives (lazyness) - cependant si ces dépendances sont déjà présentes dans
le contexte alors elles sont ramenées avec le produit
64.                     orderedProductId = (int)orderedProducts[0].Id;
65.                     OrderedProduct orderedProduct = afbContext.OrderedProducts.Where(op => op.Id == orderedProductId).Single();
66.                     Console.WriteLine("OrderedProducts[0] (one) -----");
67.                     Console.WriteLine(JsonConvert.SerializeObject(orderedProduct));
68.                     // on redemande orderedProducts[0] avec un nouveau contexte EF
69.                     // on n'obtient pas la même chose que précédemment parce que cette fois-ci les dépendances facultatives du produit
ne sont pas dans le contexte
70.                     using (AfbContext afbContext = new AfbContext())
71.                     {
72.                         OrderedProduct orderedProduct = afbContext.OrderedProducts.Find(orderedProductId);
73.                         Console.WriteLine("OrderedProducts[0] (two) -----");
74.                         Console.WriteLine(JsonConvert.SerializeObject(orderedProduct));
75.                     }
76.                     catch (Exception e)
```

```

77.         {
78.             // on signale l'erreur
79.             Console.WriteLine(string.Format("L'erreur suivante s'est produite : {0}", e.Message));
80.         }
81.     }
82. }
83. }
```

Comprenez ce code. Il vous sera utile ultérieurement.

### 7.5.2.2.2 La configuration



Le projet [afb-metier-dao-ef-skel] est configuré par le fichier [App.config] suivant :

```

1. <?xml version="1.0" encoding="utf-8"?>
2. <configuration>
3.   <configSections>
4.     <!-- For more information on Entity Framework configuration, visit http://go.microsoft.com/fwlink/?LinkId=237468 -->
5.     <section name="entityFramework" type="System.Data.Entity.Internal.ConfigFile.EntityFrameworkSection,
EntityFramework, Version=6.0.0.0, Culture=neutral, PublicKeyToken=b77a5c561934e089" requirePermission="false" />
6. ...
7.   </configSections>
8. ...
9.
10.  <startup>
11.    <supportedRuntime version="v4.0" sku=".NETFramework,Version=v4.5" />
12.  </startup>
13.  <entityFramework>
14.    <providers>
15.      <provider invariantName=" MySql.Data.MySqlClient " type=" MySql.Data.MySqlClient.MySqlProviderServices ,
MySql.Data.Entity.EF6, Version=6.9.7.0, Culture=neutral, PublicKeyToken=c5687fc88969c44d ">
16.        </provider>
17.      </providers>
18.    </entityFramework>
19.    <system.data>
20.      <DbProviderFactories>
21.        <remove invariant=" MySql.Data.MySqlClient " />
22.        <add name="MySQL Data Provider" invariant=" MySql.Data.MySqlClient " description=".Net Framework Data Provider for
MySQL" type=" MySql.Data.MySqlClient.MySqlClientFactory , MySql.Data, Version=6.9.7.0, Culture=neutral,
PublicKeyToken=c5687fc88969c44d " />
23.      </DbProviderFactories>
24.    </system.data>
25.    <connectionStrings>
26.      <add name="AfbContext" connectionString="Server=localhost;Database=e-magasin-ef;Uid=root;Pwd="
providerName=" MySql.Data.MySqlClient " />
27.    </connectionStrings>
28.  </configuration>
```

- ligne 26 : la chaîne de connexion [ADO.NET] pour se connecter à la base de données MySQL [e-magasin]. L'attribut [name="AfbContext"] n'est pas quelconque. [AfbContext] est le nom de la classe du contexte Entity Framework définie dans [AfbContext.cs] :

```

1. using System.Data.Entity;
2.
3. namespace Afb.EF
4. {
5.     public class AfbContext : DbContext
6.     {
7.         ...
8.     }
9. }
```

- ligne 26 : l'attribut [providerName=" MySql.Data.MySqlClient "] sert à indiquer le fournisseur ADO.NET à utiliser, c-à-d le pilote du SGBD qui gère la base de données, ici MySQL. Ce nom intervient à divers endroits : lignes 15, 21, 22.

- ligne 21 : le fournisseur ADO.NET [MySql.Data.MySqlClient] peut être déjà présent sur la machine, auquel cas la ligne 22 provoque un plantage. La ligne 21 enlève donc auparavant le fournisseur ADO.NET [MySql.Data.MySqlClient] de la machine ;
- ligne 26 : l'attribut [connectionString="Server=localhost;Database=e-magasin;Uid=root;Pwd="] est la chaîne de connexion ADO.NET à la base de données MySQL [e-magasin]. Cette chaîne est propriétaire. Elle change de forme avec chaque SGBD ;

### 7.5.2.2.3 Les résultats

Configurez votre projet pour exécuter le test console [ConsoleTestEf] puis exécutez le. Vous devez obtenir quelque chose qui ressemble à ceci :

```

1. Catégories -----
2. [{"Name":"dairy","Id":1,"Version":1}, {"Name":"meats","Id":2,"Version":1}, {"Name":"bakery","Id":3,"Version":1},
   {"Name":"fruit & veg","Id":4,"Version":1}]
3. Produits de la catégorie n° 1 -----
4. [{"Name":"milk","Description":"semi skimmed (1L)","Price":1.7,"LastUpdate":"2012-10-
   30T16:36:41+01:00","CategoryId":1,"Category":{"Name":"dairy","Id":1,"Version":1}, "Id":1,"Version":1},
   {"Name":"cheese","Description":"mild cheddar (330g)","Price":2.39,"LastUpdate":"2012-10-
   30T16:36:41+01:00","CategoryId":1,"Category":{"Name":"dairy","Id":1,"Version":1}, "Id":2,"Version":1},
   {"Name":"butter","Description":"unsalted (250g)","Price":1.09,"LastUpdate":"2012-10-
   30T16:36:41+01:00","CategoryId":1,"Category":{"Name":"dairy","Id":1,"Version":1}, "Id":3,"Version":1}, {"Name":"free range
   eggs","Description":"medium-sized (6 eggs)","Price":1.76,"LastUpdate":"2012-10-
   30T16:36:41+01:00","CategoryId":1,"Category":{"Name":"dairy","Id":1,"Version":1}, "Id":4,"Version":1}]
5. ClearCustomers -----
6. PlaceOrder -----
7. {"Amount":10.57,"DateCreated":"2015-10-21T16:30:06Z","ConfirmationNumber":72688071,"CustomerId":58,"Customer":
   {"Name":"1","Email":"2","Phone":"3","Address":"4","CityRegion":"55","CcNumber":"6","Id":58,"Version":null}, "Id":15,"Version
   ":null}
8.
9. OrderDetails -----
10. [{"Quantity":2,"ProductId":1,"Product":{"Name":"milk","Description":"semi skimmed (1L)","Price":1.7,"LastUpdate":"2012-10-
    30T16:36:41+01:00","CategoryId":1,"Category":
    {"Name":"dairy","Id":1,"Version":1}, "Id":1,"Version":1}, "CustomerOrderId":15,"CustomerOrder":
    {"Amount":10.57,"DateCreated":"2015-10-21T16:30:06Z","ConfirmationNumber":72688071,"CustomerId":58,"Customer":
    {"Name":"1","Email":"2","Phone":null,"Address":null,"CityRegion":null,"CcNumber":null,"Id":58,"Version":null}, "Id":15,"Version
    ":null}, "Id":2,"Product":{"Name":"cheese","Description":"mild cheddar
    (330g)","Price":2.39,"LastUpdate":"2012-10-30T16:36:41+01:00","CategoryId":1,"Category":
    {"Name":"dairy","Id":1,"Version":1}, "Id":2,"Version":1}, "CustomerOrderId":15,"CustomerOrder": {"Amount":1
    0.57,"DateCreated":"2015-10-21T16:30:06Z","ConfirmationNumber":72688071,"CustomerId":58,"Customer":
    {"Name":"1","Email":"2","Phone":null,"Address":null,"CityRegion":null,"CcNumber":null,"Id":58,"Version":null}, "Id":15,"Version
    ":null}, "Id":28,"Version":null}]
11. OrderedProducts[0] (one) -----
12. {"Quantity":2,"ProductId":1,"Product":{"Name":"milk","Description":"semi skimmed (1L)","Price":1.7,"LastUpdate":"2012-10-
    30T16:36:41+01:00","CategoryId":1,"Category":
    {"Name":"dairy","Id":1,"Version":1}, "Id":1,"Version":1}, "CustomerOrderId":15,"CustomerOrder":
    {"Amount":10.57,"DateCreated":"2015-10-21T16:30:06Z","ConfirmationNumber":72688071,"CustomerId":58,"Customer":
    {"Name":"1","Email":"2","Phone":null,"Address":null,"CityRegion":null,"CcNumber":null,"Id":58,"Version":null}, "Id":15,"Version
    ":null}, "Id":27,"Version":null}
13. OrderedProducts[0] (two) -----
14. {"Quantity":2,"ProductId":1,"Product":{"Name":"milk","Description":"semi skimmed (1L)","Price":1.7,"LastUpdate":"2012-10-
    30T16:36:41+01:00","CategoryId":1,"Category":
    {"Name":"dairy","Id":1,"Version":1}, "Id":1,"Version":1}, "CustomerOrderId":15,"CustomerOrder":
    {"Amount":10.57,"DateCreated":"2015-10-21T16:30:06Z","ConfirmationNumber":72688071,"CustomerId":58,"Customer":
    {"Name":"1","Email":"2","Phone":null,"Address":null,"CityRegion":null,"CcNumber":null,"Id":58,"Version":null}, "Id":15,"Version
    ":null}, "Id":27,"Version":null}
15. OrderedProducts[0] (two) -----
16. {"Quantity":2,"ProductId":1,"Product":null,"CustomerOrderId":15,"CustomerOrder":null,"Id":27,"Version":1}
17. Appuyez sur une touche pour continuer...

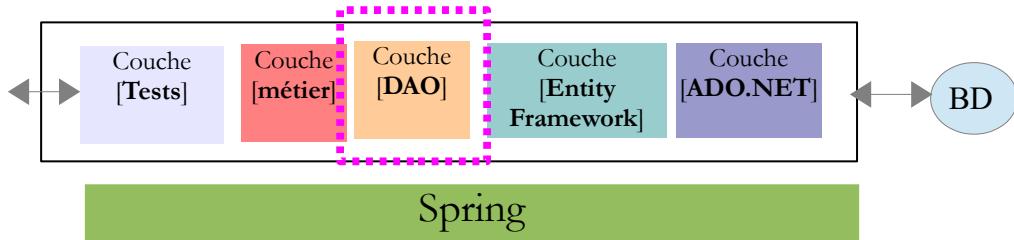
```

Observez la différence d'affichage du même objet [OrderedProduct], lignes 13-14 et lignes 15-16. Ils ont été obtenus avec deux contextes EF différents. On constate que :

- avec le 1er contexte, les champs [CustomerOrder] et [Product] ont été obtenus parce qu'ils étaient déjà présents dans le contexte lorsque le type [OrderedProduct] a été demandé. L'explication est la même pour expliquer que le champ [CustomerOrder.Customer] a lui aussi été obtenu ;
- avec le second contexte qui ne contenait pas ces informations, elles n'ont pas été fournies (lazy loading) ;
- avec le 1er contexte, on n'a pas la version du type [OrderedProduct] (fin de la ligne 14), alors que le second contexte la fournit (ligne 16) ;

Donc selon le contexte, on n'a ou pas les entités associées aux diverses clés étrangères, ici [CustomerOrder, Product, OrderedProduct]. Mais quelque soit le contexte, on a toujours les clés étrangères elles-mêmes, ici [CustomerOrderId, ProductId]. Pour obtenir les versions des entités, il faut utiliser un autre contexte que celui qui a procédé aux insertions.

### 7.5.3 La couche [DAO]



```

    ▲ Dao
    └─ ✓ C# AbstractDao.cs
    └─ ✓ C# CategoryDao.cs
    └─ ✓ C# CustomerDao.cs
    └─ ✓ C# CustomerOrderDao.cs
    └─ ✓ C# ICategoryDao.cs
    └─ ✓ C# ICustomerDao.cs
    └─ ✓ C# ICustomerOrderDao.cs
    └─ ✓ C# IDao.cs
    └─ ✓ C# IOrderedProductDao.cs
    └─ ✓ C# IProductDao.cs
    └─ ✓ C# OrderedProductDao.cs
    └─ ✓ C# ProductDao.cs
    └─ ✓ C# ShoppingCart.cs
    └─ ✓ C# ShoppingCartItem.cs
  
```

### 7.5.3.1 Le code

L'interface principale est l'interface générique [IDao<T>] :

```

1.  using Afb.Ef;
2.  using System;
3.  using System.Collections.Generic;
4.
5.  namespace Afb.Dao
6.  {
7.      // interface générique
8.      public interface IDao<T> where T : class
9.      {
10.          // nombre d'objets T
11.          int Count();
12.          // persister une entité T
13.          T Create(T entity);
14.          // modifier une entité T
15.          T Edit(T entity);
16.          // trouver une entité T via sa clé primaire
17.          T Find(Object id);
18.          // trouver toutes les entités T
19.          List<T> FindAll();
20.          // supprimer une entité T
21.          void Remove(T entity);
22.          // supprimer toutes les entités
23.          void RemoveAll();
24.      }
25.
26.  }
  
```

- ligne 13 : l'objet T rendu par la méthode *create* est l'image de la ligne insérée dans la table de la base de données. Le paramètre T de la méthode n'a pas de clé primaire ID. Celle-ci est générée par le SGBD. L'objet T rendu a la clé primaire générée. Il est important de comprendre ce point ;

Toutes les autres interfaces **dérivent** de cette interface générique et définissent parfois des méthodes supplémentaires.

L'interface [ICategoryDao] gère les accès à la table [CATEGORY] :

```

1.  using Afb.Ef;
2.  using System.Collections.Generic;
3.
4.  namespace Afb.Dao
5.  {
  
```

```

6.      // interface des catégories
7.      public interface ICategoryDao : IDao<Category>
8.      {
9.          // liste des produits d'une catégorie donnée
10.         List<Product> GetProductsByCategory(int categoryId);
11.     }
12. }
13.
14. }
```

- ligne 8 : l'interface [ICategoryDao] reprend les méthodes de l'interface [IDao<Category>] et en rajoute une ;
- ligne 11 : la méthode qui permet d'avoir tous les produits appartenant à une catégorie identifié par son n° [categoryId] ;

L'interface [ICustomerDao] gère les accès à la table [CUSTOMER] :

```

1.  using Afb.Dao;
2.  using Afb.Ef;
3.
4.  namespace Afb.Dao
5.  {
6.      // interface des clients
7.      public interface ICustomerDao : IDao<Customer>
8.      {
9.      }
10.
11. }
```

- l'interface [ICustomerDao] reprend les méthodes de l'interface [IDao<Customer>] ;

L'interface [ICustomerOrderDao] gère les accès à la table [CUSTOMER\_ORDER] :

```

1.  using Afb.Ef;
2.
3.  namespace Afb.Dao
4.  {
5.      // interface des commandes
6.      public interface ICustomerOrderDao : IDao<CustomerOrder>
7.      {
8.          // enregistrer une commande
9.          CustomerOrder PlaceOrder(Customer customer, ShoppingCart cart);
10.
11. }
12. }
```

- l'interface [ICustomerOrderDao] reprend les méthodes de l'interface [IDao<CustomerOrder>] et lui rajoute une méthode ;
- ligne 9 : la méthode [PlaceOrder] permet d'enregistrer :
  - le client [Customer] qui fait une commande ;
  - l'ordre de commande [CustomerOrder] associé ;
  - les produits [OrderedProduct] achetés par le client,

Le type [ShoppingCart] représente le chariot du client :

```

1.  using Afb.Ef;
2.  using System;
3.  using System.Collections.Generic;
4.  using System.Linq;
5.
6.  namespace Afb.Dao
7.  {
8.      [Serializable]
9.      public class ShoppingCart {
10.
11.         // data
12.         public List<ShoppingCartItem> Items { get; private set; }
13.         private int numberofItems;
14.         public int NumberOfItems {
15.             get {
16.                 return Items.Sum(item => item.Quantity);
17.             }
18.             private set {
19.                 numberofItems = value;
20.             }
21.         }
22.         public double Total { get; set; }
23.
24.         // constructeur
25.         public ShoppingCart() {
26.             Items = new List<ShoppingCartItem>();
```

```

27.     NumberOfItems = 0;
28.     Total = 0;
29. }
30.
31.     // met à jour la qté achetée d'un produit
32.     // si quantity=0, le produit est enlevé du panier
33.     // si le produit n'existe pas, il est ajouté
34.     public void Update(Product product, int quantity)
35.     {
36.         ShoppingCartItem item = null;
37.         foreach (ShoppingCartItem scItem in Items)
38.         {
39.             if (scItem.Product.Id == product.Id)
40.             {
41.                 item = scItem;
42.                 break;
43.             }
44.         }
45.
46.         if (item == null)
47.         {
48.             // add to cart
49.             ShoppingCartItem newItem = new ShoppingCartItem() { Product = product, Quantity = quantity };
50.             Items.Add(newItem);
51.         }
52.         else
53.         {
54.             if (quantity == 0)
55.             {
56.                 Items.Remove(item);
57.             }
58.             else
59.             {
60.                 item.Quantity = quantity;
61.             }
62.         }
63.     }
64.
65.     // coût du panier sans la taxe
66.     public double Subtotal()
67.     {
68.         return Items.Sum(item => item.Product.Price * item.Quantity);
69.     }
70.
71.     // coût du panier avec la taxe
72.     public void CalculateTotal(double surcharge)
73.     {
74.         Total = Subtotal() + surcharge;
75.     }
76.
77.     // raz liste d'articles
78.     public void Clear()
79.     {
80.         Items.Clear();
81.         NumberOfItems = 0;
82.         Total = 0;
83.     }

```

- ligne 12 : la liste des articles achetés par le client ;
- lignes 14-21 : la propriété [NumberOfItems] représente le nombre total d'articles présents dans le chariot ;
- ligne 22 : la propriété [Total] représente le prix à payer pour les articles achetés et la taxe de livraison ;
- lignes 34-63 : la méthode [Update] permet d'ajouter [quantity] articles de type [Product] au chariot ;
- lignes 66-68 : la méthode [SubTotal] calcule le prix à payer pour les articles achetés sans la taxe de livraison ;
- lignes 71-73 : la méthode [CalculateTotal] calcule la propriété [Total] de la ligne 22 ;
- lignes 76-80 : la méthode [Clear] vide le chariot ;

Le type [ShoppingCartItem] des articles du chariot est le suivant :

```

1.  using Afb.Ef;
2.  using System;
3.
4.  namespace Afb.Dao
5.  {
6.      [Serializable]
7.      public class ShoppingCartItem
8.      {
9.          // data
10.         public Product Product { get; set; }
11.         public int Quantity { get; set; }
12.         public double Total
13.         {
14.             get
15.             {
16.                 return (Quantity * Product.Price);

```

```

17.     }
18. }
19.
20. // constructeurs
21. public ShoppingCartItem()
22. {
23. }
24.
25. public ShoppingCartItem(Product product)
26. {
27.     this.Product = product;
28.     Quantity = 1;
29. }
30. }
31. }
```

- ligne 10 : le produit acheté ;
- ligne 11 : la quantité achetée ;
- lignes 12-18 : la propriété [Total] est le coût des [Quantity] articles achetés ;

L'interface [IOrderedProductDao] gère les accès à la table [ORDERED\_PRODUCT] :

```

1. using Afb.Dao;
2. using Afb.Ef;
3. using System.Collections.Generic;
4.
5. namespace Afb.Dao
6. {
7.     // interface des produits commandés
8.     public interface IOrderedProductDao : IDao<OrderedProduct>
9.     {
10.         // liste des articles d'une commande identifiée par sa clé primaire
11.         List<OrderedProduct> FindByOrderId(int orderId);
12.     }
13. }
```

- l'interface [IOrderedProductDao] reprend les méthodes de l'interface [IDao<OrderedProduct>] et en rajoute une ;
- ligne 11 : la méthode permet de retrouver tous les articles achetés dans une commande identifiée par son n° [orderId]

L'interface [IProductDao] gère les accès à la table [PRODUCT] :

```

1. using Afb.Dao;
2. using Afb.Ef;
3.
4. namespace Afb.Dao
5. {
6.     // interface des produits
7.     public interface IPromotionDao : IDao<Product>
8.     {
9.     }
10. }
```

Ces interfaces sont implémentées par les classes suivantes :

Interfaces	Classes
IDao<T>	AbstractDao<T>
ICategoryDao	CategoryDao
ICustomerDao	CustomerDao
ICustomerOrderDao	CustomerOrderDao
IOrderedProductDao	OrderedProductDao
IPromotionDao	ProductDao

La classe [AbstractDao] implémente l'interface [IDao] de la façon suivante :

```

1. using Afb.Ef;
2. using System;
3. using System.Collections.Generic;
4. using System.Data.Entity;
5. using System.Linq;
6.
7. namespace Afb.Dao
8. {
9.     public class AbstractDao<T> : IDao<T> where T : class
10.    {
11.        // nombre d'objets T
```

```

12.     public int Count()
13.     {
14.         using (var context = new AfbContext())
15.         {
16.             return context.Set<T>().Count<T>();
17.         }
18.     }
19.
20.    // persister une entité T
21.    public T Create(T entity)
22.    {
23.        using (var context = new AfbContext())
24.        {
25.            context.Set<T>().Add(entity);
26.            context.SaveChanges();
27.            return entity;
28.        }
29.    }
30.
31.    // modifier une entité T
32.    public T Edit(T entity)
33.    {
34.        using (var context = new AfbContext())
35.        {
36.            context.Entry(entity).State = EntityState.Modified;
37.            context.SaveChanges();
38.            return entity;
39.        }
40.    }
41.
42.    // trouver une entité T via sa clé primaire
43.    public T Find(Object id)
44.    {
45.        using (var context = new AfbContext())
46.        {
47.            return context.Set<T>().Find(id);
48.        }
49.    }
50.
51.    // trouver toutes les entités T
52.    public List<T> FindAll()
53.    {
54.        using (var context = new AfbContext())
55.        {
56.            return context.Set<T>().ToList<T>();
57.        }
58.    }
59.
60.    // supprimer une entité T
61.    public void Remove(T entity)
62.    {
63.        using (var context = new AfbContext())
64.        {
65.            context.Entry(entity).State = EntityState.Deleted;
66.            context.SaveChanges();
67.        }
68.    }
69.
70.    // supprimer toutes les entités
71.    public void RemoveAll()
72.    {
73.        using (var context = new AfbContext())
74.        {
75.            context.Set<T>().RemoveRange(context.Set<T>());
76.            context.SaveChanges();
77.        }
78.    }
79.
80.    }
81. }

```

Toutes les implémentations des différentes interfaces vont étendre la classe ci-dessus :

- lignes 12-18 : la méthode [Count] permet de compter le nombre de lignes d'une table d'entités de type T ;
- lignes 21-29 : la méthode [Create] permet d'insérer une entité de type T dans une table d'entités de type T ;
- lignes 32-40 : la méthode [Edit] permet de modifier une entité de type T dans une table d'entités de type T ;
- lignes 43-49 : la méthode [Find] permet de chercher une entité de type T dans une table d'entités de type T via sa clé primaire ;
- lignes 52-58 : la méthode [FindAll] permet de récupérer toutes les lignes d'une table d'entités de type T ;
- lignes 61-68 : la méthode [Remove] permet de supprimer une entité de type T d'une table d'entités de type T ;
- lignes 71-78 : la méthode [RemoveAll] permet de vider une table d'entités de type T ;

---

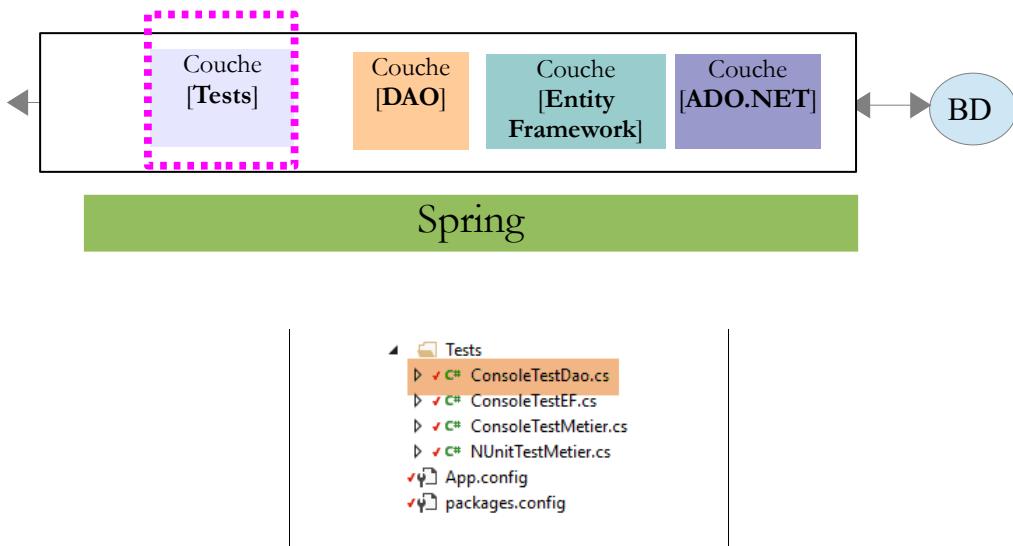
**Question 2 :** écrivez le code des autres classes.

---

#### Conseil :

- la méthode [CustomerDao.PlaceOrder] rend un type [PlaceOrder] dont le champ [Customer] n'est pas forcément initialisé (lazy loading) comme le montrent les résultats du test de la couche [DAO] au paragraphe 7.5.3.2.3, page 486 ;
  - la méthode [OrderedProductDao.FindById] rend un type [List<OrderedProduct>] où :
    - le champ [OrderedProduct.CustomerOrder] est initialisé ;
    - le champ [OrderedProduct.CustomerOrder.Customer] est initialisé ;
    - le champ [OrderedProduct.Product] est initialisé ;
    - le champ [OrderedProduct.Product.category] n'est pas initialisé ;
- On découvrira ce qui est attendu dans les résultats du test de la couche [DAO] au paragraphe 7.5.3.2.3, page 486 ;

#### 7.5.3.2 Le test



#### 7.5.3.2.1 Le code

La classe de test est la suivante :

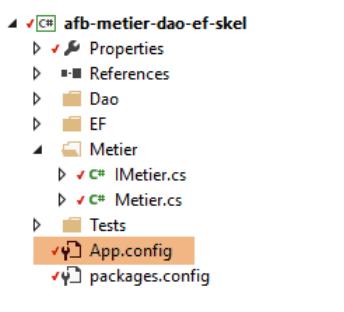
```
1.  using Afb.Dao;
2.  using Afb.Ef;
3.  using Newtonsoft.Json;
4.  using Spring.Context;
5.  using Spring.Context.Support;
6.  using System;
7.  using System.Collections.Generic;
8.
9.  namespace Afb.Tests
10. {
11.     class ConsoleTestDao
12.     {
13.         static void Main(string[] args)
14.         {
15.             // couches [DAO] injectées par Spring
16.             IApplicationContext context = ContextRegistry.GetContext();
17.             IOrderedProductDao orderedProductDao = context.GetObject("orderedProductDao") as IOrderedProductDao;
18.             ICategoryDao categoryDao = context.GetObject("categoryDao") as ICategoryDao;
19.             ICustomerDao customerDao = context.GetObject("customerDao") as ICustomerDao;
20.             ICustomerOrderDao customerOrderDao = context.GetObject("customerOrderDao") as ICustomerOrderDao;
21.
22.             try
23.             {
24.                 // catégories
25.                 Console.WriteLine("Catégories -----");
26.                 List<Category> categories = categoryDao.FindAll();
27.                 Console.WriteLine(JsonConvert.SerializeObject(categories));
28.                 // produits
29.                 Console.WriteLine("Produits de la catégorie n° 1 -----");
30.                 List<Product> products = categoryDao.GetProductsByCategory((int)categories[0].Id);
31.                 Console.WriteLine(JsonConvert.SerializeObject(products));
32.                 // reset base
33.                 Console.WriteLine("ClearCustomers -----");
34.                 customerDao.RemoveAll();
35.                 Console.WriteLine("PlaceOrder -----");
```

```

36.     // remplissage chariot
37.     ShoppingCart chariot = new ShoppingCart();
38.     // 2 produits 0
39.     chariot.Update(products[0], 2);
40.     // 3 produits 1
41.     chariot.Update(products[1], 3);
42.     // on force le calcul du prix à payer
43.     chariot.CalculateTotal(0);
44.     // persistance commande
45.     Customer customer = new Customer() { Name = "1", Email = "2", Phone = "3", Address = "4", CityRegion = "55",
46.         CcNumber = "6" };
47.     CustomerOrder order = customerOrderDao.PlaceOrder(customer, chariot);
48.     Console.WriteLine(JsonConvert.SerializeObject(order));
49.     // produits commandés
50.     Console.WriteLine("OrderDetails -----");
51.     Console.WriteLine(JsonConvert.SerializeObject(orderedProductDao.FindByOrderId((int)order.Id)));
52. }
53. catch (Exception e)
54. {
55.     // on signale l'erreur
56.     Console.WriteLine(string.Format("L'erreur suivante s'est produite : {0}", e.Message));
57. }
58. }
59. }
```

Ce code est laissé à votre compréhension. Il vous sera utile ultérieurement.

### 7.5.3.2.2 La configuration



Le projet [afb-metier-dao-ef-skel] est configuré par le fichier [App.config] suivant :

```

1.  <?xml version="1.0" encoding="utf-8"?>
2.  <configuration>
3.      <configSections>
4.          ...
5.          <!-- spring -->
6.          <sectionGroup name="spring">
7.              <section name="context" type="Spring.Context.Support.ContextHandler, Spring.Core" />
8.              <section name="objects" type="Spring.Context.Support.DefaultSectionHandler, Spring.Core" />
9.          </sectionGroup>
10.     </configSections>
11.     <!-- configuration Spring -->
12.     <spring>
13.         <context>
14.             <resource uri="config://spring/objects" />
15.         </context>
16.         <objects xmlns="http://www.springframework.net">
17.             <object id="categoryDao" type="Afb.Dao.CategoryDao,afb-metier-dao-ef" />
18.             <object id="customerDao" type="Afb.Dao.CustomerDao,afb-metier-dao-ef" />
19.             <object id="customerOrderDao" type="Afb.Dao.CustomerOrderDao,afb-metier-dao-ef" />
20.             <object id="orderedProductDao" type="Afb.Dao.OrderedProductDao,afb-metier-dao-ef" />
21. ...
22.         </objects>
23.     </spring>
24. ...
25. </configuration>
```

Ce code est également laissé à votre compréhension.

### 7.5.3.2.3 Les résultats

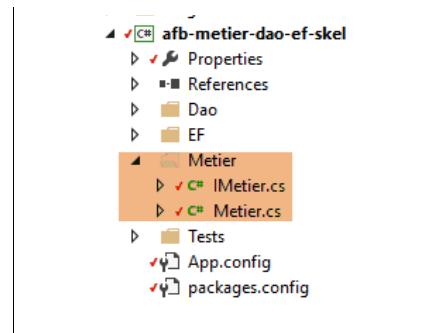
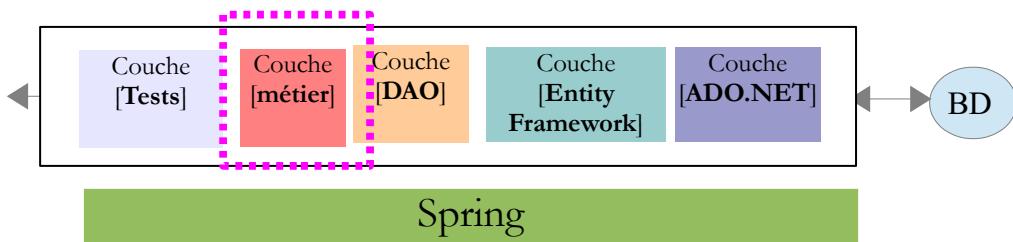
Configurez votre projet pour exécuter le test console [ConsoleTestDao] puis exécutez-le. Vous devez obtenir quelque chose qui ressemble à ceci :

```

1. Catégories -----
2. [{"Name":"dairy","Id":1,"Version":1},{"Name":"meats","Id":2,"Version":1},{"Name":"bakery","Id":3,"Version":1},
 {"Name":"fruit & veg","Id":4,"Version":1}]
3. Produits de la catégorie n° 1 -----
4. [{"Name":"milk","Description":"semi skimmed (1L)","Price":1.7,"LastUpdate":"2012-10-
30T16:36:41+01:00","CategoryId":1,"Category":null,"Id":1,"Version":1},{"Name":"cheese","Description":"mild cheddar
(330g)","Price":2.39,"LastUpdate":"2012-10-30T16:36:41+01:00","CategoryId":1,"Category":null,"Id":2,"Version":1},
 {"Name":"butter","Description":"unsalted (250g)","Price":1.09,"LastUpdate":"2012-10-
30T16:36:41+01:00","CategoryId":1,"Category":null,"Id":3,"Version":1},{"Name":"freerange eggs","Description":"medium-sized
(6 eggs)","Price":1.76,"LastUpdate":"2012-10-30T16:36:41+01:00","CategoryId":1,"Category":null,"Id":4,"Version":1}]
5. ClearCustomers -----
6. PlaceOrder -----
7. {"Amount":10.57,"DateCreated":"2015-10-
22T11:24:06+02:00","ConfirmationNumber":28532367,"CustomerId":65,"Customer":null,"Id":22,"Version":1}
8. OrderDetails -----
9. [{"Quantity":2,"ProductId":1,"Product":{"Name":"milk","Description":"semi skimmed (1L)","Price":1.7,"LastUpdate":"2012-10-
30T16:36:41+01:00","CategoryId":1,"Category":null,"Id":1,"Version":1}, "CustomerOrderId":22,"CustomerOrder": {
 {"Amount":10.57,"DateCreated":"2015-10-22T11:24:06+02:00","ConfirmationNumber":28532367,"CustomerId":65,"Customer": {
 {"Name":"1","Email":"2","Phone":"3","Address":"4","CityRegion":"55","CcNumber":6,"Id":65,"Version":1}, "Id":22,"Version":1
 }, "Id":41,"Version":1}, {"Quantity":3,"ProductId":2,"Product":{"Name":"cheese","Description":"mild cheddar
(330g)","Price":2.39,"LastUpdate":"2012-10-
30T16:36:41+01:00","CategoryId":1,"Category":null,"Id":2,"Version":1}, "CustomerOrderId":22,"CustomerOrder": {
 {"Amount":10.57,"DateCreated":"2015-10-22T11:24:06+02:00","ConfirmationNumber":28532367,"CustomerId":65,"Customer": {
 {"Name":"1","Email":"2","Phone":"3","Address":"4","CityRegion":"55","CcNumber":6,"Id":65,"Version":1}, "Id":22,"Version":1
 }, "Id":42,"Version":1}]}
10. Appuyez sur une touche pour continuer...

```

## 7.5.4 La couche [métier]



### 7.5.4.1 Le code

La couche [métier] présente l'interface [IMetier] suivante :

```

1. using Afb.Dao;
2. using Afb.Ef;
3. using System.Collections.Generic;
4.
5. namespace Afb.Metier
6. {
7.
8.     public interface IMetier
9.     {
10.
11.         // détails d'une commande identifiée par sa clé primaire
12.         List<OrderedProduct> OrderDetails(int orderId);
13.
14.         // persister une commande
15.         // customer : l'acheteur

```

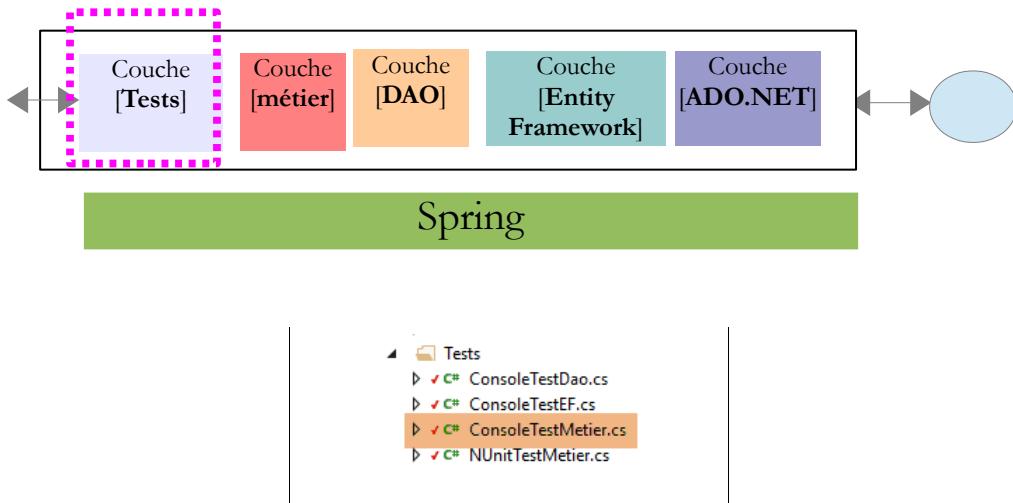
```

16.    // cart : son panier d'achats
17.    CustomerOrder PlaceOrder(Customer customer, ShoppingCart cart);
18.
19.    // liste des catégories de produits
20.    List<Category> Categories();
21.
22.    // liste des produits d'une catégorie identifiée par sa clé primaire
23.    List<Product> ProductsByCategory(int categoryId);
24.
25.    // Reset des tables [Customers, CustomerOrders, OrderedProducts]
26.    void ClearCustomers();
27. }
28. }
```

Cette interface restreint à quelques méthodes, l'ensemble des méthodes exposées par les diverses interfaces de la couche [DAO].

**Question 3 :** écrire la classe d'implémentation [Metier] de l'interface [IMetier].

#### 7.5.4.2 Le test console



##### 7.5.4.2.1 Le code

La classe [ConsoleTestMetier] est la suivante :

```

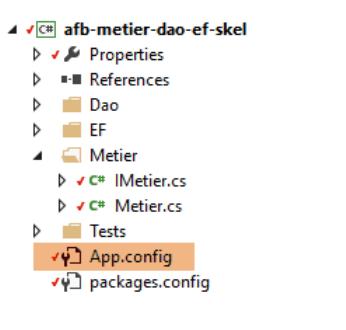
1.  using Afb.Dao;
2.  using Afb.Ef;
3.  using Afb.Metier;
4.  using Newtonsoft.Json;
5.  using Spring.Context.Support;
6.  using System;
7.  using System.Collections.Generic;
8.
9. namespace Afb.Tests
10. {
11.     class ConsoleTestMetier
12.     {
13.         static void Main(string[] args)
14.         {
15.             // instantiation couche [metier] via Spring
16.             IMetier metier = ContextRegistry.GetContext().GetObject("metier") as IMetier;
17.
18.             try
19.             {
20.                 // catégories
21.                 Console.WriteLine("Catégories -----");
22.                 List<Category> categories = metier.Categories();
23.                 Console.WriteLine(JsonConvert.SerializeObject(categories));
24.                 // produits
25.                 Console.WriteLine("Produits de la catégorie n° 1 -----");
26.                 List<Product> products = metier.ProductsByCategory((int)categories[0].Id);
27.                 Console.WriteLine(JsonConvert.SerializeObject(products));
28.                 // reset base
29.                 Console.WriteLine("ClearCustomers -----");
30.                 metier.ClearCustomers();
31.                 Console.WriteLine("PlaceOrder -----");
32.             }
33.         }
34.     }
35. }
```

```

32.     // remplissage chariot
33.     ShoppingCart chariot = new ShoppingCart();
34.     // 2 produits 0
35.     chariot.Update(products[0], 2);
36.     // 3 produits 1
37.     chariot.Update(products[1], 3);
38.     // on force le calcul du prix à payer
39.     chariot.CalculateTotal(0);
40.     // persistance commande
41.     Customer customer = new Customer() { Name = "1", Email = "2", Phone = "3", Address = "4", CityRegion = "55",
42.         CcNumber = "6" };
43.     CustomerOrder order = metier.PlaceOrder(customer, chariot);
44.     Console.WriteLine(JsonConvert.SerializeObject(order));
45.     // produits commandés
46.     Console.WriteLine("OrderDetails -----");
47.     Console.WriteLine(JsonConvert.SerializeObject(metier.OrderDetails((int)order.Id)));
48. }
49. catch (Exception e)
50. {
51.     // on signale l'erreur
52.     Console.WriteLine(string.Format("L'erreur suivante s'est produite : {0}", e.Message));
53. }
54. }
55. }
```

Ce code est laissé à votre compréhension.

#### 7.5.4.2.2 La configuration



Le projet [afb-metier-dao-ef-skel] est configuré par le fichier [App.config] suivant :

```

1. <?xml version="1.0" encoding="utf-8"?>
2. <configuration>
3.   <configSections>
4.     ...
5.     <!-- spring -->
6.     <sectionGroup name="spring">
7.       <section name="context" type="Spring.Context.Support.ContextHandler, Spring.Core" />
8.       <section name="objects" type="Spring.Context.Support.DefaultSectionHandler, Spring.Core" />
9.     </sectionGroup>
10.   </configSections>
11.   <!-- configuration Spring -->
12.   <spring>
13.     <context>
14.       <resource uri="config://spring/objects" />
15.     </context>
16.     <objects xmlns="http://www.springframework.net">
17.       <object id="categoryDao" type="Afb.Dao.CategoryDao,afb-metier-dao-ef" />
18.       <object id="customerDao" type="Afb.Dao.CustomerDao,afb-metier-dao-ef" />
19.       <object id="customerOrderDao" type="Afb.Dao.CustomerOrderDao,afb-metier-dao-ef" />
20.       <object id="orderedProductDao" type="Afb.Dao.OrderedProductDao,afb-metier-dao-ef" />
21.       <object id="metier" type="Afb.Metier.Metier,afb-metier-dao-ef">
22.         <property name="OrderedProductDao" ref="orderedProductDao" />
23.         <property name="CategoryDao" ref="categoryDao" />
24.         <property name="CustomerOrderDao" ref="customerOrderDao" />
25.         <property name="CustomerDao" ref="customerDao" />
26.       </object>
27.     </objects>
28.   </spring>
29. ...
30. </configuration>
```

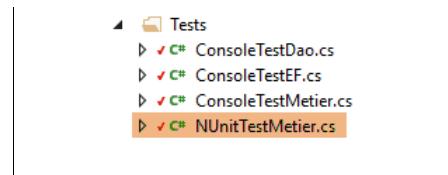
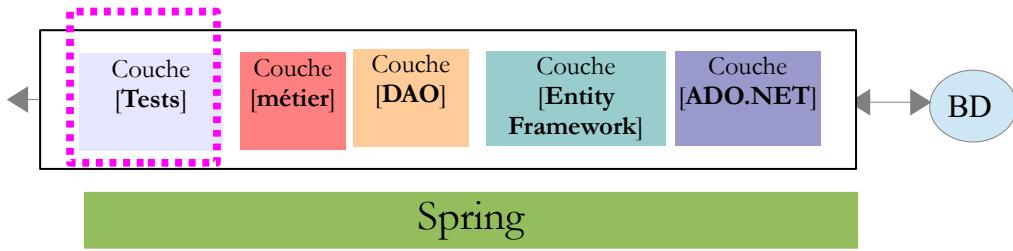
Cette configuration est laissée à votre compréhension.

### 7.5.4.2.3 Les résultats

Configurez votre projet pour exécuter le test console [ConsoleTestMetier] puis exécutez-le. Vous devez obtenir quelque chose qui ressemble à ceci :

```
1. Catégories -----
2. [{"Name":"dairy","Id":1,"Version":1}, {"Name":"meats","Id":2,"Version":1}, {"Name":"bakery","Id":3,"Version":1},
   {"Name":"fruit & veg","Id":4,"Version":1}]
3. Produits de la catégorie n° 1 -----
4. [{"Name":"milk","Description":"semi skimmed (1L)","Price":1.7,"LastUpdate":"2012-10-
   30T16:36:41+01:00","CategoryId":1,"Category":null,"Id":1,"Version":1}, {"Name":"cheese","Description":"mild cheddar
   (330g)","Price":2.39,"LastUpdate":"2012-10-30T16:36:41+01:00","CategoryId":1,"Category":null,"Id":2,"Version":1},
   {"Name":"butter","Description":"unsalted (250g)","Price":1.09,"LastUpdate":"2012-10-
   30T16:36:41+01:00","CategoryId":1,"Category":null,"Id":3,"Version":1}, {"Name":"freerange eggs","Description":"medium-sized
   (6 eggs)","Price":1.76,"LastUpdate":"2012-10-30T16:36:41+01:00","CategoryId":1,"Category":null,"Id":4,"Version":1}]
5. ClearCustomers -----
6. PlaceOrder -----
7. {"Amount":10.57,"DateCreated":"2015-10-
   22T11:43:00+02:00","ConfirmationNumber":14560392,"CustomerId":66,"Customer":null,"Id":23,"Version":1}
8. OrderDetails -----
9. [{"Quantity":2,"ProductId":1,"Product":{"Name":"milk","Description":"semi skimmed (1L)","Price":1.7,"LastUpdate":"2012-10-
   30T16:36:41+01:00","CategoryId":1,"Category":null,"Id":1,"Version":1}, "CustomerOrderId":23,"CustomerOrder":
   {"Amount":10.57,"DateCreated":"2015-10-22T11:43:00+02:00","ConfirmationNumber":14560392,"CustomerId":66,"Customer":
   {"Name":"1","Email":"2","Phone":"3","Address":"4","CityRegion":"55","CcNumber":"6","Id":66,"Version":1}, "Id":23,"Version":1
   }, "Id":43,"Version":1}, {"Quantity":3,"ProductId":2,"Product":{"Name":"cheese","Description":"mild cheddar
   (330g)","Price":2.39,"LastUpdate":"2012-10-
   30T16:36:41+01:00","CategoryId":1,"Category":null,"Id":2,"Version":1}, "CustomerOrderId":23,"CustomerOrder":
   {"Amount":10.57,"DateCreated":"2015-10-
   22T11:43:00+02:00","ConfirmationNumber":14560392,"CustomerId":66,"Customer":{"Name":"1","Email":"2","Phone":"3","Address":"4"
   , "CityRegion":"55","CcNumber":"6","Id":66,"Version":1}, "Id":23,"Version":1}, "Id":44,"Version":1}]
10. Appuyez sur une touche pour continuer...
```

### 7.5.4.3 Le test NUnit



### 7.5.4.3.1 Le code

La classe [NUnitTestMetier] est la suivante :

```
1. using Afb.Dao;
2. using Afb.Ef;
3. using Afb.Metier;
4. using NUnit.Framework;
5. using Spring.Context.Support;
6. using System.Collections.Generic;
7.
8. namespace Afb.Tests
9. {
10.
11.     [TestFixture]
12.     class NUnitTestMetier
13.     {
14.         // la couche [métier]
15.         private IMetier metier;
16.
17.         [TestFixtureSetUp]
18.         public void Init()
```

```

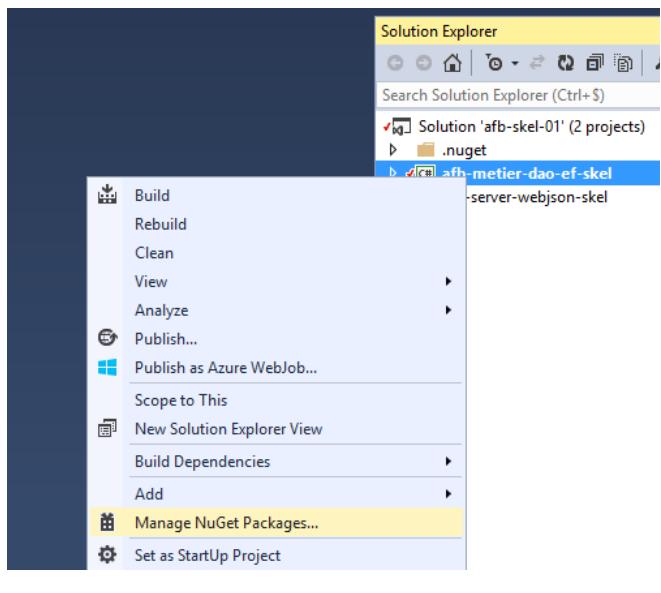
19.      {
20.          // instantiation couche [metier] via Spring
21.          metier = ContextRegistry.GetContext().GetObject("metier") as IMetier;
22.      }
23.
24.      [TestFixtureTearDown]
25.      public void Dispose()
26.      { /* ... */ }
27.
28.      [SetUp]
29.      public void BeforeTest()
30.      {
31.          metier.ClearCustomers();
32.      }
33.
34.      [TearDown]
35.      public void endTest()
36.      {
37.      }
38.
39.      [Test]
40.      public void doNothing()
41.      {
42.      }
43.
44.      [Test]
45.      public void placeOrder()
46.      {
47.          // liste des catégories
48.          List<Category> categories = metier.Categories();
49.          Assert.AreEqual(4, categories.Count);
50.          // liste des produits catégorie 0
51.          long? categoryId = categories[0].Id;
52.          List<Product> products = metier.ProductsByCategory((int)categoryId);
53.          // remplissage chariot
54.          ShoppingCart chariot = new ShoppingCart();
55.          // 2 produits 0
56.          chariot.Update(products[0], 2);
57.          // 3 produits 1
58.          chariot.Update(products[1], 3);
59.          // on force le calcul du prix à payer
60.          chariot.CalculateTotal();
61.          // persistance client
62.          Customer customer = new Customer() { Name = "1", Email = "2", Phone = "3", Address = "4", CityRegion = "55",
63.          CcNumber = "6" };
64.          CustomerOrder customerOrder = metier.PlaceOrder(customer, chariot);
65.
66.          // vérifications
67.          List<OrderedProduct> orderedProducts = metier.OrderDetails((int)customerOrder.Id);
68.          Assert.AreEqual(products[0].Name, orderedProducts[0].Product.Name);
69.          Assert.AreEqual(products[1].Name, orderedProducts[1].Product.Name);
70.          Assert.AreEqual(2, orderedProducts[0].Quantity);
71.          Assert.AreEqual(3, orderedProducts[1].Quantity);
72.          CustomerOrder customerOrder2 = orderedProducts[0].CustomerOrder;
73.          Assert.AreEqual(10.57, (double)customerOrder2.Amount, 1e-6);
74.          Customer customer2 = customerOrder2.Customer;
75.          Assert.AreEqual("1", customer2.Name);
76.          Assert.AreEqual("2", customer2.Email);
77.          Assert.AreEqual("3", customer2.Phone);
78.          Assert.AreEqual("4", customer2.Address);
79.          Assert.AreEqual("55", customer2.CityRegion);
80.          Assert.AreEqual("6", customer2.CcNumber);
81.      }
82.  }

```

Ce code est laissé à votre compréhension.

#### 7.5.4.3.2 La configuration

Il est possible d'exécuter un test NUnit à l'intérieur de Visual Studio. Pour cela, procédez comme suit :



afb-metier-dao-ef-skel - Manage NuGet Packages

Installed packages

Online

All  
nuget.org  
Microsoft and .NET  
Search Results

Updates

Sort by: Relevance

**NUnit** 1  
NUnit is a unit-testing framework for all .Net languages with a strong TDD focus.

**NUnit.Runners**  
NUnit is a unit-testing framework for all .Net languages with a strong TDD focus.

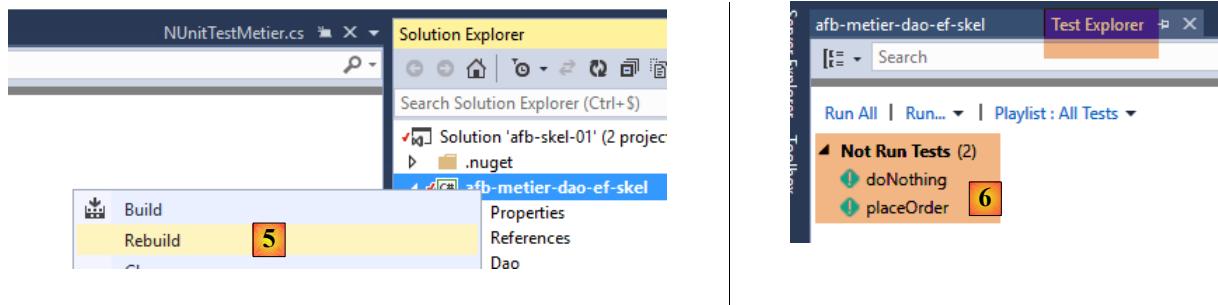
**NUnit Test Adapter for VS2012, VS2013 and VS2015**  
A package including the NUnit TestAdapter for Visual Studio 2012/13/15. With this package you don't need to install the V...

**SpecFlow.NUnit** 2

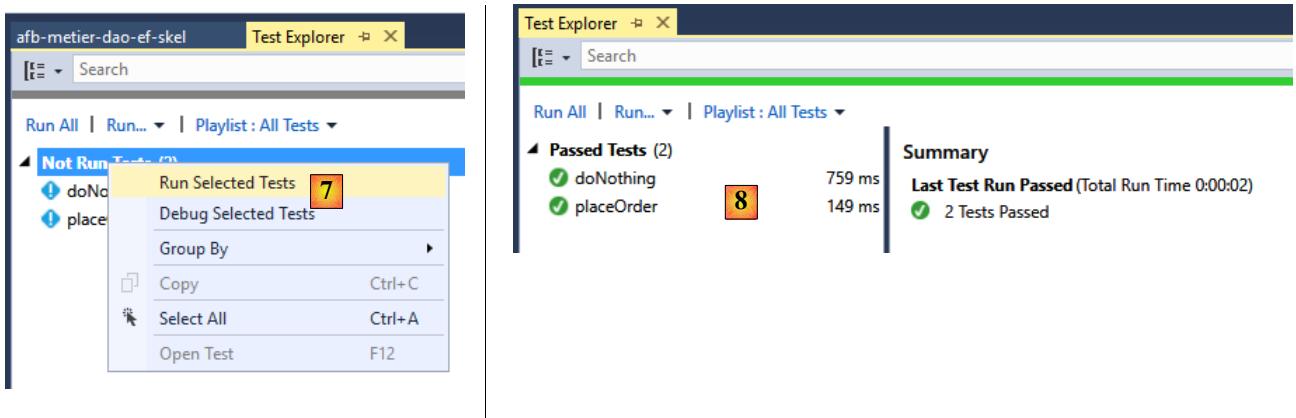
nunit  
Created by: Charlie Poole  
Id: NUnit  
Version: 2.6.4  
Last Published: 17/12/2014  
Downloads: 3072328  
License  
View License  
Zlib  
Project Information  
Report Abuse  
Description:

- installez les packages [1] et [2] ;
- fermez puis réouvez Visual Studio ;

- l'option [Test] [3] doit être présente dans le menu ;
- ouvrez la fenêtre de tests [4] ;



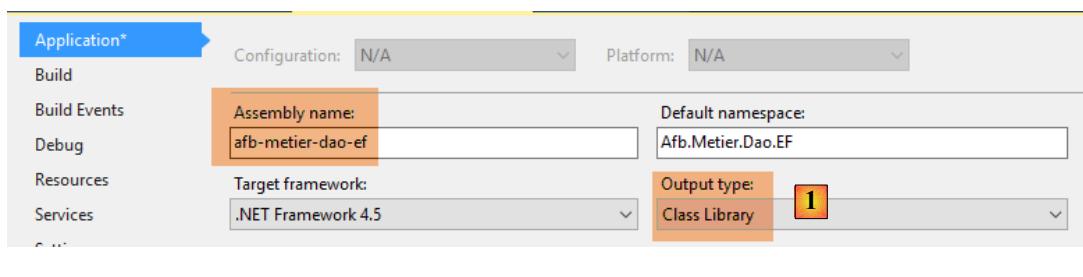
- en [5], reconstruisez votre projet ;
- en [6], les méthodes de test NUnit doivent apparaître dans la fenêtre de tests ;



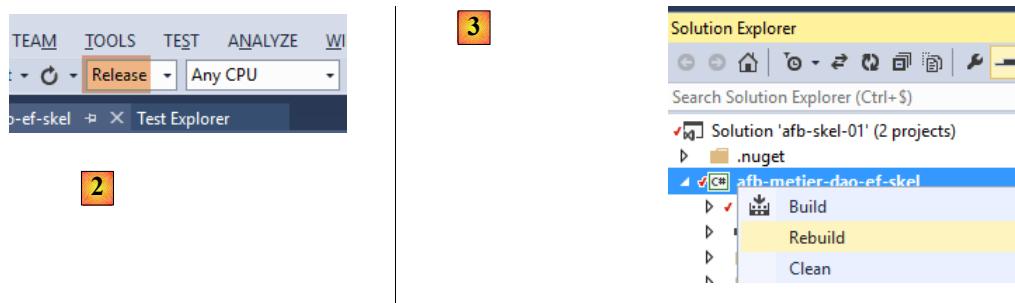
- en [7], exécutez les tests ;
- en [8], ils doivent réussir ;

### 7.5.5 Crédation des DLL des couches basses

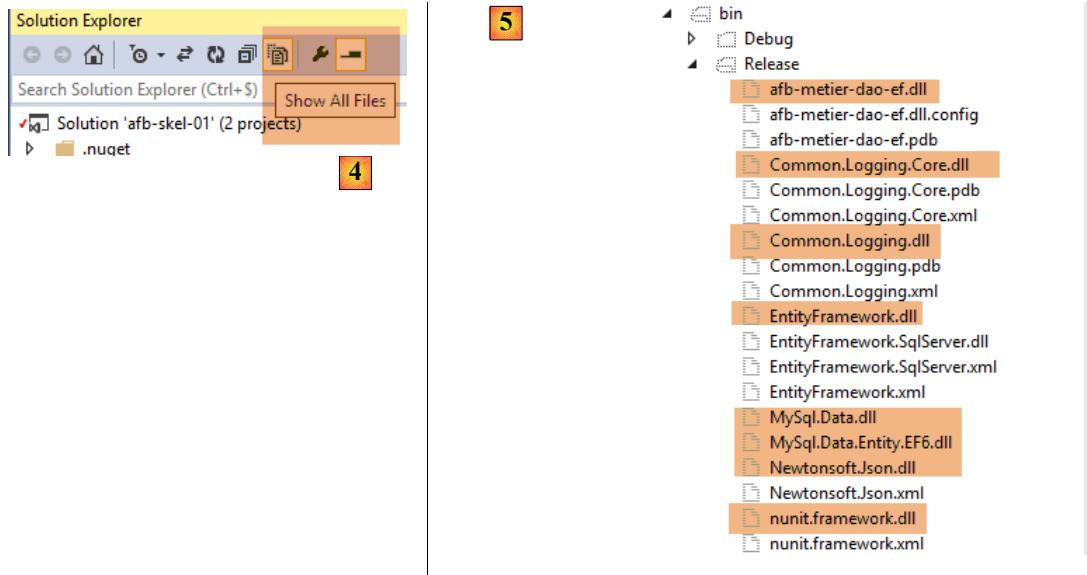
Nous allons maintenant créer les DLL du projet des couches basses. Procédez comme suit :



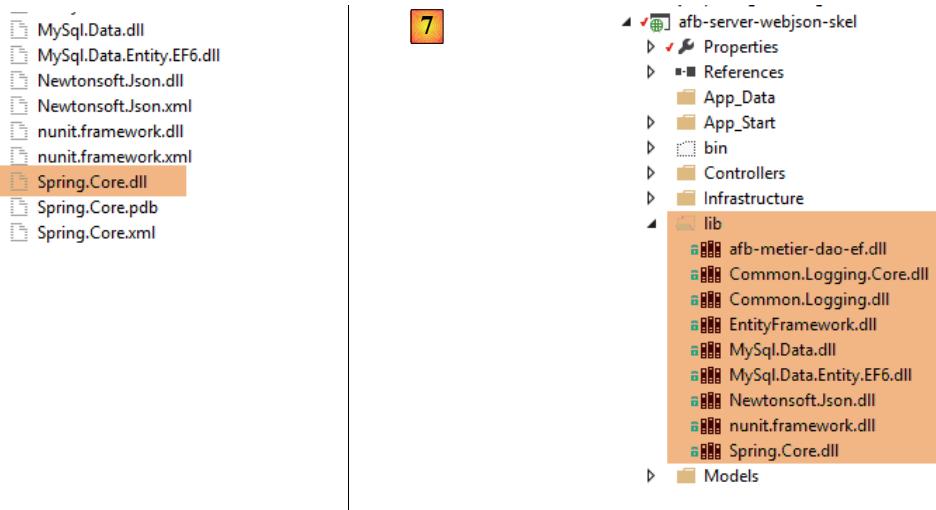
- en [1], configurez le projet pour qu'il génère une DLL ;



- en [2], configurez le projet en mode [Release] ;
- en [3], construisez le projet ;



- en [4], faites apparaître tous les fichiers du projet ;

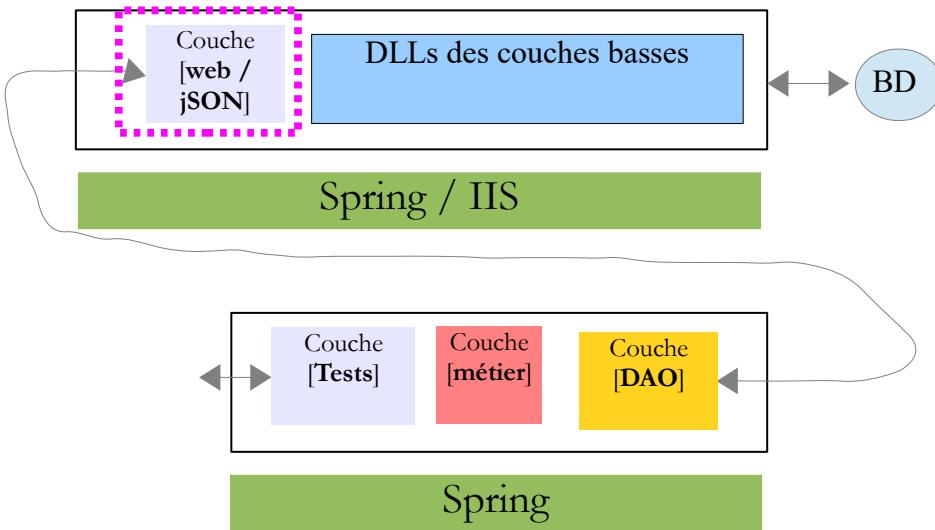


- en [5-7], prenez toutes les DLL trouvées dans le dossier [bin / release] et copiez-les dans le dossier [lib] du projet Visual Studio [afb-server-webjson-skel]

## 7.6 Ecriture de la couche web / JSON

### 7.6.1 Architecture

Nous allons maintenant construire une application client / serveur implémentant l'architecture suivante :

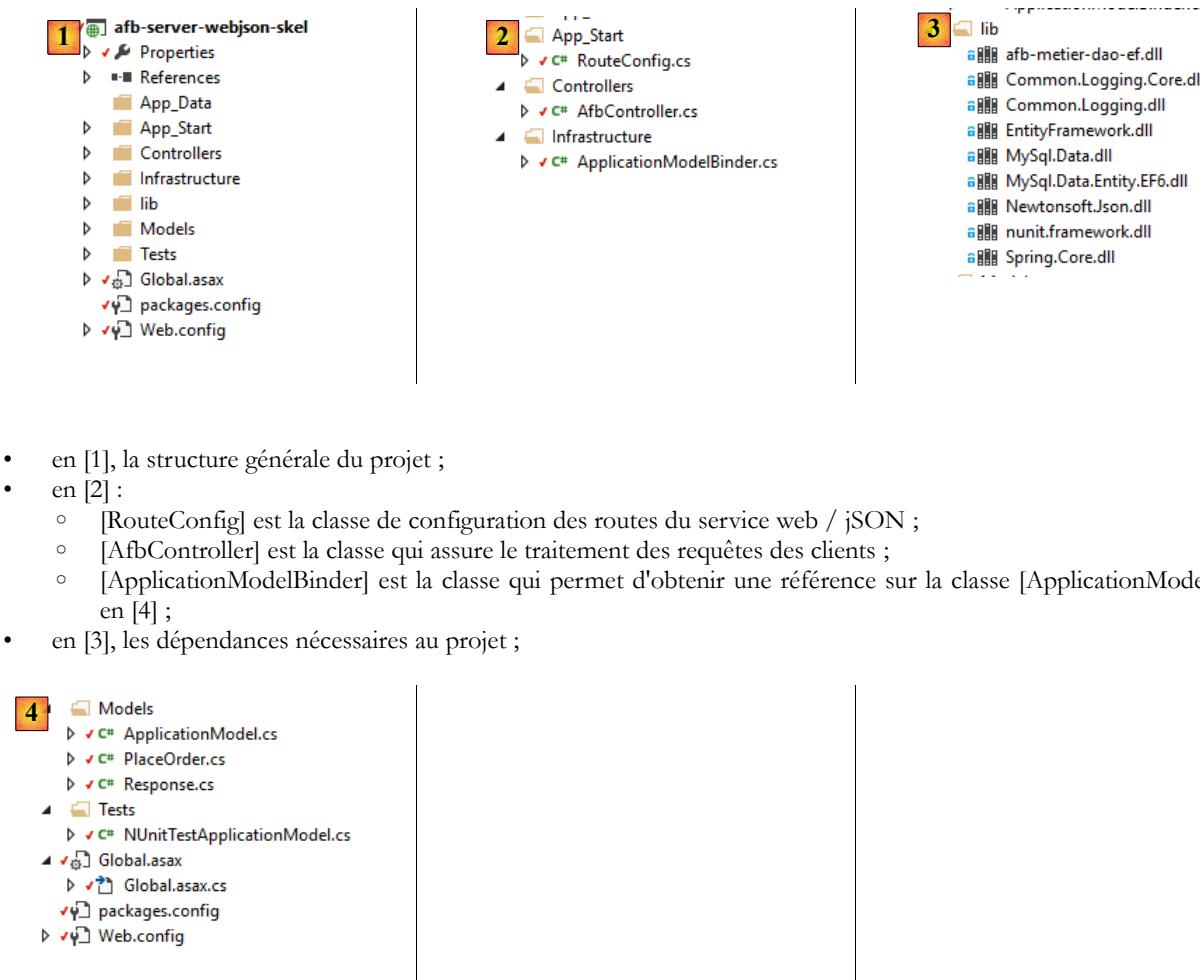


Nous allons construire l'application en deux temps :

- construction de la couche web / JSON du serveur ;
- test de cette couche avec un client web / JSON qui vous sera donné et expliqué ;

### 7.6.2 Le projet Visual Studio

Le projet Visual Studio du serveur web / JSON est le suivant :



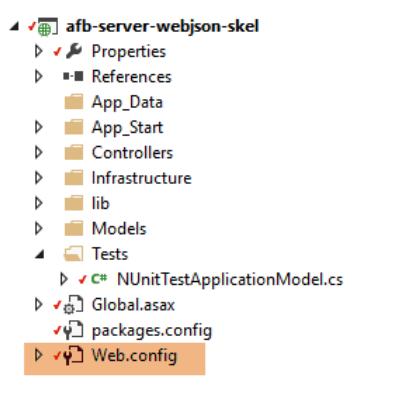
- en [1], la structure générale du projet ;
- en [2] :
  - [RouteConfig] est la classe de configuration des routes du service web / JSON ;
  - [AfbController] est la classe qui assure le traitement des requêtes des clients ;
  - [ApplicationModelBinder] est la classe qui permet d'obtenir une référence sur la classe [ApplicationModel] présente en [4] ;
- en [3], les dépendances nécessaires au projet ;

- en [4] :
  - [ApplicationModel] est la classe qui encapsule les données de portée [Application] et qui implémente l'interface [IMetier] des couches basses ;
  - [Response] est la classe générique qui encapsule la réponse faite aux clients ;
  - [PlaceOrder] est le modèle de la valeur postée à l'une des URL exposées par le service web. On a en général, un modèle par valeur postée ;
  - [Global] est la classe qui initialise l'application web / JSON à son démarrage ;
  - [Web.config] est le fichier de configuration du projet ;
  - [NUnitTestApplicationModel] est une classe de test NUnit qui teste la classe [ApplicationModel] ;

### Mise en place des dépendances

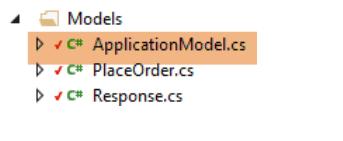
Nous avons mis précédemment dans le dossier [lib] les dépendances nécessaires au projet. Ajoutez-les toutes comme références du projet.

#### 7.6.3 Configuration

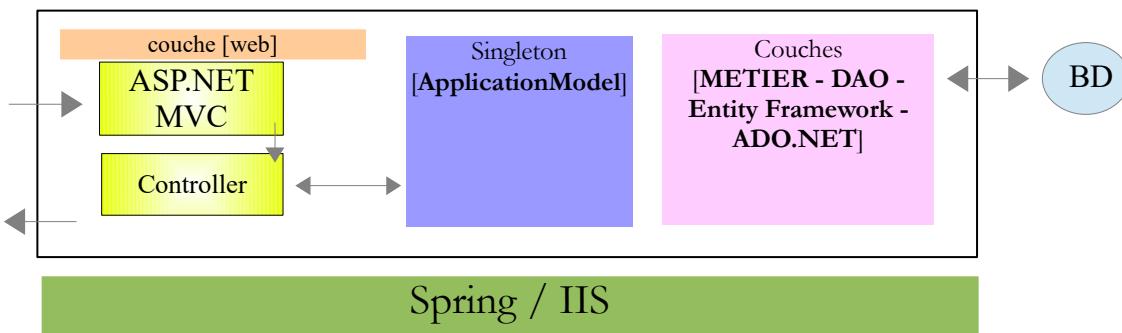


Le fichier [Web.config] qui configure l'application est identique au fichier [App.config] qui configurait le projet des couches basses. Par défaut, lorsqu'on construit un projet web, Visual Studio met beaucoup de configurations dans [Web.config]. Ici, elles s'avèrent inutiles.

#### 7.6.4 La classe [ApplicationModel]



La classe [ApplicationModel] encapsule les données de portée [Application], celles disponibles en lecture seule à toutes les requêtes de tous les utilisateurs. On peut également lui faire implémenter l'interface [IMetier] qui gère l'accès aux couches basses. Ce sera le cas ici. On a alors l'architecture MVC suivante :

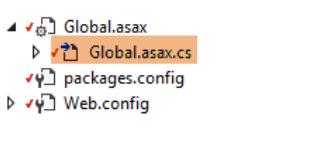


Les méthodes du contrôleur utilisent les services offerts par le singleton [ApplicationModel]. Le code de celui-ci est le suivant :

```
1.  using Afb.Dao;
2.  using Afb.Ef;
3.  using Afb.Metier;
4.  using System;
5.  using System.Collections.Generic;
6.
7.  namespace Afb.Web.Models
8.  {
9.      public class ApplicationModel : IMetier
10.     {
11.         // la couche [métier]
12.         public IMetier Metier { get; set; }
13.         // l'exception initiale
14.         public Exception InitException { get; set; }
15.
16.         // ----- interface IMetier
17.         // détails d'une commande
18.         public List<OrderedProduct> OrderDetails(int orderId)
19.         {
20.             return Metier.OrderDetails(orderId);
21.         }
22.
23.         // faire une commande
24.         public CustomerOrder PlaceOrder(Customer customer, ShoppingCart cart)
25.         {
26.             return Metier.PlaceOrder(customer, cart);
27.         }
28.
29.         // liste des catégories de produits
30.         public List<Category> Categories()
31.         {
32.             return Metier.Categories();
33.         }
34.
35.         // liste des produits d'une catégorie
36.         public List<Product> ProductsByCategory(int categoryId)
37.         {
38.             return Metier.ProductsByCategory(categoryId);
39.         }
40.
41.         // reset de la base
42.         public void ClearCustomers()
43.         {
44.             Metier.ClearCustomers();
45.         }
46.     }
47. }
```

- ligne 12 : une référence sur la couche [métier] qui permet d'implémenter l'interface [IMetier] (lignes 16-45) ;
- ligne 14 : s'il se produit une exception au démarrage du serveur web, elle est enregistrée là ;

## 7.6.5 Initialisation de l'application



Le serveur web / JSON est initialisé par la classe suivante du fichier [Global.asax.cs] :

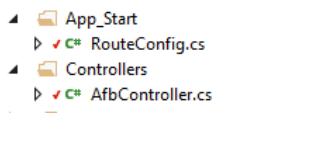
```
1.  using Afb.Metier;
2.  using Afb.Web.Infrastructure;
3.  using Afb.Web.Models;
4.  using Afb.Web.Routes;
5.  using Spring.Context.Support;
6.  using System;
7.  using System.Web.Mvc;
8.  using System.Web.Routing;
9.
10. namespace Istia.St.Afb.Web.Main
11. {
12.     public class MvcApplication : System.Web.HttpApplication
13.     {
14.         protected void Application_Start()
15.         {
16.             // auto-généré
17.             AreaRegistration.RegisterAllAreas();
```

```

18.     RouteConfig.RegisterRoutes(RouteTable.Routes);
19.     // -----
20.     // ----- configuration spécifique
21.     // -----
22.     // données de portée application
23.     ApplicationModel application = new ApplicationModel();
24.     Application["data"] = application;
25.     application.InitException = null;
26.     try
27.     {
28.         // instantiation couche [métier]
29.         application.Metier = ContextRegistry.GetContext().GetObject("metier") as IMetier;
30.     }
31.     catch (Exception ex)
32.     {
33.         application.InitException = ex;
34.     }
35.     // model binders
36.     ModelBinders.Binders.Add(typeof(ApplicationModel), new ApplicationModelBinder());
37. }
38. }
39. }
```

Ce code est laissé à votre compréhension. Du code analogue a été vu dans l'étude de cas n° 1.

## 7.6.6 Les URL exposées par le service web / JSON



Le contrôleur [AfbController] expose les URL suivantes :

```

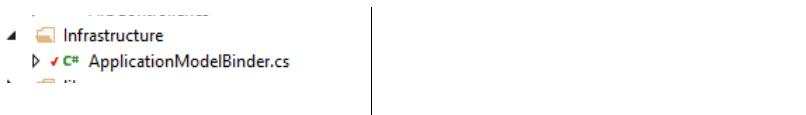
1.  using Afb.Ef;
2.  using Afb.Web.Models;
3.  using System;
4.  using System.Collections.Generic;
5.  using System.Web.Mvc;
6.
7.  namespace Afb.Web.Controllers
8.  {
9.      public class AfbController : Controller
10.     {
11.         [HttpGet]
12.         public JsonResult GetCategories(ApplicationModel application)
13.         {
14.             // obtenir les catégories
15.             ...
16.         }
17.
18.         [HttpGet]
19.         public JsonResult GetProductsByCategory(ApplicationModel application, int categoryId)
20.         {
21.             // obtenir les produits d'une catégorie
22.             ...
23.         }
24.
25.         [HttpGet]
26.         public JsonResult GetOrderDetails(ApplicationModel application, int orderId)
27.         {
28.             // obtenir les produits d'une commande
29.             ...
30.         }
31.
32.         [HttpGet]
33.         public JsonResult ClearCustomers(ApplicationModel application)
34.         {
35.             // vider les tables de clients, ordres de commande, produits achetés
36.             ...
37.         }
38.
39.         [HttpPost]
40.         public JsonResult PlaceOrder(ApplicationModel application, PlaceOrder order)
41.         {
42.             // enregistrer une commande
43.             ...
44.         }
45.     }
46. }
```

Ces URL sont reprises dans la classe de routage [RouteConfig] :

```
1.  using System.Web.Mvc;
2.  using System.Web.Routing;
3.
4.  namespace Afb.Web.Routes
5.  {
6.      public class RouteConfig
7.      {
8.          public static void RegisterRoutes(RouteCollection routes)
9.          {
10.              routes.IgnoreRoute("{resource}.axd/{*pathInfo}");
11.
12.              // les routes
13.              // vider les tables de clients
14.              routes.MapRoute(
15.                  name: "ClearCustomers",
16.                  url: "Afb/ClearCustomers",
17.                  defaults: new { controller = "Afb", action = "ClearCustomers" }
18. );
19.              // obtenir les catégories
20.              routes.MapRoute(
21.                  name: "GetCategories",
22.                  url: "Afb/GetCategories",
23.                  defaults: new { controller = "Afb", action = "GetCategories" }
24. );
25.              // obtenir les produits d'une commande
26.              routes.MapRoute(
27.                  name: "GetOrderDetails",
28.                  url: "Afb/GetOrderDetails/{orderId}",
29.                  defaults: new { controller = "Afb", action = "GetOrderDetails" }
30. );
31.              // obtenir les produits d'une catégorie
32.              routes.MapRoute(
33.                  name: "GetProductsByCategory",
34.                  url: "Afb/GetProductsByCategory/{categoryId}",
35.                  defaults: new { controller = "Afb", action = "GetProductsByCategory" }
36. );
37.              // persister une commande
38.              routes.MapRoute(
39.                  name: "PlaceOrder",
40.                  url: "Afb/PlaceOrder",
41.                  defaults: new { controller = "Afb", action = "PlaceOrder" }
42. );
43.      }
44.  }
```

Au final, les seules URL acceptées par l'application sont les URL des lignes 16, 22, 28, 34 et 39. Le code du contrôleur montre que ces URL sont toutes obtenues avec un ordre HTTP GET, sauf l'URL [Afb/PlaceOrder] obtenue avec un ordre HTTP POST.

### 7.6.7 La classe [ApplicationModelBinder]



Toutes les méthodes du contrôleur ont parmi leurs paramètres le paramètre [[ApplicationModel](#) application]. Ce paramètre est automatiquement initialisé par ASP.NET MVC grâce à la classe [ApplicationModelBinder] suivante :

```
1.  using System.Web.Mvc;
2.
3.  namespace Afb.Web.Infrastructure
4.  {
5.      public class ApplicationModelBinder : IModelBinder
6.      {
7.          public object BindModel(ControllerContext controllerContext, ModelBindingContext bindingContext)
8.          {
9.              // on rend les données de portée [Application]
10.             return controllerContext.HttpContext.Application["data"];
11.         }
12.     }
13. }
```

- ligne 10 : on rend l'objet de portée [Application] associé à la clé [data]. Cet objet est créé dans [Global.asax] lors de l'initialisation de l'application :

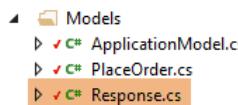
```

1. ...
2. namespace Istia.St.Afb.Web.Main
3. {
4.     public class MvcApplication : System.Web.HttpApplication
5.     {
6.         protected void Application_Start()
7.         {
8.             ...
9.             // données de portée application
10.            ApplicationModel application = new ApplicationModel();
11.            Application["data"] = application;
12.            application.InitException = null;
13.            try
14.            {
15.                // instantiation couche [métier]
16.                application.Metier = ContextRegistry.GetContext().GetObject("metier") as IMetier;
17.            }
18.            catch (Exception ex)
19.            {
20.                application.InitException = ex;
21.            }
22.            // model binders
23.            ModelBinders.Binders.Add(typeof(ApplicationModel), new ApplicationModelBinder());
24.        }
25.    }
26. }

```

- lignes 10-11 : l'objet de portée [Application] associé à la clé [data] est une instance de la classe [ApplicationModel] ;
- ligne 23 : c'est cette ligne qui associe le [IModelBinder] à la classe [ApplicationModel] et qui fait que ASP.NET MVC va automatiquement initialiser le paramètre [ApplicationModel applicationModel] des méthodes du contrôleur. Cette association ne doit être faite qu'une fois. C'est pourquoi elle se trouve dans la méthode [Application\_Start] qui n'est exécutée qu'une fois au démarrage de l'application ;

## 7.6.8 La classe [Response]



Toutes les méthodes du contrôleur rendent la valeur JSON d'une instance [Response] définie comme suit :

```

1. using Newtonsoft.Json;
2.
3. namespace Afb.Web.Models
4. {
5.     public class Response<T>
6.     {
7.         // ----- propriétés
8.         // statut de l'opération
9.         public int Status { get; set; }
10.        // l'éventuelle exception
11.        public string Exception { get; set; }
12.        // le corps de la réponse
13.        public T Body { get; set; }
14.        // la signature JSON
15.        public override string ToString()
16.        {
17.            return JsonConvert.SerializeObject(this);
18.        }
19.    }
20. }

```

- ligne 5 : la classe est paramétrée par le type T qui est le type du corps de la réponse, ligne 13. Les méthodes du contrôleur doivent normalement rendre le champ de la ligne 13. Mais elles peuvent rencontrer une exception. Pour prendre en compte celle-ci, on ajoute deux propriétés :
  - ligne 9 : un code d'erreur : 0 pour pas d'erreur, autre chose sinon ;
  - ligne 11 : le message pour préciser l'éventuelle exception, null sinon ;
- ligne 15 : le corps de la réponse qui est ce qu'attend vraiment le client qui a fait la requête ;

## 7.6.9 Les méthodes du contrôleur

### 7.6.9.1 L'URL [Afb/GetCategories]

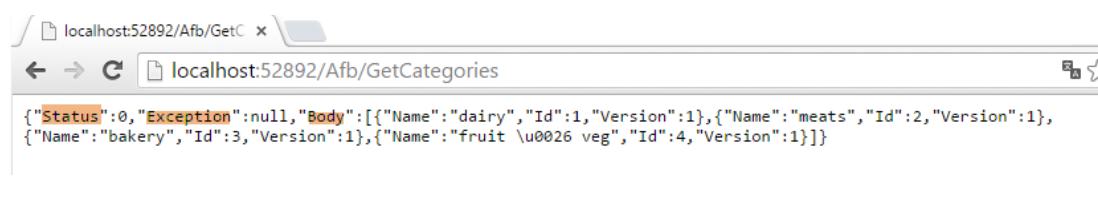
La méthode [GetCategories] du contrôleur pourrait être la suivante :

```
1.      [HttpGet]
2.      public JsonResult GetCategories(ApplicationModel application)
3.      {
4.          // obtenir les catégories
5.          Response<List<Category>> response;
6.          try
7.          {
8.              response = new Response<List<Category>>() { Status = 0, Exception = null, Body = application.Categories() };
9.          }
10.         catch (Exception e)
11.         {
12.             response = new Response<List<Category>>() { Status = 1, Exception = getMessagesFromException(e), Body = null };
13.         }
14.         return Json(response, JsonRequestBehavior.AllowGet);
15.     }
```

Ce code est laissé à votre compréhension. Pour le tester, lancez l'application web / JSON. Vous allez obtenir quelque chose comme suit :



Demandez alors l'URL [Afb/GetCategories] :



### 7.6.9.2 Les autres URL

**Question 4 :** complétez le code du contrôleur [AfbController].

Il y a une difficulté pour la méthode [PlaceOrder] qui reçoit une valeur JSON postée. On récupérera cette valeur de la façon suivante :

```
1.      [HttpPost]
2.      public JsonResult PlaceOrder(ApplicationModel application, PlaceOrder order)
3.      {
4.          // valeur JSON postée : {"customer":
5.          {"name":"1","email":"2","phone":"3","address":"4","cityRegion":"55","ccNumber":"6"}, "cart": {"Items": [{"Product": {"Id":1}, "quantity":2}, {"Product":{"Id":2}, "quantity":3}], "total":10.57}}
6.          // la valeur JSON postée est automatiquement déserialisée dans [PlaceOrder order]
7.      }
```

- lignes 4-5 : la valeur JSON doit être celle d'un type [PlaceOrder] ;

L'URL peut être testée avec l'extension Chrome [Advanced Rest Client] :

http://localhost:52892/Afb/PlaceOrder

POST

Raw Form Headers

Encode payload Decode payload

```
{"customer": {"name": "1", "email": "2", "phone": "3", "address": "4", "cityRegion": "55", "ccNumber": "6"}, "cart": {"Items": [{"Product": {"Id": 1}, "quantity": 2}, {"Product": {"Id": 2}, "quantity": 3}], "total": 10.57}}
```

application/json Set "Content-Type" header to overwrite this value.

Raw JSON Response

Copy to clipboard Save as file

```
{
  Status: 0
  Exception: null
  Body: {
    Amount: 10.57
    DateCreated: "/Date(1445588772000)"
    ConfirmationNumber: 66616792
    CustomerId: 74
    Customer: null
    Id: 30
    Version: 1
  }
}
```

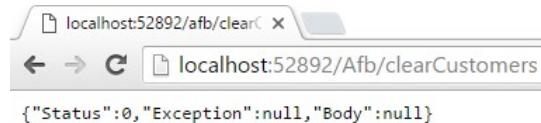
Clear Send

On obtient alors le résultat suivant :

Raw JSON Response

```
{
  Status: 0
  Exception: null
  Body: {
    Amount: 10.57
    DateCreated: "/Date(1445588772000)"
    ConfirmationNumber: 66616792
    CustomerId: 74
    Customer: null
    Id: 30
    Version: 1
  }
}
```

L'URL [/ClearCustomers] rend un type Response<Object> où l'objet contenu dans la réponse est toujours le pointeur *null*.

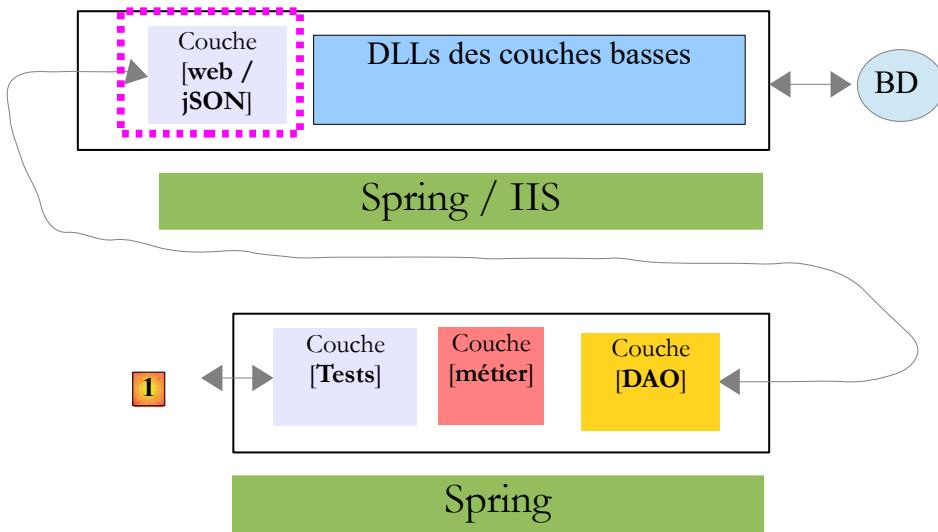


#### Note :

Pour des raisons incomprises, parfois le serveur IIS Express déclare une erreur et ne se lance pas. Il ne donne alors aucune indication sur la source de l'erreur. Dans ce cas, utilisez le serveur ASP [Ultidev Server] présenté au paragraphe 2.26, page 327 du cours. Faites pointer le serveur sur le dossier du serveur web / jSON.

## 7.7 Étude du client web / JSON

### 7.7.1 Architecture



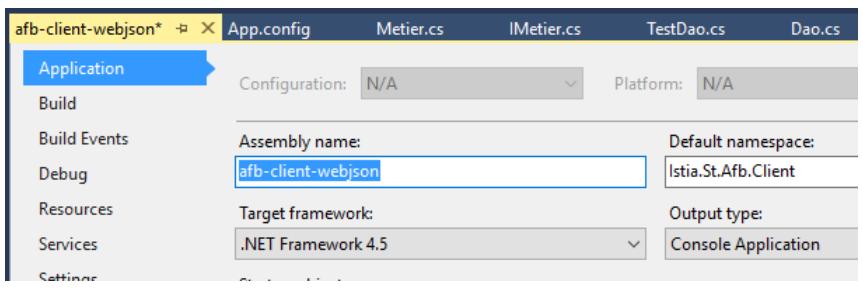
Nous allons maintenant présenter le client [1] du serveur web / JSON.

### 7.7.2 Le projet Visual Studio



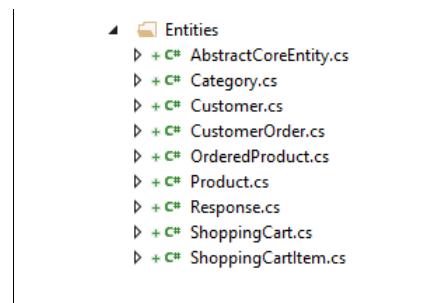
- en [1], le projet dans son ensemble ;
- en [2], les entités échangées entre le client et le serveur ;
- en [3] :
  - le dossier [Dao] rassemble les classes d'implémentation de la couche [DAO] du client ;
  - le dossier [Metier] rassemble les classes d'implémentation de la couche [métier] du client ;
  - le dossier [Tests] rassemble les classes de test du client ;

Vérifiez le nom de l'assembly de ce projet :



Il doit être celui de la copie d'écran ci-dessus. Le fichier de configuration [App.config] le référence sous ce nom.

### 7.7.3 Les entités



Les entités échangées entre le client et le serveur sont celles déjà rencontrées côté serveur avec leurs attributs [Entity Framework] en moins. Ainsi la classe [AbstractCoreEntity] est-elle devenue la suivante :

```

1.  using Newtonsoft.Json;
2.  using System;
3.
4.  namespace Afb.Client.Entities
5.  {
6.      [Serializable]
7.      public class AbstractCoreEntity
8.      {
9.
10.         public int? Id { get; set; }
11.
12.         public int? Version { get; set; }
13.
14.         public override string ToString()
15.         {
16.             return JsonConvert.SerializeObject(this);
17.         }
18.     }
19. }
```

et la classe [Product] :

```

1.  using System;
2.
3.  namespace Afb.Client.Entities
4.  {
5.      public class Product : AbstractCoreEntity
6.      {
7.          public string Name { get; set; }
8.
9.          public string Description { get; set; }
10.
11.         public double Price { get; set; }
12.
13.         public DateTime LastUpdate { get; set; }
14.
15.         public int? CategoryId { get; set; }
16.
17.         public Category Category { get; set; }
18.     }
19. }
```

## 7.7.4 Implémentation de la couche [DAO]

```
◀ Dao
  ▷ + C# Dao.cs
  ▷ + C# IDao.cs
```

La couche [DAO] du client offre l'interface [IDao] suivante :

```
1.  using System;
2.
3.  namespace Afb.Client.Dao
4.  {
5.      public interface IDao
6.      {
7.          T1 getResponse<T1,T2>(String url, T2 body);
8.      }
9.  }
```

L'interface n'a qu'une méthode, celle de la ligne 7 paramétrée par deux types T1 et T2 :

- T1 est le type de la réponse attendue ;
- T2 est le type de la valeur postée (sera *null* pour une opération GET) ;

On sait que le serveur renvoie la chaîne JSON du type [Response] suivant :

```
1.  public class Response<T>
2.  {
3.      // ----- propriétés
4.      // statut de l'opération
5.      public int Status { get; set; }
6.      // l'éventuelle exception
7.      public string Exception { get; set; }
8.      // le corps de la réponse
9.      public T Body { get; set; }
10.     // signature JSON
11.     public override string ToString()
12.     {
13.         return JsonConvert.SerializeObject(this);
14.     }
15. }
```

Le type T1 de la ligne 7 de l'interface correspond au type T du champ [body] de la réponse (ligne 9). Revenons à la signature de la méthode :

```
T1 getResponse<T1,T2>(String url, T2 body);
```

- [url] est l'URL du service web interrogée ;
- [body] est la valeur postée ou *null* s'il s'agit d'une opération GET ;

L'interface [IDao] est implémentée par la classe [Dao] suivante :

```
1.  using Afb.Client.Entities;
2.  using Newtonsoft.Json;
3.  using System;
4.  using System.Net;
5.
6.  namespace Afb.Client.Dao
7.  {
8.      public class Dao : IDao
9.      {
10.          public T1 getResponse<T1, T2>(String url, T2 body)
11.          {
12.              // client web
13.              WebClient webClient = new WebClient();
14.              // on précise qu'on va envoyer du Json
15.              webClient.Headers.Add("Content-Type", "application/json");
16.              // requête
17.              string result;
18.              if (body == null)
19.              {
20.                  // GET
21.                  result = webClient.DownloadString(url);
22.              }
23.              else
24.              {
```

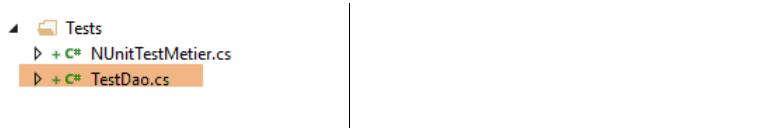
```

25.         // POST
26.         result = webClient.UploadString(url, JsonConvert.SerializeObject(body));
27.     }
28.     Response<T1> response = JsonConvert.DeserializeObject<Response<T1>>(result);
29.     // erreur ?
30.     if (response.Status != 0)
31.     {
32.         throw new Exception(response.Exception);
33.     }
34.     else
35.     {
36.         return response.Body;
37.     }
38. }
39. }
40. }

```

- ligne 13 : le type [WebClient] est la classe qui nous permet de dialoguer avec le serveur JSON. Il est fourni par l'espace de noms [System.Net] (ligne 4) ;
- ligne 15 : on fixe l'en-tête HTTP [Content-Type] pour indiquer au serveur qu'on va lui envoyer du JSON. C'est parce qu'il reçoit cet entête que le serveur va automatiquement déserialiser la valeur postée, s'il y en a une ;
- ligne 18 : le cas [body==null] correspond à une absence de valeur postée. On fera alors une requête HTTP GET. Elle se fait avec la méthode [webClient.DownloadString(url)] de la ligne 21. Comme son nom l'indique, elle rend une chaîne de caractères, la chaîne JSON envoyée par le serveur ;
- ligne 26 : le cas [body !=null] indique qu'il y a une valeur postée. On fera alors une requête HTTP POST. Elle se fait avec la méthode [webClient.UploadString] de la ligne 21. Son second paramètre est la chaîne JSON de la valeur à poster. Elle rend une chaîne de caractères, la chaîne JSON envoyée par le serveur ;
- ligne 28 : on désérialise la réponse JSON qu'on a reçue ;
- ligne 30 : on regarde son champ [Status]. On rappelle que si celui-ci est différent de 0, alors il y a eu une exception côté serveur et le message d'erreur associé est dans le champ [Response.Exception] ;
- ligne 32 : en cas d'erreur côté serveur, on lance une exception avec le message transmis par le serveur ;
- ligne 36 : sinon on rend le corps de la réponse qui est la partie qui nous intéresse ;

## 7.7.5 Tests de la couche [DAO]



La classe de test est la suivante :

```

1.  using Afb.Client.Dao;
2.  using Afb.Client.Entities;
3.  using Afb.Client.Metier;
4.  using Newtonsoft.Json;
5.  using System;
6.  using System.Collections.Generic;
7.
8. namespace Afb.Client.Tests
9. {
10.     class TestDao
11.     {
12.         static void Main(string[] args)
13.         {
14.             // initialisations
15.             string urlServer = "http://localhost:52892/Afb";
16.             IDao dao = new Afb.Client.Dao.Dao();
17.             // catégories
18.             Console.WriteLine("Catégories -----");
19.             Console.WriteLine(JsonConvert.SerializeObject(dao.getResponse<List<Category>,
20. Object>(String.Format("{0}/GetCategories", urlServer), null)));
21.             // produits
22.             Console.WriteLine("Produits de la catégorie n° 1 -----");
23.             List<Product> products = dao.getResponse<List<Product>, Object>(String.Format("{0}/GetProductsByCategory/1",
urlServer), null);
24.             Console.WriteLine(JsonConvert.SerializeObject(products));
25.             // reset base
26.             Console.WriteLine("ResetDatabase -----");
27.             dao.getResponse<Object, Object>(String.Format("{0}/ClearCustomers", urlServer), null);
28.             Console.WriteLine("PlaceOrder -----");
29.             // remplissage chariot
30.             ShoppingCart chariot = new ShoppingCart();
31.             // 2 produits 0
32.             chariot.Update(products[0], 2);
33.             // 3 produits 1

```

```

33.         chariot.Update(products[1], 3);
34.         // on force le calcul du prix à payer
35.         chariot.CalculateTotal(0);
36.         // persistance commande
37.         Customer customer = new Customer() { Name = "1", Email = "2", Phone = "3", Address = "4", CityRegion = "55",
38.             CcNumber = "6" };
39.         CustomerOrder order = dao.getResponse<CustomerOrder, PlaceOrder>(String.Format("{0}/PlaceOrder", urlServer), new
40.             PlaceOrder() { Customer = customer, Cart = chariot });
41.         Console.WriteLine(JsonConvert.SerializeObject(order));
42.         // produits commandés
43.         Console.WriteLine("GetOrderDetails -----");
44.         Console.WriteLine(JsonConvert.SerializeObject(dao.getResponse<List<OrderedProduct>,
45.             Object>(String.Format("{0}/GetOrderDetails/{1}", urlServer, order.Id), null)));

```

Ce code est laissé à votre compréhension. Ligne 15, vous adapterez l'URL à celle de votre serveur.

On obtient des résultats analogues aux suivants :

```

1. Catégories -----
2. [{"Name":"dairy","Id":1,"Version":1},{"Name":"meats","Id":2,"Version":1},{"Name":"bakery","Id":3,"Version":1},
   {"Name":"fruit & veg","Id":4,"Version":1}]
3. Produits de la catégorie n° 1 -----
4. [{"Name":"milk","Description":"semi skimmed (1L)","Price":1.7,"LastUpdate":"2012-10-
   30T15:36:41Z","CategoryId":1,"Category":null,"Id":1,"Version":1},{"Name":"cheese","Description":"mild cheddar
   (330g)","Price":2.39,"LastUpdate":"2012-10-30T15:36:41Z","CategoryId":1,"Category":null,"Id":2,"Version":1},
   {"Name":"butter","Description":"unsalted (250g)","Price":1.09,"LastUpdate":"2012-10-
   30T15:36:41Z","CategoryId":1,"Category":null,"Id":3,"Version":1},{"Name":"free range eggs","Description":"medium-sized (6
   eggs)","Price":1.76,"LastUpdate":"2012-10-30T15:36:41Z","CategoryId":1,"Category":null,"Id":4,"Version":1}]
5. ResetDatabase -----
6. PlaceOrder -----
7. {"Amount":10.57,"DateCreated":"2015-10-
   23T10:07:54Z","ConfirmationNumber":32049580,"CustomerId":79,"Customer":null,"Id":35,"Version":1}
8. GetOrderDetails -----
9. [{"Quantity":2,"ProductId":1,"Product":{"Name":"milk","Description":"semi skimmed (1L)","Price":1.7,"LastUpdate":"2012-10-
   30T15:36:41Z","CategoryId":1,"Category":null,"Id":1,"Version":1}, "CustomerOrderId":35,"CustomerOrder":
   {"Amount":10.57,"DateCreated":"2015-10-23T10:07:54Z","ConfirmationNumber":32049580,"CustomerId":79,"Customer":
   {"Name":"1","Email":"2","Phone":"3","Address":"4","CityRegion":"55","CcNumber":"6","Id":79,"Version":1}, "Id":35,"Version":1
   }, "Id":65,"Version":1}, {"Quantity":3,"ProductId":2,"Product":{"Name":"cheese","Description":"mild cheddar
   (330g)","Price":2.39,"LastUpdate":"2012-10-
   30T15:36:41Z","CategoryId":1,"Category":null,"Id":2,"Version":1}, "CustomerOrderId":35,"CustomerOrder":
   {"Amount":10.57,"DateCreated":"2015-10-23T10:07:54Z","ConfirmationNumber":32049580,"CustomerId":79,"Customer":
   {"Name":"1","Email":"2","Phone":"3","Address":"4","CityRegion":"55","CcNumber":"6","Id":79,"Version":1}, "Id":35,"Version":1
   }, "Id":66,"Version":1}]
10. Appuyez sur une touche pour continuer...

```

## 7.7.6 Implémentation de la couche [métier]



L'interface [IMetier] de la couche [métier] du client est la suivante :

```

1. using Afb.Client.Entities;
2. using System.Collections.Generic;
3.
4. namespace Afb.Client.Metier
5. {
6.
7.     public interface IMetier
8.     {
9.
10.        // détails d'une commande identifiée par sa clé primaire
11.        List<OrderedProduct> OrderDetails(int orderId);
12.
13.        // persister une commande
14.        // customer : l'acheteur
15.        // cart : son panier d'achats
16.        CustomerOrder PlaceOrder(Customer customer, ShoppingCart cart);
17.
18.        // liste des catégories de produits
19.        List<Category> Categories();
20.
21.        // liste des produits d'une catégorie identifiée par sa clé primaire
22.        List<Product> ProductsByCategory(int categoryId);
23.

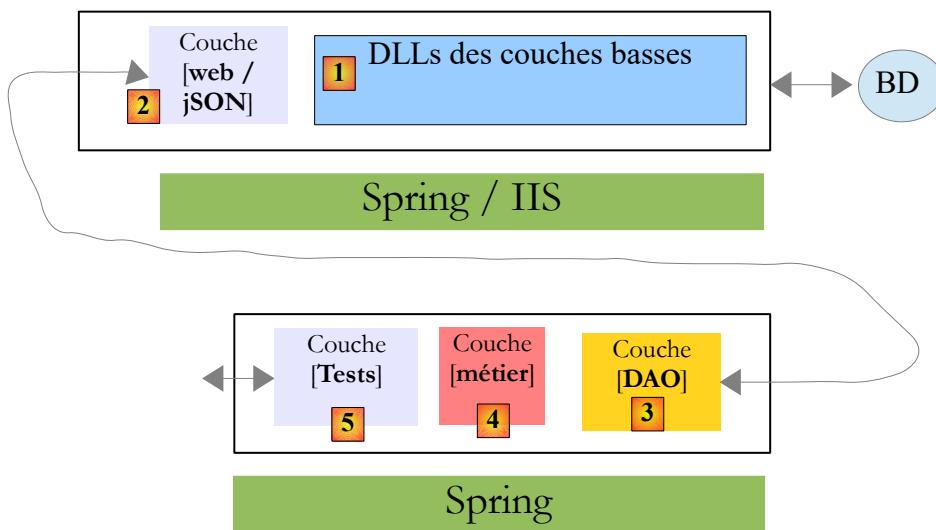
```

```

24.     // Reset des tables [Customers, CustomerOrders, OrderedProducts]
25.     void ClearCustomers();
26. }
27.

```

C'est donc la même que l'interface [IMetier] côté serveur. On peut le comprendre en revenant à l'architecture client / serveur de l'application :



- en [1], les couches basses présentent l'interface [IMetier] à la couche [web / JSON] [2] du serveur ;
- les couches [métier] [4], [DAO] [3] du client et la couche [web / JSON] [2] ont pour effet que la couche [Tests] [5] du client a l'impression de communiquer directement avec l'interface [IMetier] des couches basses du serveur. Elle ne voit pas le réseau qui sépare le client du serveur ;

La classe [Metier] qui implémente l'interface [IMetier] du client est la suivante :

```

1.  using Afb.Client.Dao;
2.  using Afb.Client.Entities;
3.  using System;
4.  using System.Collections.Generic;
5.
6. namespace Afb.Client.Metier
7. {
8.     public class Metier : IMetier
9.     {
10.         // injections Spring -----
11.         // URL service web : JSON
12.         private string UrlServer { get; set; }
13.         // couche [DAO]
14.         private IDao Dao { get; set; }
15.
16.         // détails d'une commande
17.         public List<OrderedProduct> OrderDetails(int orderId)
18.         {
19.             throw new NotImplementedException("méthode non encore implémentée");
20.         }
21.
22.         // faire une commande
23.         public CustomerOrder PlaceOrder(Customer customer, ShoppingCart cart)
24.         {
25.             throw new NotImplementedException("méthode non encore implémentée");
26.         }
27.
28.         // liste des catégories de produits
29.         public List<Category> Categories()
30.         {
31.             throw new NotImplementedException("méthode non encore implémentée");
32.         }
33.
34.         // liste des produits d'une catégorie
35.         public List<Product> ProductsByCategory(int categoryId)
36.         {
37.             throw new NotImplementedException("méthode non encore implémentée");
38.         }

```

```

39.
40.        // reset de la base
41.        public void ClearCustomers()
42.        {
43.            Dao.getResponse<List<Product>, Object>(String.Format("{0}/ClearCustomers", UrlServer), null);
44.        }
45.    }
46. }

```

La classe [Metier] reprend le code que nous venons de voir dans la classe [TestDao]. Les propriétés des lignes 12 et 14 seront injectées par Spring. Pour cela, le fichier de configuration [App.config] de l'application est le suivant :

```

1.  <?xml version="1.0" encoding="utf-8" ?>
2.  <configuration>
3.      <configSections>
4.          <!-- spring -->
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.     <!-- configuration Spring -->
11.     <spring>
12.         <context>
13.             <resource uri="config://spring/objects" />
14.         </context>
15.         <objects xmlns="http://www.springframework.net">
16.             <object id="dao" type="Afb.Client.Dao.Dao,afb-client-webjson" />
17.             <object id="metier" type="Afb.Client.Metier.Metier,afb-client-webjson">
18.                 <property name="Dao" ref="dao" />
19.                 <property name="UrlServer" value="http://localhost:52892/Afb"/>
20.             </object>
21.         </objects>
22.     </spring>
23.     <startup>
24.         <supportedRuntime version="v4.0" sku=".NETFramework,Version=v4.5" />
25.     </startup>
26. </configuration>

```

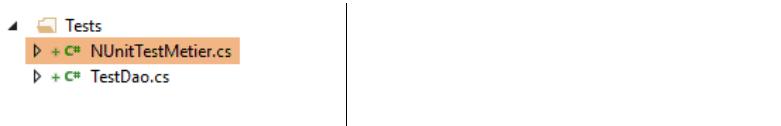
Ce code est laissé à votre compréhension. Vous adapterez l'URL de la ligne 19 à celle de votre serveur.

---

**Question 5** : compléter la classe [Metier].

---

### 7.7.7 Tests de la couche [métier]



La classe [NUnitTestMetier] est la classe de test unitaire NUnit déjà utilisée côté serveur. C'est normal puisque c'est la même interface [IMetier] qui est testée.

```

1.  using Afb.Client.Entities;
2.  using Afb.Client.Metier;
3.  using NUnit.Framework;
4.  using Spring.Context.Support;
5.  using System.Collections.Generic;
6.
7.  namespace Afb.Client.Tests
8.  {
9.
10.     [TestFixture]
11.     class NUnitTestMetier
12.     {
13.         // la couche [métier]
14.         private IMetier metier;
15.
16.         [TestFixtureSetUp]
17.         public void Init()
18.         {
19.             // instanciation couche [metier] via Spring
20.             metier = ContextRegistry.GetContext().GetObject("metier") as IMetier;
21.         }
22.
23.         [TestFixtureTearDown]
24.         public void Dispose()
25.         { /* ... */ }

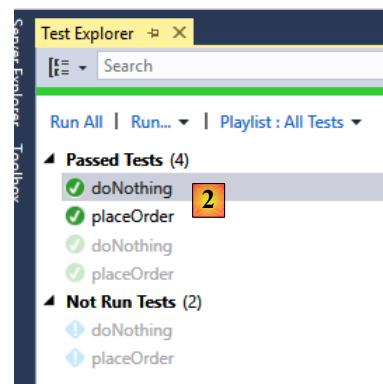
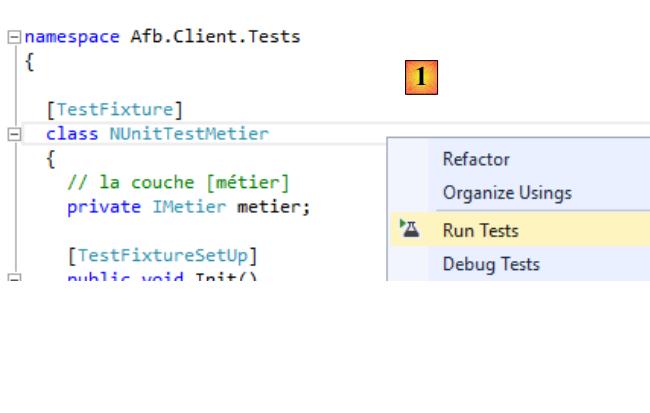
```

```

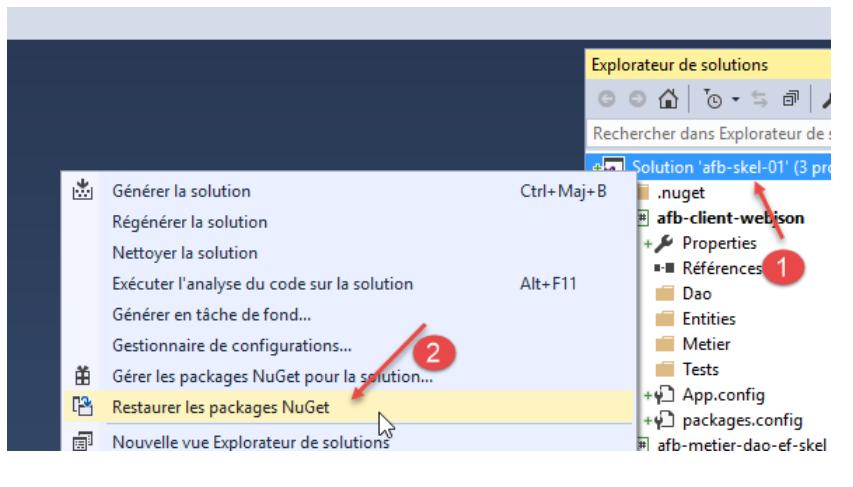
26.
27.     [SetUp]
28.     public void BeforeTest()
29.     {
30.         metier.ClearCustomers();
31.     }
32.
33.     [TearDown]
34.     public void endTest()
35.     {
36.     }
37.
38.     [Test]
39.     public void doNothing()
40.     {
41.     }
42.
43.     [Test]
44.     public void placeOrder()
45.     {
46.         // liste des catégories
47.         List<Category> categories = metier.Categories();
48.         Assert.AreEqual(4, categories.Count);
49.         // liste des produits catégorie 0
50.         long? categoryId = categories[0].Id;
51.         List<Product> products = metier.ProductsByCategory((int)categoryId);
52.         // remplissage chariot
53.         ShoppingCart chariot = new ShoppingCart();
54.         // 2 produits 0
55.         chariot.Update(products[0], 2);
56.         // 3 produits 1
57.         chariot.Update(products[1], 3);
58.         // on force le calcul du prix à payer
59.         chariot.CalculateTotal();
60.         // persistance client
61.         Customer customer = new Customer() { Name = "1", Email = "2", Phone = "3", Address = "4", CityRegion = "55",
62.             CcNumber = "6" };
63.         CustomerOrder customerOrder = metier.PlaceOrder(customer, chariot);
64.
65.         // vérifications
66.         List<OrderedProduct> orderedProducts = metier.OrderDetails((int)customerOrder.Id);
67.         Assert.AreEqual(products[0].Name, orderedProducts[0].Product.Name);
68.         Assert.AreEqual(products[1].Name, orderedProducts[1].Product.Name);
69.         Assert.AreEqual(2, orderedProducts[0].Quantity);
70.         Assert.AreEqual(3, orderedProducts[1].Quantity);
71.         CustomerOrder customerOrder2 = orderedProducts[0].CustomerOrder;
72.         Assert.AreEqual(10.57, (double)customerOrder2.Amount, 1e-6);
73.         Customer customer2 = customerOrder2.Customer;
74.         Assert.AreEqual("1", customer2.Name);
75.         Assert.AreEqual("2", customer2.Email);
76.         Assert.AreEqual("3", customer2.Phone);
77.         Assert.AreEqual("4", customer2.Address);
78.         Assert.AreEqual("55", customer2.CityRegion);
79.         Assert.AreEqual("6", customer2.CcNumber);
80.     }
81. }

```

Lancez votre serveur web / JSON puis exécutez [1] le test unitaire ci-dessus :



Il doit réussir [2].



## 7.8 Conclusion

Cette étude de cas est un exemple représentatif d'une application client / serveur réalisée avec ASP.NET MVC et Entity Framework. Vous pouvez la prendre comme base pour des projets analogues.

# Table of Contents

<b>1 LE COURS.....</b>	<b>9</b>
<b>  1.1 INTRODUCTION.....</b>	<b>9</b>
<b>1.1.1 LA PLACE DE ASP.NET MVC DANS UNE APPLICATION WEB.....</b>	<b>9</b>
<b>1.1.2 LE MODÈLE DE DÉVELOPPEMENT DE ASP.NET MVC.....</b>	<b>10</b>
<b>1.1.3 LES OUTILS UTILISÉS.....</b>	<b>11</b>
<b>1.1.4 LES EXEMPLES.....</b>	<b>13</b>
<b>  1.2 LES BASES DE LA PROGRAMMATION WEB.....</b>	<b>14</b>
<b>1.2.1 LES ÉCHANGES DE DONNÉES DANS UNE APPLICATION WEB AVEC FORMULAIRE.....</b>	<b>15</b>
<b>1.2.2 PAGES WEB STATIQUES, PAGES WEB DYNAMIQUES.....</b>	<b>16</b>
<b>1.2.2.1 Page statique HTML (HyperText Markup Language).....</b>	<b>16</b>
<b>1.2.2.2 Une page ASP.NET.....</b>	<b>18</b>
<b>1.2.2.3 Conclusion.....</b>	<b>21</b>
<b>1.2.3 SCRIPTS CÔTÉ NAVIGATEUR.....</b>	<b>21</b>
<b>1.2.4 LES ÉCHANGES CLIENT-SERVEUR.....</b>	<b>22</b>
<b>1.2.4.1 Le modèle OSI.....</b>	<b>23</b>
<b>1.2.4.2 Le modèle TCP/IP.....</b>	<b>24</b>
<b>1.2.4.3 Le protocole HTTP.....</b>	<b>26</b>
<b>1.2.4.4 Conclusion.....</b>	<b>30</b>
<b>1.2.5 LES BASES DU LANGAGE HTML.....</b>	<b>30</b>
<b>1.2.5.1 Un exemple.....</b>	<b>31</b>
<b>1.2.5.2 Un formulaire HTML.....</b>	<b>33</b>
<b>1.2.5.2.1 Le formulaire.....</b>	<b>35</b>
<b>1.2.5.2.2 Les champs de saisie texte.....</b>	<b>36</b>
<b>1.2.5.2.3 Les champs de saisie multilignes.....</b>	<b>36</b>
<b>1.2.5.2.4 Les boutons radio.....</b>	<b>37</b>
<b>1.2.5.2.5 Les cases à cocher.....</b>	<b>37</b>
<b>1.2.5.2.6 La liste déroulante (combo).....</b>	<b>37</b>
<b>1.2.5.2.7 Liste à sélection unique.....</b>	<b>38</b>
<b>1.2.5.2.8 Liste à sélection multiple.....</b>	<b>38</b>
<b>1.2.5.2.9 Bouton de type button.....</b>	<b>39</b>
<b>1.2.5.2.10 Bouton de type submit.....</b>	<b>39</b>
<b>1.2.5.2.11 Bouton de type reset.....</b>	<b>40</b>
<b>1.2.5.2.12 Champ caché.....</b>	<b>40</b>
<b>1.2.5.3 Envoi à un serveur Web par un client Web des valeurs d'un formulaire.....</b>	<b>40</b>
<b>1.2.5.3.1 Méthode GET.....</b>	<b>40</b>
<b>1.2.5.3.2 Méthode POST.....</b>	<b>43</b>
<b>1.2.6 CONCLUSION.....</b>	<b>44</b>
<b>  1.3 CONTRÔLEURS, ACTIONS, ROUTAGE.....</b>	<b>45</b>
<b>1.3.1 LA STRUCTURE D'UN PROJET ASP.NET MVC.....</b>	<b>45</b>
<b>1.3.2 LE ROUTAGE PAR DÉFAUT DES URL.....</b>	<b>48</b>
<b>1.3.3 CRÉATION D'UN CONTRÔLEUR ET D'UNE PREMIÈRE ACTION.....</b>	<b>50</b>
<b>1.3.4 ACTION AVEC UN RÉSULTAT DE TYPE [CONTENTRESULT] - 1.....</b>	<b>54</b>
<b>1.3.5 ACTION AVEC UN RÉSULTAT DE TYPE [CONTENTRESULT] - 2.....</b>	<b>55</b>
<b>1.3.6 ACTION AVEC UN RÉSULTAT DE TYPE [JSONRESULT].....</b>	<b>56</b>
<b>1.3.7 ACTION AVEC UN RÉSULTAT DE TYPE [STRING].....</b>	<b>57</b>
<b>1.3.8 ACTION AVEC UN RÉSULTAT DE TYPE [EMPTYRESULT].....</b>	<b>57</b>
<b>1.3.9 ACTION AVEC UN RÉSULTAT DE TYPE [REDIRECTRESULT] - 1.....</b>	<b>58</b>
<b>1.3.10 ACTION AVEC UN RÉSULTAT DE TYPE [REDIRECTRESULT] - 2.....</b>	<b>59</b>
<b>1.3.11 ACTION AVEC UN RÉSULTAT DE TYPE [REDIRECTToROUTERESULT].....</b>	<b>59</b>
<b>1.3.12 ACTION AVEC UN RÉSULTAT DE TYPE [VOID].....</b>	<b>60</b>
<b>1.3.13 UN SECOND CONTRÔLEUR.....</b>	<b>60</b>
<b>1.3.14 ACTION FILTRÉE PAR UN ATTRIBUT.....</b>	<b>62</b>
<b>1.3.15 RÉCUPÉRER LES ÉLÉMENTS D'UNE ROUTE.....</b>	<b>63</b>
<b>  1.4 LE MODÈLE D'UNE ACTION.....</b>	<b>65</b>
<b>1.4.1 INITIALISATION DES PARAMÈTRES DE L'ACTION.....</b>	<b>65</b>
<b>1.4.2 VÉRIFIER LA VALIDITÉ DES PARAMÈTRES DE L'ACTION.....</b>	<b>69</b>
<b>1.4.3 UNE ACTION À PLUSIEURS PARAMÈTRES.....</b>	<b>72</b>
<b>1.4.4 UTILISER UNE CLASSE COMME MODÈLE D'UNE ACTION.....</b>	<b>73</b>
<b>1.4.5 MODÈLE DE L'ACTION AVEC CONTRAINTES DE VALIDITÉ - 1.....</b>	<b>74</b>
<b>1.4.6 MODÈLE DE L'ACTION AVEC CONTRAINTES DE VALIDITÉ - 2.....</b>	<b>79</b>
<b>1.4.7 MODÈLE DE L'ACTION AVEC CONTRAINTES DE VALIDITÉ - 3.....</b>	<b>81</b>

<b>1.4.8</b> MODÈLE D'ACTION DE TYPE TABLEAU OU LISTE.....	82
<b>1.4.9</b> FILTRAGE D'UN MODÈLE D'ACTION.....	83
<b>1.4.10</b> ÉTENDRE LE MODÈLE DE LIAISON DES DONNÉES.....	84
<b>1.4.11</b> LIAISON TARDIVE DU MODÈLE DE L'ACTION.....	89
<b>1.4.12</b> CONCLUSION.....	90
<b>1.5</b> LA VUE ET SON MODÈLE.....	<b>92</b>
<b>1.5.1</b> INTRODUCTION.....	92
<b>1.5.2</b> UTILISER LE [VIEWBAG] POUR PASSER DES INFORMATIONS À LA VUE.....	95
<b>1.5.3</b> UTILISER UN MODÈLE FORTEMENT TYPÉ POUR PASSER DES INFORMATIONS À LA VUE.....	96
<b>1.5.4</b> [RAZOR] – PREMIERS PAS.....	102
<b>1.5.5</b> FORMULAIRE – PREMIERS PAS.....	106
<b>1.5.6</b> FORMULAIRE – UN EXEMPLE COMPLET.....	113
<b>1.5.6.1</b> Le modèle de portée [Application].....	114
<b>1.5.6.2</b> Le modèle de l'action [Action08Get].....	116
<b>1.5.6.3</b> Le modèle de la vue [Formulaire].....	116
<b>1.5.6.4</b> La vue [Formulaire].....	120
<b>1.5.6.5</b> Traitement du POST du formulaire.....	122
<b>1.5.6.6</b> Traitement des anomalies du POST.....	123
<b>1.5.7</b> UTILISATION DE MÉTHODES SPÉCIALISÉES DANS LA GÉNÉRATION DE FORMULAIRE.....	125
<b>1.5.7.1</b> Le nouveau formulaire.....	125
<b>1.5.7.2</b> Les actions et le modèle.....	129
<b>1.5.8</b> GÉNÉRATION D'UN FORMULAIRE À PARTIR DES MÉTADONNÉES DU MODÈLE.....	130
<b>1.5.8.1</b> Le [POST] du formulaire.....	134
<b>1.5.8.2</b> Propriété [Text].....	134
<b>1.5.8.3</b> Propriété [MultiLineText].....	135
<b>1.5.8.4</b> Propriété [Number].....	136
<b>1.5.8.5</b> Propriété [Decimal].....	137
<b>1.5.8.6</b> Propriété [Tel].....	138
<b>1.5.8.7</b> Propriété [Date].....	138
<b>1.5.8.8</b> Propriété [Time].....	139
<b>1.5.8.9</b> Propriété [HiddenInput].....	140
<b>1.5.8.10</b> Propriété [Boolean].....	141
<b>1.5.8.11</b> Propriété [Email].....	142
<b>1.5.8.12</b> Propriété [Url].....	142
<b>1.5.8.13</b> Propriété [Password].....	143
<b>1.5.8.14</b> Propriété [Currency].....	144
<b>1.5.8.15</b> Propriété [CreditCard].....	145
<b>1.5.9</b> VALIDATION D'UN FORMULAIRE.....	145
<b>1.5.9.1</b> Validation côté serveur.....	145
<b>1.5.9.2</b> Validation côté client.....	150
<b>1.5.10</b> GESTION DES LIENS DE NAVIGATION ET D'ACTION.....	153
<b>1.6</b> INTERNATIONALISATION DES VUES.....	<b>156</b>
<b>1.6.1</b> LOCALISATION DES NOMBRES RÉELS.....	156
<b>1.6.2</b> GÉRER UNE CULTURE.....	159
<b>1.6.3</b> INTERNATIONALISER LE MODÈLE DE VUE [VIEWMODEL14].....	161
<b>1.6.4</b> INTERNATIONALISER LA VUE [ACTION14GET.CSHMTL].....	167
<b>1.6.5</b> EXEMPLES D'EXÉCUTION.....	173
<b>1.6.6</b> INTERNATIONALISATION DES DATES.....	173
<b>1.6.7</b> CONCLUSION.....	179
<b>1.7</b> AJAXIFICATION D'UNE APPLICATION ASP.NET MVC.....	<b>180</b>
<b>1.7.1</b> LA PLACE D'AJAX DANS UNE APPLICATION WEB.....	180
<b>1.7.2</b> RUDIMENTS DE JQUERY ET DE JAVASCRIPT.....	181
<b>1.7.3</b> MISE À JOUR D'UNE PAGE AVEC UN FLUX HTML.....	184
<b>1.7.3.1</b> Les vues.....	184
<b>1.7.3.2</b> Le contrôleur, les actions, le modèle, la vue.....	185
<b>1.7.3.3</b> L'action [Action01Post].....	189
<b>1.7.3.4</b> La vue [Action01Error].....	190
<b>1.7.3.5</b> La vue [Action01Success].....	191
<b>1.7.3.6</b> Gestion de la session.....	191
<b>1.7.3.7</b> Gestion de l'image d'attente.....	192
<b>1.7.3.8</b> Gestion du lien [Calculer].....	192
<b>1.7.4</b> MISE À JOUR D'UNE PAGE HTML AVEC UN FLUX JSON.....	194
<b>1.7.4.1</b> L'action [Action02Get].....	194

<u><a href="#">1.7.4.2</a></u> L'action [Action02Post].....	195
<u><a href="#">1.7.4.3</a></u> Le code Javascript côté client.....	196
<u><a href="#">1.7.4.4</a></u> Le lien [Calculer].....	198
<u><a href="#">1.7.5</a></u> APPLICATION WEB À PAGE UNIQUE.....	200
<u><a href="#">1.7.6</a></u> APPLICATION WEB À PAGE UNIQUE ET VALIDATION CÔTÉ CLIENT.....	204
<u><a href="#">1.7.6.1</a></u> Les vues de l'exemple.....	204
<u><a href="#">1.7.6.2</a></u> Le modèle des vues.....	209
<u><a href="#">1.7.6.3</a></u> Les données de portée [Session].....	209
<u><a href="#">1.7.6.4</a></u> L'action serveur [Action05Get].....	211
<u><a href="#">1.7.6.5</a></u> L'action client [Calculer].....	211
<u><a href="#">1.7.6.6</a></u> L'action client [Effacer].....	215
<u><a href="#">1.7.6.7</a></u> L'action client [Retour aux Saisies].....	216
<u><a href="#">1.7.7</a></u> RENDRE ACCESIBLE SUR INTERNET UNE APPLICATION ASP.NET.....	218
<u><a href="#">1.7.8</a></u> GÉNÉRATION D'UNE APPLICATION NATIVE POUR ANDROID À PARTIR D'UNE APPLICATION À PAGE UNIQUE APU.....	218
<b><u><a href="#">1.8 CONCLUSION INTERMÉDIAIRE</a></u></b> .....	<b>219</b>
<b><u><a href="#">2 ETUDE DE CAS N° 1 : GESTION BASIQUE DE SALAIRES.</a></u></b> .....	<b>220</b>
<u><a href="#">2.1 INTRODUCTION</a></u> .....	220
<u><a href="#">2.2 LE PROBLÈME À RÉSOUTRE</a></u> .....	220
<u><a href="#">2.3 ARCHITECTURE DE L'APPLICATION</a></u> .....	222
<u><a href="#">2.4 LA BASE DE DONNÉES</a></u> .....	223
<u><a href="#">2.5 MODE DE CALCUL DU SALAIRE D'UNE ASSISTANTE MATERNELLE</a></u> .....	225
<u><a href="#">2.6 LE PROJET VISUAL STUDIO DE LA COUCHE [WEB]</a></u> .....	226
<b><u><a href="#">2.7 ÉTAPE 1 – MISE EN PLACE DE COUCHE [MÉTIER] SIMULÉE</a></u></b> .....	<b>227</b>
<u><a href="#">2.7.1</a></u> LA SOLUTION VISUAL STUDIO DE L'APPLICATION COMPLÈTE.....	227
<u><a href="#">2.7.2</a></u> L'INTERFACE DE LA COUCHE [MÉTIER].....	228
<u><a href="#">2.7.3</a></u> LES ENTITÉS DE LA COUCHE [MÉTIER].....	230
<u><a href="#">2.7.4</a></u> LA CLASSE [PAMEXCEPTION].....	233
<u><a href="#">2.7.5</a></u> IMPLÉMENTATION DE LA COUCHE [MÉTIER].....	233
<u><a href="#">2.7.6</a></u> LE TEST CONSOLE DE LA COUCHE [MÉTIER].....	235
<b><u><a href="#">2.8 ÉTAPE 2 : MISE EN PLACE DE L'APPLICATION WEB</a></u></b> .....	<b>237</b>
<b><u><a href="#">2.9 ÉTAPE 3 : MISE EN PLACE DU MODÈLE APU</a></u></b> .....	<b>242</b>
<u><a href="#">2.9.1</a></u> LES OUTILS DU DÉVELOPPEUR JAVASCRIPT.....	243
<u><a href="#">2.9.2</a></u> UTILISATION D'UNE VUE PARTIELLE POUR AFFICHER LE FORMULAIRE.....	244
<u><a href="#">2.9.3</a></u> L'APPEL AJAX [FAIRESIMULATION].....	244
<u><a href="#">2.9.4</a></u> L'APPEL AJAX [ENREGISTRERSIMULATION].....	246
<u><a href="#">2.9.5</a></u> L'APPEL AJAX [VOIRSIMULATIONS].....	247
<u><a href="#">2.9.6</a></u> L'APPEL AJAX [RETOURFORMULAIRE].....	248
<u><a href="#">2.9.7</a></u> L'APPEL AJAX [TERMINERSESSION].....	248
<u><a href="#">2.9.8</a></u> LA FONCTION JS [EFFACERSIMULATION].....	248
<u><a href="#">2.9.9</a></u> GESTION DE LA NAVIGATION ENTRE ÉCRANS.....	249
<b><u><a href="#">2.10 ÉTAPE 4 : ÉCRITURE DE L'ACTION SERVEUR [INDEX]</a></u></b> .....	<b>252</b>
<u><a href="#">2.10.1</a></u> LE MODÈLE DU FORMULAIRE.....	253
<u><a href="#">2.10.2</a></u> LE MODÈLE DE L'APPLICATION.....	254
<u><a href="#">2.10.3</a></u> LE CODE DE L'ACTION [INDEX].....	256
<u><a href="#">2.10.4</a></u> LE MODÈLE DE LA VUE [INDEX.CSHTML].....	256
<u><a href="#">2.10.5</a></u> LES VUES [INDEX.CSHTML] ET [FORMULAIRE.CSHTML].....	258
<u><a href="#">2.10.6</a></u> TEST DE L'ACTION [INDEX].....	258
<b><u><a href="#">2.11 ÉTAPE 5 : MISE EN PLACE DE LA VALIDATION DES SAISIES</a></u></b> .....	<b>259</b>
<u><a href="#">2.11.1</a></u> LE PROBLÈME.....	259
<u><a href="#">2.11.2</a></u> SAISIE DES NOMBRES RÉELS AU FORMAT FRANÇAIS.....	262
<u><a href="#">2.11.3</a></u> VALIDATION DU FORMULAIRE PAR LE LIEN JAVASCRIPT [FAIRE LA SIMULATION].....	263
<b><u><a href="#">2.12 ÉTAPE 6 : FAIRE UNE SIMULATION</a></u></b> .....	<b>264</b>
<u><a href="#">2.12.1</a></u> LE PROBLÈME.....	264
<u><a href="#">2.12.2</a></u> ÉCRITURE DE LA VUE [SIMULATION.CSHTML].....	264
<u><a href="#">2.12.3</a></u> CALCUL DU SALAIRE RÉEL.....	267
<u><a href="#">2.12.4</a></u> GESTION DES ERREURS.....	269
<b><u><a href="#">2.13 ÉTAPE 7 : MISE EN PLACE D'UNE SESSION UTILISATEUR</a></u></b> .....	<b>274</b>
<b><u><a href="#">2.14 ÉTAPE 8 : ENREGISTRER UNE SIMULATION</a></u></b> .....	<b>275</b>
<u><a href="#">2.14.1</a></u> LE PROBLÈME.....	275
<u><a href="#">2.14.2</a></u> ÉCRITURE DE L'ACTION SERVEUR [ENREGISTRERSIMULATION].....	277
<u><a href="#">2.14.3</a></u> ÉCRITURE DE LA VUE PARTIELLE [SIMULATIONS.CSHTML].....	277
<b><u><a href="#">2.15 ÉTAPE 9 : RETOURNER AU FORMULAIRE DE SAISIE</a></u></b> .....	<b>278</b>
<u><a href="#">2.15.1</a></u> LE PROBLÈME.....	278

<b>2.15.2</b> ÉCRITURE DE L'ACTION SERVEUR [FORMULAIRE].....	279
<b>2.15.3</b> MODIFICATION DE LA FONCTION JAVASCRIPT [RETOURFORMULAIRE].....	279
<b>2.16</b> ÉTAPE 10 : VOIR LA LISTE DES SIMULATIONS.....	<b>280</b>
<b>2.16.1</b> LE PROBLÈME.....	280
<b>2.16.2</b> ÉCRITURE DE L'ACTION SERVEUR [VOIRSIMULATIONS].....	280
<b>2.17</b> ÉTAPE 11 : TERMINER LA SESSION.....	<b>281</b>
<b>2.17.1</b> LE PROBLÈME.....	281
<b>2.17.2</b> ÉCRITURE DE L'ACTION SERVEUR [TERMINERSESSION].....	281
<b>2.17.3</b> MODIFICATION DE LA FONCTION JAVASCRIPT [TERMINERSESSION].....	282
<b>2.18</b> ÉTAPE 12 : EFFACER LA SIMULATION.....	<b>282</b>
<b>2.18.1</b> LE PROBLÈME.....	282
<b>2.18.2</b> ÉCRITURE DE L'ACTION CLIENT [EFFACERSIMULATION].....	282
<b>2.19</b> ÉTAPE 13 : RETIRER UNE SIMULATION.....	<b>282</b>
<b>2.19.1</b> LE PROBLÈME.....	282
<b>2.19.2</b> ÉCRITURE DE L'ACTION CLIENT [RETRIERSIMULATION].....	283
<b>2.19.3</b> ÉCRITURE DE L'ACTION SERVEUR [RETRIERSIMULATION].....	283
<b>2.20</b> ÉTAPE 14 : AMÉLIORATION DE LA MÉTHODE D'INITIALISATION DE L'APPLICATION.....	<b>284</b>
<b>2.20.1</b> AJOUT DES RÉFÉRENCES [SPRING] AU PROJET WEB.....	285
<b>2.20.2</b> CONFIGURATION DE [WEB.CONFIG].....	286
<b>2.20.3</b> MODIFICATION DE [APPLICATION_START].....	287
<b>2.20.4</b> GÉRER UNE ERREUR D'INITIALISATION DE L'APPLICATION.....	287
<b>2.21</b> OÙ EN SOMMES – NOUS ?.....	<b>290</b>
<b>2.22</b> ÉTAPE 15 : MISE EN PLACE DE LA COUCHE ENTITY FRAMEWORK 5.....	<b>291</b>
<b>2.22.1</b> LA BASE DE DONNÉES.....	291
<b>2.22.2</b> LE PROJET VISUAL STUDIO.....	295
<b>2.22.3</b> AJOUT DES RÉFÉRENCES NÉCESSAIRES AU PROJET.....	296
<b>2.22.4</b> LES ENTITÉS ENTITY FRAMEWORK.....	297
<b>2.22.5</b> CONFIGURATION DE L'ORM EF5.....	300
<b>2.22.6</b> TEST DE LA COUCHE [EF5].....	302
<b>2.22.7</b> DLL DE LA COUCHE [EF5].....	304
<b>2.23</b> ÉTAPE 16 : MISE EN PLACE DE LA COUCHE [DAO].....	<b>304</b>
<b>2.23.1</b> L'INTERFACE DE LA COUCHE [DAO].....	304
<b>2.23.2</b> LE PROJET VISUAL STUDIO.....	305
<b>2.23.3</b> AJOUT DES RÉFÉRENCES NÉCESSAIRES AU PROJET.....	305
<b>2.23.4</b> IMPLÉMENTATION DE LA COUCHE [DAO].....	306
<b>2.23.5</b> CONFIGURATION DE LA COUCHE [DAO].....	308
<b>2.23.6</b> TEST DE LA COUCHE [DAO].....	308
<b>2.23.7</b> DLL DE LA COUCHE [DAO].....	309
<b>2.24</b> ÉTAPE 17 : MISE EN PLACE DE LA COUCHE [MÉTIER].....	<b>309</b>
<b>2.24.1</b> L'INTERFACE DE LA COUCHE [MÉTIER].....	309
<b>2.24.2</b> LE PROJET VISUAL STUDIO.....	309
<b>2.24.3</b> AJOUT DES RÉFÉRENCES NÉCESSAIRES AU PROJET.....	310
<b>2.24.4</b> IMPLÉMENTATION DE LA COUCHE [MÉTIER].....	310
<b>2.24.5</b> CONFIGURATION DE LA COUCHE [MÉTIER].....	311
<b>2.24.6</b> TEST DE LA COUCHE [MÉTIER].....	312
<b>2.24.7</b> DLL DE LA COUCHE [MÉTIER].....	313
<b>2.25</b> ÉTAPE 18 : MISE EN PLACE DE LA COUCHE [WEB].....	<b>313</b>
<b>2.25.1</b> LE PROJET VISUAL STUDIO.....	314
<b>2.25.2</b> AJOUT DES RÉFÉRENCES NÉCESSAIRES AU PROJET.....	315
<b>2.25.3</b> IMPLÉMENTATION DE LA COUCHE [WEB].....	315
<b>2.25.4</b> CONFIGURATION DE LA COUCHE [WEB].....	316
<b>2.25.5</b> TEST DE LA COUCHE [WEB].....	316
<b>2.26</b> ÉTAPE 19 : RENDRE ACCESSIBLE SUR INTERNET UNE APPLICATION ASP.NET.....	<b>321</b>
<b>2.27</b> ÉTAPE 20 : GÉNÉRATION D'UNE APPLICATION NATIVE POUR ANDROID.....	<b>332</b>
<b>2.27.1</b> L'ARCHITECTURE DE L'APPLICATION.....	332
<b>2.27.2</b> REFACTORISATION DU PROJET [PAM-WEB-02].....	334
<b>2.27.3</b> TEST DU PROJET REFACTORISÉ.....	337
<b>2.27.4</b> CRÉATION DU BINAIRE ANDROID.....	339
<b>2.28</b> ANNEXES.....	<b>343</b>
<b>2.28.1</b> INSTALLATION DU GESTIONNAIRE D'ÉMULATEURS GENYMOTION.....	343
<b>2.29</b> CONCLUSION.....	<b>345</b>
<b>3</b> ÉTUDE DE CAS N° 2 : GESTION DE BILANS DE FORMATION.....	<b>346</b>
<b>3.1</b> LES COMPÉTENCES MISES EN OEUVRE.....	346

<b>3.2 LE PROBLÈME.....</b>	<b>346</b>
<b>3.3 L'ARCHITECTURE DE L'APPLICATION.....</b>	<b>347</b>
<b>3.4 LA BASE DE DONNÉES.....</b>	<b>347</b>
<b>3.5 LES COUCHES [DAO, ENTITY FRAMEWORK].....</b>	<b>351</b>
<b>3.5.1 INSTALLATION ET TESTS.....</b>	<b>351</b>
<b>3.5.2 L'INTERFACE DE LA COUCHE [DAO].....</b>	<b>352</b>
<b>3.5.3 VERSIONS COURTES ET LONGUES DES ENTITÉS.....</b>	<b>357</b>
<b>3.6 LA COUCHE [WEB / ASP.NET MVC].....</b>	<b>359</b>
<b>3.6.1 L'IMPLÉMENTATION DE DÉMARRAGE.....</b>	<b>359</b>
<b>3.6.2 CONFIGURATION DE L'APPLICATION.....</b>	<b>361</b>
<b>3.6.3 INITIALISATION DE L'APPLICATION.....</b>	<b>362</b>
<b>3.6.4 LE MODÈLE DE PORTÉE [APPLICATION].....</b>	<b>363</b>
<b>3.6.5 LE ROUTAGE DES URL.....</b>	<b>366</b>
<b>3.6.6 LA PAGE [TEST.CSHTML] ET SON MODÈLE [TESTMODEL].....</b>	<b>366</b>
<b>3.6.7 LE PATRON DES VUES.....</b>	<b>367</b>
<b>3.6.8 L'ACTION [TEST].....</b>	<b>369</b>
<b>3.6.9 LE BEAN [SESSIONMODEL].....</b>	<b>370</b>
<b>3.7 IMPLÉMENTATION DE L'APPLICATION.....</b>	<b>370</b>
<b>3.7.1 ÉTAPE 1.....</b>	<b>370</b>
<b>3.7.2 ÉTAPE 2.....</b>	<b>376</b>
<b>3.7.3 ÉTAPE 3.....</b>	<b>377</b>
<b>3.7.4 ÉTAPE 4.....</b>	<b>377</b>
<b>3.7.5 ÉTAPE 5.....</b>	<b>377</b>
<b>3.7.6 ÉTAPE 6.....</b>	<b>378</b>
<b>3.7.7 ÉTAPE 7.....</b>	<b>379</b>
<b>3.7.8 ÉTAPE 8.....</b>	<b>381</b>
<b>3.7.9 ÉTAPE 9.....</b>	<b>382</b>
<b>3.7.10 ÉTAPE 10.....</b>	<b>384</b>
<b>3.7.11 ÉTAPE 11.....</b>	<b>385</b>
<b>3.7.12 ÉTAPE 12.....</b>	<b>385</b>
<b>3.7.13 ÉTAPE 13.....</b>	<b>386</b>
<b>3.7.14 ÉTAPE 14.....</b>	<b>387</b>
<b>4 ÉTUDE DE CAS N° 3 : APPLICATION E-COMMERCE.....</b>	<b>388</b>
<b>4.1 LES COMPÉTENCES MISES EN OEUVRE.....</b>	<b>388</b>
<b>4.2 LE PROBLÈME.....</b>	<b>388</b>
<b>4.3 PRÉSENTATION DES COUCHES BASSES.....</b>	<b>389</b>
<b>4.3.1 MISE EN PLACE DE L'ENVIRONNEMENT DE TRAVAIL.....</b>	<b>389</b>
<b>4.3.2 LA BASE DE DONNÉES.....</b>	<b>390</b>
<b>4.3.3 LES ENTITÉS MANIPULÉES PAR LES COUCHES BASSES.....</b>	<b>394</b>
<b>4.3.4 L'INTERFACE [IMETIER] DES COUCHES BASSES.....</b>	<b>400</b>
<b>4.3.5 LA CONFIGURATION DES COUCHES BASSES.....</b>	<b>401</b>
<b>4.3.6 VÉRIFICATION DU FONCTIONNEMENT DES COUCHES BASSES.....</b>	<b>402</b>
<b>4.4 ECRITURE DE LA COUCHE WEB / MVC.....</b>	<b>405</b>
<b>4.4.1 LES VUES DE L'APPLICATION.....</b>	<b>406</b>
<b>4.4.2 LE PROJET VISUAL STUDIO.....</b>	<b>408</b>
<b>4.4.3 CONFIGURATION.....</b>	<b>409</b>
<b>4.4.4 LA CLASSE [APPLICATIONMODEL].....</b>	<b>410</b>
<b>4.4.5 LA CLASSE [SESSIONMODEL].....</b>	<b>412</b>
<b>4.4.6 LES CLASSES [PARENTVIEWMODEL] ET [PARENTMODELBINDER].....</b>	<b>412</b>
<b>4.4.7 INITIALISATION DE L'APPLICATION.....</b>	<b>413</b>
<b>4.4.8 ROUTAGE DE L'APPLICATION.....</b>	<b>414</b>
<b>4.4.9 LE MODÈLE DES VUES.....</b>	<b>414</b>
<b>4.4.10 LA PAGE D'ACCUEIL [CATEGORIES.CSHTML].....</b>	<b>417</b>
<b>4.4.11 LA PAGE DES PRODUITS VENDUS [PRODUCTS.CSHTML].....</b>	<b>420</b>
<b>4.4.12 LA MÉTHODE [ADDTOCART] DU CONTRÔLEUR.....</b>	<b>422</b>
<b>4.4.13 LA PAGE DU PANIER [CART.CSHTML].....</b>	<b>425</b>
<b>4.4.14 LE LIEN [CONTINUER LES ACHATS].....</b>	<b>427</b>
<b>4.4.15 LA MÉTHODE [UPDATEQUANTITY] DU CONTRÔLEUR.....</b>	<b>428</b>
<b>4.4.16 LA PAGE DE PAIEMENT [CHECKOUT.CSHTML].....</b>	<b>430</b>
<b>4.4.17 LA MÉTHODE [CHECKOUT] DU CONTRÔLEUR.....</b>	<b>432</b>
<b>4.4.18 LA PAGE DE CONFIRMATION [CONFIRMATION.CSHTML].....</b>	<b>433</b>
<b>4.4.19 LE LIEN [VIDER LE CHARIOT].....</b>	<b>435</b>
<b>5 ÉTUDE DE CAS N° 4 : GESTION DE RENDEZ-VOUS.....</b>	<b>436</b>

<b>5.1 INTRODUCTION.....</b>	436
<b>5.1.1 LE PROJET EXISTANT.....</b>	436
<b>5.1.2 MÉTHODES DE TRAVAIL.....</b>	438
<b>5.2 LES VUES DE L'APPLICATION WEB.....</b>	438
<b>5.3 TRAVAIL À FAIRE.....</b>	441
<b>6 ÉTUDE DE CAS N° 5 : GESTION BASIQUE DE NOTES D'ÉLÈVES.....</b>	442
<b>6.1 LES COMPÉTENCES MISES EN OEUVRE.....</b>	442
<b>6.2 LE PROBLÈME.....</b>	442
<b>6.3 L'ARCHITECTURE DE L'APPLICATION WEB.....</b>	443
<b>6.4 LES PROJETS VISUAL STUDIO DE L'APPLICATION.....</b>	444
<b>6.5 LA BASE DE DONNÉES.....</b>	445
<b>6.6 LA COUCHE [MÉTIER] ET LES ENTITÉS MANIPULÉES PAR CELLE-CI.....</b>	448
<b>6.7 IMPLÉMENTATION DE LA COUCHE [WEB].....</b>	452
<b>6.7.1 LE PROJET VISUAL STUDIO.....</b>	452
<b>6.7.2 CONFIGURATION DU PROJET WEB.....</b>	453
<b>6.7.3 INITIALISATION DE L'APPLICATION.....</b>	454
<b>6.7.4 LE MODÈLE DE PORTÉE [APPLICATION].....</b>	456
<b>6.7.5 LE MODÈLE DE PORTÉE [SESSION].....</b>	456
<b>6.7.6 LE ROUTAGE DE L'APPLICATION.....</b>	457
<b>6.8 L'URL [/ECOLE/EXO00].....</b>	457
<b>6.9 L'URL [/ECOLE/EXO01].....</b>	459
<b>6.9.1 INTRODUCTION.....</b>	459
<b>6.9.2 GESTION DU LIEN [AFFICHER LES ÉLÈVES].....</b>	462
<b>6.10 L'URL [/ECOLE/EXO02].....</b>	464
<b>7 ÉTUDE DE CAS N° 6 : CLIENT / SERVEUR JSON.....</b>	466
<b>7.1 LES COMPÉTENCES MISES EN OEUVRE.....</b>	466
<b>7.2 LE PROBLÈME.....</b>	466
<b>7.3 LA BASE DE DONNÉES.....</b>	467
<b>7.4 MISE EN PLACE DE L'ENVIRONNEMENT DE TRAVAIL.....</b>	470
<b>7.5 ECRITURE DES COUCHES BASSES.....</b>	471
<b>7.5.1 LE PROJET VISUAL STUDIO.....</b>	472
<b>7.5.2 LA COUCHE [ENTITY FRAMEWORK].....</b>	473
<b>7.5.2.1 Le code.....</b>	473
<b>7.5.2.2 Le test.....</b>	476
<b>7.5.2.2.1 Le code.....</b>	476
<b>7.5.2.2.2 La configuration.....</b>	478
<b>7.5.2.2.3 Les résultats.....</b>	479
<b>7.5.3 LA COUCHE [DAO].....</b>	479
<b>7.5.3.1 Le code.....</b>	480
<b>7.5.3.2 Le test.....</b>	485
<b>7.5.3.2.1 Le code.....</b>	485
<b>7.5.3.2.2 La configuration.....</b>	486
<b>7.5.3.2.3 Les résultats.....</b>	486
<b>7.5.4 LA COUCHE [MÉTIER].....</b>	487
<b>7.5.4.1 Le code.....</b>	487
<b>7.5.4.2 Le test console.....</b>	488
<b>7.5.4.2.1 Le code.....</b>	488
<b>7.5.4.2.2 La configuration.....</b>	489
<b>7.5.4.2.3 Les résultats.....</b>	490
<b>7.5.4.3 Le test NUnit.....</b>	490
<b>7.5.4.3.1 Le code.....</b>	490
<b>7.5.4.3.2 La configuration.....</b>	491
<b>7.5.5 CRÉATION DES DLL DES COUCHES BASSES.....</b>	493
<b>7.6 ECRITURE DE LA COUCHE WEB / JSON.....</b>	494
<b>7.6.1 ARCHITECTURE.....</b>	494
<b>7.6.2 LE PROJET VISUAL STUDIO.....</b>	495
<b>7.6.3 CONFIGURATION.....</b>	496
<b>7.6.4 LA CLASSE [APPLICATIONMODEL].....</b>	496
<b>7.6.5 INITIALISATION DE L'APPLICATION.....</b>	497
<b>7.6.6 LES URL EXPOSÉES PAR LE SERVICE WEB / JSON.....</b>	498
<b>7.6.7 LA CLASSE [APPLICATIONMODELBINDER].....</b>	499
<b>7.6.8 LA CLASSE [RESPONSE].....</b>	500
<b>7.6.9 LES MÉTHODES DU CONTRÔLEUR.....</b>	501

<u>7.6.9.1</u> L'URL [Afb/GetCategories].....	501
<u>7.6.9.2</u> Les autres URL.....	501
<b>7.7 ÉTUDE DU CLIENT WEB / JSON.....</b>	<b>503</b>
<u>7.7.1</u> ARCHITECTURE.....	503
<u>7.7.2</u> LE PROJET VISUAL STUDIO.....	503
<u>7.7.3</u> LES ENTITÉS.....	504
<u>7.7.4</u> IMPLÉMENTATION DE LA COUCHE [DAO].....	505
<u>7.7.5</u> TESTS DE LA COUCHE [DAO].....	506
<u>7.7.6</u> IMPLÉMENTATION DE LA COUCHE [MÉTIER].....	507
<u>7.7.7</u> TESTS DE LA COUCHE [MÉTIER].....	509
<b>7.8 CONCLUSION.....</b>	<b>511</b>