Setting up TP .Net MVC Training

22/03/2023

Table of Contents

Document follow-up:

Version	**Action**	**No one**	**Date**
V1.0	Creating the document	Joffrey LEFRERE	07/03/2019
V1.1	Editing the document	Aurélie BROS	12/03/2019
V1.2	Editing the document	Daphnée HOT	08/04/2019
V1.3	Editing the document	Aurélie BROS	25/04/2019
V1.4	Editing the document	Louis TEMPELAERE	04/12/2020
V1.5	Added examples of different types of controls in the part: Form – Step 5	Louis TEMPELAERE	17/02/2021
V1.6	Editing the document for project update to .Net6	Nicolas TRAN	

Purpose of the document

This document describes a practical work dedicated to employees trained in dual skills and destined in particular to evolve on the EI sites of the CM-CIC. It follows from the observation that the LOCAL(C#) part of the dual competence training delivered until then by HN Institut is not in line with the work subsequently requested from the client.

Principle

The TP aims to simulate an application allowing the evaluation of a training. This application will be structured as follows:

- A home page (Appendix 1)
- A form information page (Appendix 2)
- A validation page of the information entered on the form (Appendix 3)
- A page showing the list of reviews left (Appendix 4)

Needs

In order to carry out this TP, it will be necessary to make available to future employees:

- This specification document explaining the work to be done.
- A starting Visual Studio solution that contains the appropriate routing and folders.
- A file with the HN logo
- A file with a list of surnames, first name and O/N indicator in order to constitute the list appearing on the "List Page": "DataAvis.xml"
- A fully coded and commented Visual Studio solution. It will be consulted in case of blockage.

Specifications

Home page

In order to start calmly, we propose to accompany you step by step in the development of this first page (Appendix 1).

On this page you will practice:

- To format an HTML web page
- To create a menu
- To create navigation links between 2 pages

As a first step, it is necessary to create the solution that will serve as the basis for all developments. Retrieve the solution supplied with the TP. This is a APS.NET MVC (Vue Controller Model) solution, in which the controller and routing are already created to save you time. Indeed, you will not have to code this part within the CIC. Open TPLOCAL1.sln to open this solution. The technical choice of the VMC is similar to the one you will find at the CIC. The operation is as follows:

- The template contains the data used by the site. In C#, all data is in classes, so the model will be in the form of classes.
- The view contains what is seen on the web browser screen. It will consist of cshtml pages, which allows both html code and c# code.
- Finally, the controller defines the logic of the website, for example the links between the pages, the controls of the data entered by the user, the calls to the databases (not present in this TP), the readings of files...

Step 1: Display of logo and title

The purpose of this part is to align the logo and the title (Appendix 6).

Now we will create the first cshtml view page. To do this, in the Solution Explorer window (Appendix 5), right-click on the View/Home folder, then Add and View. Click Add. Name the Index view and then add.

This addition creates acshtml page, in which we will put html code, and which is recognized by the MVC engine. This page contains 2 parts:

- The header with the <head> tag : this section allows you to give the title of the page, the encoding...
- The bodys <body>: main part of the page.

IN THE HEADER: First of all, we will name the page with the <title> tag: `<title>HN - TP LOCAL</title>`This tag allows you to have the name of the page that appears in the internet tab.

IN THE MAIN PART OF THE PAGE:

To put a title on the page, you must use the <header> tag, it manages the header of an html page:

```
<header>
   <h1>Welcome to TP HN</h1>
</header>
```

The <h1> tag is used to define a level 1 title and will be used later in the page. CSS.

Now that the title is filled in, you must put the logo. To do this, in the solution folder, in the "resources" subfolder , put the file containing the image, making sure that the file name is GroupHN.png. If the file is in the folder but is not visible in the solution, add an existing item on the "resources" folder, and choose the file in Windows Explorer.
On the Index page. cshtml, you must add a tag <img>:

```
<img src="~/resources/GroupeHN.png" class="image" alt="GroupeHN" />
```

The src attribute indicates the location of the file and alt indicates what the image contains. The ~ indicates that it is a path, and autocompletion of the code normally proposes the path (Appendix 11). To use autocompletion, when it appears, use the arrows on the keyboard and enter or the mouse.

Run the debug by typing F5 (on the cshtml page). If the code does not compile, comment out the parts that are bugging (for example, the ValidationForm method of the controller).
A line is commented out when it has "// " at the beginning for a C# page, and <!-- --> for an HTML page.

The logo and title are one above the other. To align the logo and title, you need to create a page. CSS. The .CSS allow formatting of HTML CS pages.

To create a. CSS, as well as for the creation of a CSHTML page click on the "Resources" folder, then add new element, choose Web/stylesheet. It will be named Index.css First of all,

you have to take care of the image, by adding the following code in the CSS page :

```
.image
{
   width: 150px;
   height: 150px;
}
```

We just created the image class. The width and height attributes are used to resize the image in width and height.

For the title, you have to put in the CSS page :

```
h1
{
   margin-right: 170px;
   margin-left: 170px;
   text-align: center;
}
```

margin-right and margin-left are used to define a margin to the right and left of the text. text-align is used to set text alignment. To link the css page to the html page, you must create a link between these 2 pages in the cshtml page. To do this, use a <link> tag in the <head> section :

```
<link rel="stylesheet"  href="~/ressources/Index.css"/>
```

At this point, the title and logo are formatted, but when you run the debug, theystill haven't aligned.

To align the logo and the title, add 2 elements in the image and h1 classes: display:inline-block;

```
   vertical-align:middle;
```

The first line is used to make a block of the 2 elements and the second line is used to align the block.

Step 2: Creating the menu

The purpose of this part is to show you how to create a menu (Appendix 7).

The menu will be in the <body> part of the cshtml page.

The code for a menu is as follows:

```
<nav>
   <ul>
      <li><a href="#">Fill in the form</a></li>
      <li><a href="#">Avis list</a></li>
   </ul>
</nav>
```

Explanation of tags:

- <nav>: main navigation link, used among others for the main menu,
- <ul>: Bulleted list,
- <li>: Item in a bulleted list,

- <a>: Tag for hyperlinks, it must be associated with the href attribute to indicate to which page the link should lead. Here there is "#" because the pages are not yet created, it will be replaced later.

We can see thanks to the debug that the menu appeared on the page, but it is not very pretty. To improve the visual, you must modify the CSS page :

```
Nav
{
  width: 150px;
  border: 2px solid #8c014c;
  background-color: #8c014c;
  color: white;
}
li
{
  margin-top: 10px;
  margin-bottom: 15px;
} a

{
  color: chocolate;
}
```

2 CSS classes have been created, nav, which corresponds to the <nav> html and li tag for the <li> html tag.

Attribute explanation:

- width: width of the block in pixels (ability to do this as a percentage)
- border: definition of the border of the block, here the structure is thickness of the border (2 px) type of border (solid) color (#8c014c)
- background-color: color of the background of the block, either color (yellow, blue ...) or reference (#8c014c)
- color: color of the text
- margin-top: margin above the line
- margin-bottom: margin below the line

And here the menu is created, it will only remain to put the links.

Step 3: Creating the page body

The purpose of this step is to see how to fill in and put a link in the body of the page (Appendix 8).

As before, changes are made in the <body> part of the cshtml page.

To create the body of the page, you must add the following code:

```
  <section>
    <p>
      <a href="#">Fill out form </a>
    </p>
  </section>
```

Explanation of tags:

- <section>: page section, used to group content,
- <p>: paragraph.

Look at what your page looks like now that all the elements are in place. There is a problem, the body of the page is not in front of the menu but below. This is normal, you must modify the CSS page. In the nav class, we must add:

```
float: left;
```

The float property is used to make an object floating, it can be usedfor a menu or an image and can take 2 values that are left or right.

Now the body of the page and the menu are aligned but a little too pasted. Just create a new class to define a margin on the left for the section tag and voila:

```
section
{
  margin-left: 170px;
}
```

Step 4: Establishment of links

The purpose of this part is to show you how to create a link.

First, simply replace "#" with "/Home/Index/Form" in the <a href="/Home/Index/Form" tags>Fill out form </a>. This path means that the controller is called HomeController, and more precisely its Index function, to which the Form parameter is supplied. The controller does not need to be modified.

Next, you must create the Form.cshtml page. Be careful, you must create this view in the Views/Shared folder for the routing to work properly (if you are interested, you will find an explanatory link at the end of the statement however you will not need to know the reason once on site).

The return View(id) code; links to the Training page. The View method allows you to refer to the desired view, here without other parameters. Later, we will also use the overloaded form of the View method with the page to be called and a data model.

When you click on the newly set link, you arrive on a blank page which is normal because it is not yet encoded.

When the AvisList.cshtml page is created, you will be able to set up its link in the menu.

To see the code for the Index page. see Appendix 9 and for the index page code.css see Appendix 1-0.

Form

From this part, you will be less guided. If you encounter difficulties, look for the solution on the internet, then if you are unable to do so, consult the solution provided by HN.

This involves creating a page that will allow the user to fill in his personal data as well as his opinion on a training course (Appendix 2).

On this page you will practice:

- To create a drop-down list
- To set up various controls according to the information expected in each of the fields
- To handle error messages

The development of this page is organized in 6 steps:

Step 1: Display of logo and title

See previously (1st step homepage). Be careful, the title is different from that of the home page.

Step 2: Creating the menu

See previously (2nd step homepage). Be careful, the sections are different from those on the home page.

3rd^{me} Step: Creation of the "Personal Information" input table

As its name suggests, the user must be able to enter in this table several data that are specific to him.

The table must have 2 columns, one for the title of the information requested, the other for its information. However, there is an exception for the title that will have to be centered on the 2 columns.

Here are the different fields that the table should have with its restrictions / controls:

Left column (titles)	**Columnof title (type of fields, possible controls)**
Personal information	
Name	Input field
Forename	Input field
Gender	- Drop-down list with 4 choices: • Select a gender • Man • Woman • Other - By default the field must be filled in with "Select a gender"
Address	Input field
Zip code	- Input field - Control: must be on 5 numeric characters thanks to a regular expression
Town	Input field
Email address	- Input field - Format control to implement

Regex (= regular expression) possible: ^([\w]+)@([\w]+)\.([\w]+)$

(Source: https://lgmorand.developpez.com/dotnet/regex/)

The data is to be stored in a class in the Models folder.

4^{thme} Step: Creation of the input table "Training information followed"

This table allows the user to enter information about the training they have completed.

The table must have 2 columns, one for the title of the information requested, the other for its information. However, there is an exception for the title that will have to be centered on the 2 columns.

Here are the different fields that the table should have with its restrictions / controls:

Left column (titles)	**Right column (type of fields, any controls)**
Training information	
Start date training	- Date input field - The date must be less than 01/01/2021
Type of training	- Input field - Drop-down list with 4 choices: • Select a course • Cobol Training • Object training • Dual Skills Training - By default the field must be filled in with "Select a training"

Step 5: Creation of the "Training Notice" input table

This table allows the user to give his opinion on the training he has taken.

The table must have 2 columns, one for the title of the information requested, the other for its information. However, there is an exception for the title that will have to be centered on the 2 columns.

Here are the different fields that the table should have with its restrictions / controls:

Left column (titles)	**Right column**
Training Reviews	
Cobol Formation	- Input field
C# Formation	- Input field

For controls, it is possible to do them in HTML or in the declaration of variables in a class (see Required and StringLength Attribute in MVC - Dot Net Tutorials and Regular Expression Attribute in MVC - Dot Net Tutorials) or finally in the controller when clicking on the Validate button.

Some explanations:

- In HTML, in the input tag, it is possible to define in the "type" field the type of the data. Depending on the type, the visual is variable. For example, `<input type="date"/>` gives a calendar. This type of control is very convenient but not very flexible, as there are a limited number of types.
- In the variable declaration, Required can be added above the attribute variable to check that the data is filled in:

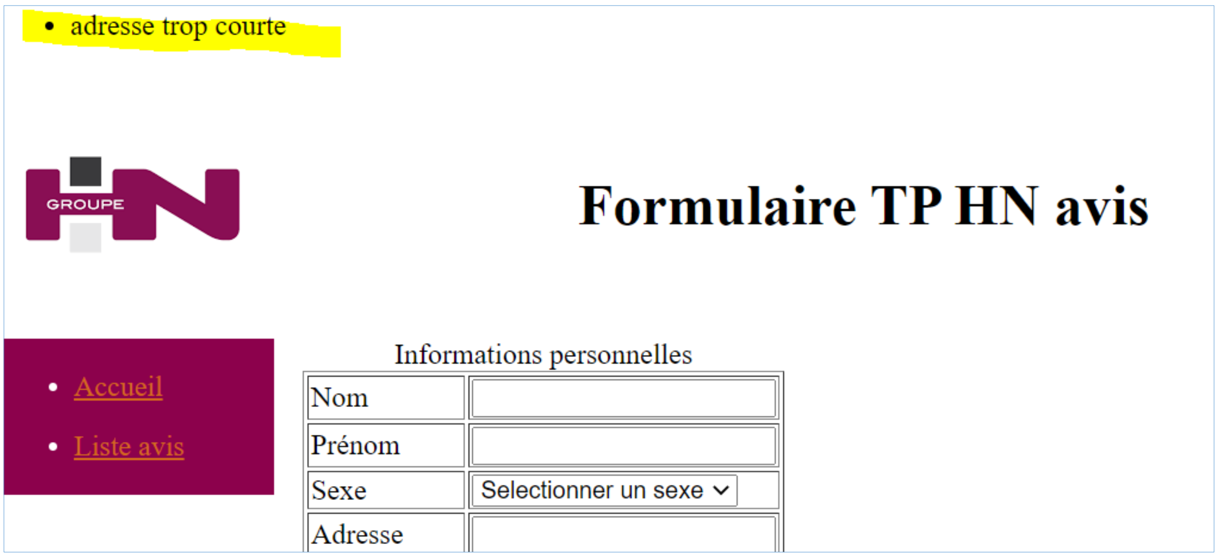
```
[Required]
public string Formation { get;  set; }
```

It is also possible to make more advanced checks in this way, for example on the size of the data to be entered by the user, with regular expressions, or the data range of the date.

- Finally, it is also possible to make controls in the function called by a button. For example, assuming that the model where the data is calls modelv, we can return an error if the address data is empty or not long enough:

```
if (modelv. Address == null || modelv. Address.Length < 5)
{
        ModelState.AddModelError("",  "address too short");
}
```

And to display errors, add in the cshtml page at the very beginning of the page after the @model TPLOCAL1. Models.FormModel, the code: @Html.ValidationSummary(false,""). We then have the following visual:


- adresse trop courte

Formulaire TP HN avis

- Accueil
- Liste avis

Informations personnelles

Nom	
Prénom	
Sexe	Selectionner un sexe
Adresse	

Step 6: Create a button

This is to create a "Validation" button. Here's what is expected for this button:

- Must generate at least one error message if at least one of the controls defined on the page is KO.
- Must display the validity page correctly completed incase all controls on the page are OK.

For information, we can use the action field of the form tag to call the ValidationForm method of the HomeController for this button. In this method, we will check the data and send data as parameters to the right page with the model.

Validation page

This involves creating a page that will allow the user to consult the information he has just filled in on the form page (Appendix 3).

On this page you will practice:

- To convey data from one page to another

This page must include the same elements as the form page with the following differences:

- In the first line of the body of the page should be the mention "Your form has been taken into account", centered on the page.
- The columns on the right of the tables must be filled in with the information entered by the user on the form page. For information, to display a data of a C# class, in the cshtml page, we define the class at the top of the page as follows:
  @model TPLOCAL1. Models.FormModel (with the relative path TPLOCAL1. Models and FormModel the name of the model class).
  We note with the at sign the C# data in the cshtml pages. Then, to use a data from this template in the cshtml page, we write @Model.DataName.
  The same template will be used for the validation page as for the form page.
- No information should be modifiable by the user.
- The date will be displayed in French format (Day/month/year).

List page

This involves creating a page that will allow the user to consult the information that is present in a file (Appendix 4).

On this page you will practice:

- to create a list from data present in a file
- to condition the formatting of a field according to its value

The development of this page is organized in 3 steps:

Step 1: Display of logo and title

See previously (1st step homepage). Be careful, the title is different from that of the home page.

Step 2: Creating the menu

See previously (2nd step homepage). Attention, the sections are different from those of the home page.

3rd^{me} Step: Creation of the list of reviews

This is to display a list whose information comes from a file made available to you.

This list will be presented in the form of a 3-column table whose data will not be editable. It will have as many rowsas there are people listed in the source file plus the table title row and the column title row. In the first column will appear the names of the people in the file, in the second their first name and in the last column a top indicating whether or not the people have left a review on the training. However, there is an exception for the title that will have to be centered on the 3 columns.

Here are the different fields that the table should contain:

Column 1	**Column 2**	**Column 3**
Title label table "List"		
Title label column "Name"	Title label column "First name"	Wording title column "Notice given"
Name 1	First Name 1	- Notice given in the information of: • "Yes" if the top present in the source file is "Y" • "No" otherwise - Show field by: • green if the top present in the source file is "O" • red otherwise
Name 2	Given name 2	- Notice given in the information of: • "Yes" if the top present in the source file is "Y" • "No" otherwise - Show field by: • green if the top present in the source file is "O" • red otherwise
Name 3	Given name 3	- Notice given in the information of: • "Yes" if the top present in the source file is "Y" • "No" otherwise - Show field by: • green if the top present in the source file is "O" • red otherwise
...	...	...

The data will be retrieved from the DataNotice file.xml using these classes present in Appendix 11. The file is to be placed in the XML File folder of the solution.

To go through the list, the use of a foreach loop is imposed.

Bonus

Display multiple error messages simultaneously

The goal is as follows: on the form page, in case there are several errors, display all the corresponding error messages (instead of displaying only the error message of the first error encountered).

Make the menu and header dynamic

The aim would be:

- Code the menu and header once and call the code on each page.
- To condition the information of the menu and the header to the page by which they are called.

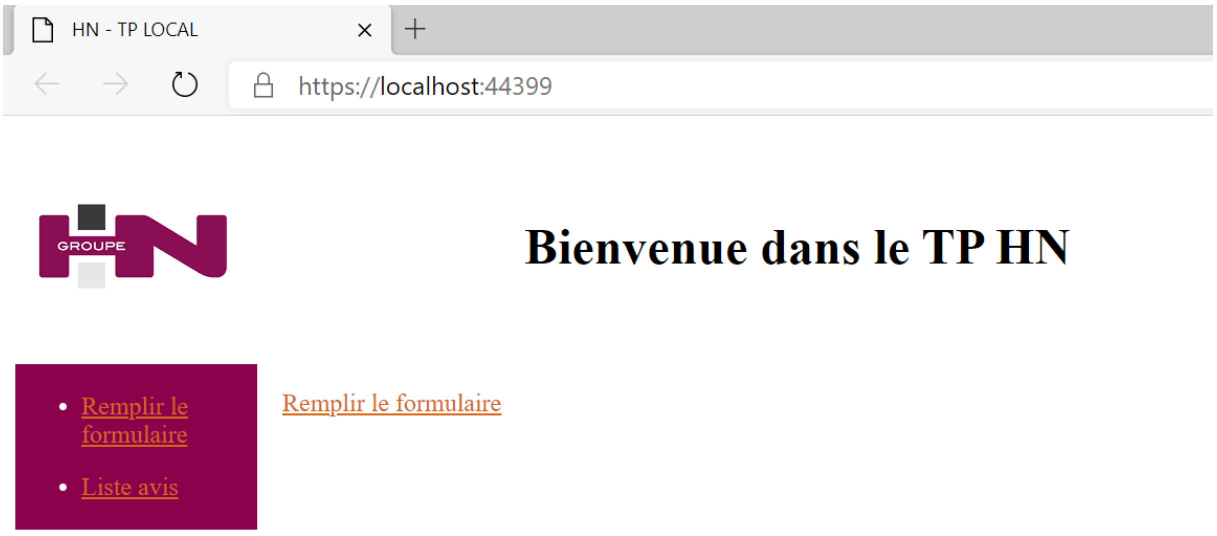
For more information about MVC

You can read openclassroom's course on Learn ASP.NET MVC - OpenClassrooms to see how the controller and routing work. However, this is not necessary for work at the CIC and only serves for your general knowledge.

Annexes

Appendix 1: Visual home page

(Back to paragraph)



HN - TP LOCAL

https://localhost:44399

Bienvenue dans le TP HN

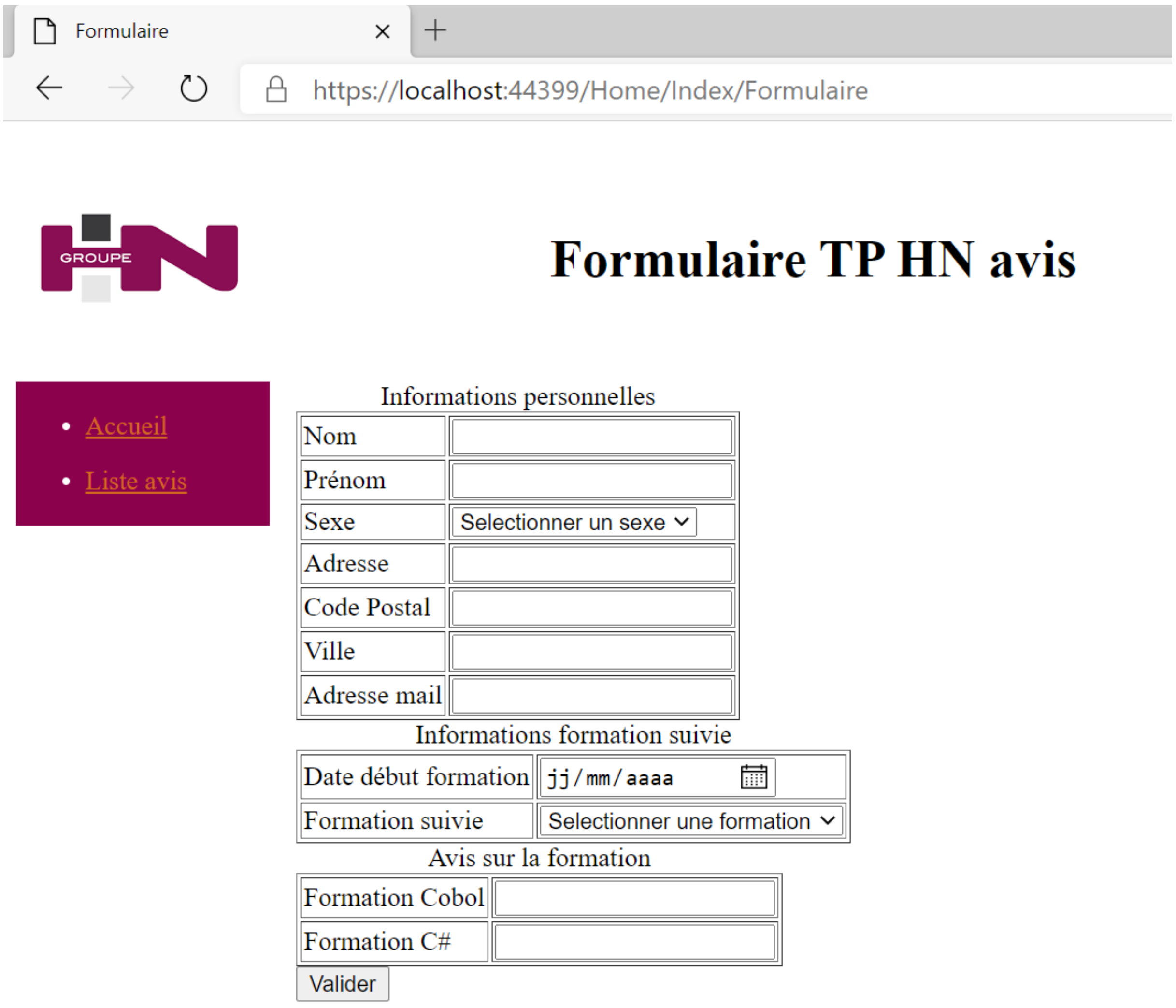
- Remplir le formulaire
- Liste avis

Remplir le formulaire

Appendix 2: Visual form

(Back to paragraph "Principle")

(Back to "Form" paragraph)



Formulaire

https://localhost:44399/Home/Index/Formulaire

Formulaire TP HN avis

- Accueil
- Liste avis

Informations personnelles

Nom	
Prénom	
Sexe	Selectionner un sexe
Adresse	
Code Postal	
Ville	
Adresse mail	

Informations formation suivie

Date début formation	jj/mm/aaaa
Formation suivie	Selectionner une formation

Avis sur la formation

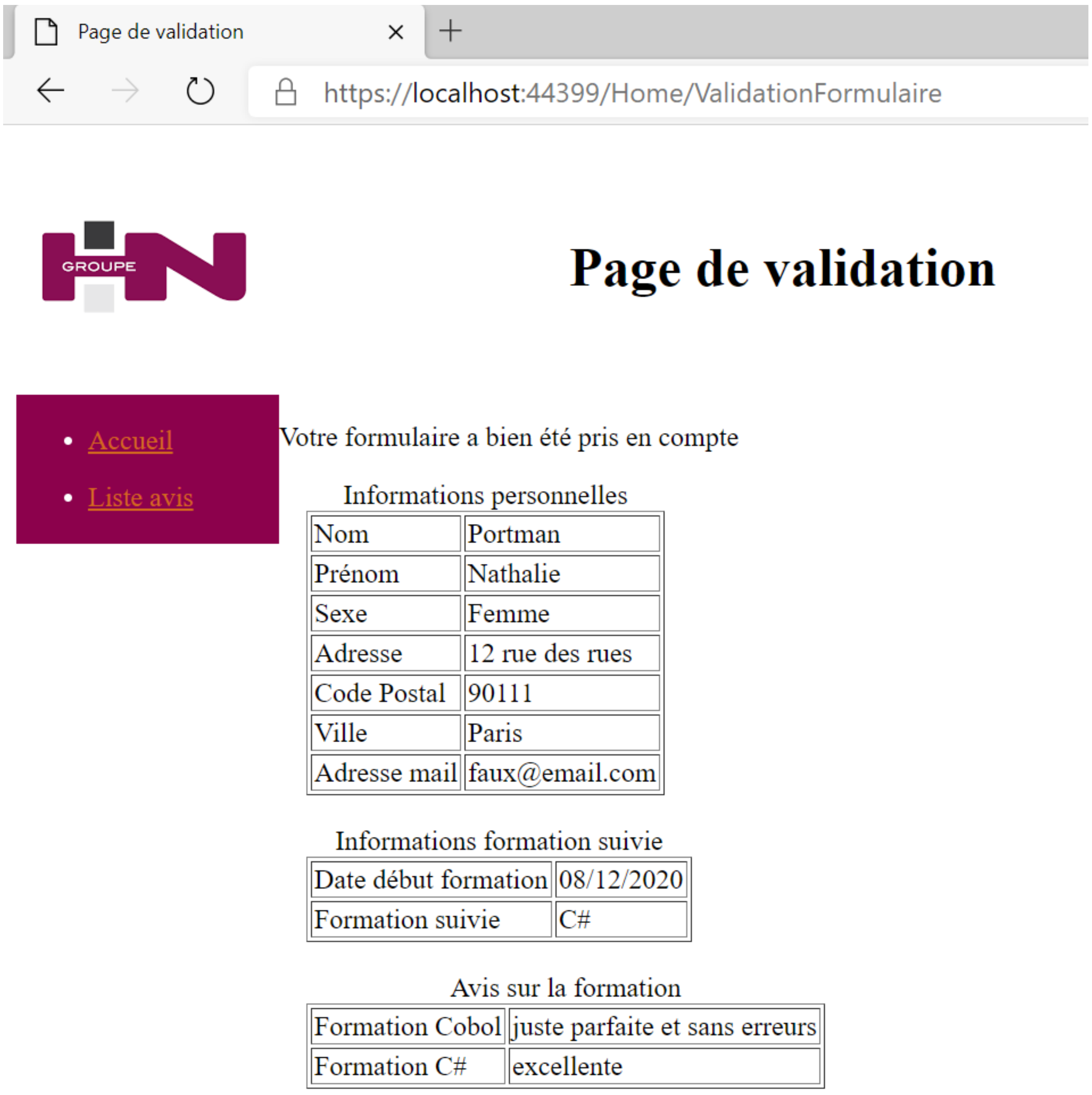
Formation Cobol	
Formation C#	

Valider

Appendix 3: Visual validation page

(Back to paragraph "Principle")

(Back to "Validation page " paragraph)



Page de validation

- Accueil
- Liste avis

Votre formulaire a bien été pris en compte

Informations personnelles

Nom	Portman
Prénom	Nathalie
Sexe	Femme
Adresse	12 rue des rues
Code Postal	90111
Ville	Paris
Adresse mail	faux@email.com

Informations formation suivie

Date début formation	08/12/2020
Formation suivie	C#

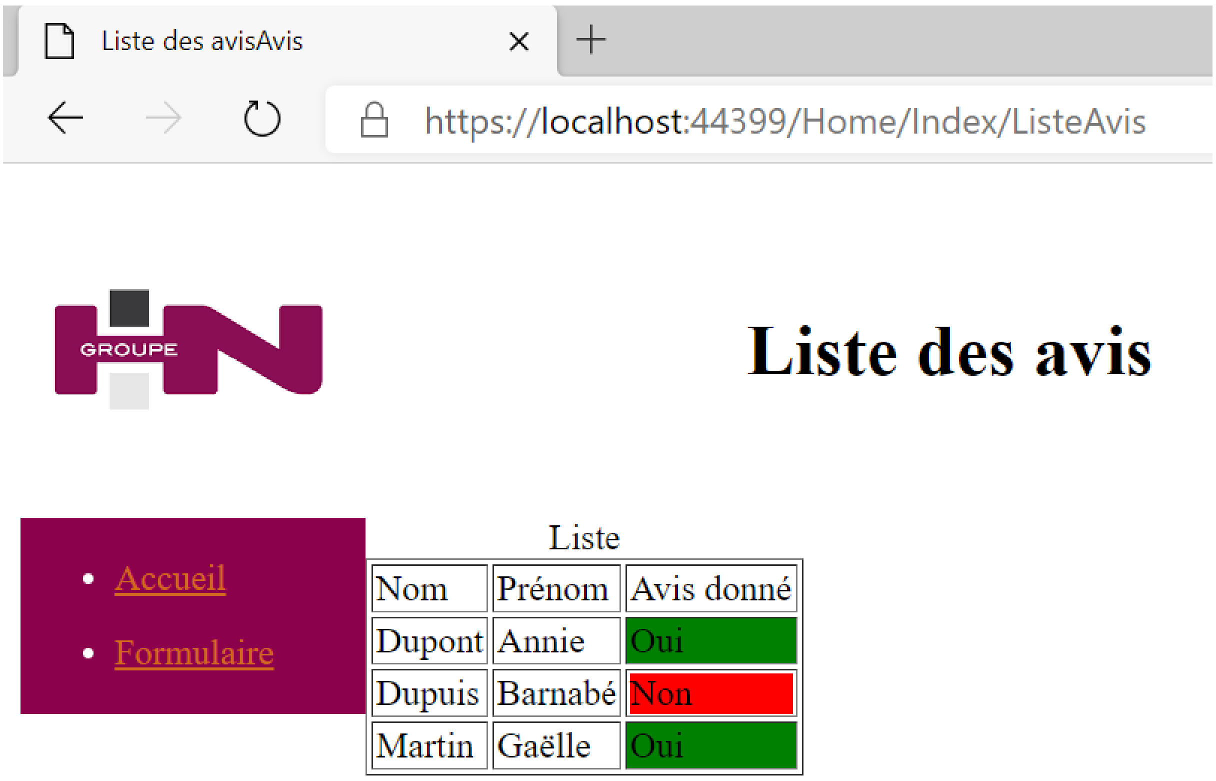
Avis sur la formation

Formation Cobol	juste parfaite et sans erreurs
Formation C#	excellente

Appendix 4: Visual list

(Back to paragraph "Principle")

(Back to "List page" paragraph)



Liste des avisAvis

https://localhost:44399/Home/Index/ListeAvis

Liste des avis

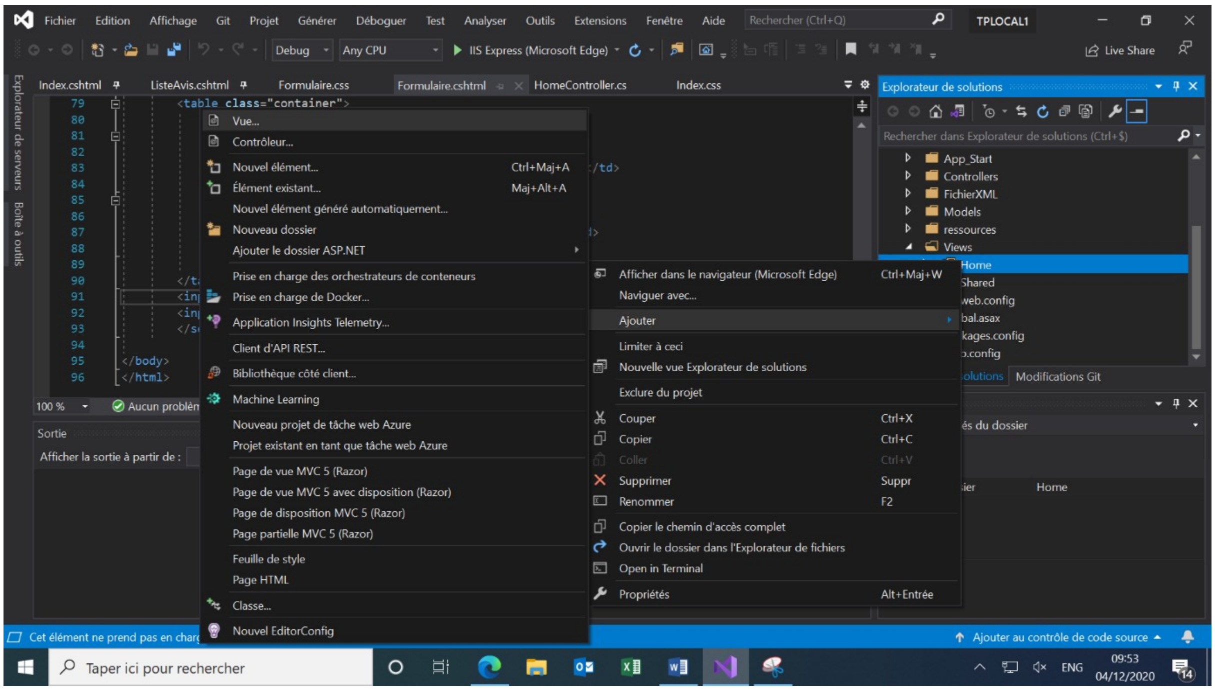
- Accueil
- Formulaire

Liste

Nom	Prénom	Avis donné
Dupont	Annie	Oui
Dupuis	Barnabé	Non
Martin	Gaëlle	Oui

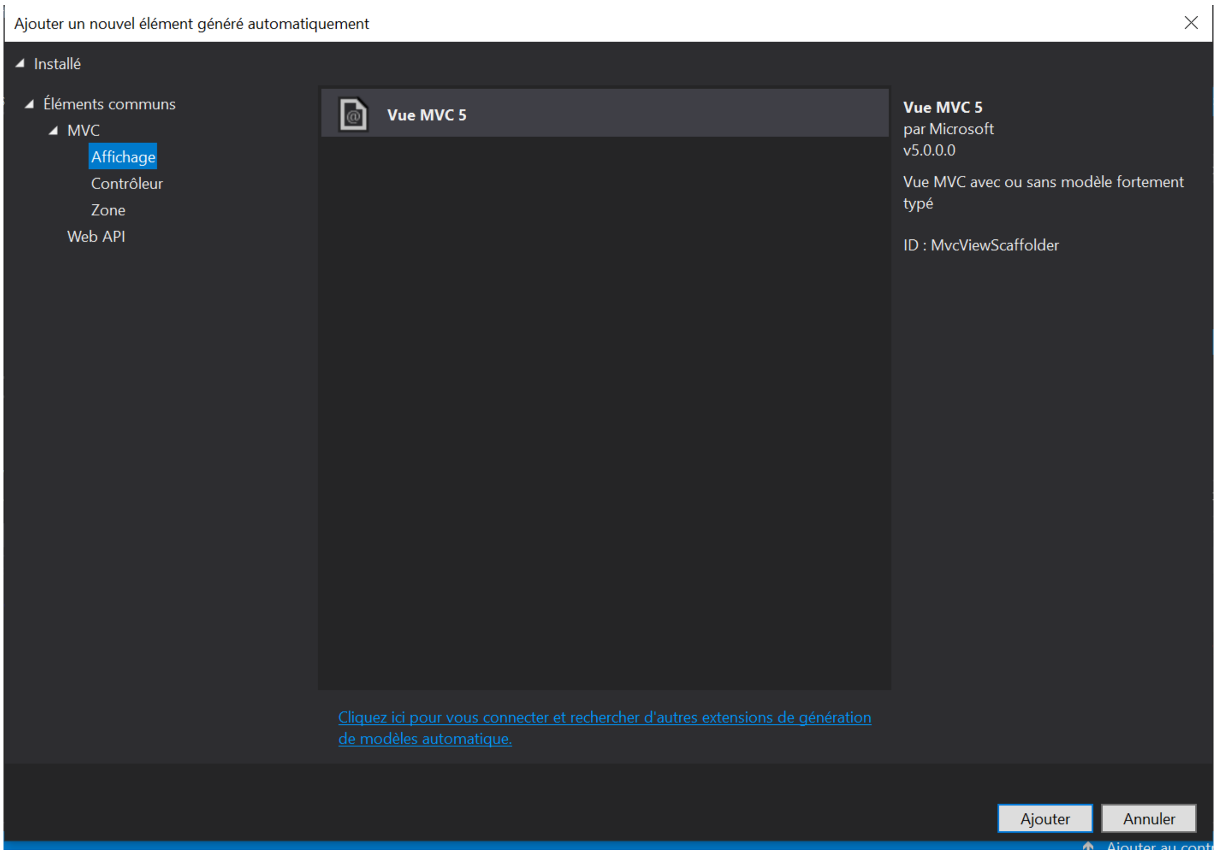
Appendix 5: Solution Explorer

([Back to paragraph](#))


Fichier
Edition
Affichage
Git
Projet
Générer
Déboguer
Test
Analyser
Outils
Extensions
Fenêtre
Aide
Rechercher (Ctrl+Q)
TPLOCAL1
Debug
Any CPU
IIS Express (Microsoft Edge)
Live Share
Index.cshtml
ListeAvis.cshtml
Formulaire.css
Formulaire.cshtml
HomeController.cs
Index.css
Explorateur de solutions
Rechercher dans Explorateur de solutions (Ctrl+$)
App_Start
Controllers
FichierXML
Models
ressources
Views
Home
Shared
web.config
Vue...
Contrôleur...
Nouvel élément... Ctrl+Maj+A
Élément existant... Maj+Alt+A
Nouvel élément généré automatiquement...
Nouveau dossier
Ajouter le dossier ASP.NET
Prise en charge des orchestrateurs de conteneurs
Prise en charge de Docker...
Application Insights Telemetry...
Client d'API REST...
Bibliothèque côté client...
Machine Learning
Nouveau projet de tâche web Azure
Projet existant en tant que tâche web Azure
Page de vue MVC 5 (Razor)
Page de vue MVC 5 avec disposition (Razor)
Page de disposition MVC 5 (Razor)
Page partielle MVC 5 (Razor)
Feuille de style
Page HTML
Classe...
Nouvel EditorConfig
Afficher dans le navigateur (Microsoft Edge) Ctrl+Maj+W
Naviguer avec...
Ajouter
Limiter à ceci
Nouvelle vue Explorateur de solutions
Exclure du projet
Couper Ctrl+X
Copier Ctrl+C
Coller Ctrl+V
Supprimer Suppr
Renommer F2
Copier le chemin d'accès complet
Ouvrir le dossier dans l'Explorateur de fichiers
Open in Terminal
Propriétés Alt+Entrée
Modifications Git
100 %
Aucun problème
Sortie
Afficher la sortie à partir de :
Ajouter au contrôle de code source
Taper ici pour rechercher
ENG
09:53
04/12/2020

```
<table class="container">
                          </td>
</body>
</html>
```

Cet élément ne prend pas en charg

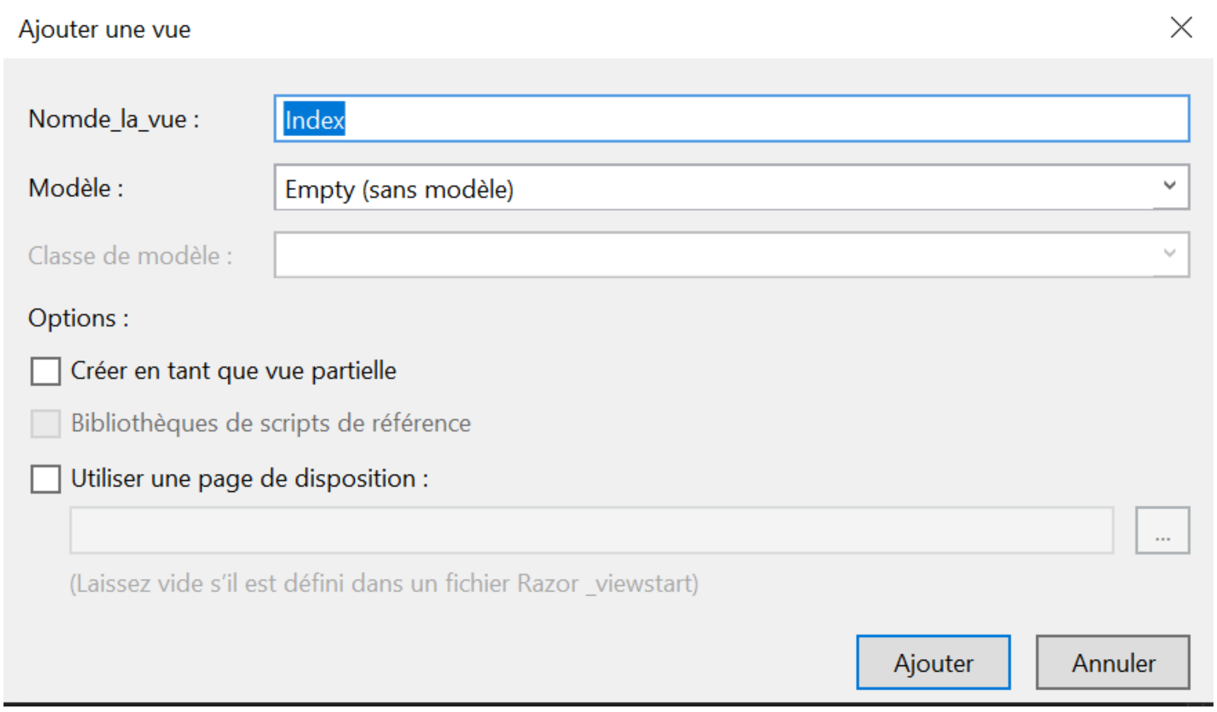


Vue MVC 5
par Microsoft
v5.0.0.0

Vue MVC avec ou sans modèle fortement typé

ID : MvcViewScaffolder

Cliquez ici pour vous connecter et rechercher d'autres extensions de génération de modèles automatique.



Ajouter une vue

Nomde_la_vue : Index

Modèle : Empty (sans modèle)

Classe de modèle :

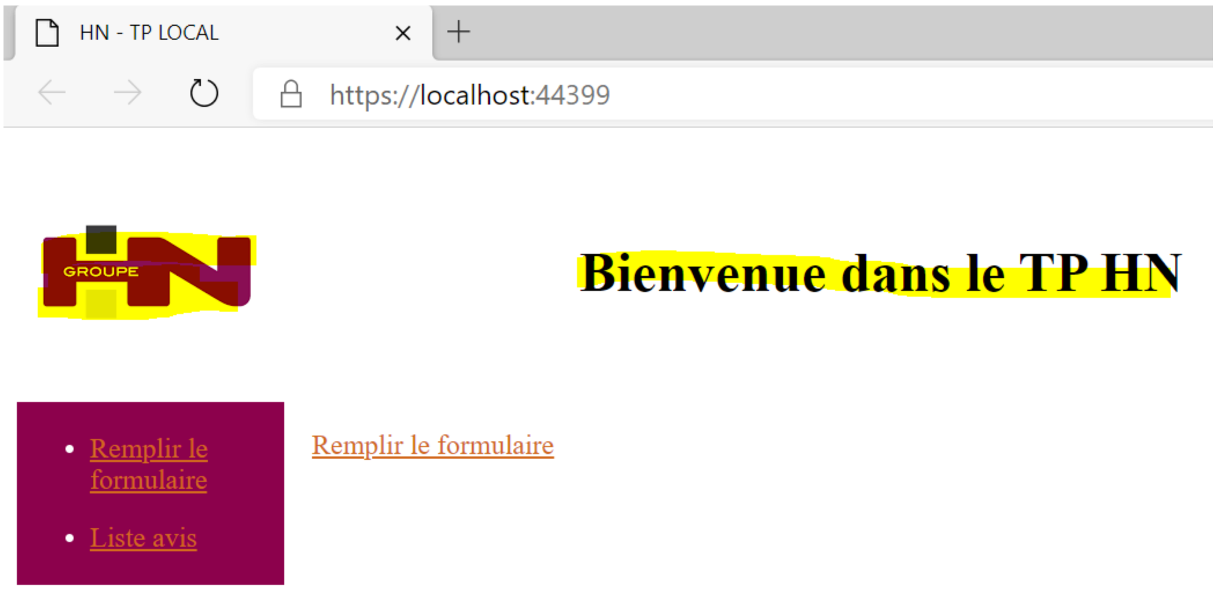
Options :

- [ ] Créer en tant que vue partielle
- [ ] Bibliothèques de scripts de référence
- [ ] Utiliser une page de disposition :

(Laissez vide s'il est défini dans un fichier Razor _viewstart)

Appendix 6: Display of Logo and Title

(Back to paragraph)



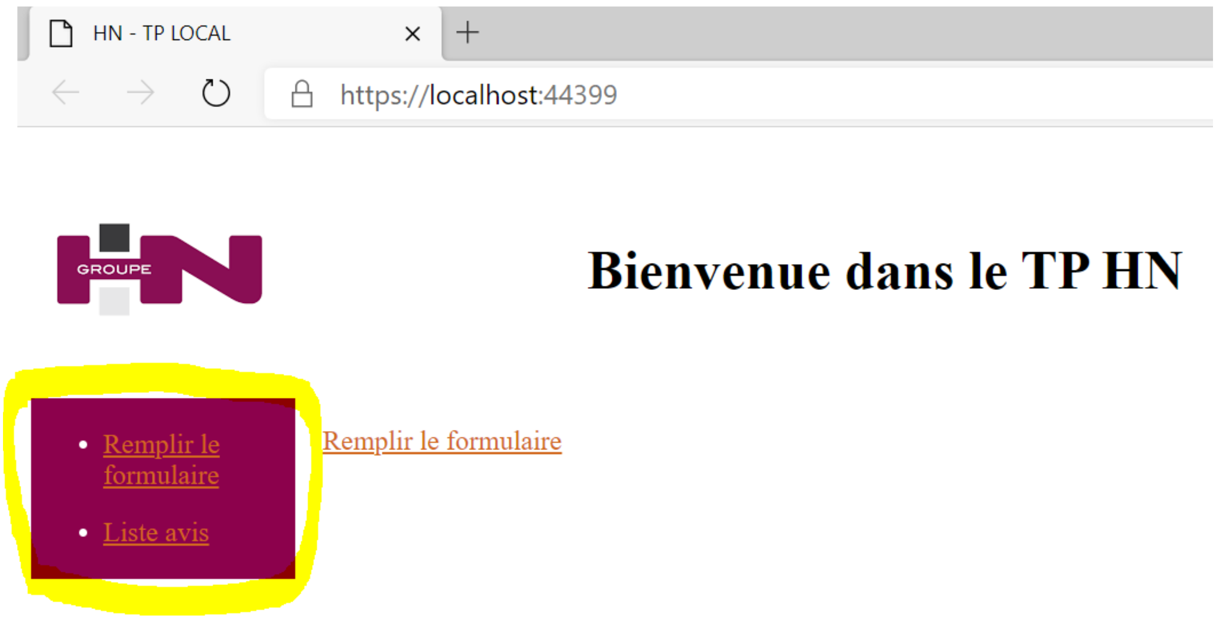
Bienvenue dans le TP HN

- Remplir le formulaire
- Liste avis

Remplir le formulaire

Appendix 7: Creating the menu

(Back to paragraph)



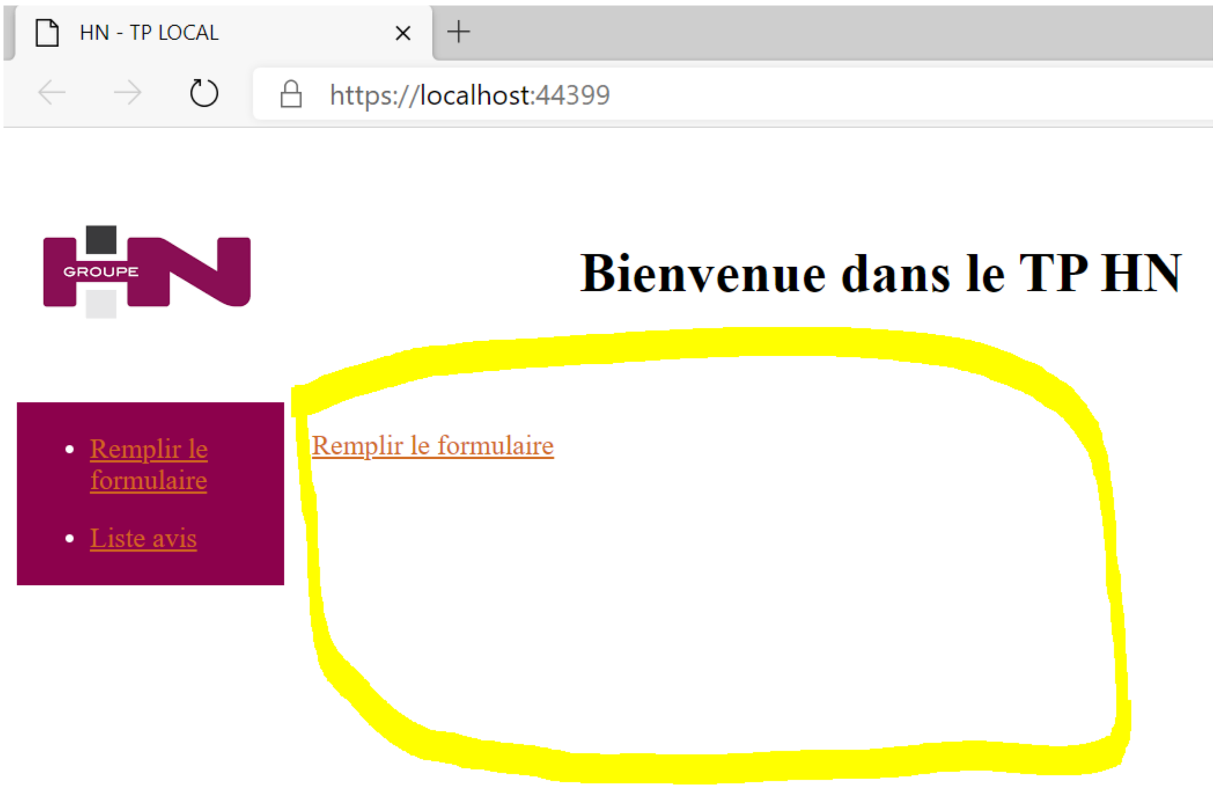
Bienvenue dans le TP HN

- Remplir le formulaire
- Liste avis

Remplir le formulaire

Appendix 8: Page Body Creation

(Back to paragraph)



HN - TP LOCAL

https://localhost:44399

Bienvenue dans le TP HN

- Remplir le formulaire
- Liste avis

Remplir le formulaire

Appendix 9: Index. CSHTML

(Back to paragraph)

```
<! DOCTYPE html>
<html>
        <head>
                <title>HN - TP LOCAL</title>
                <link rel="stylesheet" href="~/resources/Index.css"/>
        </head>
        <body>
                <header>
                        <img src="~/resources/GroupHN.png" class="image"  alt="GroupHN" />
                        <h1>Welcome to TP HN</h1>
                </header>

                <nav>
                        <ul>
                                <li><a href="/Home/Index/Form">Fill out form</a></li>
                                <li><a href="/Home/Index/AvisList">Avis List</a></li>
                        </ul>
                </nav>

                <section>
                        <p>
                                <a href="/Home/Index/Form">Fill out form </a>
                        </p>
                /section>
        </body>
</html>
```

Appendix 10: Index.css

(Back to paragraph)

```css
.image {
   width: 150px;
   height: 150px;
   display: inline-block;
   vertical-align: middle;
}

h1 {
   margin-right: 170px;
   margin-left: 170px;
   text-align: center;
   display: inline-block;
   vertical-align: middle;
}

nav {
   float: left;
   width: 150px;
   border: 2px solid #8c014c;
   background-color: #8c014c;
   color: white;
}

li {
   margin-top: 10px;
   margin-bottom: 15px;
}

a {
   color: chocolate;
}

section {
margin-left: 170px;
}
```

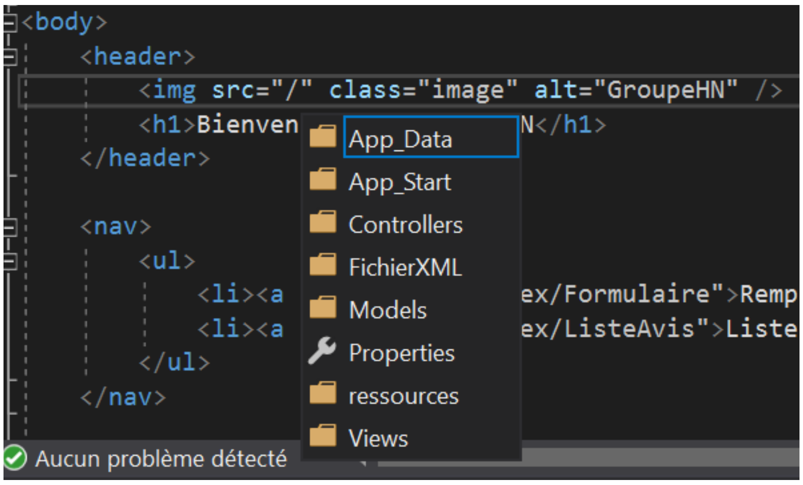
Appendix 11: Retrieval of the List of Notices

(Back to paragraph)

```
using System;
using System.Collections.Generic;
using System.IO;
using System.Linq;
using System.Web;
using System.Xml;

namespace  TPLOCAL
{
		public class ListReviews
		{
				///  <summary>
				///  Function to retrieve the list of reviews contained in an XML file
				///  </summary>
				///  <param name="file">file path</param>
				public List<Reviews> GetAvis(string file)
				{
						//  Instantiate the list as empty
						List<Reviews> ListReviews = new List<Reviews>();
						//  Creating an XMLDocument object  to retrieve data from the physical file
						XmlDocument xmlDoc = new XmlDocument();
						//  Reading the file from a StreamReader object
						StreamReader streamDoc = new StreamReader(file);
						string dataXml = streamDoc.ReadToEnd();
						//  Loading Data into the XmlDocument
						xmlDoc.LoadXml(dataXml);

						//  Retrieving the nodes to  pass them as a Notice object and then adding them to the
						'ListNotices' list
						//  We loop on each node of  type XmlNode having for path "root/row" (cf structure of
						the xml file)
						//  The SelectNodes method  retrieves all nodes with the specified path
						foreach  (XmlNode  node in xmlDoc.SelectNodes("root/row"))
						{
								//  Retrieving data in child nodes
								string name = node["Name"]. InnerText;
								string first name = node["Firstname"]. InnerText;
								string avisdonne = node["Avis"]. InnerText;

								//  Creating the Notice Object to Add to the Results List
								Avis avis = new Avis
								{
								   Name = name,
								   First name = firstname,
								   AvisDonne = avisdonne
								};

								Adding the object to the list
								listNotice.Add(notice);
						}

						//  The list formed by the treatment is returned to the calling method
						return listReviews;
				}
```

```
        }

        // . :Info:.
        //  This class can be checked out in a new C# page but as part of the TP it can be left in the same page.
        //  It is necessary to avoid as much as possible to have the same page with several classes inside.
        //  Even if it works, it can complicate code readability and ultimately maintenance.
        /// <summary>
        /// Object grouping data related to reviews.
        /// \n Can be modified
        /// </summary>
        public class Reviews
        {
        /// <summary>
        /// Surname
        /// </summary>
        public string Name { get;  set; }
        /// <summary>
        /// Forename
        /// </summary>
        public string Prenom { get;  set; }
        /// <summary>
        /// Notice given (Possible values : O or N)
        /// </summary>
        public string AvisDonne { get;  set; }
        }
}
```

Appendix 12: autocompletion of the path of a CRS:



```
<body>
    <header>
        <img src="/" class="image" alt="GroupeHN" />
        <h1>Bienven                    N</h1>
    </header>

    <nav>
        <ul>
            <li><a                     ex/Formulaire">Remp
            <li><a                     ex/ListeAvis">Liste
        </ul>
    </nav>
```