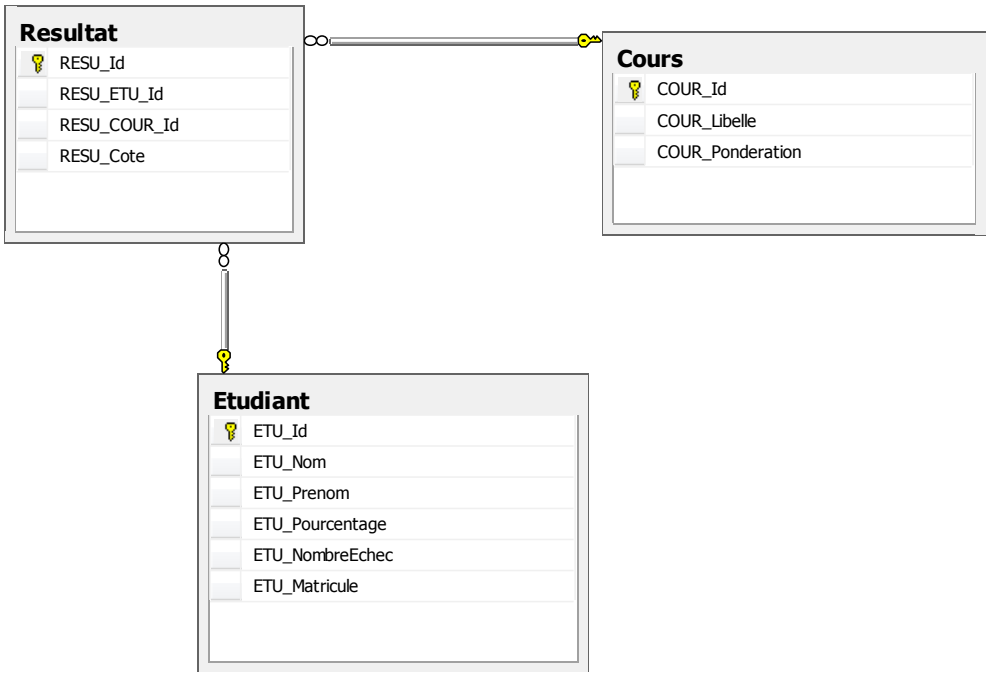
Divers points pour réaliser votre projet

1. Accès DB via EntityFramework (EF)

Si vous souhaitez accéder à la base de données via EntityFramework, il faudra implémenter une couche « Business » qui fera le lien entre les Entities (classes d'EntityFramework mappées avec les tables de la base de données) et les Models qui vont être exploités par vos Controller.
En effet, ces Entities sont générées par EF, et on ne peut y ajouter des DataAnnotations sans risquer de les perdre (par exemple lors d'une mise à jour de la BD → Model EF)

Exemple avec DB EtudiantCours :



Avec EF, une classe Cours est créée :

```
//------------------------------------------------------------------------------
// <auto-generated>
//     This code was generated from a template.
//
//     Manual changes to this file may cause unexpected behavior in your application.
//     Manual changes to this file will be overwritten if the code is regenerated.
// </auto-generated>
//------------------------------------------------------------------------------
namespace WebAppDbAccess.DAL
{
    public partial class Cours
    {
        public Cours()
        {
            this.Resultat = new HashSet<Resultat>();
        }
        public int COUR_Id { get; set; }
        public string COUR_Libelle { get; set; }
        public int COUR_Ponderation { get; set; }
        public virtual ICollection<Resultat> Resultat { get; set; }

    }
}
```

Notez les commentaires « `Manual changes to this ...` ».
En test, ajouter des DataAnnotations, modifier DB, update Entities, et vérifier le résultat.

Une solution est de créer une classe qui utilise EF et renvoie les Models.
Créons un CoursModels avec quelques DataAnnotations :

```
public class CoursModels
{
    public int COUR_Id { get; set; }

    [Required]
    [StringLength(50)]
    [Display(Name = "Libellé")]
    public string COUR_Libelle { get; set; }

    [Range(1,int.MaxValue)]
    [Display(Name = "Pondération")]
    public int COUR_Ponderation { get; set; }
}
```

Une classe qui appelle EF et renvoie le CoursModels :

```
public static class EtudiantCoursBL
{
    public static List<CoursModels> SelectAllCours()
    {
        List<CoursModels> rtn = new List<CoursModels>();
        using (DAL.EtudiantCoursEntities context = new DAL.EtudiantCoursEntities())
        {
            foreach (var item in context.Cours)
            {
                rtn.Add(new CoursModels
                {
                    COUR_Id = item.COUR_Id,
                    COUR_Libelle = item.COUR_Libelle,
                    COUR_Ponderation = item.COUR_Ponderation
                });
            }
        }
        return rtn;
    }
}
```

Et un Controller Cours avec Index qui renvoie une liste de CoursModels obtenue via la BL :

```
public ActionResult Index()
{
    return View(BL.EtudiantCoursBL.SelectAllCours());
}
```

Remarque, en utilisant Linq, on peut se passer de cette couche de mapping, il suffit d'implémenter l'appel de Stored Procedure qui renvoie les Models :

```
public class MainDAL : DataContext
{
    public MainDAL()
        : base(ConfigurationManager.ConnectionStrings["DefaultConnection"].ConnectionString)
    {
    }

    [Function(Name = "[dbo].[CoursSelectAll]")]
    public ISingleResult<CoursModels> CoursSelectAll()
    {
        IExecuteResult result = ExecuteMethodCall(
        this, ((MethodInfo)(MethodInfo.GetCurrentMethod())));
        return ((ISingleResult<CoursModels>)(result.ReturnValue));
    }
}
```

2. Poster une liste d'objet

Modifier la vue liste des cours

```
@using (Html.BeginForm())
{
    for (int i = 0; i < Model.Count(); i++)
```

```
    {
        <tr>
            <td>
                @Html.TextBoxFor(x => Model.ToList()[i].COUR_Id)
            </td>
            <td>
                @Html.TextBoxFor(x => Model.ToList()[i].COUR_Libelle)
            </td>
            <td>
                @Html.TextBoxFor(x => Model.ToList()[i].COUR_Ponderation)
            </td>
            <td>
                @Html.ActionLink("Edit", "Edit", new { /* id=item.PrimaryKey */ }) |
                @Html.ActionLink("Details", "Details", new { /* id=item.PrimaryKey */ }) |
                @Html.ActionLink("Delete", "Delete", new { /* id=item.PrimaryKey */ })
            </td>
        </tr>
    }
    <tr>
        <td colspan="4">
            <input type="submit" value="Save" />
        </td>
    </tr>
}
```

Ajouter l'action POST et vérifier en DEBUG que le model est bien complété

```
        [HttpPost]
        public ActionResult Index(List<CoursModels> model)
        {
            return View(BL.CoursBL.GetAllCours());
        }
```

3. jQuery : + / -

```
     <td>
        <a href="#" onclick="plus(@i)">Plus</a>
    </td>

    function plus(i) {
        // récupérer l'élément nommé [i].COUR_Ponderation
        var el = $("[name='[" + i + "].COUR_Ponderation']");
        // récupérer la valeur
        var ponder = el.val();
        // l'augmenter
        ponder++;
        // affecter la nouvelle valeur
        el.val(ponder);
    }
```

4. Liste déroulante chargée avec ajax, liée

Créer Action Edit d'un cours

Dans l'action load d'une liste de statut :

```
List<Statut> lstStatut = new List<Statut>();
lstStatut.Add(new Statut { Id = 0, Libelle = "------" });
lstStatut.Add(new Statut { Id = 1, Libelle = "Tous" });
lstStatut.Add(new Statut { Id = 2, Libelle = "Réussi" });
lstStatut.Add(new Statut { Id = 3, Libelle = "Echec" });

ViewBag.ListStatut = new SelectList(lstStatut, "Id", "Libelle", 0);
```

Et d'une liste d'étudiant vide

```
ViewBag.ListEtudiant = new SelectList(new List<EtudiantModels>(), "ETU_Id", "ETU_Nom");
```

Dans la vue, on définit et charge la liste statut comme suit :

```
@Html.DropDownList("Statut", (SelectList)ViewBag.ListStatut, new { @class = "form-control" })
```

Lorsqu'elle est modifiée, on souhaite charger la liste étudiant.

On écrira donc un script sur l'évènement change qui va récupérer la liste des étudiants selon le statut sélectionné :

```
<script>
    $('#Statut').change(function () {
        $.ajax({
            type : 'GET',
            datatype : 'json',
            url: '@Url.Action("GetEtudiant","Cours")',
            data: { courid : @Model.COUR_Id, statut : $('#Statut').val() },
            success :
                function(response){
                    if (response.result == "OK")
                    {
                        var items;
                        $('#Etudiant').html("");
                        $.each(response.etudiant, function(k,v){
                            items +=
                                        "<option value='" + v.ETU_Id + "'>" +
                                        v.ETU_Nom + " " + v.RESU_Cote + "</option>";
                        })
                        $('#Etudiant').html(items);
                    }
                }
        })
    })
</script>
```

On effectue un appel ajax qui va renvoyer un tableau JSON sur base du COUR_Id et du Statut sélectionné.

Il faut évidemment créer l'action GetEtudiant associée :

Et avant cela, on crée le model Etudiant :

```
    public class EtudiantModels
    {
        public int ETU_Id { get; set; }
        public string ETU_Nom { get; set; }
        public decimal ETU_Pourcentage { get; set; }
        public int ETU_NombreEchec { get; set; }
        public string ETU_Matricule { get; set; }
        public decimal RESU_Cote { get; set; }
    }
```

Puis l'action GET :

```
        [HttpGet]
        public JsonResult GetEtudiant(int courid, int statut)
        {
            List<EtudiantModels> lstEtu;
            using (MainDAL dal = new MainDAL())
            {
                lstEtu = dal.EtudiantSelectFromCourId(courid).ToList();
            }
            switch (statut)
            {
                case 0:
                    lstEtu = new List<EtudiantModels>();
                    break;
                case 2:
                    lstEtu = lstEtu.FindAll(x => x.RESU_Cote >= 10M);
                    break;
                case 3:
                    lstEtu = lstEtu.FindAll(x => x.RESU_Cote < 10M);
                    break;
            }

            return Json(new { result = "OK", etudiant = lstEtu }, JsonRequestBehavior.AllowGet);
        }
```

Notez ceci :

- l'action renvoie un objet JsonResult càd un objet au format Json
- le return est créé via la méthode Json qui renvoie un objet JsonResult
- le paramètre `JsonRequestBehavior.AllowGet` spécifie si l'action peut être appelée en GET

Qu'est-ce que JSON ?

Une syntaxe connue du JavaScript pour stocker et échanger des données.

Exemple :

```
{"employees":[
    {"firstName":"John", "lastName":"Doe"},
    {"firstName":"Anna", "lastName":"Smith"},
    {"firstName":"Peter", "lastName":"Jones"}
]}
```

Un objet JSON est facilement utilisable en JavaScript :

```
<script>
   var text = '{"name":"John Johnson","street":"Oslo West 16","phone":"555 1234567"}';
   var obj = JSON.parse(text);
   document.getElementById("demo").innerHTML = obj.name + "<br>" + obj.street + "<br>" + obj.phone;
</script>
```

Une donnée JSON est écrite sous la forme « nom » / « valeur » : `"firstName":"John"`

Un objet JSON est écrit entre accolade : `{"firstName":"John", "lastName":"Doe"}`

Si l'objet JSON contient une liste d'objet JSON, on l'écrira entre crochet :

```
"employees":[
    {"firstName":"John", "lastName":"Doe"},
    {"firstName":"Anna", "lastName":"Smith"},
    {"firstName":"Peter","lastName":"Jones"}
]
```

Il est aisé d'utiliser un objet JSON en JavaScript :

```
var employees = [
    {"firstName":"John", "lastName":"Doe"},
    {"firstName":"Anna", "lastName":"Smith"},
    {"firstName":"Peter","lastName": "Jones"}
];
var firstEmployee = employees[0].firstName + " " + employees[0].lastName;
```

C'est ce qui est effectué dans fonction change :

```
var items;
$.each(response.etudiant, function(k,v){
     items += "<option value='" + v.ETU_Id + "'>" + v.ETU_Nom + " " + v.RESU_Cote + "</option>";
})
```

L'appel ajax a donc renvoyé un objet JSON avec 2 données (result et etudiant) :

```
return Json(new { result = "OK", etudiant = lstEtu }, JsonRequestBehavior.AllowGet);
```

La donnée etudiant est une liste d'objet JSON, on peut donc la parcourir avec la fonction JQuery each qui prend en paramètre le tableau à parcourir et la fonction qui va être exécutée pour chaque élément du tableau.

Cette fonction doit avoir la signature (key, value) :

- key : l'index ou la clé de l'élément du tableau
- value : la valeur de l'élément du tableau

Dans ce cas, un élément est un objet JSON qui est la conversion d'un objet de la classe EtudiantModels.

```
public class EtudiantModels
{
    public int ETU_Id { get; set; }
    public string ETU_Nom { get; set; }
    public decimal ETU_Pourcentage { get; set; }
    public int ETU_NombreEchec { get; set; }
    public string ETU_Matricule { get; set; }
    public decimal RESU_Cote { get; set; }
}
{"ETUD_Id":"125", "ETU_Nom":"Hecquet", "ETU_Pourcentage":50.0, "ETU_NombreEchec":10,
"ETU_Matricule":"PSR08112", "RESU_Code":8.5}
```

4. Poster un fichier

Dans la vue, pour que le post d'un fichier fonctionne, il faut absolument une balise « form » avec l'attribut « enctype = "multipart/form-data" » :

```
@using (Html.BeginForm("Article", "Home", FormMethod.Post, new { enctype = "multipart/form-data" }))
```

Et un input type « file » :

```
<input type="file" name="monfichier" value="coucou" />
```

Dans le controller on ajoutera un paramètre « HttpPostedFile » du même nom que celui de l'input type « file » :

```
[HttpPost]
public ActionResult Cours(CoursModel model, HttpPostedFileBase monfichier)
{
    if (file != null && file.ContentLength > 0)
    {
        string path = Path.Combine(Server.MapPath("~/Img"), model.artId.ToString() + ".jpg");
        file.SaveAs(path);
        // ajouter le cours
    }
    return RedirectToAction(nameof(Index));
}
```

On enregistrera le fichier dans un répertoire (dans cet exemple : « Img »).

Pour afficher la photo :

```
<img src="~/Img/@(Model.ToList()[i].COUR_Id).jpg" height="40" />
```

La suite au prochain cours ….

5. Gestion des erreurs

```
<system.web>
  <customErrors mode="On" />
</system.web>
```

Redirection vers Shared.Error.cshtml + log dans DB par exemple …

6. Session

```
Session["ItemSession"] = _lst;
(List<ObjetModel>) Session["ItemSession"]
```

7. DateTimePicker

8. DataTable