**Laboratoire R15**

Génération d’un projet BD-First

Une image contenant texte, capture d’écran, conception

Description générée automatiquement

Mise en situation : Imaginez une BD qui permet de réunir les informations sur plusieurs types d’employés. Chaque sous-type d’employé (Programeur, Producteur, Artiste, Vendeur, etc.) « hérite » (généralisation) de la table Employe.

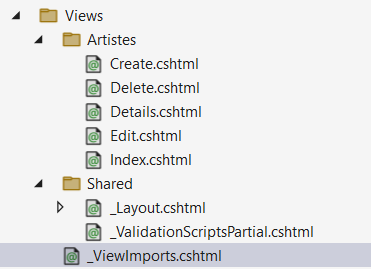
Pour simplifier les choses et rendre ce laboratoire moins répétitif, nous nous concentrerons seulement sur les tables Employe et Artiste et nous ne noterons que quelques informations sur les employés.

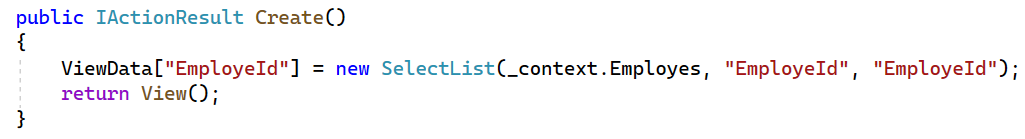
Une image contenant texte, capture d’écran, conception

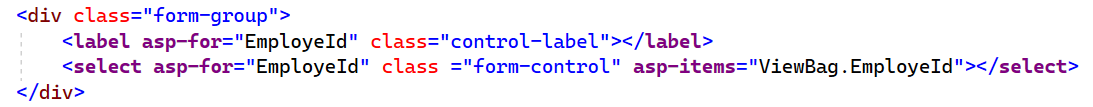
Description générée automatiquement

**Étape 1 - Une génération paresseuse 🦥🛌**

Puisqu’on part d’une BD existante, la génération du projet sera plutôt simple initialement. Suivez les étapes ci-dessous. (Ça suit assez strictement les diapos)

* Exécutez le script ***Script\_bd\_R15\_Employes\_create.sql*** dans SSMS. Cela créera la base de données **R15\_Employes**. Pour ce labo il n’y aura pas d’autres manipulations à faire du côté de la BD.
  + Remarquez : Y a-t-il des vues ? des procédures ? des déclencheurs ?
* **4-10**  Créez le projet ASP.NET Core avec Visual Studio appelé **R15\_Labo**. (Vous utiliserez ce projet pour la rencontre R15 et R16). Préparez les dossiers **Controllers, Models** et **Views**, copiez du dossier **Pages** les fichiers qui nous intéressent dans **Views** puis supprimez le dossier **Pages**. (Le dossier **Data** sera créé automatiquement plus tard)
* **11-12**  Obtenez le string de connexion pour la BD que vous venez de créer et ajoutez-le dans appsettings.json.
* **13-18**  Générez les Models et le DbContext. (N’oubliez pas que si vous n’avez pas build votre projet depuis l’installation des packages EntityFramework, ça ne marchera pas)
  + N’oubliez pas de configurer le DbContext dans Program.cs.
* **19-20**  Générez un contrôleur et les vues pour le Model « **Artiste** ».
  + Assurez-vous que le script de validation et les tags helpers fonctionneront dans vos vues.
* **21**  Configurez le routage dans Program.cs. La vue à afficher par défaut est Artistes/Index
* Assurez-vous que le projet fonctionne :
  + Il faut être capable d’accéder aux cinq vues générées pour les Artistes. (Index, Create, Edit, Details et Delete)
  + N’utilisez pas d’opérations CRUD sur les artistes pour le moment ! (**N’activez pas les formulaires disponibles dans Create, Edit et Delete**) À cause de la relation d’héritage entre **Employe** et **Artiste**, les vues auto-générées ne sont pas tout à fait adaptées à nos données.
* Jetez un coup d’oeil à la vue **Create.cshtml**
  + Créer un « artiste » se résume à choisir une spécialité et choisir un **EmployeID**. Remarquez que la vue ne nous laisse que choisir des **EmployeID** qui existent dans la BD ! C’est une astuce très intéressante pour s’assurer de ne pas violer la contrainte étrangère FK.
    - Le seul bémol est que les IDs ne sont pas très parlants et ne nous permettent pas de faire un choix très éclairé... si on voyait les noms des employés ce serait mieux. (Nous améliorerons la manière de créer un artiste plus tard)
    - On peut voir comment l’action **Create** du contrôleur **Artistes** fait pour envoyer cette liste d’IDs à la vue (Usage d’un objet SelectList et du ViewData/ViewBag) :





**Étape 2 - Vues, procédures et déclencheurs à la rescousse 🦸‍♂️**

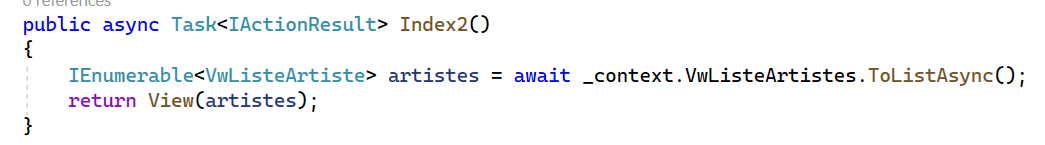
Le projet Web ne sait pas qu’un Artiste est aussi un Employe. Ces deux entités sont donc complètement indépendantes dans les contrôleurs et les vues.

Nous allons améliorer certaines fonctionnalités pour que la gestion des employés artistes soit meilleure :

1. **L’affichage des données des employés artistes. (Grâce à une vue)**
2. **La création d’un employé artiste. (Grâce à une procédure)**
3. **La suppression d’un employé artiste. (Grâce à un déclencheur)**

## **1) Ajout d’une action pour voir toutes les infos d’un artiste (Grâce à une vue) :**

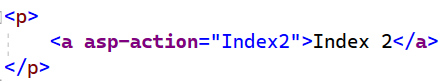
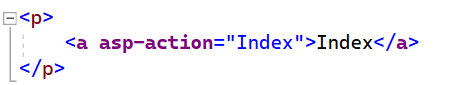
* + **31-32**  Ajoutez une action dans le contrôleur Artistes qui retourne les données de la vue VwListeArtiste à une vue :



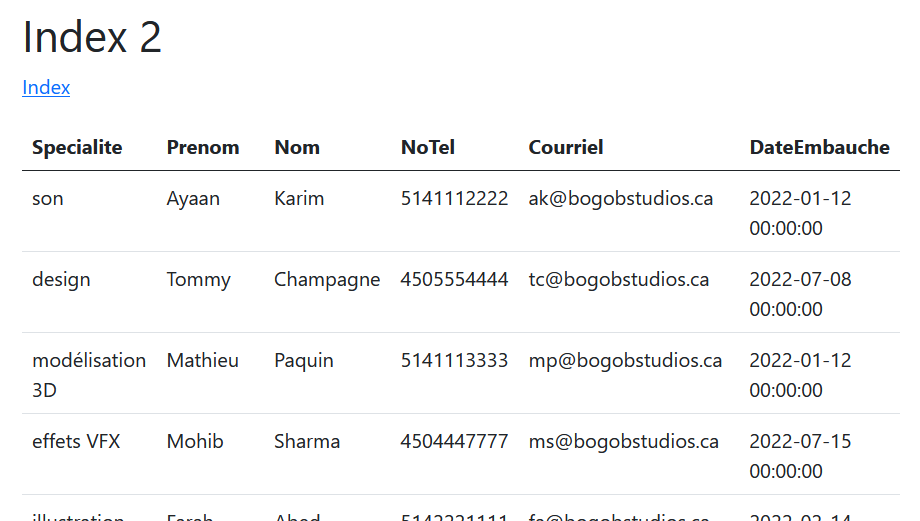
* + Nous allons devoir créer une vue Razor pour afficher la **liste de VwListeArtiste**. Copiez-collez la vue Index.cshtml existante pour les artistes, mais renommez-la avec le nom de votre nouvelle action.
  + Affichez la liste de VwListeArtiste dans cette vue. (Cela demandera quelques modifications et ajouts pour tout afficher)



* N’hésitez pas à ajouter des boutons pour pouvoir naviguer entre Index et Index2 :

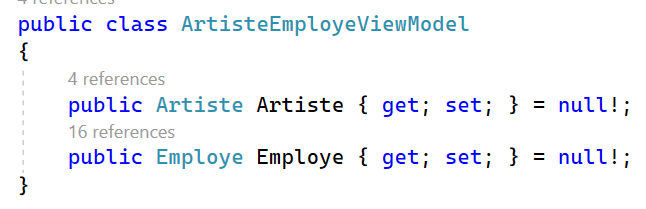


* + Dans votre nouvelle vue Razor, nous devrions voir ceci, au minimum :



## **2) La création d’un employé artiste. (Grâce à une procédure)**

* En vue de modifier l’action Create, créons un ViewModel qui encapsule les données d’un artiste et d’un employé (Sans constructeur svp !) :



Nous aurons besoin de ce ViewModel pour modifier Create.

(Vous pouvez mettre tous les ViewModel dans un dossier ViewModels)

* + Modifier la vue Create :
    - AVANT de changer cette vue, nous allons nous créez une vue auto-générée de type de type « Create » pour le Model « Employe » :
      * Cela nous permettra d’avoir le code que nous allons ajouter dans notre vue modifiée pour l’artiste, soit tout ce qui est lié à un employé.

Une image contenant texte, capture d’écran, Police, nombre

Description générée automatiquement

Une image contenant texte, capture d’écran, nombre, Police

Description générée automatiquement

Nous pourrons prendre dans la vue TemplEmploye tout ce qui est lié à la création d’un Employé.

* + - MODIFICATIONS de la vue CREATE pour un artiste :
      * Changez le @model de la vue par un ArtisteEmployeViewModel
      * Retirez tout le « form-group » d’**EmployeId**. (Car on en aura plus besoin)
      * Comme le @model a changé, on doit mettre « Artiste.Specialite » au lieu de juste « Specialite » dans les Tag Helpers : (La propriété Specialite n’est plus à la racine du @model de la Vue, mais plutôt dans la section « Artiste » du ViewModel utilisé désormais)

Une image contenant texte, Police, ligne, capture d’écran

Description générée automatiquement

* + - * Maintenant... on doit ajouter des « form-group » pour toutes les propriétés d’un employé ! (Prenom, Nom, NoTel et Courriel) ... Vous pouvez les prendre dans la vue TempEmploye q’on a fait auparavant.

Une image contenant texte, capture d’écran, Police

Description générée automatiquement

* + - Copiez-collez les « form-group » des quatre propriétés dont on a besoin dans la vue Create.cshtml des artistes ! (Vous aurez Employe. dans les Tag Helpers. C’est OK) Vous pouvez supprimer la vue TempEmploye après.
    - Votre vue devait ressembler à ceci : (Elle ne marchera pas pour le moment)

Une image contenant texte, capture d’écran, Police, nombre

Description générée automatiquement

* + Dans ArtistesController, modifiez le deuxième Create qui gère la création d’un artiste :
    - D’abord, à partir de maintenant notre action va recevoir un paramètre de type ArtisteEmployeViewModel. Arranger le Bind pour n’avoir que Artiste et Employe.
    - **34-40**   Ensuite, le code à l’intérieur de l’action va être remplacé par un appel de la procédure nommée **Employes.USP\_AjouterArtiste**.
    - **ATTENTION** (C’est une procédure sans retour ! Mais avec des paramètres!!!)

Dans le ppt, on a comment appeler une procédure avec des paramètres qui retourne des enregistrements ET comment appeler une procédure sans paramètres qui ne retourne pas d’enregistrements.

* + - **Il va falloir ici faire un MIX des deux exemples ici**.
    - Vérifiez comment la procédure fonctionne dans le script de la BD. Tous les paramètres nécessaires seront disponibles dans le paramètre artisteEmploye reçu par l’action !

Une image contenant texte, Police, ligne, capture d’écran

Description générée automatiquement

Vous pouvez maintenant tester la création d’un artiste avec votre vue Create !

* + - AVANT la création de cette artiste par la vue, vous avez les données suivantes dans SSMS pour le dernier enregistrement des tables Employe et Artiste :

Une image contenant texte, capture d’écran, Police, ligne

Description générée automatiquement

* + - Utilisez la vue pour créer l’artiste (et l’employé) ci-dessous :

Une image contenant texte, capture d’écran, Police, nombre

Description générée automatiquement

* + - Vérification, dans la BD, avec SSMS, qu’une rangée a été ajoutée dans les tables Employe ET Artiste.

Une image contenant texte, capture d’écran, Police, ligne

Description générée automatiquement

* + ATTENTION. Nous aurions pu utiliser la vue **VwListeArtiste** à la place du ViewModel ! Pour cet exemple, nous avons utilisé un ViewModel pour le bénéfice de l’apprentissage, car ce n’est pas dans toutes les situations qu’une Vue SQL aurait été à portée de main.

## **3. La suppression d’un employé artiste. (Grâce à un déclencheur)**

* + Faut-il modifier le Delete pour que supprimer un artiste supprime la rangée dans Employe qui lui est associée.
    - **42**  Vérifiez comment fonctionne le déclencheur dans le script de la BD et vérifiez s’il est bel et bien appelé quand on utilise la vue existante **Delete.cshtml** pour supprimer un artiste.
    - (Normalement, il n’y a rien à modifier et ça marche déjà ... Merci les déclencheurs ! Avec le DB-First, le but est de limiter le plus possible la quantité de code dans l’application Web.)

**Remarque 1** : Vous avez peut-être remarqué que nous n’avons pas utilisé les services vus en Prog Web transactionnelle / Prog Web orientée services. (Le DbContext est injecté directement dans le contrôleur) Pour le moment on se concentre à faire fonctionner notre projet DB-First. Si vous ne suivez pas Prog Web services cette session, c’est possible que vous ne voyiez pas ce que signifie « services ».

**Remarque 2** (Exercice supplémentaire présent dans le corrigé) : Nous aurions pu faire beaucoup de modifications supplémentaires sur nos vues pour les artistes. Par exemple : adapter la vue **Edit.cshtml** pour qu’on puisse modifier toutes les propriétés d’un employé artiste. (et pas juste sa spécialité) Si vous doutez de vos habiletés avec ASP.NET Core, prenez le temps d’essayer de le faire. (Cela implique de modifier les deux actions « **Edit** » dans le contrôleur, de modifier la vue **Edit.cshtml** et d’utiliser **ArtisteEmployeViewModel**) C’est possible de le faire avec une procédure stockée ou directement en C#. Dans le corrigé, c’est en C# car il y a peu de code à modifier dans le contrôleur.

Précision : ModelState pourrait vous donner du fil à retordre, voici avec quoi vous pourriez le remplacer pour vérifier si tout est valide :

Une image contenant texte, capture d’écran, Police, ligne

Description générée automatiquementUne image contenant texte, capture d’écran, Police, nombre

Description générée automatiquement

(À droite : votre nouveau formulaire Edit devrait ressembler à ça)