**Laboratoire #8** (Partie 1 sur 2)

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.

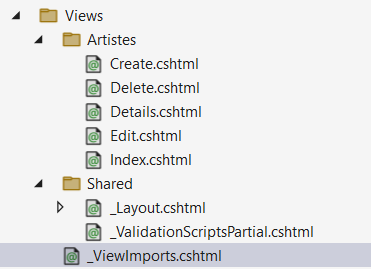
Pour simplifier les choses et rendre ce laboratoire moins répétitif, nous nous concentrerons seulement sur les tables Employe et Artiste et nous ne noterons que quelques informations sur les employés.

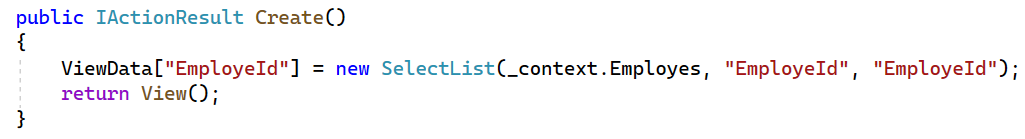
Une image contenant texte, capture d’écran, Police, conception

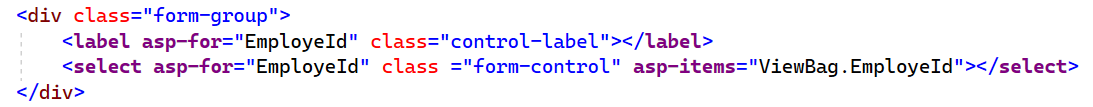
Description générée automatiquement

**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 **bd\_employes\_create.sql** dans SSMS. 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 ?
* **4-7**  Créez le projet ASP.NET Core avec Visual Studio, préparez les dossiers **Controllers** et **Views**, supprimez le dossier **Pages**. (Les dossiers **Data** et **Models** seront créés automatiquement plus tard)
  + Si vous voulez, vous pouvez conserver le \_Layout dans le dossier **Pages** et le déplacer dans le dossier Views/Shared qu’
* **8-9**  Obtenez le string de connection pour la BD que vous venez de créer et ajoutez-le dans appsettings.json.
* **10-14**  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.
* **15-17**  Préparez un **\_Layout** pour vos vues. (Ou utilisez celui qui existait déjà dans le dossier **Pages**)
* **18-21**  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.
* **22**  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 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) :

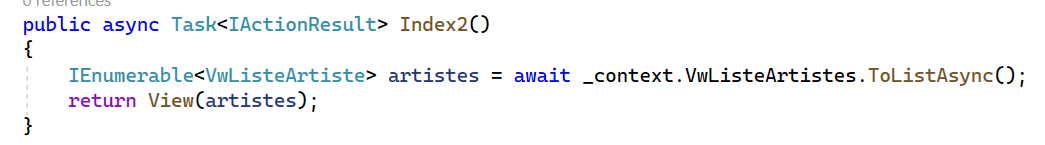




**Étape 2 - Vues, procédures et déclencheurs à la rescousse 🦸‍♂️**

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 mieux :

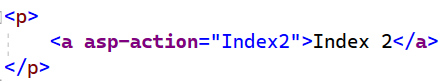
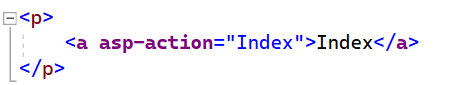
* La création d’un employé artiste. (Grâce à une procédure)
* L’affichage des données des employés artistes. (Grâce à une vue)
* La suppression d’un employé artiste. (Grâce à un déclencheur)
* Ajout d’une action pour voir toutes les infos d’un artiste :
  + **53-53**  Ajoutez une action dans le contrôleur Artistes qui retourne les données de la vue VwListeArtiste à une vue :



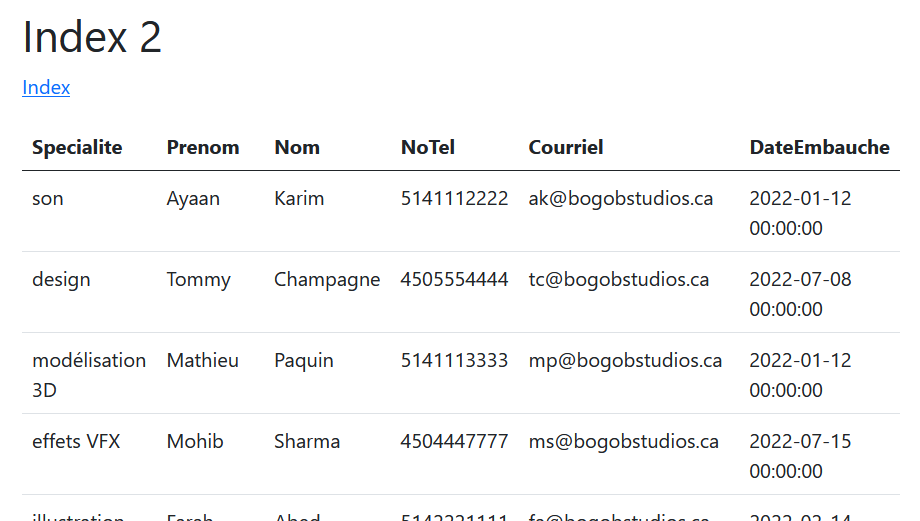
* + Nous allons devoir créer une vue Razor pour afficher la **liste de VwListeArtiste**. Copiez-collez la vue Index.cshtml existante pour les artistes, mais renommez-la avec le nom de votre nouvelle action.
  + Affichez la liste de VwListeArtiste dans cette vue. (Cela demandera quelques modifications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



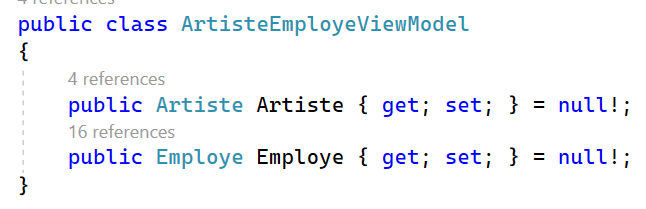
* N’hésitez pas à ajouter des boutons pour pouvoir naviguer entre Index et Index2 :



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

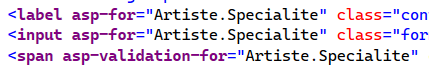


* En vue de modifier l’action Create, créons un ViewModel qui encapsule les données d’un artiste et d’un employé (Sans constructeur svp !) :



Nous aurons besoin de ce ViewModel pour modifier Create.

* Modification des actions Create
  + Le but sera de créer une rangée dans la table **Artiste** et de, simultanément, créer une rangée dans la table **Employe**.
  + Bien entendu, il y a deux actions Create :
    - Une pour afficher la vue avec le formulaire de création.
    - Une qui reçoit les données du formulaire et procède à la création dans la BD.
  + Modifier le premier Create pour afficher la vue avec le formulaire :
    - On retire juste la ligne qui met une SelectList dans ViewData ! Comme on créera un Employé et un Artiste simultanément, nous n’aurons plus besoin de « Choisir » un employé existant.
  + Modifier la vue Create :
    - 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)

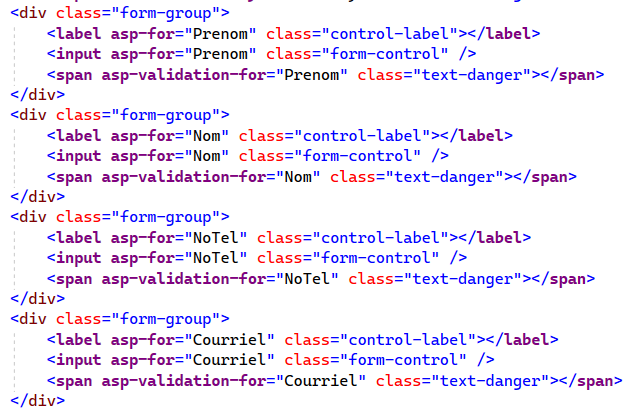


* + - Maintenant... on doit ajouter des « form-group » pour toutes les propriétés d’un employé ! (Prenom, Nom, NoTel et Courriel) ... Trichons un peu ! Auto-générez une Vue Razor/ de type « Create » pour le Model « Employe » :

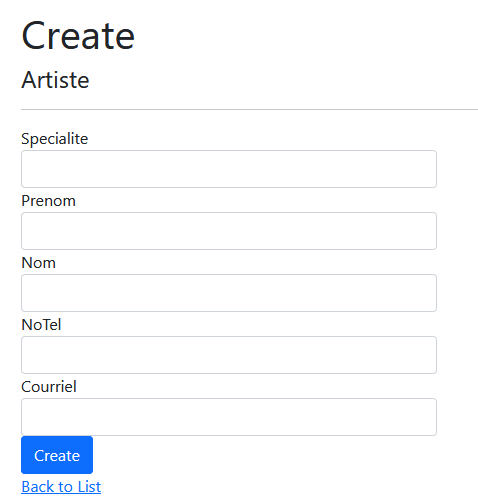
Une image contenant texte, capture d’écran, nombre, Police

Description générée automatiquement

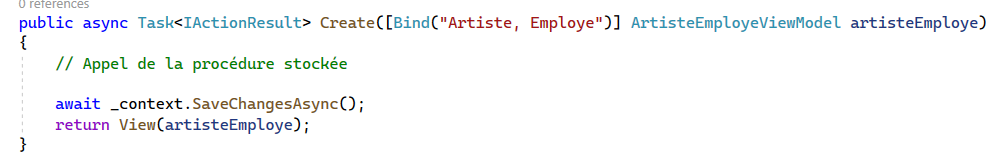
* + - Copiez-collez les « form-group » des quatre propriétés dont on a besoin dans la vue Create.cshtml des artistes ! (Il faudra ajouter Employe. dans les Tag Helpers) Vous pouvez supprimer la vue qu’on vient de générer après.



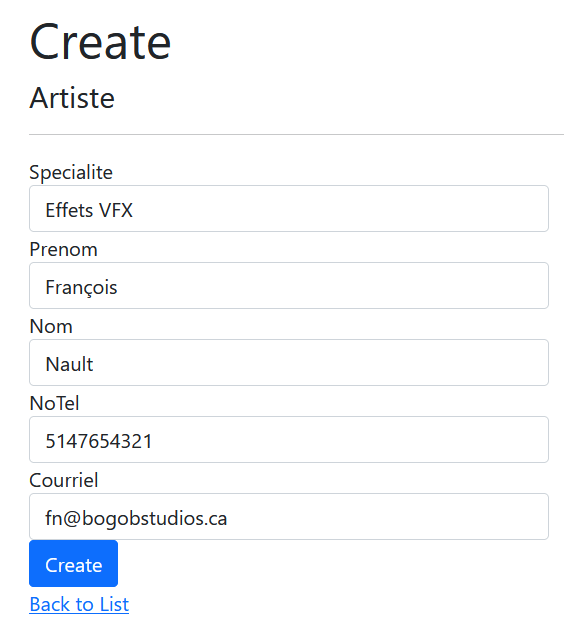
* + - Votre vue devait ressembler à ceci : (Elle ne marchera pas pour le moment)



* + Modifier 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.
    - **63-67**  Ensuite, le code à l’intérieur de l’action va être remplacé par un appel de la procédure nommée **Employes.USP\_AjouterArtiste**. (C’est une procédure sans retour !) Vérifiez comment elle fonctionne dans le script de la BD. Tous les paramètres nécessaires seront disponibles dans le paramètre artisteEmploye reçu par l’action !



* + - Vous pouvez maintenant tester la création d’un artiste avec votre vue Create ! Vérifiez, dans la BD, avec SSMS, qu’une rangée a été ajoutée dans les tables Employe ET Artiste.



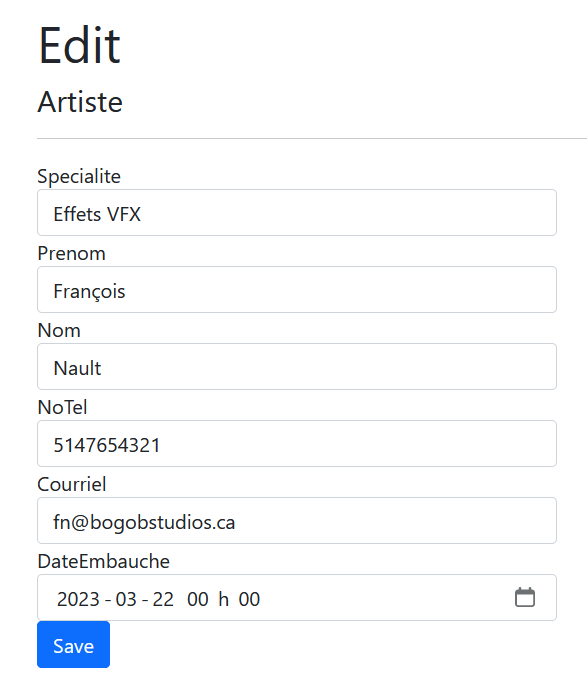
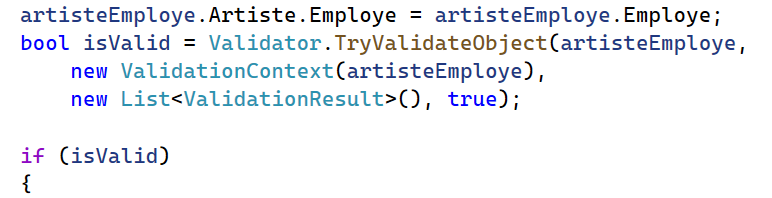
De votre côté l’id pourrait être 31.

* + - 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’un Vue SQL aurait été à portée de main.
  + « Modifier » le Delete pour que supprimer un artiste supprime la rangée dans Employe qui lui est associée.
    - **68**  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** : Vous avez peut-être remarqué que nous n’avions pas encore utilisé les services vus en Prog Web transactionnelle / Prog Web orientée services. (Le DbContext est injecté directement dans le contrôleur) Ça viendra... un peu plus tard ! Pour le moment on se concentrait 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** (pour ajouter Artiste.ArtisteID, Artiste.EmployeID et Employe.EmployeID en champs cachés)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 :



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

**Laboratoire #8** (Partie 2 sur 2)

Révision de LINQ

Dans le cours Prog Web orientées services, il n’y aura pas vraiment de manipulations complexes pour « Retrieve » des données. Ce cours est donc une bonne opportunité de faire des agrégations, des jointures, etc. avec les données.

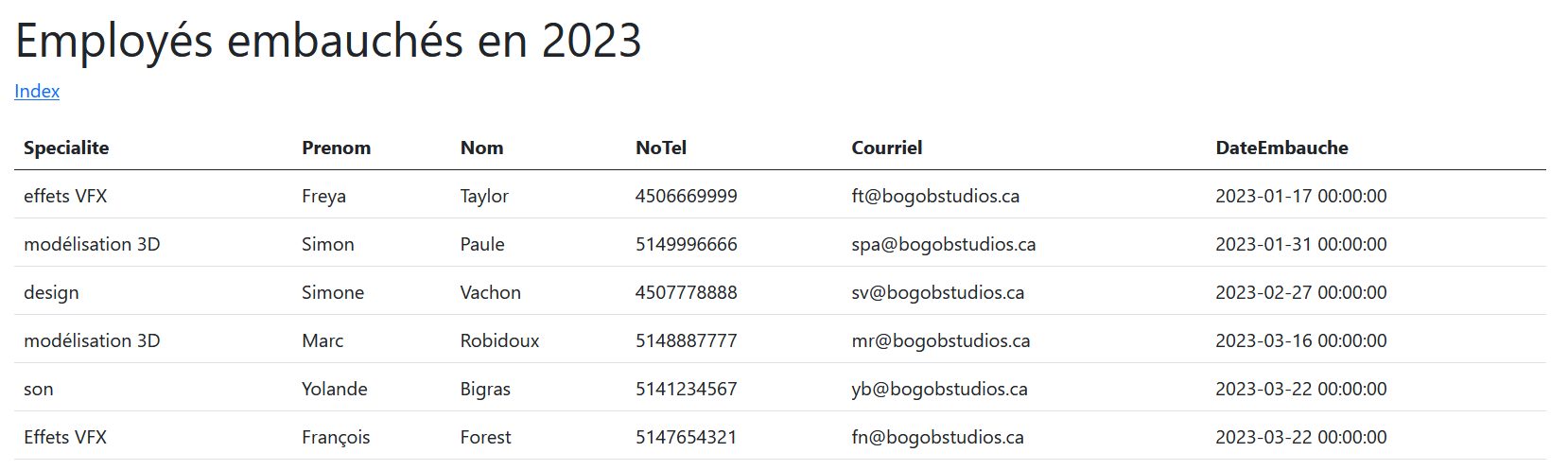
Pour les exercices qui suivent, vous aurez à intégrer plusieurs **Views** fournies à votre projet. Il ne vous restera qu’à compléter des actions dans le contrôleur artiste pour préparer les données à afficher dans les vues fournies. Vous pouvez retrouver le squelette des six actions à faire dans le fichier **actions.txt**.

N’hésitez pas à installer le package **EntityFrameworkCore.Proxies** si vous voulez vous simplifier la vie pour certaines requêtes qui nécessitent les deux tables.

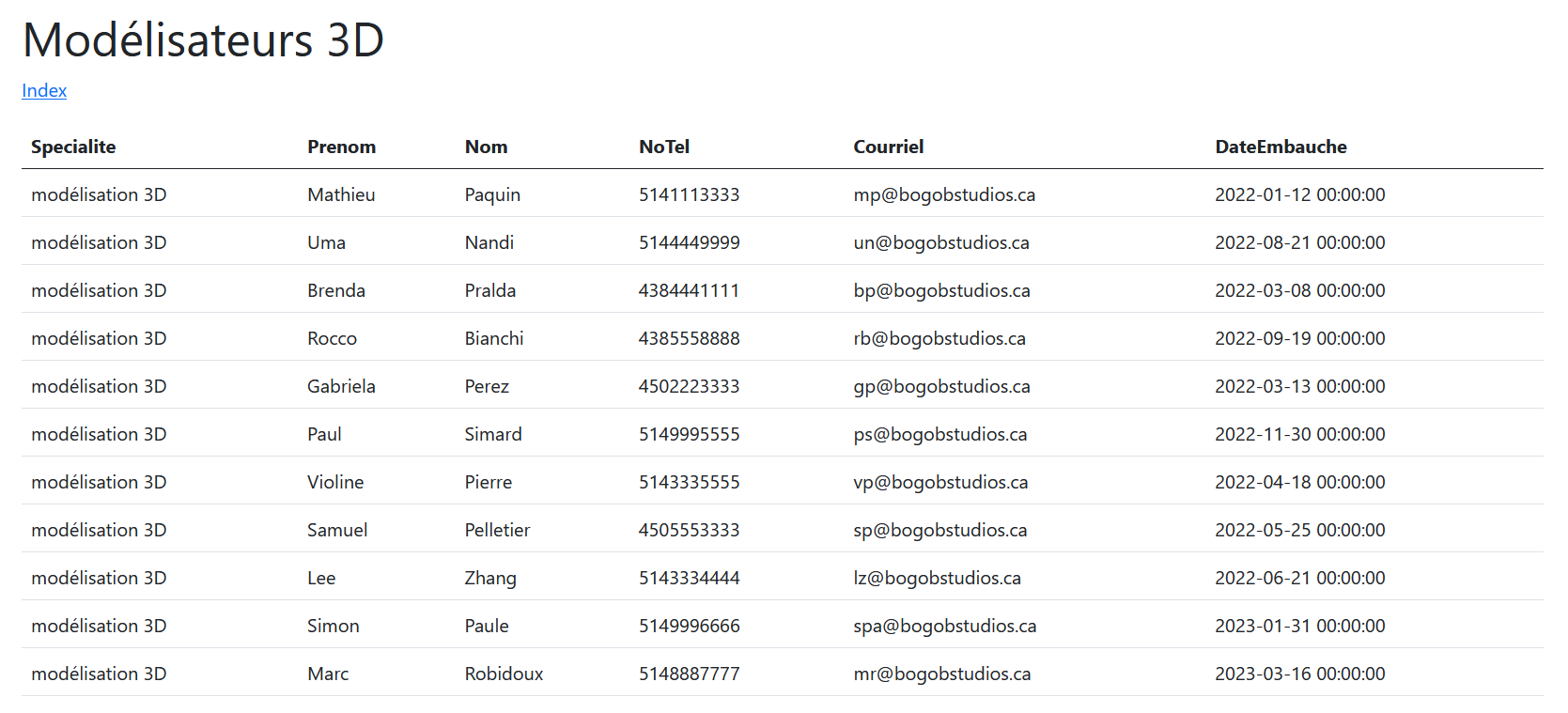


**Étape 3 - C'est plus rapide en SQL 😩**

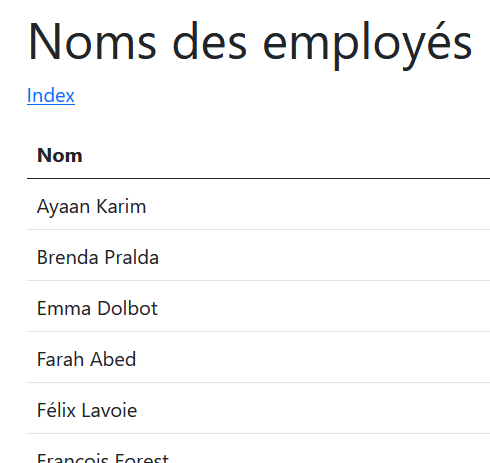
* Quels employés ont été embauchés en 2023 ?
  + Utilisez **VwListeArtiste**.



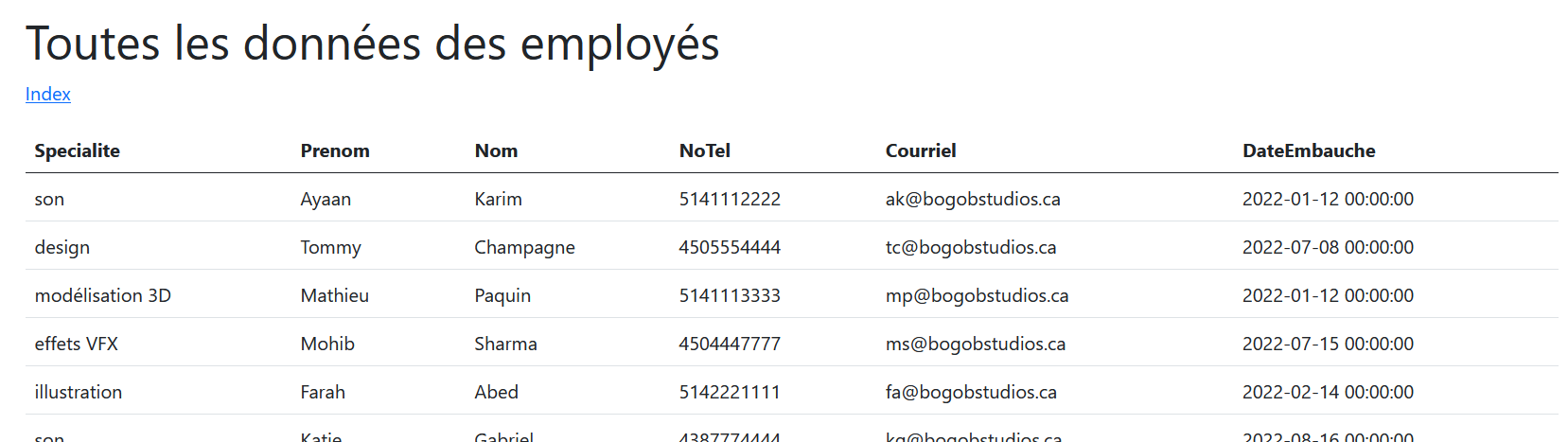
* Quels employés ont la spécialité « modélisation 3D » ?
  + Utilisez **VwListeArtiste**.



* Le prénom et le nom de tous les employés, classés par ordre alphabétique de prénom ?
  + Votre requête LINQ doit retourner une simple liste de string (**IEnumerable<string>**) comme ça pas besoin de créer de ViewModel.



* Toutes les données des employés artistes avec jointure. (Pas le droit d’utiliser la vue **VwListeArtiste** !)
  + Utilisez **ArtisteEmployeViewModel**. (Donc on envoie **IEnumerable<ArtisteEmployeViewModel>** à la vue Razor) Il est très possible que vous n’ayez pas à utiliser Join() pour y arriver !



* Combien d’artistes par spécialité ?
  + Vous aurez à créer un ViewModel nommé **NbSpecialiteViewModel** avec les propriétés « **string Specialite** » et « **int Nb** ». Envoyez un **IEnumerable<NbSpecialiteViewModel>** à la vue Razor.
  + Comme pour la jointure, il est possible que vous trouviez une solution sans GroupBy() !

