**Semaine 09** (Partie 1 sur 2)

Migrations



Bleu et orange : État initial de la BD.

Orange : Devra être supprimé ou modifié.

Fushia : Devra être ajouté.

**Résumé des fonctionnalités :**

* Il y a des **chanteurs** et des **chansons**. (Les chansons appartiennent à un chanteur) On peut les afficher sur le site.
* Il y aura des utilisateurs, on pourra s’**inscrire**, se **connecter** et se **déconnecter**. On pourra voir nos propres informations dans notre **profil**.
* Les utilisateurs pourront **mettre des chanteurs dans leur liste de favoris**.
* C’est un peu comme le TP2 de Prog Web services, mais pris par surprise un dimanche matin en peignoir et en pantoufles.

**Attention !**

* Dans ce labo, beaucoup de portions de codes sont déjà faites et devront simplement être décommentées au bon moment. (Si vous aviez eu à tout coder vous-mêmes, nous n’aurions pas pu tester autant de migrations, ou bien ça aurait pris 10 heures. Vous allez surtout modifier du SQL. De toute façon, généralement, le code à décommenter est censé paraître simple ou intuitif pour vous)
* Si vous prenez le temps de regarder ce que fait le code à décommenter, cela vous aidera à vous lancer dans votre projet perso éventuellement.
* Durant le projet perso, les solutions du S09\_Labo pourront être très intéressantes pour compléter certaines parties.

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

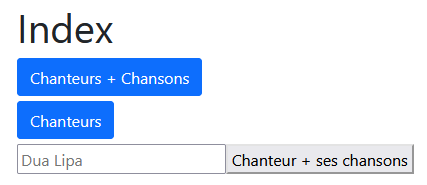
Description générée automatiquement**

Il y a déjà un dossier Sql\_Scripts avec 5 fichiers de migrations. Certains devront être modifiés. Attendez que ce soit indiqué avant d’en exécuter un.

Parfois, vous allez devoir ajouter l’option **--force** pour la commande qui scaffold le DbContext et les Models. Cela permettra d’écraser les anciens Models.

**Étape 0 - Échauffaudage 🏗**

* **08**  Avec SSMS ou Visual Studio, exécutez manuellement le script dans le fichier **InitialCreate.sql** pour créer la BD du labo. Si vous nommez bien la BD, le string de connexion déjà configuré devrait marcher.
  + Si vous faites des erreurs dans les migrations n’hésitez pas à supprimer la BD et réexécuter toutes les migrations jusqu’à la version critique plus tard.
* **11**  Appliquez la migration **1.0**
  + À ce stade, le projet Web devrait fonctionner ! Essayez ces trois boutons pour être sûrs qu’ils fonctionnent et faites le tour des actions dans MusiqueController.



**Note : Entrez la valeur avant de cliquer sur le bouton**

**Étape 1 - Migrations printanières** 🦆🦆🦆

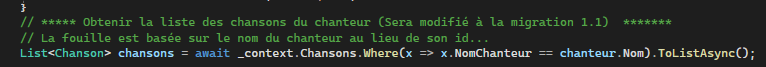
Dans cette section, **nous allons compléter et exécuter les migrations 1.1, 1.2 et 1.3.**

* Vous devrez remplir le fichier de la **migration 1.1** en entier vous-mêmes.  
  + **25-28**  Si vous jetez un coup d’œil à la migration 1.0, vous remarquez que la clé primaire de la table Chanteur est ... le nom des artistes. (un nvarchar) Quelle horreur ! (Imaginez le risque d’erreurs en entrant ainsi plusieurs fois le nom des artistes…) **Créez une clé artificielle avec une colonne IDENTITY à la place et changez la clé étrangère dans la table Chanson.**
  + Avec Evolve, **exécutez la migration 1.1**.
  + Avec la commande ci-dessous, qu’on vous fournit car vous êtes des *cutie-pie*, écrasez les anciens Models par les nouveaux. Avec l’option **--force**, on s’assure d’écraser les Models existants.

(*Assurez-vous que la fenêtre de console de déboggage  soit fermée AVANT d’essayer de regénérer les Models existants*)

**dotnet ef dbcontext scaffold Name=S09\_Labo Microsoft.EntityFrameworkCore.SqlServer -o Models --context-dir Data --data-annotations --force**

* + Vous allez devoir modifier légèrement l’action **Musique/UnChanteurEtSesChansons** car la manière de faire le lien entre un chanteur et ses chansons a changé. Voir image ci-bas :



* + Si tout fonctionne ensuite, vous pouvez passer à la suite.
* **Pour la migration** **1.2**, **nous allons créer une vue SQL**.
  + Vérifier l**’action** **Chanteurs** et la **View** **Chanteurs**. Ils affichent le **nom**, la **date de naissance** et le **nombre de chansons** pour chaque chanteur. (**L’id du chanteur sera nécessaire aussi**)
  + Créez la **vue** **SQL** nécessaire dans la **migration 1.2.** Elle devrait avoir 4 colonnes.
  + Exécutez la migration 1.2 avec **Evolve**.
  + **Faites encore un scaffold pour recréer les Models et le DbContext. (Cela ajoutera la vue)**
  + Dans l’action Chanteurs (et la vue Chanteurs), remplacez le code pour utiliser la vue SQL au lieu de construire les données avec une requête LINQ.
    - Le ViewModel ne servira plus : À la place on va envoyer à la vue une **List<VwChanteurNbChanson>**.
    - Dans le contrôleur, l’action devrait maintenant n’avoir que 1 ou 2 lignes de code. (Utiliser la vue simplifie beaucoup l’action)
    - Dans la **View Razor**, **remplacez le @model** et modifier les étiquettes pour l’affichage au besoin.
  + **Si la vue Razor Chanteurs marche encore bien**, vous pouvez passer à la suite, **sinon, modifiez la vue Razor pour aller avec les noms des colonnes que vous avez utilisé dans votre vue SQL**, qui est maintenant dans le modèle VwChanteurNbChanson.cs.
* Pour la migration **1.3**, nous allons créer une **procédure stockée.**
  + Vérifiez l’action **UnChanteurEtSesChansons** et la View **UnChanteurEtSesChansons**. Un ViewModel envoie les infos du chanteur et la liste de ses chansons à la vue Razor.
  + **Faites une procédure stockée** qui retourne toutes les informations des chansons de l’artiste recherché. Ces informations sont dans la table Musique.Chanson.
  + Exécutez la migration 1.3 avec **Evolve**.
  + Pas besoin de scaffold : la procédure stockée ne fait pas partie du DbContext.
  + **Modifiez l’action UnChanteurEtSesChansons** :
    - Vous devrez appeler la procédure stockée que vous venez de créer pour récupérer une List<Chanson>. C’est à la fin des notes de la rencontre 17
  + Il n’y a rien à modifier dans la vue car on utilise le même ViewModel.
  + Si la vue Razor **UnChanteurEtSesChansons** marche encore bien, vous avez fini la partie 1.