**Laboratoire #10** (Partie 1 sur 2)

Migrations



Bleu et orange : État initial de la BD.

Orange : Devra être supprimé ou modifié.

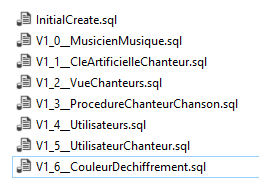
Fushia : Devra être ajouté.

**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 le TP2 la semaine prochaine.
* Durant le TP2, les solutions du labo 9 et 10 pourront être très intéressantes pour compléter certaines parties.

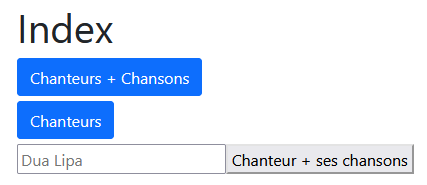
****

Il y a déjà un dossier Sql\_Scripts avec 8 fichiers de migrations. Certains devront être modifiés. Attendez que ce soit indiqué avant d’en exécuter un avec Evolve.

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 🏗**

* **12-13**  Avec SSMS ou Visual Studio, exécutez manuellement le script dans le fichier **InitialCreate.sql** pour créer la BD du labo 10. 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.
* **7**  Installez Evolve. (Rappel : c’est à répéter à chaque fois que vous changez d’ordinateur au cégep)
* **10**  Appliquez la migration **1.0** à l’aide de la commande Evolve. Vous pouvez déjà spécifier les schémas Musique ET Utilisateurs pour que la commande soit toujours pareille (à part la version) puisque vous allez l’exécuter plusieurs fois.
  + 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, progressivement.

* Vous devrez remplir le fichier de la migration **1.1** en entier vous-mêmes.
  + **20**  Si vous jetez un coup d’oeil 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=Lab10 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.
    - Si tout fonctionne ensuite, vous pouvez passer à la suite.
* Pour la migration **1.2**, nous allons créer une vue.
  + 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<VwChanteurNbChansons>.
    - 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.
  + Complétez la procédure stockée : elle est très simple, elle retourne toutes les colonnes de toutes les rangées de la table Musique.Chanson de l’artiste recherché.
  + Exécutez la migration 1.3 avec **Evolve**.
  + Pas besoin de scaffold : la procédure stockée ne fait pas partie du DbContext.
  + Modifiez l’action UnChanteurEtSesChansons (Rien à modifier dans la vue car on utilise le même ViewModel)
    - Il y aura une petite requête LINQ pour récupérer l’id du chanteur.
    - Ensuite, vous devrez appeler la procédure stockée que vous venez de créer pour récupérer une List<Chanson>. C’est à la fin des notes de cours 9.
  + Si la vue Razor **UnChanteurEtSesChansons** marche encore bien, vous avez fini la partie 1.

**Laboratoire #9** (Partie 2 sur 2)

Authentification

**Étape 2 - Problèmes identitaires 👤😨**

Dans cette section, nous allons implémenter la gestion des utilisateurs et de leurs chanteurs favoris. Le travail est déjà commencé... mais il y a des maladresses impardonnables dans les migrations. C’est parti pour les migrations **1.4**, **1.5** et **1.6**. (N’oubliez pas d’utiliser Evolve pour réexécutez les migrations 1.0 à 1.3 si vous avez changé d’ordi entre-temps) Pas besoin de scaffold puisque votre DbContext et vos Models devraient déjà être accordés à la version **1.3** de la BD.

* Pour la migration **1.4**, qui est déjà bien entamée, nous allons faire quelques modifications.
  + Il y a deux problèmes colossaux :
    - Les **mots de passe** sont **stockés en clair**. (Et quand quelqu’un s’authentifie, on compare si le mot de passe fourni est identique au mot de passe en clair)
    - La **couleur préférée** n’est **pas chiffrée** ! 🌈🙄 Bon ok... ce n’est pas grave, mais faisons semblant que la couleur préférée d’une personne est extrêmement secrète et permet de voler facilement son identité. 🕵️‍♂️ Il faut donc absolument **chiffrer** cette donnée !
  + **38-40, 43, 50**  Pour le hachage du mot de passe, il faudra modifier :
    - Le type de MotDePasseHache.
    - La procédure de création d’un utilisateur.
    - La procédure d’authentification d’un utilisateur.
  + **38, 41-42, 44-49**  Pour le chiffrement de la couleur préférée, il faudra :
    - Modifier le type de CouleurPrefere.
    - Créez les clés nécessaires pour chiffrer / déchiffrer.
    - Modifier la procédure d’inscription.
  + Une fois que c’est réglé, exécutez la migration **1.4** avec **Evolve**.
  + Faites un **scaffold** pour créer les nouveaux **Models** et remplacer le **DbContext**.
  + **52-58**  Décommentez les actions POST **Inscription** et **Connexion** dans **UtilisateursController**. Ils sont identiques à celles dans les notes de cours.
  + **51**  Décommentez les configurations requises dans **Program.cs**. Tout est déjà là.
  + **59**  Décommentez le bloc de code dans l’action **Index** de **MusiqueController**. Il sert à dire bonjour à l’utilisateur connecté. (S’il y en a un)
  + Testez les boutons Inscription, Connexion et Déconnexion. Tout devrait marcher.
  + **60**  Décommentez l’action Profil et ajoutez-lui un petit quelque chose pour qu’elle fonctionne juste avec les utilisateurs authentifiés.
    - Dans la vue Profil, il y aura des choses à décommenter et des █ à remplacer par des @.
  + Testez le bouton qui affiche le **profil** de l’utilisateur actuellement connecté.
    - C’est normal que la couleur préférée ne soit pas affichée.
* La migration **1.5** est déjà complétée. C’est une table de liaison Many-To-Many entre Utilisateur et Chanteur pour que les utilisateurs puissent mettre des chanteurs dans leurs favoris.
  + Exécutez la migration 1.5 avec **Evolve**.
  + Faites un **scaffold** pour créer les nouveaux **Models** et remplacer le **DbContext**.
  + Décommentez la propriété Utilisateur dans le UtilisateurEtFavorisViewModel.
  + Décommentez l’action **AjouterFavori** dans **MusiqueController**. N’oubliez pas le [Authorize].
  + Dans la vue **Chanteurs**, décommentez le **<th>** et le **<td>** qui permettront d’ajouter des chanteurs à nos favoris.
  + Dans l’action **Profil** de **UtilisateursController**, remplacez le **return View()** par la fin alternative qui permet d’envoyer dans le profil la liste des chanteurs favoris de l’utilisateur authentifié.
  + Dans la vue **Profil**, vous pouvez décommenter le tableau des artistes favoris ainsi que changer le @model dans le haut de la page pour utiliser un ViewModel qui contient l’utilisateur et sa liste d’artistes.
  + Testez (à nouveau) le bouton qui affiche le **profil** APRÈS avoir ajouté des artistes dans votre liste de favoris dans la vue **Chanteurs**.
    - Vous devriez vous la liste des chanteurs que vous avez ajoutés à vos favoris désormais.
* La migration **1.6** est presque terminée ! Elle permettra à un utilisateur de vérifier sa propre couleur préférée après avoir validé son mot de passe dans son profil.
  + **48**  Modifiez la migration, il reste trois choses à ajouter :
    - L’ouverture de la clé symétrique.
    - La conversion et le déchiffrement de la couleur préférée. (Cette colonne sera renommée en « Couleur », c’est normal)
    - La fermeture de la clé symétrique.
  + Exécutez la migration 1.6 avec **Evolve**.
  + Faites un **scaffold** pour créer les nouveaux **Models** et remplacer le **DbContext**.
  + Décommentez l’action **Couleur** dans **UtilisateursController**. Elle reçoit un mot de passe en paramètre, le compare avec celui dans la BD, et retourne la couleur déchiffrée dans la vue **Profil** à l’aide de la procédure stockée dans la migration 1.6. (Il se peut que la propriété **.Couleur1** doive être renommée **.Couleur**, je ne sais pas ce qui a poussé le scaffolding à lui donner ce nom)
  + Décommentez le bloc qui accompagne la couleur préférée dans la vue **Profil**. Elle permettra de valider son mot de passe pour accéder à notre couleur préférée quand on est connecté.
  + Testez la nouvelle option du profil qui permet de vérifier notre couleur préférée !
    - Si ça marche, vous avez terminé ~~de décommenter frénétiquement du code~~ ce labo.