**Wrap-Up R22- R23**

On récapitule tout !!

Le but de cette séance (et du travail à faire à la maison) est de réaliser un grand laboratoire qui résume toutes les notions vues depuis l’examen intra, avant de commencer le projet final.

Nous ne commencerons pas un projet à partir de zéro (vous aurez largement l’occasion de relever ce défi lors du projet final), mais nous allons plutôt passer en revue les notions suivantes :

* Intégration de **Views**
* Appels de **Procédures Stockées**
* Intégration d’**images**
* **Chiffrement** d’informations sensibles

Le projet initial vient avec tous les paquets NuGet nécessaires pour effectuer le labo, dont **EntityFrameworkCore.Proxies**, pour vous simplifier la vie pour certaines requêtes.

Dans ce Wrap-Up, nous allons gérer une **billetterie de spectacles**. Les indications sont aussi détaillées que possible, mais il y a très peu de code à décommenter. Ce laboratoire vous permettra de bien vous préparer en vue de votre projet de fin de session.

**Étape 0 – Mise en place du script InitialCreate.sql**

Cette étape préliminaire vous permettra de **terminer la mise en place des bases de l’application**.  
La structure principale du projet Visual Studio est déjà en place, mais vous devrez **compléter certains éléments progressivement, au fil des prochaines étapes**.

**Pour le script InitialCreate.sql vous devez compléter les étapes suivantes :**

1. Créer la Base de données R22\_Billeterie (**Rencontre 15**),
2. Ajouter le code pour permettre l’utilisation de FILESTREAM avec la base de données (**Rencontre 19**).
3. Ajouter le code permettant la configuration d’une clé symétrique (**Rencontre 18**). Cette clé symétrique servira à chiffrer les données de carte bancaire lors de l’étape 7.

Lorsque **le script InitialCreate.sql sera complété**, **exécutez-le** puis utilisez la commande suivante pour faire le **scaffold** dans VisualStudio :

* + *dotnet ef dbcontext scaffold Name=R22\_Billeterie Microsoft.EntityFrameworkCore.SqlServer -o Models --context-dir Data --context R22\_BilleterieContext --data-annotations –force*

**Étape 1 – Création de la table Client**

Une image contenant texte, capture d’écran, Police

Description générée automatiquement

Dans cette étape, nous allons créer la **table** **Client**. Le **contrôleur** **Clients** et les **vues** associées sont déjà créés, mais vous devrez compléter certaines actions dans le **contrôleur Clients**.

La table **Client** servira à stocker les informations nécessaires pour **s’inscrire** et **se connecter** à l’application. Lorsqu’un utilisateur s’inscrit, il devra fournir les informations suivantes : **prénom, nom, courriel et mot de passe**.

Pour se connecter, seulement le **courriel** et le **mot de passe** seront demandés.

**Vous devez compléter les étapes suivantes :**

1. **Compléter le fichier V1\_0\_\_CreationDesClients.sql :** 
   1. Créer la table Client (champs et contrainte de clé primaire)
   2. Utiliser ALTER TABLE pour ajouter une contrainte d’unicité sur le champ Courriel
   3. Compléter la procédure stockée **Clients.USP\_CreerClient** qui permet d’inscrire un nouvel utilisateur. (Les paramètres nécessaires sont déjà déclarés dans la procédure)
   4. Compléter la procédure stockée **Clients.USP\_AuthClient** qui permet d’authentifier un utilisateur lors de sa connexion. (Les paramètres nécessaires sont déjà déclarés dans la procédure)

**Précision :** Les insertions initiales sont déjà fournies.

1. **Exécuter la migration avec la commande *evolve*, puis faite le scaffold.** 
   1. *evolve migrate sqlserver -c "Data Source=.;Initial Catalog=R22\_Billeterie;Integrated Security=True;Persist Security Info=False;Pooling=False;Encrypt=False" -s Clients -s Spectacles --target-version 1.0*

***Précision :***Remarquez que la **commande Evolve** crée aussi **deux schémas :** **Clients** et **Spectacles**. Assurez-vous que ces schémas sont bien créés si jamais vous faites des tests directement dans SSMS.

* 1. *dotnet ef dbcontext scaffold Name=R22\_Billeterie Microsoft.EntityFrameworkCore.SqlServer -o Models --context-dir Data --context R22\_BilleterieContext --data-annotations –force*

1. Quatre actions sont fournies dans le contrôleur **Clients** : **Index**, **Inscription** (GET et POST), **Connexion** (GET et POST) et **Deconnexion**.
   1. Compléter le code de l’action **Inscription (POST).**
   2. Compléter le code de l’action **Connexion (POST)**. Dans cette action, après avoir exécuté la procédure, **vous devez créer le cookie** et lui donner la **bonne propriété**. Attention à l’attribut du **Client** que vous envoyez dans le cookie! Regardez dans l’action **Index**quel attribut est utilisé pour identifier le **Client**.

**Précisions :**

* Les ***ViewModels*** associés sont déjà créés.
* Trois **vues** sont fournies : Inscription, Connexion et Index.

**Important :** Veuillez noter que dans **program.cs**, la fonction **MapControllerRoute** pointe sur l’action **/Clients/Index.** Ceci est une **mesure temporaire** afin que nous puissions tester que tout fonctionne du côté de l’identification en attendant d’ajouter les données sur les spectacles et les billets.

1. **Tester que tout fonctionne en faisant les actions suivantes :** 
   1. Vérifiez que lors du démarrage de l’application, vous voyez la mention : « Bonjour visiteur ».
   2. Allez sur la page de connexion et connectez-vous avec un des utilisateurs enregistrés. Le retour à l’index devrait afficher : « Bonjour Chantal Vallière » si vous avez utilisé la cliente Chantal Vallières.
   3. Déconnectez-vous.
   4. Allez sur la page d’inscription d’un utilisateur et créé un nouvel utilisateur.
   5. Retournez sur la page de connexion et vérifiez que vous êtes capable de vous connecter avec ce nouvel utilisateur (et vérifiez votre base de données avec SQL Server).

**Étape 2 – Ajout des spectacles et des billets**

Une image contenant texte, capture d’écran, conception

Description générée automatiquement

1. Complétez la migration **V1\_1\_\_AjoutSpectaclesEtBillets.sql**. Les tables **Spectacle**, **Representation** et **Billet** sont déjà créées, mais il manque certains éléments. La colonne **Prix** dans la table **Spectacle** représente le **prix unitaire d’un billet**. La colonne **CoutBillet** dans la table **Billet** représente le nombre de billets achetés multiplié par le prix unitaire d’un billet. Les insertions sont déjà fournies. **Les éléments à compléter sont les suivants :**
   1. En vous basant sur le diagramme, complétez les contraintes de clés étrangères dans la migration **V1\_1\_\_AjoutSpectaclesEtBillets.sql**.
   2. Ajoutez aussi les contraintes suivantes :
      1. La colonne DateHeureRepresentation prend la valeur de **GETDATE()** par défaut.
      2. Le prix du billet est positif ou nul.
      3. Le nom du spectacle est unique.
2. Exécutez la migration **V1\_1\_\_AjoutSpectaclesEtBillets.sql** puis faites le scaffold.
3. Exécutez ensuite la migration **V1\_1\_1\_\_UpdateCoutBillet.sql** qui mettra à jour le champ **CoutBillet** dans la table **Spectacles.Billet.** Vous n’avez rien à modifier dans le code de cette migration.
4. Générez automatiquement un contrôleur et des vues Razor pour le modèle **Spectacle** (**Rencontre 15**).
5. Modifiez les paramètres de routing dans **program.cs** pour que nous arrivions directement sur la page **/Spectacles/Index** par défaut.
6. Lancez votre application et vérifiez que vous êtes capables de naviguer dans les différentes pages de l’application. Nous reviendrons aux clients plus tard.

**Étape 3 – Ajout d’une vue pour avoir plus d’informations sur les spectacles**

**Les éléments à compléter sont les suivants :**

1. Complétez la migration **V1\_2\_\_AjoutVueSpectacles.sql.** Celle-cicontient une **vue SQL** nommée **Spectacles.VW\_SpectaclesRepresentationSpectateurs** qui permettra de voir les informations suivantes :
   * L’id du spectacle (très important !)
   * Nom du spectacle
   * Date de début
   * Date de fin
   * Nombre de représentations prévues (DISTINCT est votre ami ici)
   * Le nombre total de billets vendus
   * Le prix d’une place
2. Exécutez la migration puis faites le scaffold.
3. Copiez-collez le contenu de la vue Razor **Spectacles/Index** dans une nouvelle vue Razor nommée **Spectacles/IndexAncien**. Faites de même pour l’action **Index** dans le contrôleur **Spectacles**.
4. Adaptez la vue **Spectacles/Index** et l’action correspondante de manière à afficher les informations de la vue **VW\_SpectaclesRepresentationsSpectateurs :**
   1. Pensez à changer le modèle à la première ligne de la vue.
   2. Ajoutez les colonnes nécessaires pour afficher les informations envoyées par la vue SQL.
   3. Ne gardez que les liens vers la vue **Details**.

Une image contenant texte, ligne, Police, nombre

Description générée automatiquement

**Étape 4 – Amélioration de la page Détails**

Une image contenant texte, capture d’écran, conception

Description générée automatiquement

Pour l’instant, la vue **Détails** d’un spectacle n’est pas vraiment « sexy ». On aimerait pouvoir montrer l’affiche et l’ensemble des dates des représentations des spectacles.

**Les éléments à compléter sont les suivants :**

1. Complétez la **V1\_3\_\_AjouterTableAffiche** :
   1. Créez une table **Affiche** ayant les champs suivants : AfficheID, Identifiant, AfficheContent et SpectacleID. **Précision :** Les images seront stockées via le champ **AfficheContent**. (Voir la rencontre 19 pour préparer une table qui stockera des images)
   2. Liez les tables **Affiche** et **Spectacle** par un lien de clé étrangère
   3. Insérez des données dans cette table (le code des insertions est fourni). **Attention : pensez à changer le chemin amenant vers les images des affiches pour que les insertions fonctionnent !**
   4. Lancez la migration puis faites le scaffold.
2. Nous allons construire un ***ViewModel*** spécifique pour passer des données à la **vue Razor** **Spectacles/Details** et nous allons modifier cette dernière. **Les étapes à suivre sont les suivantes :** 
   1. Ouvrez le fichier *SpectaclesAfficheRepresentationViewModel.cs* et décommentez les lignes définissant les trois attributs contenus dans cette classe. Ces attributs sont les suivants :
      * Une instance de la vue créée à l’étape précédente
      * Une liste de représentations
      * Une chaîne de caractères qui va contenir l’image à afficher
   2. Copiez la vue Razor **Spectacles/Details** et renommez la copie **Spectacles/DetailsAncien.cshtml**
   3. Copiez-collez l’action **Details** dans le contrôleur et renommez une des deux copies **DetailsAncien**
   4. Modifiez l’action **Details** de façon à construire une instance du *ViewModel* **SpectaclesAfficheRepresentationViewModel** qui recueillera les données pertinentes et passez ce *ViewModel* à la vue Razor.
      1. Faites **une requête par attribut du *ViewModel*** (une pour la vue SQL, une pour les représentations et une pour l’affiche)
      2. **Ensuite, construisez une instance du *ViewModel*** et passez-la en paramètre au ***return View().***
   5. Modifiez la vue Razor **Spectacles/Détails** de façon à afficher tous les détails du *ViewModel* :
      1. Changez le **modèle** à la première ligne du fichier.
      2. Adaptez les ***DisplayNameFor*** et ***DisplayFor*** à ce nouveau modèle.
      3. Ajoutez un champ pour afficher l’image (copier-coller un des couples *dt* et *dd* et faites les modifications nécessaires). Pour afficher l’image, vous aurez à remplacer **la ligne @Html.DisplayFor** par un **tag <img>** puis remplir celui-ci pour afficher les images. (**Rencontre 19**).
      4. Lancez votre application une première fois pour vérifier que tout fonctionne bien lorsque vous cliquez sur un lien ***Détail***.
      5. Enfin, créez un tableau (prenez modèle sur la façon dont la page Index est construite) pour afficher les dates des représentations. Pour l’instant, votre tableau n’aura qu’une seule colonne.

Une image contenant texte, capture d’écran

Le contenu généré par l’IA peut être incorrect.

**Étape 5 – On réintroduit les clients**

Cette étape-ci est assez simple. Nous allons ajouter des liens en haut des pages web pour que l’utilisateur puisse s’inscrire, se connecter et se déconnecter de la plateforme. De plus, dans la vue Index, nous mettrons un petit texte ***’’Bonjour visiteur !’’*** si l’utilisateur n’est pas authentifié, ou ***’’Bonjour [Prénom] [Nom] !’’*** s’il l’est.

**Les éléments à compléter sont les suivants :**

1. Dans la vue Razor **Spectacles/Index**, ajoutez les liens vers les actions **Clients/Inscription**, **Clients/Connexion** et **Clients/Deconnexion** du contrôleur **Clients**.

Une image contenant texte, Police, nombre, ligne

Le contenu généré par l’IA peut être incorrect.

1. Dans le contrôleur **Clients**, modifiez les actions de façon que les renvois vers la vue **Clients/Index** se fassent plutôt vers la vue **Spectacles/Index** (Inscription en mode POST, Connexion en mode POST et Deconnexion).
2. Dernier petit raffinement, nous voulons afficher ***’’Bonjour visiteur !’’*** ou ***’’Bonjour [Prénom] [Nom] !’’*** dans la vue **Spectacles/Index**. Comme indice, reprenez ce qu’on faisait pour la vue **Clients/Index**.

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

Le contenu généré par l’IA peut être incorrect.

1. Assurez-vous que tout fonctionne ! Refaites des tests de connexion, déconnexion, enregistrement d’un client, connexion avec ce nouveau client, déconnexion.

**Étape 6 – Ajout d’une page Profil pour les clients**

Dans cette étape, nous allons permettre à un client authentifié sur l’application d’afficher les informations le concernant. Nous allons construire une vue Clients/Profil ayant pour modèle la classe Client.

**Les éléments à compléter sont les suivants :**

1. Création de la vue Razor **Clients/Profil** :
   1. Copiez-Coller la vue razor **Spectacles/DetailsAncien** en **Clients/Profil**.
   2. Dans **ProfilClientViewModel.cs,** décommentez la propriété **Client**.
   3. Dans la vue Razor **Profil**, changez le modèle passé à la vue pour le ViewModel **ProfilClientViewModel.cs,**
   4. Changez ***ViewData["Title"]*** pour que le titre de la page soit **"Profil"**.
   5. Changez le titre *h1* pour Profil.
   6. Changez le titre *h4* pour Client.
   7. Changez les champs affichés pour montrer les informations du **Client**.
   8. Enlevez le lien vers **Spectacles/Edit** et ajustez le lien vers Index pour qu’il vous redirige vers **Spectacles/Index**.

1. Création de l’action **Profil** dans le contrôleur **Client** :
   1. Copiez-Coller l’action **DetailsAncien** du contrôleur **Spectacles** dans le contrôleur **Clients** et renommez-la **Profil**.
   2. Rajoutez l’annotation **[Authorize]** à cette action.
   3. Enlevez le paramètre *id*. Nous allons récupérer l’*id* du client connecté grâce au *cookie*. L’idée est qu’on accède à cette action seulement si on est connecté à l’application.
   4. Adaptez le code de cette action pour envoyer le bon **client** à la vue Razor (Construisez un objet **ProfilClientViewModel** et envoyez-le à la vue). Au cas où un petit malin arrive quand même à exécuter l’action sans être authentifié, vous le renverrez vers la vue **Spectacles/Index**.

1. Ajout d’un **lien vers le profil** dans **Spectacles/Index** :
   1. Afficher ce lien uniquement si le client est connecté. Un simple test sur la valeur de ViewData["PrenomNom"] fera l’affaire.
2. Les deux images ci-dessous vous montre les vues **Spectacles/Index** lorsqu’un utilisateur est connecté et la vue **Client/Profil**.

**Voir la page suivante pour le résultat attendu.**

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

Description générée automatiquement

Une image contenant texte, capture d’écran, Police

Description générée automatiquement

**Étape 7 – Ajout de moyens de paiement pour le client et affichage de ces moyens**

Une image contenant texte, capture d’écran, conception

Description générée automatiquement

Nous avons maintenant une application permettant à un client de s’enregistrer, de se connecter, d’afficher les informations le concernant, de consulter la liste des spectacles, ainsi que le détail d’un spectacle. Mais, le client ne peut pas acheter de billets. Pour acheter des billets, il va falloir enregistrer des cartes bancaires. Nous voudrons également afficher les informations de ces cartes dans la vue Clients/Profil.

1. Complétez la migration **V1\_4\_AjoutMoyenPaiement.sql**.
   1. Créez une table **CarteBancaire** faisant partie du schéma **Client**. Les informations sensibles de cette table doivent être chiffrées. Les champs sont les suivants : **CarteBancaireID, Numero** (donnée chiffrée), **Echeance** (donnée chiffrée), **ClientID** (clé étrangère)
   2. Créez une table **CarteBancaireEnClair** (voir modèle logique), qui nous permettra de récupérer les données déchiffrées des cartes bancaires pour les afficher dans la vue **Clients/Profil**.
   3. Complétez la procédure **USP\_AjouterCarteBancaire** (paramètres imposés).
   4. Complétez la procédure **USP\_RecupererCarteBancaire** (paramètres imposés). Cette procédure doit retourner les données sous la même forme que la table **CarteBancaireEnClair** (soit une colonne *Numero* et une colonne *Echeance*).
   5. Les insertions dans la table **CarteBancaire** sont fournies. Vous pouvez en ajouter d’autres si vous voulez
   6. Lancez la migration puis faites le scaffold.
2. Ouvrez le fichier **ProfilClientViewModel.cs** et décommentez la propriété **Cartes**. Ce *ViewModel* nous permettra de passer les données pertinentes au profil. Notez que la propriété **Carte**s est une liste de **CarteBancaireEnClair**. Pourquoi en clair ? Tout simplement car si on allait chercher une liste de **CarteBancaire**, nous aurions les données chiffrées !
3. Modifiez l’action **Profil** dans le contrôleur **Client** pour envoyer un objet de type **ProfilClientViewModel**.
   1. Normalement, vous avez déjà le client. Il vous suffit donc d’aller chercher une liste de **CarteBancaireEnClair** grâce à la procédure **USP\_RecupererCarteBancaire** (Rencontre 15).
   2. Adaptez l’objet **ProfilClientViewModel** créé à l’étape 6 et envoyez-le à la vue.
4. Dans la vue **Clients/Profil**, ajoutez un tableau pour afficher les cartes bancaires du client.

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

Description générée automatiquement