**Wrap-Up R22**

On récapitule tout !!

Le but de la séance (et de votre travail à la maison) est de faire un grand labo récapitulant toutes les notions que nous avons vu ensemble depuis l’examen intra avant d’attaquer le projet final.

Nous n’allons pas démarrer le projet à partir de zéro (vous aurez amplement le temps de vous confronter à cette problématique lors du projet final), mais nous allons passer en revue les différentes 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. En vue de votre projet de fin de session, l’idée est de faire pratiquer la débrouillardise en vous montrant comment utiliser, réutiliser et adapter le code auto-généré par les outils de VisualStudio.

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

Cette étape préliminaire vous permet de finir de mettre en place les fondements de l’application. La structure fondamentale du projet VisualStudio est déjà présente, mais vous allez devoir compléter les éléments au fur et à mesure. Vous devez compléter les étapes suivantes :

* Créer la Base de Donnée R22\_Billeterie (**Rencontre 17**),
* Ajouter le code pour permettre l’utilisation de FILESTREAM avec la base de données (**Rencontre 19**).
* 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.

Mettez dans un coin les deux commandes PowerShell suivantes :

*evolve migrate sqlserver -c "Data Source=.\SQLEXPRESS;Initial Catalog=R22\_Billeterie;Integrated Security=True;Persist Security Info=False;Pooling=False;Encrypt=False" -s Clients -s Spectacles --target-version x.x*

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

Remarquez que la commande evolve permet la création de deux schémas : Clients et Spectacles. Assurez-vous que ces schémas sont bien créés si jamais vous faites des tests directement dans SSMS.

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

Une image contenant texte, capture d’écran, Police

Description générée automatiquement

Dans cette étape, nous allons mettre en place la table Client, ainsi que le contrôleur et les vues associés. La table client va contenir les informations d’identification pour l’application. Enregistrer un utilisateur va permettre de demander les informations suivantes : prénom, nom, courriel et mot de passe. L’information demandée pour s’identifier sont les suivantes : courriel et mot de passe. Vous devez compléter les étapes suivantes :

* Compléter le fichier **V1\_0\_\_CreationDesClients.sql**. Il comporte les étapes suivantes :
  + Création de la table Client
  + Ajout de la contrainte d’unicité des courriels
  + Ajout de la procédure stockée pour enregistrer un nouvel utilisateur. Les paramètres sont les suivants (vous pouvez reprendre ce que nous avons vu au cours de la **rencontre 18**) :
    - Nom
    - Prénom
    - Courriel
    - Mot de Passe
  + Ajout de la procédure stockée identifier un utilisateur sur l’application. Les paramètres sont les suivants (vous pouvez reprendre ce que nous avons vu au cours de la **rencontre 18**) :
    - Courriel
    - Mot de Passe
  + Les insertions initiales sont déjà données
* Lancez la commande *evolve*, puis *scaffoldez* vos données.
* Le contrôleur **Clients** et les vues associées sont déjà fourni. Trois vues sont fournies : Inscription, Connexion et Index. Quatre actions sont fournies dans le contrôleur : **Index**, **Inscription**, **Connexion** (GET et POST) et **Deconnexion**. Vous avez juste à compléter le code des actions **Inscription** et **Connexion** pour faire fonctionner l’ensemble. Les *ViewModels* associés sont déjà prêt. Attention à la propriété que vous envoyez dans le cookie ! Regardez l’action **Index**.
* Veuillez noter que dans **program.cs**, la fonction **UseEndPoints** 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. Pour tester que tout fonctionne faite les actions suivantes :
  + Vérifiez que lors de la première connexion à l’application vous voyez la mention : « Bonjour visiteur ».
  + Allez sur la page de connexion et connectez-vous avec un des utilisateurs enregistrés. Le retour à l’index devrait afficher la phrase suivante : « Bonjour Chantal Vallière » si vous avez utilisé la cliente Chantal Vallière.
  + Déconnectez vous.
  + Allez sur la page d’enregistrement d’un utilisateur et enregistrez-en un.
  + Revenez sur la page de connexion et vérifiez que vous êtes capables de vous connecter avec ce nouvel utilisateur (et vérifier votre base de données avec SQL Server).

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

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

Description générée automatiquement

Suivez les étapes suivantes :

* Complétez la migration **V1\_1\_\_AjoutSpectaclesEtBillets.sql**. Les insertions sont données. Les éléments à compléter sont les suivants :
  + Créez les tables **Spectacle**, **Representation** et **Billet** (toutes font partie du schéma Spectacles). La colonne **Prix** dans la table **Spectacle** représente le prix d’UNE place. La colonne **CoutBillet** dans la table **Billet** représente le nombre de billets achetés multiplié par le prix d’une place.
  + Complétez les contraintes de clés étrangères
  + On ajoute les contraintes suivantes :
    - La colonne DateHeureRepresentation prend la valeur de GETDATE() par défault
    - Le prix du billet est positif ou nul
    - Le nom du spectacle est unique
* Lancez la migration puis scaffoldez vos nouvelles tables
* Pour vous simplifier la vie, générez automatiquement un contrôleur et des vues Razor pour le modèle **Spectacle** (**Rencontre 15**).
* Modifiez les paramètres de l’appel à *app.UseEndPoint* dans **program.cs** pour que nous arrivions directement sur la page **/Spectacles/Index** par défault.
* 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’information sur les spectacles**

Suivez les étapes suivantes :

* Complétez la migration 1.2. Il s’agit d’ajouter une vue 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)
  + Nombre de billets vendues
  + Le prix d’une place
* Appliquez la migration et scaffoldez vos modèles.
* Copiez-collez la vue Razor **Spectacles/Index** en **Spectacles/IndexAncien**. Faites de même pour l’action **Index** dans le contrôleur **Spectacles**.
* Adaptez la vue **Spectacles/Index** et l’action correspondante de manière à afficher les informations de la vue **VW\_SpectaclesRepresentationsSpectateur** :
  + Pensez à changer le modèle à la première ligne de la vue.
  + Ajoutez colonnes nécessaires pour afficher les informations envoyées par la vue SQL.
  + Ne gardez que les liens vers la vue **Details**.

Une image contenant texte, capture d’écran, 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 de représentation des spectacles. Il va falloir faire les étapes suivantes :

* Complétez la **migration 1.3** :
  + Ajoutez une table Affiche ayant les champs suivants : AfficheID, Identifiant, AfficheContent, SpectacleID (Rencontre 19)
  + Liez les tables **Affiche** et **Spectacle** par un lien de clé étrangère
  + Insérez des données dans cette table (le code des insertions est donné). **Attention : pensez à changer le chemin amenant vers les images des affiches pour que les insertions fonctionnent !**
  + Lancez la migration et scaffoldez les nouveaux modèles.
* 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 :
  + 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 crée à l’étape précédente
    - Une liste de représentations
    - Une chaîne de caractère qui va contenir l’image à afficher
  + Copiez la vue Razor **Spectacles/Details** et renommez la copie **Spectacles/DetailsAncien.cshtml**
  + Copiez-collez l’action **Details** dans le contrôleur et renommez une des deux copies **DetailsAncien**
  + 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.
    - Contentez vous de faire une requête par attribut du *ViewModel* (une pour la vue SQL, une pour les représentations et une pour l’affiche)
    - Construisez une instance du *ViewModel* et passez la en paramètre de *return View().*
  + Modifiez la vue Razor **Spectacles/Détails** de façon à afficher tous les détails du *ViewModel* :
    - Changez le modèle à la première ligne du fichier.
    - Adaptez les *DisplayNameFor* et *DisplayFor* à ce nouveau modèle.
    - Ajoutez un champ pour afficher l’image (copier-coller un des couples *dt* et *dd* et faites les modifications nécessaire). Pour afficher l’image, vous aurez juste à remplacer la ligne @Html.DisplayFor par un tag <img>. Je vous laisse remplir le tag vous-même (**Rencontre 19**).
    - Lancez votre application une première fois pour vérifier que tout fonctionne lorsque vous cliquez sur un lien *Détail*.
    - Créez un tableau (prenez modèle sur la façon dont la page Index est construite) pour afficher la date des représentations. Pour l’instant, votre tableau n’aura qu’une seule colonne.

Une image contenant texte, capture d’écran

Description générée automatiquement

**Étape 5 – On réintroduit les clients**

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

* 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

Description générée automatiquement

* Dans la vue **Clients/Connexion**, changez le lien définit à la ligne 2 de façon à renvoyez à la vue **/Spectacles/Index**
* Même chose pour la vue **Clients/Inscription**
* Dans le contrôleur **Clients**, modifiez les actions de façon que les renvois vers la vue **Clients/Index** se fasse plutôt vers la vue **Spectacles/Index** (Inscription en mode POST, Connexion en mode POST et Deconnexion).
* Dernier petit raffinement, nous voulons afficher *Bonjour visiteur !* ou *Bonjour Prenom 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, Police, nombre

Description générée automatiquement

* Assurez que tout fonctionne ! Refaites un tour 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. Suivez le guide :

* Création de la vue Razor **Clients/Profil** :
  + Copiez-Coller la vue razor **Spectacles/DetailsAncien** en **Clients/Profil**.
  + Dans la vue Razor, changez le modèle passé à la vue pour la classe **Client**.
  + Changez *ViewData[‘’Title’’]* pour que le titre de la page soit ‘’Profil’’.
  + Changez le titre *h1* pour Profil.
  + Changez le titre *h4* pour Client.
  + Changez les champs affichés pour montrer les informations du **Client**.
  + Enlevez le lien vers **Spectacles/Edit** et ajustez le lien vers Index pour qu’il vous redirige vers **Spectacles/Index**.
* Création de l’action **Profil** dans le contrôleur **Client** :
  + Copiez-Coller l’action **DetailsAncien** du contrôleur **Spectacles** dans le contrôleur **Clients** et renommez la Profil.
  + Rajoutez l’annotation **[Authorize]** à cette action.
  + 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.
  + Adaptez le code de cette action pour envoyer le bon **client** à la vue Razor. Au cas où un petit malin arrive quand même à exécuter l’action sans être authentifié, vous le renverrez vers la vue **Spectacles/Index**.
* Ajout d’un lien vers le profil dans **Spectacles/Index** :
  + L’idée est d’afficher ce lien uniquement si le client est connecté.
  + Pour y arriver, un simple test sur la valeur de ViewData[‘’PrenomNom’’] fera l’affaire.
* Les deux images ci-dessous vous montre les vues **Spectacles/Index** lorsqu’un utilisateur est connecté et la vue **Client/Profil**.

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. Et 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.

* Complétez la migration **V1\_4\_AjoutMoyenPaiement.sql**. Cette migration nous permettra d’avoir le modèle complet.
  + Créez une table **CarteBancaire** faisant partie du schéma **Client**. Les informations sensibles de cette table doivent être chiffrée. Les champs sont les suivants :
    - CarteBancaireID
    - Numero (donnée chiffrée)
    - Echeance (donnée chiffrée)
    - ClientID (clé étrangère)
  + Créez une table **CarteBancaireEnClair**, qui nous permettra de récupérer les données déchiffrées des cartes bancaires pour les afficher dans la vue **Clients/Profil**.
  + Complétez la procédure **USP\_AjouterCarteBancaire** (paramètres imposés).
  + Complétez la procédure **USP\_RecupererCarteBancaire** (paramètres imposés). Cette procédure doit retourner ses données sous la même forme que la table **CarteBancaireEnClair** (soit une colonne *Numero* et une colonne *Echeance*).
  + Les insertions dans la table **CarteBancaire** sont données. Vous pouvez en ajouter d’autres si ça vous tente.
  + Exécuter la migration et scaffoldez vos modèles.
* Ouvrez le fichier **ProfilClientViewModel.cs** et décommentez les lignes définissant les attributs de cette classe. Ce *ViewModel* nous permettra de passer les données pertinentes au profil. Notez que le premier attribut est un objet **Client**, tandis que le deuxième est une liste de **CarteBancaireEnClair**. Pourquoi en clair ? Tout simplement car si on allait chercher une liste de **CarteBancaire**, nous aurions les données chiffrées !
* Modifiez l’action **Profil** dans le contrôleur **Client** pour envoyer un objet de type **ProfilClientViewModel**.
  + 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).
  + Construisez un objet **ProfilClientViewModel** et envoyez-le à la vue.
* Dans la vue **Clients/Profil**, ajoutez un tableau pour afficher les cartes bancaires du client. Pensez à changer le modèle et à réadapter le code existant !

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

Description générée automatiquement

**Étape 8 – Achat de billets par un client**

Nous sommes rendus à l’étape ultime, celle où notre client va acheter des billets. L’idée est qu’une fois qu’un client s’est rendu sur le détail d’un spectacle, un lien apparaisse pour acheter des billets seulement s’il est identifié au site, qu’il puisse réserver le nombre de billets qu’il veut (la salle de spectacle a les mêmes propriétés que la salle sur demande à Poudlard). Suivez les étapes suivantes :

* Modifiez la vue **Spectacles/Détails** pour qu’elle affiche un lien vers l’achat de billets si l’utilisateur est connecté et à défaut un lien vers la page de connexion des utilisateurs.
  + Ajoutez une colonne **Action** au tableau affichant les représentations. Faites un test sur la valeur de ViewData[‘’PrenomNom’’] pour savoir quel lien doit être afficher.
  + Si le lien est celui invitant à se connecter, alors il doit rediriger vers la vue **Clients/Connexion**.
  + Si le lien est celui permettant d’acheter des billets, alors il doit rediriger vers la vue **Billets/Achat** (cette vue et ce contrôleur n’existent pas encore, nous allons bientôt nous en occuper). Pensez à fixer le paramètre **asp-route-id** avec *RepresentationId*.
* Modifiez l’action **Détails** du contrôleur **Spectacles** pour fixer la valeur de *ViewData[‘’PrenomNom’’]* si l’utilisateur est connecté (vous avez déjà ce code dans une autre action, copiez le !!).
* Les images ci-dessous vous illustrent à quoi doit ressembler la page **Details** avec un utilisateur connecté, et un visiteur.

Une image contenant texte, capture d’écran, Page web, Site web

Description générée automatiquementUne image contenant texte, capture d’écran

Description générée automatiquement

* Maintenant que c’est fait, il est temps de réfléchir à ce que nous voulons faire avec notre duo Action et Vue **Achat**. Plusieurs façons sont évidemment possibles, mais nous allons essayez de nous simplifier la vie.
  + Dans un premier temps, auto-générez un contrôleur et des vues pour le modèle **Billet**.
  + Supprimez toutes les vues générées, sauf **Create**.
  + Supprimez toutes les méthodes du contrôleur, sauf les deux méthodes **Create**.
  + Renommez la vue et les méthodes **Create** en **Achat**.
  + Rendez l’action **Achat** en mode GET accessible uniquement si l’utilisateur est connecté avec l’annotation **[Authorize]**.
* Modification de l’action **Create** (GET) :
  + Nous allons tout changer dans cette méthode. L’idée est qu’avec un clique sur le lien, nous savons déjà quelle représentation et quel client sont concernés par la transaction, ainsi que le prix d’une place. Nous allons les passer grâce à **ViewData**.
  + Changez le type de retour pour *public async Task<IActionResult>*.
  + Ajoutez un paramètre de type entier *id* à la méthode. Ce paramètre permettra de récupérer l’id de la représentation pour laquelle on veut acheter des billets.
  + Récupérer *ClientId* grâce au cookie.
  + Récupérez le prix d’une place. Pour y arriver, vous pouvez faire une première requête permettant d’avoir SpectacleID, puis une deuxième permettant de récupérer le prix de la place.
  + Mettez ces valeurs dans *ViewData[‘’RepresentationId’’]*, *ViewData[‘’ClientId’’]* et *ViewData[‘’Prix’’]*.
* Modification de la vue Razor **Billets/Achat** :
  + Remplacer les champs liés à RepresentationId, ClientId et CoutBillet par les lignes suivantes :



* + Pensez à remplacer toutes les mentions de **Create** par **Achat**
* Modification de l’action **Create** (POST)
  + Pour faire simple, enlevez la vérification du *ModelState*
  + Enlevez toutes les lignes en dessous du bloc if.
  + Remplacez la valeur de l’attribut *CoutBillet* par *CoutBillet\*NbtBillet*.
  + Remplacez les paramètres de la méthode *RedirectToAction* pour pouvoir revenir à la vue **Spectacles/Index**.
  + Tout devrait marcher !! Vous devriez voir que le nombre total de billets vendu a changé.

**Étape 9 – Si vous voulez approfondir**

Nous pouvons penser à bcp d’autres améliorations possibles :

* Un client devrait pouvoir ajouter ou supprimer un moyen de paiement par lui-même.
* On pourrait vouloir ajouter un spectacle, puis ajouter des représentations à l’intérieur du début de l’affiche et de la fin de l’affiche.
* On pourrait accorder une réservation uniquement si un moyen de paiement a été enregistré.
* On pourrait vouloir afficher la recette totale d’un spectacle.
* On pourrait afficher l’ensemble des billets réservés par un client.
* On pourrait afficher la recette totale d’un spectacle.

Bref, il y a bcp de choses que je vous laisse explorer par vous-même.