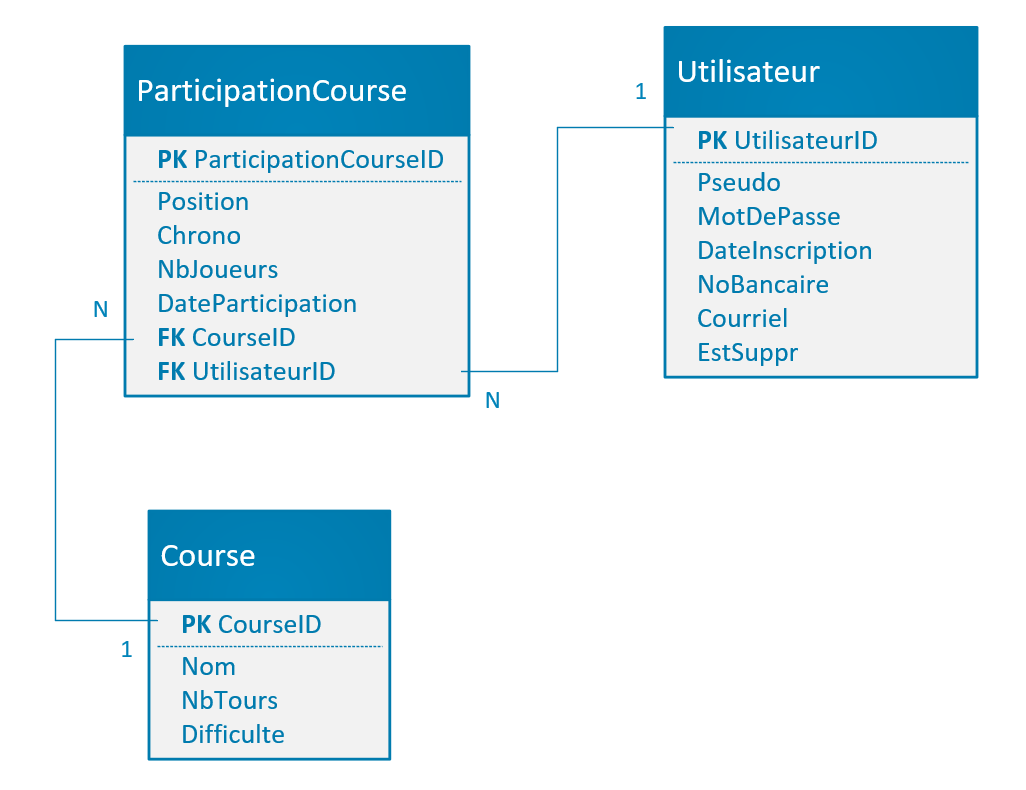
**TP2 - Partie 1 (15%)**

(Notions des semaines 4 à 10)



**Consignes :**

* Ceci est la première partie de deux évaluations. (**TP2 partie 1** et **TP2 partie 2**) La deuxième partie vaut **25%** de la note finale et vous continuerez de travailler sur le projet que vous avez fait dans la partie 1.
* Le **TP2** est à faire de manière individuelle. Attention au plagiat.
* Le **TP2 partie 1** vise surtout à réviser les notions de la **semaine 9** et **10**, mais les semaines 4 à 7 du cours seront encore pratiquées vu que le côté Database-First de l’application nous amène à rédiger du SQL.
* La page « Amis » n’existe pas encore : c’est normal. Ce sera pour la partie 2.
* Il faudra utiliser **GitKraken** pour le TP2. Donnez accès au répertoire **PRIVÉ** de votre TP2 à votre enseignant avant le 15 avril pour ne pas perdre de points. ([chantal.vallieres@cegepmontpetit.ca](mailto:chantal.vallieres@cegepmontpetit.ca) ou [maxime.pelletier@cegepmontpetit.ca](mailto:maxime.pelletier@cegepmontpetit.ca)) Vous êtes tenus de créer des commits en quantité stratégique avec des descriptions claires. <https://info.cegepmontpetit.ca/git> Nous utiliserons tout de même la remise sur Léa, mais nous inspecterons votre répertoire Git.

**Mise en situation :**

* L’application Web **Sussy Kart** propose un petit jeu de course codé en JavaScript.
* Le site met à disposition des statistiques sophistiquées (« Scoreboards ») sur le jeu et les participations. Un utilisateur authentifié peut aussi voir les données de son profil.
* Lorsqu’on s’inscrit, on doit fournir son numéro de compte bancaire. (Parfois les joueurs gagnent des prix en argent qui sont virés dans leur compte de banque s’ils battent des records... 🙄) C’est toutefois une donnée sensible que la base de données devra chiffrer.
* Pour l’instant, il y aura 6 courses, 8000 utilisateurs et 25000 participations dans la base de données. Cela augmentera sûrement dans la partie 2 lorsque la *performance* sera évaluée.

**1 - Migrations versionnées avec Evolve**

Tous les changements que vous ferez à la BD devront faire partie de migrations standardisées exécutables avec Evolve. Notez qu’il est interdit de modifier les migrations **1.0** à **1.3**. (Rien ne vous empêche de « supprimer » ou modifier des objets SQL créés dans ces migrations, mais il faudra le faire à l’aide de nouvelles instructions SQL dans les migrations suivantes !) Assurez-vous que si on ouvre votre projet avec une BD vide, toutes les migrations fonctionnent à l’aide de la commande Evolve.

**TP2 partie 1** : Les versions des migrations seront **2.0**, **2.1**, etc.

**TP2 partie 2** : Les versions des migrations seront **3.0**, **3.1**, etc.

**2 - Models et DbContext générés par scaffolding**

Tous vos **Models** et votre **DbContext** devront être générés grâce à la commande **dotnet ef scaffold** que nous avons abordée. Vous ne pouvez pas modifier les **Models** et le **DbContext** qui ont été générés par cette commande. Lors de la correction, le **DbContext** et les **Models** doivent déjà être présents dans le projet. (La seule chose à « ajouter » par le correcteur sera l’exécution des migrations)

**3 - Gestion des utilisateurs**

Une gestion des utilisateurs basique devra être implémentée :

* Il faudra pouvoir s’**inscrire** (avec procédure stockée), se **connecter** (avec procédure stockée) et se **déconnecter** de l’application.
  + Il y a déjà une procédure pour créer un utilisateur, mais il faudra l’écraser dans une nouvelle migration.
* Le schéma logique au début de ce document illustre bien les 7 données de la table Utilisateur.
* 8000 utilisateurs sont déjà inscris, (Migration 1.2) mais ils n’ont pas de **mot de passe** et de **numéro de compte bancaire** car la table Utilisateur est incomplète...
  + Modifiez la table utilisateur pour ajouter les données manquantes.
  + Tout hachage, chiffrement et déchiffrement doit se faire en SQL.
  + Pour les 8000 utilisateurs existants, vous pouvez remplir leur mot de passe avec « patate » et leur numéro de compte bancaire avec « 123456789 ». Pour les futures inscriptions, l’utilisateur fournira ces données lui-même.
    - Assurez-vous de quand même sécuriser ces deux données pour les utilisateurs pré-existants.
    - ATTENTION ! Si jamais les mots de passe en clair transmis à la BD (Lors de l’inscription et de la connexion) sont des paramètres de type **nvarchar** (plutôt que **varchar**), lorsque vous allez **update** les utilisateurs actuels pour leur donner le mot de passe **'patate'**, utilisez plutôt **N'patate'**. Le N devant la valeur permet d’encoder en nvarchar.

**4 - Participation**

Lorsqu’un joueur authentifié complète une partie, un bouton apparait sous la course et permet de soumettre son score à l’application. (Seule la participation du joueur 1 est soumise... Donc le personnage rouge. Les autres personnages sont juste des « invités » dont on ne garde pas les données) Le bouton est déjà fonctionnel et envoie un « **ParticipationViewModel** » à une action du contôleur **Jeu**, qu’il faudra compléter. Vous remarquez qu’une participation est liée à une **course** et à un **utilisateur**. Vous devez utiliser une procédure stockée pour cette insertion.

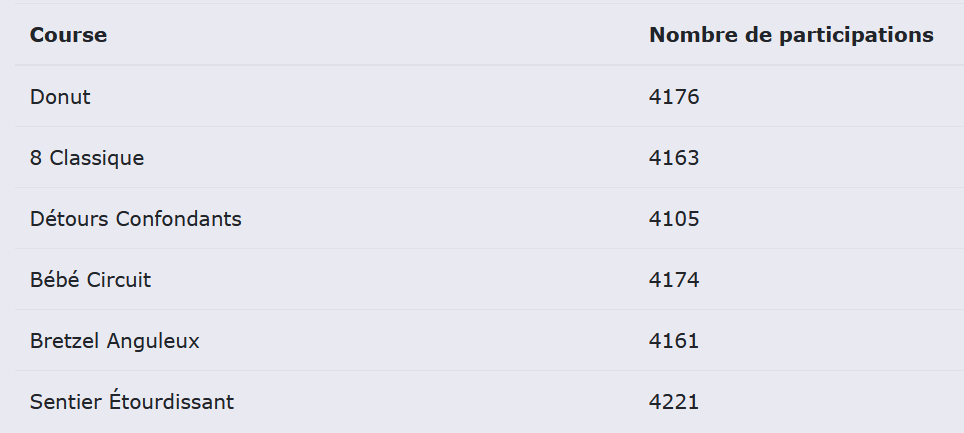
**5 - Statistiques (Scoreboards)**

Au total, il y a 7 pages de statistiques. Vous devrez rendre fonctionnelles 4 pages sur 7. Si vous complétez plus de 4 pages, nous allons tout de même retirer des points s’il y a des erreurs dans certaines statistiques, donc prudence !

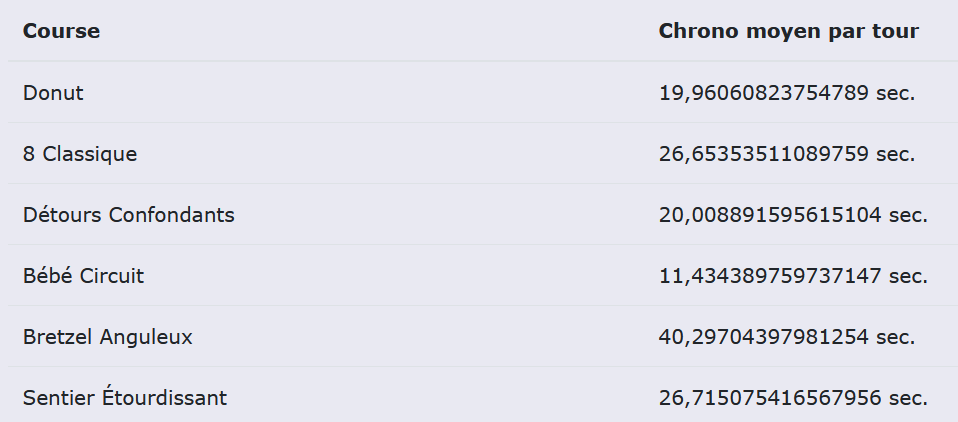
Vous n’avez pas le droit d’utiliser de Vue SQL, sauf pour la page « **Toutes les participations** », qui le demande explicitement. Pour autres pages, utilisez LINQ et créez un **ViewModel** au besoin.

⭐ Merci de mettre style="filter:grayscale(100%);" sur les trois images qui mènent vers les pages non complétées. (Ça rend les images noires et blanches)

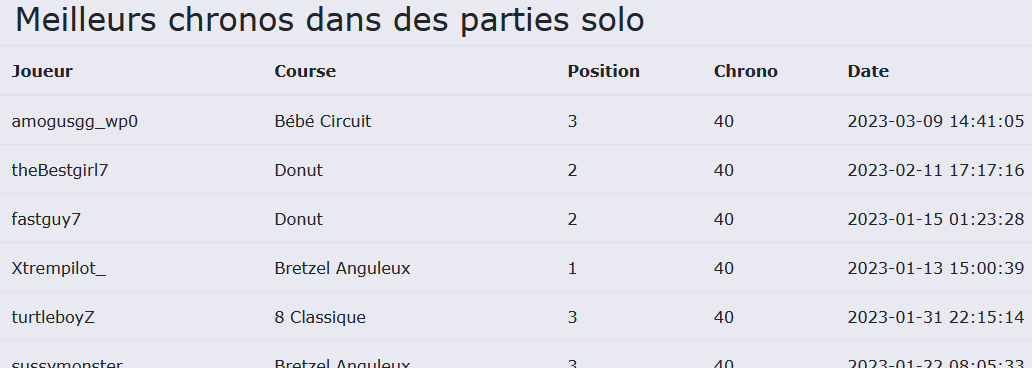
* La page « **Toutes les participations** » doit absolument être fonctionnelle.
  + Il faudra créer une vue SQL qui gère les jointures pour nous pour l’affichage de ces statistiques, ce qui simplifiera le code dans le contrôleur. (Il restera juste à faire des .Where, des .Skip, des .Take et des .OrderBy avec LINQ)
  + On ne doit voir que 30 participations à la fois dans la page. Trouvez le moyen de pouvoir consulter les participations, page par page.
  + N’hésitez pas à modifier FiltreParticipationVM.cs pour y ajouter une liste qui contient les données à afficher.
  + Un système de filtre est déjà partiellement mis en place, il doit être fonctionnel.
* Choisissez une de ces deux pages :
  + « **Participations par course** »
    - Pour chacune des 6 courses existantes, on veut afficher combien d’utilisateurs distincts ont participé. Exemple : un total de 2840 joueurs différents ont joué au moins une fois le circuit « Donut »



* + « **Chrono moyen par course par tour** »
    - Pour chacune des 6 courses existantes, on veut afficher le temps moyen, en secondes, pour compléter un tour de la course. (Attention, les circuits n’ont pas toujours que 3 tours) Exemple : le temps moyen par tour pour le circuit « Donut » est 8 secondes.



* Choisissez une de ces deux pages :
  + « **Meilleurs chronos solo** »
    - On affiche les participations avec les 30 meilleurs chronomètres générés lors de parties solos. (Parties à 1 joueur)



* + « **Meilleurs chronos à 4** »
    - On affiche les 30 participations avec les meilleurs chronomètres pour les parties à 4 joueurs.



**6 - Affichage du profil**

Seul un utilisateur authentifié peut aller sur la page **Profil**. Cette page affiche les informations de l’utilisateur authentifié qui la consulte. Le numéro de compte bancaire doit être affiché en clair.

**Grille de correction**

|  |  |  |
| --- | --- | --- |
| Section | Critères | Points |
| Migrations | - Toutes les migrations sont fonctionnelles avec la commande Evolve.  - Les migrations sont bien versionnées et segmentées. (Tout n’est pas glissé dans une même migration) | 1  1 |
| Models et DbContext | - Les Models et le DbContext sont présents et sont compatibles avec la base de données. | 1 |
| Utilisateur | - Inscription  - Connexion  - Déconnexion  - Protection des mots de passe et des numéros de compte bancaire | 3  2  1  5 |
| Participation | - On peut générer de nouvelles données de participations valides. | 3 |
| Statistiques | Trois pages de statistiques ont été implémentées et les données sont exactes :  - Page avec toutes les participations filtrées  - Page au choix 1  - Page au choix 2 | 4  3  3 |
| Profil | - Le profil affiche bien les données de l’utilisateur qui consulte la page. | 3 |
| Git | - Git n’a pas été utilisé.  - L’enseignant n’a pas été ajouté en tant que collaborateur à temps.  - La quantité de commits est trop petite. (Nous ne jugerons pas le moment des commits, juste la quantité. Si vous êtes à la dernière minute, mais que vous faites bel et bien un commit par fonctionnalité / section du barème, ça va !) | -5 |
| Total | | 30 |

**Et dans la partie 2 ?**

En gros, nous allons ajouter des images sur le site, optimiser la performance de l’application, créer des tests et ajouter quelques migrations supplémentaires (pour la gestion des amis) qui impliquent entre autre un déclencheur vu qu’il n’y en a pas dans la partie 1.