# TP1 : JuliePro

## **Consignes (15% de la note finale)**

* Lisez toutes les instructions et la grille avant de commencer
* Créez un Repository **PRIVÉ** et ajoutez votre enseignant comme collaborateur
* Vous DEVEZ faire au moins les migrations et les *commits* demandés mais vous pouvez en faire plus sans problème, tant que vous les documentés correctement.
* Le dossier TP1\_fichiersComplementaires contient :
  + Images du layout et les photos des *Trainers*
  + Le \_Layout
  + Les fichiers de Boostrap.css et min.css
  + Les classes de départ INCOMPLÈTES
  + La View Home/Index incomplète
  + Les *Seed* pour intégrer les données de départ (enlever les commentaires au fur et à mesure que vous créez les classes modèles correspondantes)

## **Étude de cas JuliePro**

Julie Loiselle est propriétaire de l’entreprise d’entrainement personnel JuliePro. Elle offre à ses clients (customer) un suivi personnalisé des entrainements et des objectifs. L’entreprise emploi une douzaine d’entraineurs (trainer) chevronnés et spécialisés (Speciality : perte de poids, althérophilie, course, réabilitation, etc). Un Trainer a une seule Speciality.

Chaque client est assigné à un entraineur qui planifie des séances (ScheduledSession) à chaque semaine (environ 2 à 5 selon le client) qui sont effectuées parfois avec l’entraineur, parfois seuls. Le client peut visualiser les séances à venir ainsi que les séances passées et cocher une séance lorsqu’elle est complétée.

L’entraineur fait un suivi régulier et vérifie que ses clients suivent bien le programme et communique avec lui si les séances s’espacent. Chaque séance est d’un entrainement particulier (CrossFit, Yoga, Course, Step, etc). Chaque entrainement (training) est soit cardio, musculaire, étirement et nécessite 1 ou plusieurs équipements. Le même équipement peut être utilisé dans plusieurs entrainements.

L’entraineur défini également avec son client un objectif courant (objective) : soit de perte de poids ou de distance. Lorsqu’il est atteint, le client indique la date et peut en déterminer un autre avec son entraineur. Bien qu’il ne se concentre que sur un objectif à la fois, le client pourra donc visualiser la liste de ses objectifs atteints.

## **Étape 1 Compléter le diagramme de classes en fonction des informations contenues dans l’étude de cas :**

1. Ouvrez le fichier Visio *TP1 diagramme de classes*
2. Complétez le diagramme en ajoutant les liaisons et des classes (si nécessaire)

## **Modèle à compléter**



## **Étape 2 Créer les projets de la solution :**

1. À partir du *repository* cloné de GitHub Classrooms
2. Créer une branche à partir de **Main** et appelée **CreationInitiale**

Ne pas travailler directement dans la branche **Main**. Vous ajoutez une fonctionnalité à la fois.

1. Créez les projets nécessaires (*DataLayers*)
2. Faites les opérations nécessaires au bon fonctionnement (références, ajout de code, etc)
3. Dans le projet MVC, remplacez/ajoutez les fichiers suivants par ceux dans le dossier des FichiersComplementaires :
   * wwwroot\lib\bootstrap\dist\css\**bootstrap.css**
   * wwroot\lib\bootstrap\dist\css**\bootstrap.min.css**
   * Views\Shared\**\_Layout.cshtml**
   * Copiez dans **root/images/imagesLayout tile.png et JulioProImage.png**
   * wwroot\**favicon.ico**
   * Views\Home\**Index.cshtml**
4. Implémentez FontAwesome pour tous les boutons
5. Comment-Commit-Push
6. Ajoutez le code pour populer la BD (*Seed*) NOTE : Le code est partiellement en commentaire et sera décommenté au fur et à mesure
7. Créez le JulieProDbContext dans le projet JuliePro\_DataAccess, dans le dossier Data
8. Dans le projet **JuliePro\_DataAccess,** créez la classe **ModelBuilderGenerator** (même niveau de le DbContext donc dans le dossier Data)
9. Copiez le contenu du fichier **ModelBuilderGenerator**
10. Ajoutez le code suivant dans le fichier du **DbContext**

protected override void OnModelCreating(ModelBuilder modelBuilder)

        {

            base.OnModelCreating(modelBuilder);

            //Générer des données de départ

            modelBuilder.GenerateData();

    }

### Créer les *PartialView* pour les boutons des formulaires

1. Créez les *PartialView* pour afficher les boutons :

* Create et BackToList
* Update et BackToList
* Delete et BackToList

Référez-vous aux fichiers de laboratoiresi nécessaire.

#### Fusion des branches

1. Comment-Commit-Push
2. Fusionnez la branche **CreationInitiale** dans **Main**

## **Étape 3 Ajouter des fonctionnalités une par une :**

Vous ferez le développement du modèle en trois phases, à partir des classes. Vous devez ajouter annotations, certaines propriétés, et les classes intermédiaires nécessaires. Créez des *ViewModels* lorsque nécessaire.

NOTE : Bien que la charge de travail soit répartie en trois principales fonctionnalités, vous aurez à faire des liaisons entre certaines classes pour lier les fonctionnalités donc modifier des classes créées précédemment.

### Fonctionnalité des entraineurs et spécialités (*Trainers-speciality*) :

1. Créez la branche **FCT**\_**TrainerSpeciality**
2. Créez les classes en BRUN seulement, copiez le contenu à partir des fichiers complémentaires et complétez les éléments manquants
3. Ajoutez le nécessaire pour le respect des règles d’affaires (annotations) et les liaisons (clés, propriétés de navigation)
4. Faites migration/Update-Database
5. Dans le fichier ***ModelBuilderDataGenerator*** décommentez le contenu des sections suivantes :
   1. ***Speciality***
   2. ***Trainer***
6. Compilez
7. Si vous avez des erreurs dans le *Seed,* ajustez en fonction de vos classes modèles
8. Faites migration/Update-Database

**Trainer Index**

1. Pour compléter la fonctionnalité :
   1. créez le controller **Trainer,** l’action **Index**
      1. Liste des *Trainers* avec leur *Speciality*
      2. BONUS : Liste des *Trainers* triés par *LastName* et *FirstName*
   2. Créez la View **Index** qui doit afficher la liste des entraineurs avec leur spécialité
      1. Remplacer le contenu généré par le fichier complémentaire fourni Trainer\_View\_Index
      2. Complétez la View **Index**
2. Comment-Commit-Push

**Trainer Upsert**

1. Modifiez le controller **Trainer,** ajoutezl’action **Upsert**
   1. Créez la View **Upsert** qui doit afficher les informations d’un Trainer et une liste déroulante des spécialités (*Speciality*)
      1. Remplacer le contenu généré par le fichier complémentaire fourni Trainer\_View\_Index
      2. Complétez la View **Upsert** (utilisez les *PartialView* pour les boutons)
2. Comment-Commit-Push
3. Ajoutez le code nécessaire et modifiez la *View* **Upsert** afin d’utiliser *Summernote* pour avec un champ richtext pour la *Biography*
4. Comment-Commit-Push

**Trainer Delete**

1. Modifiez le controller **Trainer,** ajoutezl’action **Delete**
   1. Créez la View **Delete** qui doit afficher les informations d’un Trainer (vous pouvez copier-coller la View Upsert et la modifier)
      1. Remplacer le contenu généré par le fichier complémentaire fourni Trainer\_View\_Index
      2. Complétez la View **Upsert** (utilisez les *PartialView* pour les boutons)

#### Fusion des branches

1. Comment-Commit-Push
2. Fusionnez la branche **FCT**\_**TrainerSpeciality** dans **Main**

### Fonctionnalité des clients et objectifs (*Customer, objectives*) :

1. Créez la branche **FCT**\_**CustomersObjectives**
2. Les tables doivent être générées mais vous n’avez PAS à créer les *Controllers* et *Views* pour cette fonctionnalité pour l’instant.
3. Créez les classes en JAUNE seulement, copiez le contenu à partir des fichiers complémentaires et complétez les éléments manquants
4. Ajoutez le nécessaire pour le respect des règles d’affaires (annotations) et les liaisons (clés, propriétés de navigation)
5. Faites migration/Update-Database

#### Fusion des branches

1. Comment-Commit-Push
2. Fusionnez la branche **FCT**\_**CustomersObjectives** dans **Main**

### Fonctionnalité des session d’entrainement- (*ScheduledSession*) :

Les tables doivent être générées mais vous n’avez PAS à créer les *Controllers* et *Views* pour cette fonctionnalité pour l’instant.

1. Créez la branche **FCT**\_**ScheduledSession**
2. Créez les classes en VERT seulement modifiez les classes existantes si nécessaire (navigation)
3. Ajoutez le nécessaire pour le respect des règles d’affaires (annotations) et les liaisons (clés, propriétés de navigation)
4. Faites migration/Update-Database

#### Fusion des branches

1. Comment-Commit-Push
2. Fusionnez la branche **FCT**\_ **ScheduledSession** dans **Main**

| **Grille de correction: TP 1** | | | | | | |
| --- | --- | --- | --- | --- | --- | --- |
| **Composantes de la tâche et critères d’évaluation** | **Très bien** | **Bien** | **Passable** | **Insuffisant** | **Incomplet** | **Note** |
| Mise en place d’une application respectant le patron MVC   * DataLayers * Type de projets | L’application est répartie en plusieurs projets utilisant le type adéquat et correctement identifiés.  L'étudiant démontre une très bonne maîtrise des bonnes pratiques. | L’application est répartie en plusieurs projets utilisant le type adéquat et sont plus ou moins identifiés correctement.  L'étudiant démontre une bonne maîtrise des bonnes pratiques. | L’application est répartie en plusieurs projets utilisant le type adéquat et sont plus ou moins identifiés correctement.  L'étudiant démontre une maîtrise minimum des bonnes pratiques. | L’application est répartie en plusieurs projets ou ceux-ci n’utilisant pas le type adéquat et ne sont pas correctement identifiés correctement.  Et/ou l'étudiant ne démontre pas une maîtrise suffisante des bonnes pratiques. | L’application n’est pas répartie en plusieurs projets ou ceux-ci n’utilisant pas le type adéquat et ne sont pas correctement identifiés correctement.  Et l'étudiant ne démontre pas une maîtrise suffisante des bonnes pratiques. | /15 |
| 15-12 | 11-8 | 9 | 8-5 | 4-0 |
| Modèle de classe   * Classes * Annotations * Relations | Le modèle de données contient toutes les classes et attributs requis.  Les liaisons entre les classes sont toutes présentes et reflètent les besoins d’affaires.  Les annotations sont complètes et correspondent aux règles d’affaires.  L'étudiant démontre une très bonne maîtrise de la conception de modèle de données | Le modèle de données contient toutes les classes et attributs requis.  Les liaisons sont toutes présentes entre les classes reflètent les besoins d’affaires.  Les annotations sont presque complètes et correspondent aux règles d’affaires.  L'étudiant démontre une bonne maîtrise de la conception de modèle de données | Le modèle de données contient toutes les classes et attributs requis malgré quelques erreurs.  Quelques liaisons entre les classes ne reflètent pas les besoins d’affaires ou sont incomplètes.  Ou les annotations sont incomplètes et/ou ne correspondent pas toutes aux règles d’affaires.  L'étudiant démontre une maîtrise acceptable de la conception de modèle de données | Le modèle de données ne contient pas toutes les classes ou les attributs requis ou contient plusieurs erreurs.  Plusieurs liaisons entre les classes ne reflètent pas les besoins d’affaires ou sont incomplètes.  Et/ou les annotations sont incomplètes et/ou ne correspondent pas aux règles d’affaires.  Et/ou l'étudiant ne démontre pas une connaissance minimum du EF Core | Le modèle de données ne contient pas toutes les classes ou les attributs requis et/ou contient plusieurs erreurs majeures.  Les liaisons entre les classes ne reflètent pas les besoins d’affaires ou sont absentes.  Et/ou les annotations sont incomplètes et/ou ne correspondent pas aux règles d’affaires.  Et l'étudiant ne démontre pas une connaissance suffisante du EF Core. | /20 |
| 20-17 | 16-13 | 12 | 11-6 | 5-0 |
| Mise en place d’entity framework Core   * Connexion à la Base de données * DbContext * Migrations et updates | L’application contient une connexion fonctionnelle à la BD et un DbContext adéquat.  Le projet contient au moins les trois migrations demandées.  L'étudiant démontre une très bonne connaissance du EF Core | L’application contient une connexion fonctionnelle à la BD et un DbContext adéquat.  Le projet contient au moins les trois migrations demandées.  L'étudiant démontre une bonne connaissance du EF Core | L’application contient une connexion fonctionnelle à la BD et un DbContext comportant quelques erreurs.  Le projet contient au moins les trois migrations demandées.  L'étudiant démontre une connaissance du EF Core | L’applicationne ne contient pas une connexion fonctionnelle à la BD et/ou un DbContext comportant plusieurs erreurs ou manques.  Et/ou le projet ne contient pas les trois migrations demandées.  Et/ou l'étudiant ne démontre pas une connaissance minimum du EF Core | L’application ne contient pas une connexion fonctionnelle à la BD et un DbContext comportant plusieurs erreurs ou manques.  Et le projet ne contient pas les trois migrations demandées.    Et l'étudiant ne démontre pas une connaissance suffisante du EF Core. |  |
| 15-12 | 11-8 | 9 | 8-5 | 4-0 | /15 |
| Mise en place des Controllers, des Views, PartialViews et des ViewsModels adéquats :  **Views et Controllers complexes :**   * Partials Views * CRUD Trainer | Les Views et Controllers autogénérées sont adéquatement implémentées*.*  Les views et controllers complexes sont tous adéquatement conçus et sont pleinement fonctionnels.  L'étudiant démontre une très bonne maîtrise des concepts MVC . | Les Views et Controllers autogénérées sont adéquatement implémentées malgré quelques erreurs*.*  Les views et controllers complexes sont tous adéquatement conçus et sont fonctionnels  malgré quelques erreurs mineures.  L'étudiant démontre une bonne maîtrise des concepts MVC. | Les Views et Controllers autogénérées sont adéquatement implémentéesmais comportent plusieurs erreurs*.*  Les views et controllers complexes sont presque tous adéquatement conçus et sont fonctionnels  malgré quelques erreurs.  L'étudiant démontre une maîtrise minimum des concepts MVC et de l’implantation de librairies. | Les Views et Controllers autogénérées ne sont pas adéquatement implémentéesou comportent plusieurs erreurs*.*  Les views et controllers complexes ne sont pas tous adéquatement conçus et ne sont fonctionnels  ou contiennent des erreurs majeures.  Et/ou l'étudiant ne démontre pas une maîtrise suffisante des concepts MVC et de l’implantation de librairies. | Les Views et Controllers autogénérées ne sont pas adéquatement implémentéeset comportent plusieurs erreurs*.*  Les views et controllers complexes sont manquants.  Et l'étudiant ne démontre pas une maîtrise suffisante des concepts MVC et de l’implantation de librairies. | /25 | |
| 25-21 | 20-16 | 15 | 14-8 | 7-0 |
| Gestion de code source et documentation   * *Commit* documentés * Branches demandées * Identification des objets/fichiers/branches | La documentation accompagnant les *commit* est claire et adéquate et les branches sont correctement effectuées.  Les objets, fichiers et branches sont tous correctement identifiés.  L'étudiant démontre une très bonne maîtrise du gestionnaire de code source. | La documentation accompagnant les *commit* est claire et adéquate, à l’exception de quelques ambiguïtés et/ou les branches sont presque toutes correctement effectuées.  Les objets, fichiers et branches sont tous correctement identifiés.  L'étudiant démontre une bonne maîtrise du gestionnaire de code source. | La documentation accompagnant les *commit* plusieurs ambiguïtés et/ou les branches sont presque toutes correctement effectuées.  Et/ou les objets, fichiers et branches sont presque tous correctement identifiés.  L'étudiant démontre une maîtrise correcte du gestionnaire de code source. | La documentation accompagnant les *commit* plusieurs ambiguïtés et/ou les branches ne sont pas toutes correctement effectuées ou sont absentes.  Et/ou les objets, fichiers et branches sont presque tous correctement identifiés.  L'étudiant ne démontre pas une connaissance suffisante du gestionnaire de code source. | La documentation accompagnant les *commit* plusieurs ambiguïtés et/ou les branches ne sont pas toutes correctement effectuées ou sont absentes.  Et les objets, fichiers et branches sont incorrectement identifiés.  L'étudiant ne démontre pas une connaissance suffisante du gestionnaire de code source. | /15 |
| 15-12 | 11-8 | 9 | 8-5 | 4-0 |
| Qualité du français écrit  Termes techniques  Code orthographique et grammatical | L’étudiant démontre une très bonne maîtrise du français écrit. | L’étudiant démontre une bonne maîtrise du français écrit. | L’étudiant démontre une maîtrise minimum du français écrit. | L’étudiant ne démontre pas une maîtrise suffisante du français écrit. | L’étudiant ne démontre pas une maîtrise minimum du français écrit. |  |
| 10-9 | 8-7 | 6 | 5-3 | 2-0 | /10 |

| **Grille de correction: TP 1** | |
| --- | --- |
| Tâche : Application du JuliePro. Correction négative | |
| Retard : La note obtenue sera diminuée de 10 % (5pts) par jour de retard (tel que stipulé dans le plan de cours et la PIEA) pour un maximum de 5 jours, après ce délai le devoir sera considéré comme non remis et obtiendra la note de 0% |  |
| **Note finale** | **\_\_/100** |