# Énoncé du laboratoire

# UE Programmation réseaux, web et mobiles
# AA Programmation.Net

3e Bachelier en informatique
Haute École de la Province de Liège (HEPL)

Daniel Schreurs <daniel.schreurs@hepl.be>

2021 – 2022

## 1 Introduction

Ce travail est demandé dans le cadre de l'activité d'apprentissage (AA) "Programmation.Net" de l'unité d'enseignement (UE) "Programmation réseaux, web et mobiles" de la section 3e Bachelier en informatique de gestion, 3e Bachelier en informatique systèmes orientation informatique industrielle et 3e Bachelier en informatique systèmes orientation réseaux télécommunications.

### 1.1 Objectifs

L'objectif de ce projet est de vous familiariser avec une application qui combine différentes architectures afin de comprendre au travers d'un cas réel l'intérêt des architectures. De plus, ce projet vous permettra de renforcer vos connaissances de l'écosystème Dotnet. Enfin, nous mettons au centre du projet les tests et aimerions vous amener vers un développement piloté par les tests ainsi qu'une gestion avancée du code source d'un projet.

Ce projet est calibré pour un groupe de *2 étudiants*. Chacun devra être parfaitement capable d'expliquer le code développé par l'autre. La plateforme GitHub permet de publier le code source de vos projets. Elle doit être utilisée dès le début et régulièrement durant toute la réalisation de celui-ci. Nous aimerions ainsi vous habituer à l'outil de gestion de versions git. Nous voulons vous faire ressentir sa puissance et la facilité qu'il amène pour collaborer sur un même projet. Notez qu'on s'attend à retrouver des branches pour les fonctionnalités principales, des messages pertinents et structurés ainsi qu'une répartition des commits équilibrée au sein de votre équipe.

### 1.2 Obtenir son dépôt GitHub

1. Avant de commencer la procédure de création d'équipe, choisissez votre co-équipier;
2. Utilisez ce lien et sélectionnez votre nom dans la liste proposée;
3. Si vous êtes le premier étudiant de votre équipe: créez une nouvelle équipe en respectant le format suivant : `23XX-nom1-prenom1-nom2-prenom2`, par exemple : `2321-knuth-donald-lamport-leslie`[1].
4. Sinon, choisissez l'équipe existante créée par votre co-équipier;
5. Validez votre choix pour terminer la création du repository.

### 1.3 Conventions

Pour réaliser les étapes suivantes, il s'agit d'abord de consulter et lire les différentes ressources mises à votre disposition avant.

Vous devez créer une seule solution contenant plusieurs projets pouvant interagir. Des librairies de classes, des projets de tests unitaires, des projets ASP.NET (Web) ainsi qu'un projet WPF seront ajoutés à cette unique solution.

Nommez correctement la solution ainsi que les projets qu'elle contient. Tous les noms de fichiers, de classes, ainsi que vos commentaires, doivent nécessairement être en anglais. Aussi, il est indispensable, dès la création de votre solution, de la mettre sous contrôle de version ( cf. section Obtenir son dépôt GitHub). Cela attestera de la répartition du travail d'équipe fourni. Un dépôt avec un seul commit la veille de la remise, ne sera pas considéré comme une utilisation pertinente de l'outil git. Enregistrez donc régulièrement les états stables de vos projets. N'oubliez pas d'ajouter à la racine de votre dépôt un fichier `.gitignore` avant votre premier git add . (voir gitignore.io)

## 2 Technologies à mettre en œuvre

### 2.1 Entity Framework Core

#### 2.1.1 Ressources

- Transparents du cours de théorie : disponibles sur la plateforme Moodle;
- ce tutoriel , l'onglet *EF Basics* donne un complément d'information au cours de théorie;
- Documentation Entity Framework Core.

### 2.2 Création de la BD

Il s'agira de travailler avec une base de données SQLite que vous allez peupler à partir du fichier de films utilisé dans le cadre du cours de SGBD. Il est nécessaire d'avoir au moins 1000 films et leurs détails en utilisant le mécanisme *Code First* d'*Entity Framework Core*.

1. Créer une solution à partir de votre dépôt. Celle-ci contiendra tous les projets à la racinex.
2. Dans cette solution, créez un premier projet *librairie de classes (.NET Core)* destiné à contenir les classes de données ainsi que la classe dérivant de DbContext. Créez les classes en fonction des données qu'elles doivent contenir et qui doivent apparaitre dans la base de données. Par exemple :

[1] Il n'y a aucun espace blanc dans le nom d'équipe et aucune majuscule. Il n'y a que des tirets pour séparer les mots. Suivez la notation *kebab case*.

- Une classe *movie*,[2] qui permet au moins d'enregistrer le titre, la date de sortie, le score, la durée, le genre[3] du film;
- Une classe *actor*, qui permet au moins d'enregistrer le nom et le prénom de l'acteur.
- Une classe *comment*, qui permet au moins d'enregistrer un texte, une cote (valeurs possibles : 1 à 5) et la date d'un commentaire.

3. Ajouter *Entity Framework Core* et *Entity Framework Core SQLite* à partir du gestionnaire de projet *NuGet*.
4. Ajouter un projet *Console* dont l'objectif sera de :
   (a) Lire le fichier texte contenant les films;
   (b) Créer puis remplir la base de données correspondante avec au moins 1000 films;
5. Référencer la première librairie et intégrer les packages nécessaires pour pouvoir compiler et exécuter l'application.

### 2.3 Tests et vérifications

1. Créer un projet de tests unitaires *NUnit* permettant la vérification en bonne et due forme des méthodes CRUD de la DAL.
2. Vérifier que la base de données a bien été créée à l'aide de l'outil DataGrip (ou équivalent).
3. La base de données créée correspond-elle précisément à ce qui est attendu ?
4. Nous vous demandons de vérifier à l'aide de quelques requêtes SQL sauvegardées[4] que qfuelques films de référence soient bien enregistrés parmi les autres. En effet, certains ont quelques spécificités qui vous permettront de vérifier que votre base de données est cohérente et correspond à ce qui est demandé. Voici quelques exemples :
   - Star Wars (plusieurs épisodes);
   - Harrison Ford ou Tom Hanks (acteurs connus dans des films connus à repérer facilement dans quelques films.
   - Harrison Ford (dans Star Wars et Apocalypse Now);
   - Mrs Doubtfire (dans lequel Robin Williams joue plusieurs rôles).

[2] Les identifiants originaux présents dans le fichier texte doivent être conservés dans la BD.
[3] Il n'est pas souhaitable de stocker à chaque fois le genre dans le film. Les genres sont des entités connues à l'avance et doivent pouvoir évoluer sans intervention manuelle.
[4] Il faut conserver ces requêtes dans un fichier SQL. Nous pourrions les demander.

## 2.4 Architecture de projet

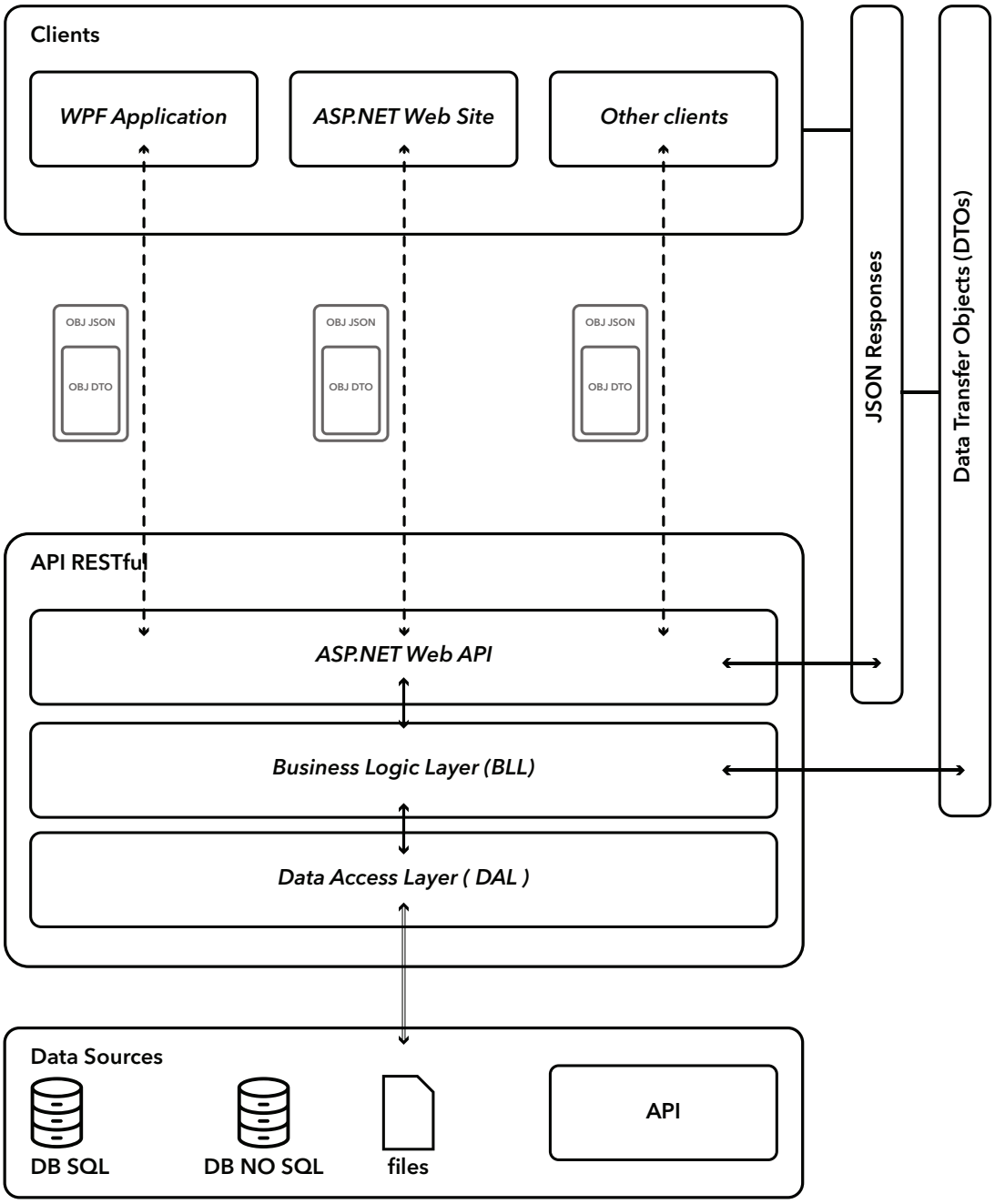


Figure 1: Architecture de référence

Cette architecture modulaire en couche a été présentée durant le cours de théorie. Elle doit être nécessairement respectée pour réussir ce laboratoire. Bien sûr, des améliorations peuvent être proposées, discutées préalablement et documentées dans votre `readme.md`.

### 2.4.1 Création des couches DAL, BLL, Web API et objets DTO

À ce stade du projet, vous avez déjà la couche d'accès aux données. Elle contient les classes utilisées pour créer les tables de la base de données ainsi que l'objet dérivant de *DbContext* donnant accès à la BD. Il s'agit maintenant de :

1. Créer une classe qui permet d'encapsuler cette couche d'accès à la BD et qui contient des méthodes

permettant d'effectuer des opérations de base sur celle-ci (sélection, création, modification, suppression). Toutes les méthodes ne doivent pas être écrites. Seules les méthodes nécessaires aux couches supérieures seront créées. On veillera à ne jamais charger en mémoire la totalité des données qui se trouvent en BD.

2. Créer une librairie de classe BLL qui représente la couche Business Logic Layer. Celle-ci ajoute une référence à la librairie DAL.

3. Créer une librairie de classes DTO (Data Transfer Object) qui contient la liste des objets DTO utilisés dans les couches suivantes (voir ci-dessous).

4. Créer, dans cette BLL, une classe qui permet d'encapsuler l'accès à la couche DAL. Elle contient des méthodes qui apportent de la logique métier à partir des méthodes de base. Les méthodes de la BLL apportent une plus-value à la méthode de base présente dans la couche DAL et utilise la notion d'objet DTO qui représente les données transférées de et vers les couches de niveau supérieur. Pour rappel, un objet de la DAL ne remonte pas en ligne directe vers la couche qui contient l'interface utilisateur puisque les données de la BD ne correspondent pas à l'utilisation dans les interfaces utilisateur. Toutes ces méthodes utilisent la technologie LINQ pour interroger les tables de la base de données. Les objets DTO à créer contiennent les données exclusivement réservées aux couches supérieures. Ajouter au moins ces méthodes suivantes :

   - `GetListActorsByIdMovie(...)` : récupère la liste des acteurs d'un film et retourne une List¡LightActorDT La classe LightActorDTO est une classe *légère* qui comprend toutes les données de la classe Actor, mais qui ne contient pas de liste de films dans lesquels l'acteur a pu jouer.

   - `GetListMovieTypesByIdMovie(...)` : récupère la liste des genres d'un film et retourne une `List<MovieTypeDTO>`. Un film peut être de plusieurs types, raison pour laquelle une collection est retournée.

   - `FindListMovieByPartialActorName(...)` : récupère la liste des films (`List<MovieDTO>`) dans lequel l'acteur dont on donne un nom partiellement ou entièrement a joué. La classe `MovieDTO` est une classe *légère* qui ne contient pas les détails des noms des acteurs ni des genres. Des paramètres sont ajoutés à cette méthode pour limiter le nombre d'objets retournés. On pense au fait qu'il doit être possible de paginer toutes les ressources.

   - `GetFavoriteActors(...)` : retourne la liste des acteurs ayant joué dans 2 films au moins. On créera une classe ActorDTO qui contient uniquement le nom et le prénom de l'acteur ainsi que le nombre de films auquel il aura participé. Cette méthode retourne une `List<ActorDTO>`[5]. Cette méthode trie par défaut sur le nombre de films des acteurs.

   - `GetFullMovieDetailsByIdMovie(...)` : récupère la totalité des données d'un film dont on donne l'Id. Cette méthode retourne un objet de la classe FullMovieDTO qui réutilise une classe déjà créée plus tôt et qui contient les détails en supplément.

   - `InsertCommentOnMovieId(...)` : insère un nouveau commentaire sur un film identifié par son Id. Les données doivent être initialisées dans les couches supérieures dans un objet `CommentDTO` passé en paramètre à cette méthode.

   - `GetComments(...)` : retourne la liste des commentaires validés.

   - `ValidateCommentById(...)` : Rend visible un commentaire identifié par son Id.

5. Créer un nouveau projet de type API (Choisir d'abord ASP .NET Core Web Application) configuré pour HTTP. Ce projet permet d'exposer sur le réseau au moins les méthodes créées dans la BLL au travers des routes HTTP telles que définies dans le cours de théorie. Prenez soin de vérifier que vous respectez les bonnes pratiques énoncées dans le cours théorique.

[5] Regardez ici les classes que vous avez déjà créées, l'une d'elles peut-elle servir ?

### 2.4.2 Tests et vérifications

1. Créer un projet de tests unitaires *NUnit* par couche, soit 2 projets.
2. Préparer des jeux de tests pertinents qui testent chacune des méthodes développées spécifiquement dans les couches BLL et WebAPI. On s'attend à un jeu de test utile qui couvre les cas limites. Par exemple, pensez aux problèmes liés au réseau ainsi qu'aux ressources non trouvées.

# 3 Application WPF

Il s'agit de créer une application WPF respectant le pattern MVVM.

1. La fenêtre principale présente les films triés par date et popularité de façon paginée dans un tableau. Dix films seront chargés à la fois. Un mécanisme *user-friendly* permet de parcourir les films 10 par 10. Le tableau comporte plusieurs colonnes qui seront personnalisées :
   - L'affiche,
   - Le titre : afficher une chaine de caractères,
   - La durée : Afficher celle-ci en heures et minutes,
   - Le genre : Grâce au ViewModel, ceux-ci sont convertis en une liste de petites images ou icônes faisant partie des ressources de l'application. Ces images/ icônes sont visualisées dans cette colonne du tableau.
   - Un call to action permet de faire apparaitre une boite de dialogue secondaire.
2. Une zone d'encodage de texte permet d'effectuer une recherche de films par nom partiellement encodé. Bien entendu, on s'attend ici à une recherche intelligente qui couvre les mêmes fonctionnalités que l'application web. (Voir la section Application Web ASP.NET).
3. La boite de dialogue secondaire est une boite modale qui permet d'afficher les détails d'un film ainsi que d'ajouter, visualiser et valider les commentaires associés à ce film. Elle reçoit, dans son constructeur, l'Id du film sélectionné.
   (a) Créer les composants permettant d'afficher les détails d'un film (pas le même tableau que dans la boite de dialogue précédente);
   (b) Ajouter un composant adapté permettant de visualiser les commentaires déjà encodés pour le film, ils sont triés par date.
   (c) Permettre la gestion des commentaires : valider[6] , invalider, supprimer et modifier un commentaire;
   (d) Si des commentaires sont associés au film, le niveau de popularité de celui-ci complété par le nombre de commentaires est affiché à côté de son nom et est basé sur la moyenne des rates des commentaires qui lui sont associés. Par exemple : 3 commentaires donnant une moyenne de 3.5 pour Forest Gump donnera un affichage : `FOREST GUMP 3.5 (3)`

   Remarque : il faut créer plusieurs classes `XxxViewModel`. Réfléchissez aux données à manipuler dans vos vues.

Note générale : on souhaite avoir des applications agréables à utiliser, soit une application *user-friendly*. Il ne faut pas réinventer la roue, mais s'inspirer des applications que vous utilisez. Votre application doit donc être un minimum ergonomique et respecter les conventions habituelles. Aussi, il nous semble évident

[6] Par défaut, les commentaires sont invalides. Il faut les valider pour les rendre visibles afin d'avoir un contrôle sur le contenu visible sur le web.

que l'utilisateur puisse redimensionner sa fenêtre sans casser l'apparence ainsi que la possibilité d'arriver de différentes manières à ses fins. Utilisez à bon escient les techniques vues en 2e.

# 4 Application Web ASP.NET

Le framework ASP.NET Core permet de créer des applications Web en implémentant le modèle MVC (Model View Controller). Il s'agit donc d'ajouter un nouveau projet Web du type *ASP.NET Web Application (.NET Core)* dans la solution principale. Ajoutez ensuite, les éléments de code (Controller + View + Modèle) permettant de gérer les éléments suivants :

- Page d'accueil : liste des films paginée. Les informations des films reprises dans cette liste sont :
  - Titre, affiche, runtime, vote average;
  - Une jauge représentant l'appréciation moyenne du film (déduites des commentaires attachés à celui-ci).

  Il est attendu d'afficher l'affiche du film que vous pouvez récupérer depuis l'API The Movie Database. Réfléchissez au moment et à l'endroit où vous gérer la récupération de cette image. Nous attendons une solution performante qui minimise le nombre de requêtes vers l'API The Movie Database.
- Page détail d'un film : liste toutes les informations relatives à un film ainsi que :
  - Ses commentaires s'ils ont été validés (Voir section Application WPF);
  - Le casting avec une photo de profil des acteurs récupérée depuis The Movie Database.
  - Il faut permettre de soumettre un nouveau commentaire. Une validation par HTML n'est pas suffisante, mais nécessaire pour l'expérience utilisateur.
- La page détail d'un acteur : vous n'avez que très peu d'information sur les acteurs dans le fichier de texte de départ. Vous devez donc récupérer toutes les informations depuis l'API The Movie Database. Pensez à une stratégie pour minimiser le nombre de requêtes.
- Un moteur de recherche qui s'applique aux films et acteurs. Il va de soi qu'une recherche partielle est permise et qu'elle n'est pas sensible à la casse ainsi qu'aux caractères spéciaux. En revanche, 3 mots en majuscules permettent de paramétrer la rechercher :
  - Le mot-clé **AND** permet d'exprimer un et logique;
  - Le mot-clé **OR** permet d'exprimer un ou logique;
  - Le mot-clé **NOT** permet d'exclure des éléments de la recherche;

  Vous êtes libre de choisir la syntaxe, en revanche, celle-ci doit être clairement documentée sur la page. Il parait donc logique que les résultats soient triés par pertinence. Les résultats de recherche sont des liens qui permettent d'accéder à la ressource. Il ne peut y avoir de doublons. Les résultats sont triés par pertinence. Par exemple, un film qui satisfait plusieurs conditions est plus pertinent qu'un autre qui n'en satisfait qu'une.

Servez-vous des outils mis à votre disposition, layout de page déjà existant, css déjà écrit, etc. Souvenez-vous que les pages sont toutes bootstrappées et peuvent facilement être agencées selon votre volonté. Respectez l'architecture MVC. Il est nécessaire d'avoir au moins un triplet (modèle+vue+controleur) par entité que vous exposez.

# 5 Échéance et règles d'évaluation

### 5.1 Échéance

La totalité du dossier sera évaluée durant la session d'examens de janvier oralement. N'oubliez pas qu'il y a une partie d'évaluation continue. Il est donc nécessaire de fournir un travail régulier durant les semaines de laboratoire.

### 5.2 Règles d'évaluation

L'évaluation établissant la note du cours de Programmation .NET est réalisée de la manière suivante. En cas d'évaluation en présentiel :

- Théorie (30%) : évaluation écrite
- Laboratoire (70%): évaluation orale et continue

En cas d'évaluation à distance :

- Théorie et laboratoire (100%) : évaluation orale

Note finale : moyenne de la note de théorie (30%) et de la note de laboratoire (70%). Cette procédure est d'application tant en 1ère qu'en 2e session.

Une partie de l'évaluation non présentée entraine automatiquement une seconde évaluation en seconde session, quelle que soit la cote calculée. En 2e session, un report de note est possible pour chacune des parties de laboratoire qui auraient été validées ainsi que pour la note de théorie pour des notes supérieures ou égales à 10/20. Toutes les évaluations (théorie ou laboratoire) ayant des notes inférieures à 10/20 sont à représenter.