

DÉPARTEMENT TECHNIQUE

INFORMATIQUE ET SYSTÈMES

Rue Frinoise 12 – 7500 TOURNAI

Axelle DAPREMÉE

Gestion d'une base de données avec  
Entity Framework 6

Année académique 2018-2019

Rapport de stage – Travail de fin d’études présenté pour l’obtention du diplôme de bachelier en Informatique et systèmes - Orientation : Technologie de l’informatique



Remerciements

Je remercie l’équipe Hygie-Care et son directeur, Monsieur Frédéric Foucart, de mon stage pour une durée de 15 semaines.

Je remercie Pierre Merry, mon maître de stage entreprise, ainsi que Kevin Marra, étant tous les deux développeurs au sein de l’équipe Hygie-Care, pour m’avoir guidée et aidée lors de l’évolution de mon projet, et pour m’avoir apporté de nouvelles connaissances utiles au développement du projet.

Je remercie également mon maître de stage institut, Monsieur Emmanuel Lestienne, pour avoir répondu à mes questions quant à la réalisation de mon travail de fin d’études.

Je remercie Monsieur Emmanuel Wilfart pour son aide apportée par rapport à certains points précis du déroulement de mon projet et de mon travail de fin d’études.

Table des matières

[1. Introduction 4](#_Toc9931714)

[2. Cahier des charges 5](#_Toc9931715)

[2.1 Présentation de l’entreprise 5](#_Toc9931716)

[2.2 Le projet 6](#_Toc9931717)

[2.3 Outils utilisés 8](#_Toc9931718)

[3. Mémoire 9](#_Toc9931719)

[3.1 Analyse 9](#_Toc9931720)

[**3.1.1 Qu’est-ce qu’Entity Framework et pourquoi ?** 9](#_Toc9931721)

[**3.1.2 Pourquoi choisir Entity Framework 6 plutôt que Entity Framework Core ?** 11](#_Toc9931722)

[**3.1.3 Architecture** 12](#_Toc9931723)

[**3.1.4 Le modèle de données : Entity Data Model** 14](#_Toc9931724)

[**3.1.5 Les relations de base de données** 15](#_Toc9931725)

[**3.1.6 DbContext : Qu’est-ce que c’est ?** 17](#_Toc9931726)

[**3.1.7 Les types d’entités** 18](#_Toc9931727)

[**3.1.8 La méthode « Code First »** 20](#_Toc9931728)

[**3.1.9 Les requêtes vers la base de données** 22](#_Toc9931729)

[**3.1.10 DataAnnnotations** 25](#_Toc9931730)

[**3.1.11 Les migrations** 27](#_Toc9931731)

[**3.1.12 Les logs** 29](#_Toc9931732)

[**3.1.13. Eager loading – Lazy loading – Explicit loading** 30](#_Toc9931733)

[**3.1.14 Les conventions du Code First** 32](#_Toc9931734)

[3.2 Mise en œuvre et exemples réalisés 33](#_Toc9931735)

[**3.2.1 Ajout d’Entity Framework sur Visual Studio** 33](#_Toc9931736)

[**3.2.2 Définition et utilisation de DbContext** 35](#_Toc9931737)

[4. Conclusions 42](#_Toc9931738)

[5. Bibliographie 43](#_Toc9931739)

[6. Annexes 44](#_Toc9931740)

[6.1 Annexe 1 : Schémas d’ORM 44](#_Toc9931741)

[6.2 Annexe 2 : Fonctionnalités d’ORM implémentée par Entity Framework 6 45](#_Toc9931742)

[6.3 Annexe 3 : Comparaison entre Entity Framework 6 et Entity Framework Core 46](#_Toc9931743)

[6.4 Annexe 4 : Choix d’approche Entity Framework (schéma) 50](#_Toc9931744)

[6.5 Annexe 5 : Exemple d’un service de log 51](#_Toc9931745)

[6.6 Annexe 6 : Ajout du package NuGet Entity Framework 52](#_Toc9931746)

# **Introduction**

Dès mon arrivée en tant que stagiaire au sein de l’équipe Hygie-Care, j’ai été chargée du développement d’un dossier patient informatisé, en m’adaptant et en ajoutant ce nouveau module sur un de leurs sites web existants.

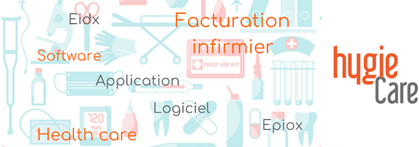
Au commencement de mon projet de stage, j’ai tout de suite été confrontée à des bases de données et à Entity Framework ainsi qu’au « code first ».

J’ai donc pris le choix de réaliser mon mémoire sur l’Entity Framework, étant une technologie d’accès aux données open source de Microsoft, car c’est une fonctionnalité très utilisée, que j’ai rapidement trouvée très intéressante.

Dans ce travail de fin d’études, je vais donc expliquer le projet effectué en entreprise, et effectuer un développement théorique et pratique concernant l’Entity Framework 6, tout en m’appuyant sur les documentations trouvées et les connaissances acquises au cours de mon stage.

# **Cahier des charges**

## 2.1 Présentation de l’entreprise



Hygie-Care est une société belge située à Tournai et fondée en 2015 par Frédéric Foucart. C’est une société de développement informatique axée sur la création d’applications web et mobile. Le secteur d’activité ciblé est celui des soins de santé ambulatoire. Ce qu’ils veulent offrir aux professionnels des soins à domicile et aux autres acteurs de la santé : *« Une technologie de pointe, adaptée à leurs besoins réels, véritablement pensée par eux et pour eux, aussi pratique qu’évolutive ».*



Cette société est étroitement liée à Hygie-Consult, qui a été fondée par Frédéric Foucart, gérant de l’entreprise Hygie-Care, ainsi que son épouse Catherine Clinquart. Cette société est entre autres un client d’Hygie-Care, car elle utilise quotidiennement l’application Epiox développée par Hygie-Care, et travaille dans le secteur des soins de santé, et plus précisément dans le suivi de tarification en soins infirmiers. Hygie-Consult est également active dans le milieu de la consultance en stratégie et management, dans l’accompagnement au changement, l’hygiène et l’épidémiologie.

L’équipe Hygie-Care :

Frédéric Foucart : Chef de projet

Pierre Merry : Analyste Développeur

Kévin Marra : Analyste Développeur

François Racquez : Technico-Commercial / Helpdesk

Ruben Letist : Web Designer

## 2.2 Le projet

Le projet sur lequel j’ai travaillé lors de mon stage, consiste en un dossier patient informatisé, qui permet de remplacer les dossiers papiers contenant toutes les informations d’un patient et tenus par les infirmières à domicile.

Le but premier de ce projet, était de le rendre plus ergonomique et plus facile d’utilisation, car l’utilisateur final est un infirmier qui pourrait ne pas avoir de bonnes connaissances en informatique.

Au début de mon stage, j’ai signé une clause de confidentialité en addendum de ma convention de stage, je ne peux donc parler en détail du projet que j’ai effectué, ni en montrer l’aspect visuel.

Le projet que j’ai effectué était un module à ajouter dans une de leurs applications web existantes. Dès le début, j’ai passé une semaine de documentation, suivie de quelques jours de familiarisation avec le projet.

Les principales fonctionnalités que j’ai développées dans ce module sont :

* La possibilité de créer des prises en charge de patients, ainsi qu’une page d’identification du patient, cette dernière pouvant être éditée.
* Compléter les données importantes relatives à un patient, telles que les numéros de contact pour les proches, les personnes du milieu médical, …
* La possibilité pour l’infirmière de partager la prise en charge d’un patient avec d’autres collègues.
* Compléter une anamnèse avec des questions/réponses, renseigner les antécédents d’un patient, …
* Ajouter les paramètres vitaux d’un patient et compléter des échelles d’évaluations.

Je ne peux en dire plus quant au travail que j’ai effectué lors du stage. Tout ce travail a été réalisé en suivant la méthode agile Scrum, qui est une méthode de gestion de projet permettant de développer une fonctionnalité en une durée déterminée (un sprint) et de les délivrer au client. On voit par conséquent une réelle avancée du projet, et l’un des avantages majeurs est que le client peut suivre l’évolution du projet et donner son avis ainsi que demander des modifications, sans que cela ait un gros impact sur le développement. La productivité de l’équipe est donc améliorée.

Il y a 3 rôles dans Scrum :

* Le Scrum Master : c’est la personne qui s’assure que les principes et les valeurs de Scrum soient respectées,. Cette personne cherche à améliorer la communication au sein de l’équipe ainsi que la productivité.
* Le product owner : c’est l’expert métier ayant un rôle de client, qui connait les fonctions spécifiques importantes. Il établit les priorités des fonctionnalités à développer ou à corriger en cas de bug. À la fin d’un sprint, il valide les fonctionnalités qui ont été développées.
* L’équipe : en général composée de 6 à 10 personnes mais pouvant contenir 200 personnes. Les rôles de chaque personne dans l’équipe peuvent être très différents : développeur, testeur, web designer, … Chaque personne apporte son savoir-faire et accomplit les tâches.

Un sprint est le cycle de vie de Scrum, ou une itération, pouvant durer une semaine voire plusieurs semaines. Dans l’entreprise où j’ai effectué mon stage, les sprints durent 2 semaines. À chaque fin de sprint, il y a une réunion ou Sprint Review, permettant à l’équipe de faire une démonstration de tout ce qui a été développé/réalisé lors du sprint ; le product owner peut donc valider ou non les fonctionnalités ayant été démontrées.

Toutes les tâches à exécuter sont réunies dans le product backlog, qui est en fait un référentiel des exigences initiales ayant été dressées avec le client. Ce backlog peut évoluer tout au long du développement, suivant les besoins du client. On y retrouve des User Stories, qui sont des descriptions des fonctionnalités attendues, et qui sont toutes définies avec le début de phrase suivant : En tant qu’utilisateur, je dois...

Chaque jour, il y avait aussi une réunion de 15 minutes exclusivement pour l’équipe, qui permet d’avoir une idée de l’avancement de projet, ainsi que de connaître les éventuels problèmes rencontrés par un membre de l’équipe. Lors de cette « réunion d’avancement », chacun dit ce qu’il a fait la veille, ce qu’il va faire aujourd’hui et les éventuels problèmes qu’il a rencontrés. Après cette réunion, un membre de l’équipe peut donc aider un autre membre en difficulté s’il pense lui trouver une solution.

## 2.3 Outils utilisés

Les principaux logiciels que j’ai utilisés sont :

* Visual Studio 2017 pour le développement de l’application web ;
* Microsoft SQL Server Management Studio.

J’ai également utilisé :

* Le design pattern MVC (Modèle – Vue – Contrôleur) : solution pour séparer l’affichage des informations, les actions de l’utilisateur et l’accès aux données ;
* ASP .NET Identity : permettant la gestion de rôle et d’authentification ;
* Entity Framework (Méthode Code First) ;
* ASP .NET C# ;
* AJAX ;
* Javascript / Jquery ;
* SendGrid API pour l’envoi de mails ;
* ChartJS pour la création de graphiques ;
* LINQ (Language Integrated Query) : syntaxe de requête en C# ou VB.NET, permettant de traiter (récupérer, ajouter, modifier, supprimer) des données provenant de différentes sources ou formats ;
* Git : Gestion de code sources pour travailler en équipe.

# **Mémoire**

## 3.1 Analyse

### 3.1.1 Qu’est-ce qu’Entity Framework et pourquoi ?

Entity Framework est un mappeur objet-relationnel ou ORM. Il est le principal mappeur fourni par Microsoft pour .NET Framework en tant que produit « Open source », et est la technologie d’accès aux données recommandée par Microsoft.

Entity Framework permet de réduire des inadéquations d’impédances entre les mondes relationnels et « orienté-objet ». Ainsi, les développeurs peuvent écrire leur application et interagir avec des données stockées dans une base de données à l’aide d’objets .NET fortement typés (un objet fortement typé possède un type qui décrit réellement celui-ci). Ils ne doivent donc plus écrire de longs codes pour raccorder les deux mondes et accéder aux données.

Tableau 1 : Date de sortie des différentes versions d’Entity Framework

|  |  |
| --- | --- |
| Version | Date de sortie |
| 1.0 | 11/08/2008 |
| 4.0 | 12/04/2010 |
| 4.1 | 12/04/2011 |
| 4.1 Mise à jour | 25/07/2011 |
| 4.3.1 | 29/02/2012 |
| 5.0 | 11/08/2012 |
| 6.0 | 17/10/2013 |
| 6.1 | 17/03/2014 |
| 6.2 | 26/10/2017 |
| Core 1.0 | 27/06/2016 |

Les données ont été obtenues grâce au PDF Apprenez Entity Framework et à l’annonce de sortie de la dernière version d’EF 6 (liens repris dans les sources).

*Qu’est-ce qu’un ORM ?*

ORM, ou Object-Relationnal Mapping, est une technique de programmation permettant de transformer automatiquement des données provenant de sources externes (base de données) en objets, qui permettent d’exécuter des instructions, afin de transmettre les modifications à la source de données. Les actions reliées à la source sont donc encapsulées derrière une « façade objet », ce qui simplifie les traitements et réduit les coûts de développement d’une solution ayant une ou plusieurs sources de données. L’utilisation de « best practices » permet également de diminuer les risques de dérapage lors du développement. Des schémas en rapport avec les ORM sont présents dans l’annexe 1

*Qu’est-ce qu’une inadéquation d’impédance ?*

Il s’agit d’un terme d’ingénierie qui caractérise une perte de signal. Dans le domaine de l’électricité, il n’y a pas d’inadéquation d’impédance si dans un circuit alternatif l’impédance électrique de la source correspond à celle du récepteur. La puissance électrique serait donc maximisée.

Dans le domaine des bases de données, ce terme est utilisé en pensant à des bases de données relationnelles et des langages orientés objet, mais il est facilement généralisé à tous les types de base de données et de langages. Il correspond à un problème de liaison entre un langage de programmation et un langage de base de données. Voici une liste d’exemples de ces problèmes, qui peuvent rendre la programmation de bases de données difficile :

- Différences de syntaxe (Gestion de la casse, représentation des littéraux, …) ;

- Différences de systèmes de types ;

- Différences de sémantique (Différence entre NULL de SQL et de C#) ;

- Différences de pouvoir expressif (Besoin de coder une partie du traitement) ;

- Différences de modèle d’exécution ;

- Différences d’environnement d’exécution (deux runtime distincts) ;

- Différences « philosophiques ».

Une liste des fonctionnalités ORM implémentées par Entity Framework 6 (retrouvée sur la documentation officielle de Microsoft) est disponible en annexe 2 (page 41).

### 3.1.2 Pourquoi choisir Entity Framework 6 plutôt que Entity Framework Core ?

Entity Framework Core, qui a été publié en 2016, est une refonte complète d’Entity Framework 6. Contrairement à son prédécesseur, EF Core est une multiplateforme ; il peut donc s’exécuter sur .NET Core ou .NET Framework. Il fournit néanmoins une expérience de développement semblable à celle d’EF 6.

Entity Framework Core apporte de nouvelles fonctionnalités non implémentées dans EF 6 (exemples : clés secondaires, mises à jour par lot, …). Cependant, puisque c’est une nouvelle base de code, il lui manque également des fonctionnalités présentes dans EF6. Nous pouvons constater cela dans les tableaux de comparaison repris en Annexe 1.

*Lequel choisir ?*

Le mieux est d’utiliser Entity Framework Core pour une nouvelle application, mais vérifiez d’abord à ce que les deux conditions reprises ci-dessous soient remplies :

- L’application a besoin des fonctionnalités de .NET Core.

- Entity Framework Core possède toutes les fonctionnalités nécessaires.

Si votre application s’exécute sur Windows et en .NET Framework 4.0 ou version ultérieure, et qu’Entity Framework 6 possède toutes les fonctionnalités nécessaires, le mieux est d’utiliser ce dernier.

Pour les applications existantes en Entity Framework 6, il est déconseillé de migrer l’application vers Entity Framework Core dû aux modifications importantes qui lui ont été apportées.

3.1.3 Architecture

L’architecture d’Entity Framework est constituée de 3 couches : 

Figure 3 : Schéma de l’architecture d’Entity Framework, créé à partir de la documentation de TutorialsPoint

* **Data Providers** : Cette couche représente les fournisseurs spécifiques aux sources. Elle est chargée de la traduction du langage SQL commun comme LINQ, via une arborescence de commandes, en expressions SQL natives, et les exécute pour le bon système de gestion de base de données, ou DBMS en anglais pour *Database Management System.*
* **Entity Client** : Cette couche permet aux développeurs de travailler avec les entités sous forme de colonnes et de lignes en utilisant des requêtes SQL d’entité, sans avoir à générer des classes afin de représenter le schéma conceptuel.

Elle représente les couches d’Entity Framework et est la couche principale. Ces couches sont exposées à la couche supérieure et sont regroupées en un groupe appelé Entity Data Model.

*Qu’est-ce qu’une entité ?*

Une entité représente un objet spécifique relatif à une table de la base de données. Dans notre table Patient contenant plusieurs patients, le patient A est représenté par une entité. Chaque entité doit avoir une clé unique dans un ensemble d’entités, qui est une collection des instances d’un type d’entité spécifique. Avec les types d’entité, l’héritage est possible. Un type d’entité peut donc être dérivé d’un autre.

Voici les 3 couches dont je parle ci-dessus :

* **Storage Layer (couche de stockage) :** Contient le schéma de base de données dans un format XML.
* **Entity Layer (couche d’entité) :** Contient les entités et les relations dans un fichier XML.
* **Mapping Layer (couche de mappage) :** Contient un mappage des entités et des relations définies dans la couche conceptuelle avec les relations et les tables existantes qui ont été définies dans la couche logique, tout cela dans un fichier XML.

Le service de métadonnées étant également représenté par la couche Entity Client et fournissant une API centralisée, permet d’accéder aux métadonnées stockées des 3 couches d’Entity Data Model.

* **Object Services** : C’est le « contexte objet » qui représente la session d’interaction entre les applications et les sources de données. Son utilité principale est d’effectuer des opérations (ajout, suppression d’instances d’entités) et de sauvegarder les changements dans la base de données grâce à des requêtes. Cette couche est la couche ORM d’Entity Framework, car elle représente les données des instances d’objet des entités**.**

*Qu’est-ce qu’un modèle conceptuel ?*

C’est un modèle des objets de l’application qui pourrait ne pas correspondre avec le schéma de la base de données, ou même être très différent. Ce modèle peut être défini dans Visual Designer, qui génère ensuite les classes qui seront utilisées dans l’application.

Entity Framework permet de transférer le modèle conceptuel à la base de données, permettant ainsi de travailler directement avec les objets du modèle.

Figure 5 : Schéma représentant les relations du modèle conceptuel avec l’application (créé à partir de la documentation de TutorialsPoint)

### 3.1.4 Le modèle de données : Entity Data Model

C’est un modèle spécifiant le modèle conceptuel des données en utilisant plusieurs techniques de modélisation, il contient des concepts qui décrivent la structure des données sans prendre en compte la forme de stockage, c’est entreautres une extension du modèle de relations d’Entity.

Comme cela a déjà été dit ci-dessus, Entity Data Model est constitué de 3 parties principales étant connues comme les bases d’Entity Framework :

* **The Storage Schema Model :** Modèle qui est également appelé Storage Schema Definition Layer (SSDL) et qui est une représentation schématique du stock de données en « backend ».
* **The Conceptual Model :** Modèle qui est également appelé Conceptual Schema Definition Layer (CSDL). C’est le modèle d’Entity pour lequel nous écrivons des requêtes.
* **The Mapping Model :** C’est un modèle ayant un rôle de mappage entre les deux modèles précédents.

ADO .NET Entity Framework utilise un langage de définition de schéma (Schema Definition Language ou SDL) afin de définir un schéma EDM. SDL définit des types simples semblables à d’autres types primitifs, tels que String, Int32, Double, Decimal, DateTime, …

Le modèle de données possède 3 concepts clés pour décrire la structure des données :

* **Entity Type :** c’est un bloc de construction fondamental. Dans un modèle, les types d’entités sont construits à partir des propriétés et décrivent la structure des concepts de niveau supérieur.
* **Association Type :** C’est un autre bloc de construction fondamental qui décrit les relations dans le modèle de données Entity (relation entre deux types d’entités). Les entités à chaque bout des associations peuvent être accédées grâce à des propriétés de navigation ou par des clés étrangères.
* **Property :** Il existe des propriétés qui définissent la structure et les caractéristiques d’un type d’entité. Par exemple, un Patient possède un identifiant unique, un numéro NISS, un nom, un prénom, … Une propriété peut contenir des données primitives ou des données structurées.

### 3.1.5 Les relations de base de données

Quand on parle de base de données, la première chose qui nous vient à l’esprit, c’est un ensemble de tables qui ont des relations entre elles. Il y a en effet 3 types de relations entre des tables, et la détermination d’un type de relation dépend de la manière dont les colonnes qui constituent les tables sont définies.

Pour illustrer ces différentes relations, je vais utiliser comme exemple certaines des tables et des relations présentes dans la base de données du projet que j’ai réalisé.

* **One-to-Many Relationship :** C’est la relation la plus utilisée, dans laquelle une ligne dans une table A peut correspondre à plusieurs lignes dans une table B, mais une ligne dans la table B ne peut correspondre qu’à une seule ligne dans la table A.

Exemple : Dans la base de données de mon projet, deux tables ont cette relation : une table Antecedent (contenant les différents antécédents d’un patient au moment d’une prise en charge), et une table PriseEnCharge.

Un antécédent a été enregistré lors d’une prise en charge à une date précise, il ne peut donc correspondre qu’à une seule prise en charge. Mais une prise en charge peut correspondre à plusieurs antécédents renseignés par le patient.



Figure 6 : Schéma d’une relation One-to-Many, créé à partir de la structure de base de données du projet de stage.

* **Many-to-Many Relationship :** Pour ce type de relation, une ligne dans une table A peut correspondre à plusieurs lignes dans une table B et inversement. Cette relation peut être créée en ajoutant une table de jonction, dont la clé primaire sera une clé étrangère pour les deux autres tables. Lors de mon projet, j’ai utilisé plusieurs fois ce type de relations, en voici un exemple :

La table PatientInfi est la table de jonction entre la table Patient et la table Infirmier, cela permet donc d’établir une relation telle qu’un(e) patient(e) peut avoir plusieurs infirmier(e)s différent(e)s, et un(e) infirmier(e) peut avoir plusieurs patients.



Figure 7 : Schéma d’une relation Many-to-Many, créé à partir de la structure de base de données du projet de stage.

* **One-to-One Relationship :** Pour ce type de relation, une ligne dans la table A ne peut pas correspondre à plus d’une ligne dans la table B et inversement. Cette relation est créée en ajoutant des clés primaires à chaque colonne ou en leur ajoutant des contraintes uniques. Ce type de relation est très peu utilisé mais peut être utile pour :
* Diviser une table ayant beaucoup de colonnes ;
* Isoler une partie de la table pour des raisons de sécurité ;
* Stocker des données temporaires qui pourraient donc être facilement perdues en supprimant la table ;
* Stocker des informations qui sont une sous-catégorie de la table principale.

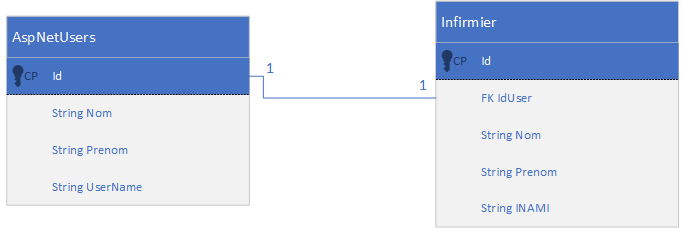
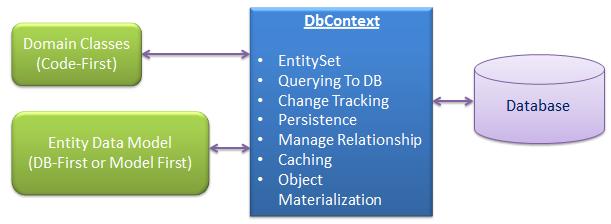


Figure 8 : Schéma d’une relation One-to-One, créé à partir de la structure de base de données du projet de stage.

### 3.1.6 DbContext : Qu’est-ce que c’est ?

### 

Comme cela a été cité précédemment dans ce travail, Entity Framework permet de récupérer, insérer, mettre à jour ou supprimer des données en utilisant des entités (Common Language Runtime objects), et d’établir des relations, qui ont été définies dans le modèle entre ces entités, dans la base de données. Mais il possède également plusieurs autres fonctionnalités grâce au DbContext faisant partie du namespace System.Data.Entity.DbContext, et qui est la classe primaire responsable de l’interaction entre les données et les objets. Elle a pour but de simplifier l’interaction avec Entity Framework et permet de réduire le nombre de méthodes et de propriétés requises pour accéder aux tâches les plus utilisées, celles-ci étaient bien plus difficiles à coder dans les versions précédentes.



**Cette classe,** qui représente donc la classe **intermédiaire entre la base de données et le domaine de classe** (Code-First) ou l’Entity Data Model (Database-First ou Model-First), **permet** :

* De définir des « sets » d’entités, qui représentent les tables de la base de données ;
* D’effectuer des requêtes vers la base de données ;
* De gérer le traçage des objets ;
* De gérer la persistance des données, c’est-à-dire, de permettre à l’application de se terminer sans que les données soient perdues. Ces informations sont transmises au serveur de base de données. Rendre un objet, ou une entité persistant(e) signifie sauvegarder ses données vers un support non volatile de telle sorte qu’un objet, ou une entité, identique à cet objet ou cette entité sera recréé(e) à la prochaine session.
* De gérer les relations entre les entités ;
* De gérer la mise en cache des données récupérées de la base de données ;
* De matérialiser les informations reçues de la base de données en objets.

La manière dont on ajoute le contexte de base de données dans une application est expliquée dans la partie « Mise en œuvre » à la page ..

### 3.1.7 Les types d’entités

Il existe deux types d’entités, qui permettent aux développeurs d’utiliser leurs propres classes avec le modèle de données, sans devoir modifier les classes de données.

* **Les entités POCO :**

POCO signifie Plain-Old CLR Objects, ce sont des objets pouvant être utilisés comme des objets du domaine existant avec le modèle de données.

Les classes de données étant reliées aux entités sont définies dans le modèle de données, et la plupart des requêtes générées par l’outil de modèle de données Entity (Entity Data Model tools), sont supportées.  
Un template POCO peut aussi être utilisé afin de générer, à partir d’un modèle conceptuel, des types d’entités « persistance-ignorant » c’est-à-dire, que chaque partie de code fonctionnant dans la couche logique de métier (ou supérieure) n’a pas connaissance du design actuel de la base de données, du moteur de base de données, ni de quand et comment les objets sont récupérés ou conservés en base de données.



* **Dynamic Proxy**

Lors de la création d’instances de types d’entités POCO, des instances d’un type généré dynamiquement, agissant comme un proxy pour l’entité, sont créées. On en parle souvent en tant que classe englobante de l’entité POCO.

Il est possible de surcharger certaines propriétés de l’entité afin d’effectuer des actions automatiquement lorsque l’on accède à cette propriété. Le mécanisme est également utilisé pour supporter le « lazy loading » dont nous parlerons plus tard, ou le suivi de changements automatiques. Ces techniques sont également d’application pour les modèles ayant été créés avec le code first ou le Designer.

Pour qu’Entity Framework supporte le « lazy loading », et pour qu’il traque les changements dans les classes POCO, il faut que ces classes suivent certaines exigences :

- Doivent être publiques ;

- Ne doivent pas non plus être déclarées avec le mot-clé sealed ;

- Elles ne peuvent pas être abstraites ;

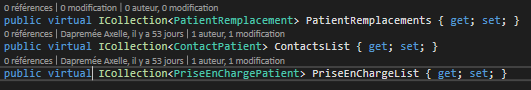
- Elle doivent avoir un constructeur public ou protégé, qui n’a aucun paramètre ;

- Les propriétés de navigation doivent être publiques et virtuelles, et les collections doivent être déclarées avec le mot clé ICollection<T>.

Il faut utiliser un constructeur protégé sans paramètres si la méthode CreateObject est nécessaire pour créer un proxy pour l’entité POCO, mais pour cela la classe doit suivre les recommandations citées ci-dessus.

La classe ne doit pas implémenter les interfaces IEntityWithChangeTracker ou IEntityWithRelationships car elles sont déjà implémentées par le proxy.

L’option ProxyCreationEnabled doit être activée, elle peut toutefois être désactivée s’il est nécessaire de désactiver la création d’objets proxy.



### 3.1.8 La méthode « Code First »

Il existe 3 méthodes permettant de créer un modèle Entity :

* Code First
* Database First
* Model First

Lors de mon stage, la méthode que j’ai apprise est l’approche Code First. C’est donc de celle-ci que je vais parler plus en détail. Le choix de méthode dépend des préférences des développeurs, car chacune d’elle a ses avantages et ses inconvénients.

Tout d’abord, afin de comprendre la différence entre ces 3 méthodes, voici une définition de chacune d’elles :

* **Code First :** Cette approche permet de créer une base de données directement à partir du code alors qu’elle est inexistante auparavant. On peut également ajouter de nouvelles tables directement en définissant les modèles de classes et en ajoutant les « DbSet » dans la classe dérivée de la classe DbContext. Il est aussi possible de créer un code de modèle semblable à une base de données existante afin d’accéder à celle-ci.
* **Database First :** Cette approche est une alternative aux deux autres. Le modèle Entity Framework est créé à partir d’une base de données existante, il faut ensuite créer des modèles (classes, propriétés, …) qui seront la liaison entre la base de données et le contrôleur.
* **Model First :** En utilisant cette approche, le développeur définit un modèle en utilisant un designer Entity Framework qui génèrera une procédure SQL, qui une fois exécutée, créera la base de données en fonction du modèle créé par le développeur. Les modèles sont stockés dans des fichiers EDMX et peuvent toujours être lus et édités.

Un schéma pouvant aider le développeur à choisir l’approche qui lui convient est disponible à l’annexe 4.

*Pourquoi le Code First est intéressant ?*

Le Code First est constitué d’un ensemble de parties de code, semblable à un puzzle, la partie principale étant les classes du domaine. Ces dernières n’ont rien à voir avec Entity Framework, car ce code a un contexte permettant de gérer les interactions entre ces classes et la base de données.

Code First ajoute un constructeur de modèle inspectant les classes et utilise ensuite plusieurs règles ou conventions afin de déterminer comment ces classes décrivent un modèle avec leurs relations, et comment appliquer cela à la base de données. Tout cela se passe au chargement de l’application ; le modèle n’est donc jamais visible car il se trouve seulement en mémoire.

Si le modèle change, la base de données peut être mise à jour en utilisant les migrations. Cette partie est expliquée à la page 27.

**Les avantages du Code First :**

* Fonctionne tout aussi bien pour une nouvelle base de données ou une base de données existante ;
* Les modèles développés deviennent la base de données ;
* Il n’est pas nécessaire de créer une classe d’extension faisant office de liaison entre le code et la base de données ;
* Le développeur a plus de contrôle sur son application ;
* Le développeur peut gérer la version de la base de données et revenir à une version antérieure en cas de nécessité (voir partie Migration du projet) ;
* Il est considéré comme le meilleur pour les projets sur le long terme.

### 3.1.9 Les requêtes vers la base de données

Il est possible d’utiliser des procédures stockées à la place de, ou en coordination avec, la génération de commandes automatiques d’Entity Framework.

Ces procédures stockées peuvent servir à exécuter des logiques prédéfinies sur des tables de bases de données, à insérer, mettre à jour ou supprimer des entités.  
Les commandes automatiques sont sécurisées et tout aussi efficaces, voire plus, que les procédures créées par un développeur. Parfois, le développeur veut avoir le contrôle total sur ce qui est exécuté sur le stockage et préfère donc les procédures stockées.

Lors de mon stage, je n’ai pas utilisé de procédures stockées, mais j’ai écrit des requêtes directement dans le code avec LINQ to Entities, en utilisant le DbContext et les DbSet.

Nous pouvons donc écrire une requête pour récupérer une liste d’entités comprise dans la base de données, soit pour l’utiliser afin d’afficher ces données, ou afin de traiter ces données. Dès que l’on effectue une modification sur ces données, nous devons appeler la méthode DbContext.SaveChanges() qui a déjà été citée dans les points précédents.

LINQ to Entities est un langage déclaratif, il suffit de se concentrer sur les informations dont nous avons besoin sans avoir à réfléchir à la manière dont nous pouvons obtenir cette information. Cela permet de faire gagner du temps au développeur, car il peut passer plus de temps à travailler sur les données.

Il y a plusieurs mots-clés à connaître afin d’écrire ce type de requêtes. Ceux-ci peuvent être combinés afin d’obtenir les résultats souhaités dans la base de données. Je vais les expliquer brièvement ci-dessous, et des exemples seront donnés dans la partie Mise en œuvre de ce travail.

* **Ascending :** on effectue un tri de l’élément le plus bas dans la sélection vers l’élément le plus haut. C’est le tri par défaut ;
* **By :** utilisé pour indiquer le champ ou l’expression utilisé(e) pour grouper les données ;
* **Descending :** on effectue un tri de l’élément le plus haut dans la sélection vers l’élément le plus bas ;
* **Equals :** utilisé entre les parties gauches et droites d’une déclaration « join » afin de joindre les données de la source de gauche (primaire) avec celles de la source de droite (secondaire) ;
* **From :** utilisé afin de spécifier la source de données utilisé pour obtenir les informations requises et définir une variable de sélection. Celle-ci a le même rôle qu’une variable utilisée pour une itération ;
* **Group :** utilisé afin de grouper la sortie en utilisant la valeur clé spécifiée. Il est possible d’utiliser plusieurs déclarations « group » afin de créer plusieurs niveaux d’organisation de la sortie. L’ordre de ces groupes détermine l’emplacement d’une valeur précise dans la liste retournée par la requête. Le mot clé Group peut être associé à By afin de créer un contexte spécifique ;
* **In :** peut être utilisé de plusieurs manières : par exemple, pour déterminer la source de données d’une requête. S’il est utilisé avec un join, il sert pour chaque source de données utilisée pour cette déclaration ;
* **Into :** permet de spécifier un identifiant utilisé comme une référence pour les requêtes telles que join, group et select ;
* **Join :** crée une seule source de données à partir de deux sources différentes, il peut spécifier un inner, group ou left-outer join, le premier étant celui par défaut ;
* **Let :** définit une variable de sélection pouvant être utilisée afin de stocker des résultats d’une sous-expression. Il est en général utilisé afin de fournir une sortie énumérée additionnelle ou pour augmenter l’efficacité d’une tâche ;
* **On :** spécifie le champ ou l’expression utilisée pour implémenter un « join », cette valeur permet de définir un élément commun entre les deux sources de données ;
* **Orderby :** permet de trier les données en fonction d’un champ d’une source de données. Il est possible d’utiliser plusieurs déclarations de ce genre afin de créer plusieurs niveaux de tri. L’ordre des déclarations détermine l’ordre dans lequel les données se retrouvent dans la liste ;
* **Where :** définit ce que la requête doit obtenir de la source de données ;
* **Select :** détermine la sortie de la requête en spécifiant quelle information celle-ci doit retourner. Cette déclaration définit également le type de données récupérées.

Les requêtes de projection : améliorent l’efficacité de l’application en recherchant uniquement des champs spécifiques dans la base de données à l’aide du mot clé select et une syntaxe spécifique : select new {retrieve.Id, retrieve.Name, retrieve.Age}.

Pour obtenir une seule entité grâce à la recherche, il est possible d’utiliser la méthode First() ou FirstOrDefault() qui, toutes deux, retournent le premier élément d’une séquence. Ces deux méthodes sont légèrement différentes, la première provoque une exception si aucune valeur n’est récupérée, la deuxième retourne une valeur nulle dans le même cas. Il existe également les méthodes Single() ou SingleOrDefault() qui retournent un seul élément spécifique d’une séquence.

Si le développeur a besoin de récupérer une liste, dans ce cas il peut utiliser la méthode ToList().

EF 6 supporte les opérations asynchrones afin de récupérer et mettre à jour des données. Ce type d’opération est utile pour rendre l’application plus réactive aux actions de l’utilisateur, et pour augmenter la performance moyenne de l’application.

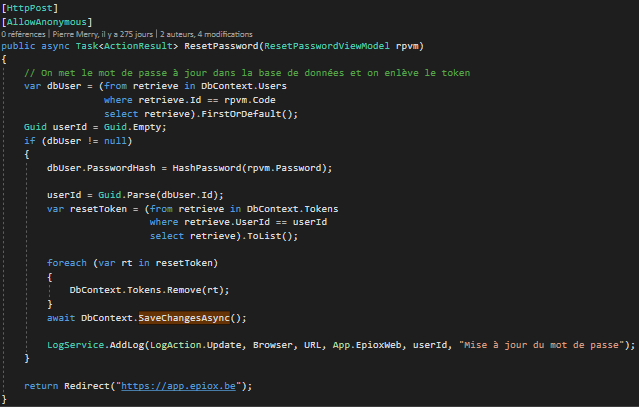
Il existe plusieurs façons d’exécuter ces opérations, mais les plus simples sont les mots-clés async/await ayant été introduits dans la version 4.5 du .NET Framework.

Pour utiliser ces mots-clés, il faut ajouter les namespaces System.Data.Entity (pour les méthodes asynchrones) et System.Threading.Tasks.

Il faut ajouter le mot-clé async dans la déclaration de la fonction où nous souhaitons ajouter une requête asynchrone ; cette fonction retournera donc un type Task.

Il suffit ensuite d’appeler la méthode SaveChangesAsync() pour mettre à jour des données. Si l’on souhaite récupérer une liste, la méthode ToListAsync est disponible.

Avant la méthode SaveChangesAsync ou la requête vers la base de données, il faut ajouter le mot-clé await.



### 3.1.10 DataAnnnotations

DataAnnotations, ou en français « Annotations de données », permet d’ajouter des configurations supplémentaires aux paramètres de bases apportés par le modèle de conventions d’Entity Framework. Le code first part du principe que les classes suivent ces conventions et qu’elles fonctionneront donc automatiquement.

Si les configurations de bases ne suffisent pas, il existe deux manières d’ajouter des configurations supplémentaires :

* Les annotations de données que j’ai utilisées lors de mon stage. Elles sont également supportées par de nombreuses applications .NET comme ASP .NET MVC pour les validations côté client.
* Fluent API, qui permet de décrire les nouvelles configurations directement dans le code.

Ces annotations de données surchargent donc les conventions par défaut du Code First.

Les différents attributs disponibles et supportés par EF sont compris dans deux namespace :

* System.ComponentModel.DataAnnotations : les attributs ayant un impact sur la nullabilité ou la taille des colonnes :
  + **Key :** permet de spécifier quelle propriété sera la clé primaire de l’entité ;
  + **ConcurrencyCheck :** permet de marquer une ou plusieurs propriétés pour la vérification de concurrence.
  + **Timestamp :** permet de marquer une seule propriété pour la vérification de concurrence, tant que le type de cette propriété est une chaine de byte et non-nulle.
  + **Required :** spécifie si une propriété est requise pour insérer les informations de l’entité dans la table de base de données ;
  + **MinLength :** spécifie la longueur minimale de la propriété ;
  + **MaxLength :** spécifie la longueur maximale de la propriété ;
  + **StringLength :** semblable à « MaxLength » mais peut seulement être appliqué à une propriété de type « string ».
* System.ComponentModel.DataAnnotations.Schema : les attributs ayant un impact sur le schéma de la base de données.
  + **Table :** permet de changer le nom de la table ;
  + **Column :** permet de changer le nom de la colonne ;
  + **Index :** crée un index qui créera automatiquement un index correspondant dans la base de données ;
  + **ForeignKey :** permet d’ajouter une liaison entre la classe actuelle et une autre ;
  + **NotMapped :** permet d’éviter que la propriété soit stockée en base de données si cela n’est pas nécessaire ;



* + **InverseProperty :** utilisé pour éviter que le code first ne soit pas capable de lier les propriétés des deux classes.

Des exemples des attributs utilisés lors de mon stage sont disponibles dans la partie mise en œuvre.

### 3.1.11 Les migrations

Il existe différentes stratégies d’initialisation de bases de données avec la méthode Code-First telles que :

* CreateDatabaseIfNotExists
* DropCreateDatabaseIfModelChanges
* DropCreateDatabaseAlways.

Le problème avec ces stratégies est que, même si la base de données existe déjà, les stratégies vont la supprimer et la recréer à chaque modification de celle-ci, ayant pour conséquence la perte des données qui y étaient stockées.

EF a donc introduit un outil de migration permettant de mettre à jour le schéma de base de données lorsque le modèle change, tout en évitant le problème de perte de données. Cet outil utilise un initialiseur de base de données appelé : MigrateDatabaseToLatestVersion.

Ces migrations sont la façon la plus recommandée pour faire évoluer le schéma de base de données tout en utilisant la méthode Code-First. Elles permettent, comme indiqué ci-dessus, de mettre à jour le schéma de base de données en générant des migrations qui traquent les modifications, mais également de créer une base de données fonctionnant avec notre modèle EF.

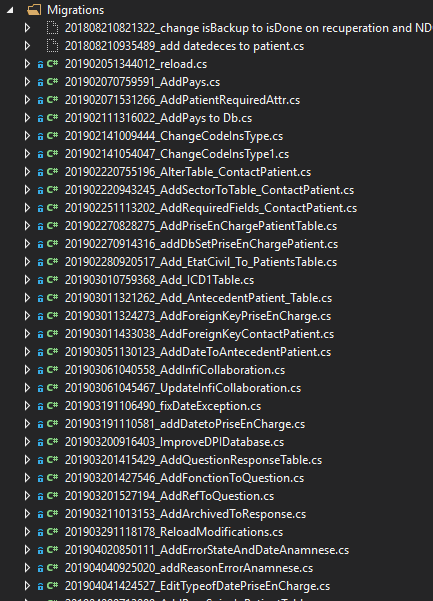
Il y a deux types de migrations :

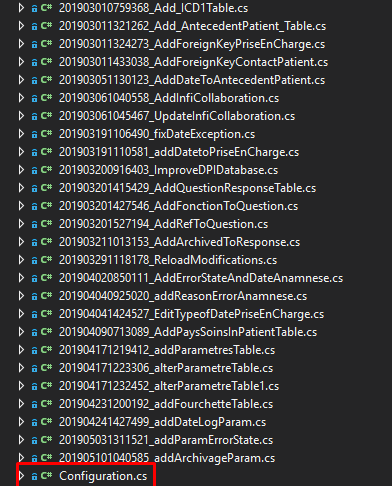
1. *Migrations automatisées :*

Permettent de ne pas devoir effectuer les migrations manuellement pour chaque changement effectué sur les classes du domaine.

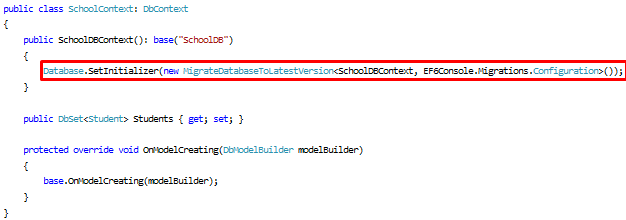
Pour les activer, il faut exécuter cette commande dans la console de gestion de package : « enable-migrations -EnableAutomaticMigration :$true ».

Une fois que la commande a été exécutée, une classe Configuration est créée, héritant de DbMigrationConfiguration. Cette classe se trouve dans le dossier Migration du projet.





Il faut ensuite changer l’initialiseur de la base de données comme ceci :



Dans la base de données, une table système a été créée : \_MigrationHistory, comprenant l’historique des changements dans la base de données pour chaque migration.

Après avoir effectué ces étapes, le développeur peut ajouter de nouvelles classes dans le domaine ou en supprimer, et en exécutant l’application la base de données sera automatiquement mise à jour. Par contre, si des propriétés sont ajoutées, modifiées ou supprimées des classes du domaine, une exception est lancée, spécifiant qu’il y aura une perte des données de la colonne affectée par la modification. Si cela ne pose pas de problème au développeur, il peut autoriser cet évènement en passant à l’état true la propriété AutomaticMigrationDatalossAllowed dans le constructeur de la classe Configuration.

1. *Migrations basées sur le code (Code-Based)*

Ce type de migration permet d’avoir un meilleur contrôle sur les migrations et permet de configurer certains paramètres, comme la valeur par défaut d’une colonne.

Ce type de migration se déroule en 3 étapes pour la première fois :

- Dans la console de gestion de package, il faut exécuter la commande « enable-migrations ». Celle-ci créera également la classe configuration ou il faut ajouter l’initialiseur de base de données comme cela a été fait dans le point précédent.

- Ensuite, toujours dans la console, il faut exécuter la commande « Add-Migration migrationName ». Celle-ci crée une classe de migration avec les méthodes Up() (contenant le code pour créer des objets de base de données) et Down() (contenant le code pour supprimer des objets). Il est possible d’écrire des codes personnalisés permettant de mettre en place des configurations supplémentaires.

- Une fois que la classe de migration est créée, il faut mettre à jour la base de données grâce à la commande « Update-Database ». Si la base de données n’existe pas encore, elle sera créée à ce moment-là.

Pour les prochaines modifications, il ne faudra plus exécuter la première étape. Une exception sera lancée si le développeur tente d’exécuter une application dont la base de données n’a pas été mise à jour avec une migration.

En cas de problème, il est possible de revenir à une version précédente de la base de donnée, en utilisant la commande « update-database -TargetMigration :EpioxDb-v1.

### 3.1.12 Les logs

Il est possible d’envoyer des commandes ou un SQL équivalent à des requêtes à la base de données, pour réaliser des opérations CRUD (Create, Read, Update and Delete).

Avant Entity Framework 6, les développeurs devaient utiliser des utilitaires de traçages tiers ou un outil de traçage de base de données pour traquer des requêtes et des commandes vers la base de données. Maintenant, les logs permettent de stocker toutes les opérations exécutées par EF très simplement. Toutes ces actions sont stockées en utilisant DbContext.Database.Log.

Les actions comprises dans le log sont :

* Les requêtes SQL ;
* Les paramètres ;
* Si la commande est asynchrone ou non ;
* Un indicateur (timestamp) du moment auquel l’exécution de la commande a débuté et le temps nécessaire pour exécuter celle-ci ;
* Si la commande a été effectuée avec succès ou si elle a échoué avec des indications à propos du résultat.

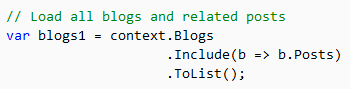
Dans mon projet de stage, je n’ai pas utilisé les logs d’Entity Framework car les développeurs ont créé leur propre service de log, où tout est stocké dans une table de la base de données. L’exemple de ce code est situé en Annexe 5.

### 3.1.13. Eager loading – Lazy loading – Explicit loading

Si les modèles ont été associés correctement, Entity Framework permet de charger les données associées de 3 manières différentes :

**a) Eager loading :**

Processus pendant lequel une requête pour un type d’entité charge également les entités liées. Il est réalisé en utilisant la méthode Include(). Ce chargement est également possible sur plusieurs niveaux.

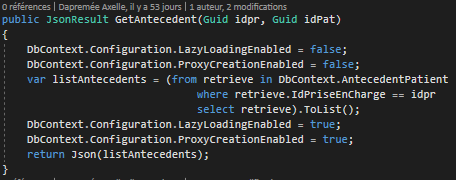


**b) Lazy loading (chargement paresseux) :**

Processus pendant lequel une entité ou une collection d’entités est automatiquement chargée de la base de données la première fois qu’une propriété faisant référence à l’entité est accédée. Dans le cas où le développeur utilise des entités de type POCO, ce chargement est effectué en créant des instances de types proxy dérivés, et en surchargeant ensuite les propriétés virtuelles pour ajouter l’ancre de chargement.

Il est conseillé de désactiver ce chargement lorsque le développeur veut effectuer de la sérialisation de données. La plupart des processus de sérialisation accèdent à chaque propriété d’une entité. L’accès aux propriétés déclenche le « lazy loading », qui provoque la sérialisation d’autres entités. De plus en plus d’entités sont donc chargées, ce qui peut amener au chargement complet de la base de données dans certains cas.

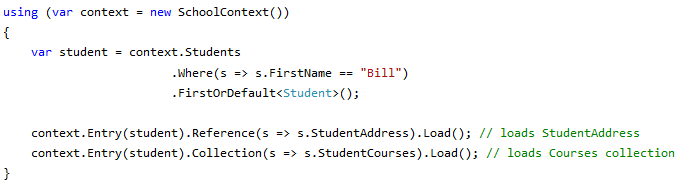
Voici comment désactiver le lazy loading :



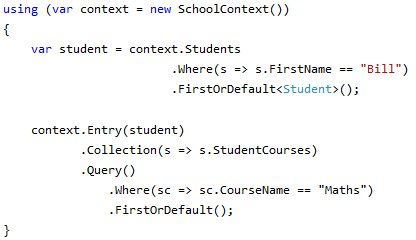
Il est également possible d’éviter le lazy loading en rendant certaines propriétés non virtuelles ou en le désactivant pour toutes les entités à partir de la classe héritant de DbContext. Si nécessaire, il est toujours possible de charger les entités liées en utilisant un des deux autres types de chargement.

1. **Explicit loading :**

Même en désactivant le lazy loading, il est toujours possible d’effectuer ce type de chargement en utilisant la méthode Load(), qui est un appel explicite.



Afin de filtrer la requête, et qu’elle soit effectuée sur seulement certaines entités, il faut utiliser la méthode Query(). Et ce, que ce soit pour une référence ou une collection, étant donné que, dans le cas d’une collection, elle est plus utile pour changer seulement certaines parties de celle-ci. Avant d’utiliser cette méthode, il est conseillé de désactiver le lazy loading.



### 3.1.14 Les conventions du Code First

**a) La clé primaire :**

Une propriété est une clé primaire si cette propriété a pour nom « ID » ou le nom de la classe suivi de « ID ». De plus, si le type de cette propriété est Guid, cette propriété sera configurée comme étant la colonne d’identité.

**b) Les relations :**

Avec Entity Framework, il est possible d’accéder aux entités par leurs relations grâce aux propriétés de navigation. Chaque objet peut avoir une propriété de navigation pour chaque relation dans laquelle elle intervient.

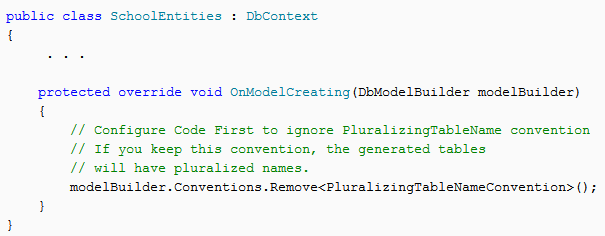
Si une propriété possède le même type de données que la propriété de la clé primaire, en terminant également par le mot « ID », cette propriété sera représentée comme une clé étrangère. Lorsqu’une clé étrangère est détectée, le Code First déduit le type de relation en fonction de la nullabilité de la clé étrangère. Si celle-ci est nullable, une relation facultative sera enregistrée, sinon elle sera requise et le Code First ajoutera donc une suppression en cascade sur la relation.

**c) Les types complexes :**

Lorsque le Code First rencontre une définition de classe où il ne retrouve pas de clé primaire et qu’aucune n’est ajoutée même avec les annotations de données ou Fluent API, il enregistre cette classe comme étant de type complexe. Pour cela, il faut également qu’il n’y ait pas de propriété référençant des types d’entités, et qu’il ne soit pas lui-même référencé par une propriété de collection sur un autre type.

1. **Supprimer les conventions :**

Il est possible de supprimer chaque convention définie dans le namespace System.Data.Entity.ModelConfiguration.Conventions.Pour cela, il faut ajouter une ligne de code dans la méthode OnModelCreating(), voici un exemple repris de la documentation de Microsoft :

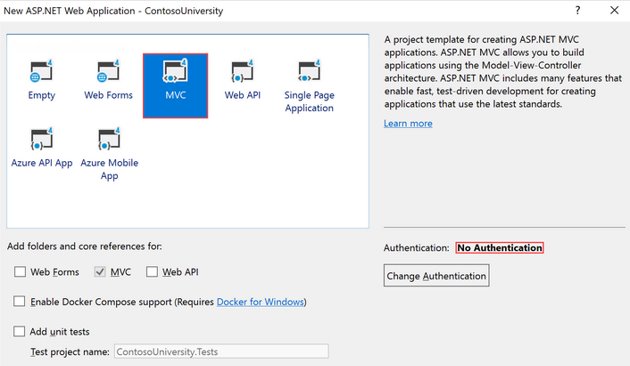


### 

## 3.2 Mise en œuvre et exemples réalisés

### 3.2.1 Créer un nouveau projet

Lors de mon stage j’ai travaillé sur un projet déjà existant, mais voici une brève démonstration pour démarrer un projet de zéro. Sous Visual Studio, il faut créer un nouveau projet en sélectionnant une application web ASP .NET Framework. Voici la fenêtre qui s’affiche après cette sélection :



Afin de travailler avec le « pattern » ou patron de conception MVC, il faut veiller à ce que celui-ci soit sélectionné. MVC est un motif d’architecture permettant d’organiser le code. Ce motif est composé de trois types de modules ayant chacun une responsabilité distincte : les modèles qui contiennent les données à afficher, les vues qui contiennent la présentation de l’interface graphique, et les contrôleurs qui contiennent la logique d’actions effectuées par l’utilisateur.

Si le développeur souhaite une application basique, il faut veiller à ce que l’authentification ne soit pas comprise dans le projet en cours de création. Toutefois, si le développeur souhaite réaliser une application qui stocke des données sensibles devant de ce fait être protégées d’une quelconque attaque (vol d’information ou de propriété intellectuelle), il est intéressant, voire fortement conseillé, d’utiliser un système de gestion d’authentification. ASP .NET fournit plusieurs méthodes permettant d’authentifier un utilisateur (authentification anonyme, basique, avec certificat, avec ports, avec formulaires, en utilisant des cookies, …).

### 3.2.2 Ajout d’Entity Framework sur Visual Studio

Une fois que le projet est créé, il faut ajouter le package Entity Framework. Dans les paragraphes suivants, je vais expliquer comment obtenir ce package, qui est constitué de :

- **Entity Framework Tools** : Outils qui incluent le Designer et l’assistant de modèle d’Entity Framework, et qui sont nécessaires pour le développement dit « database first » ou « model first ». Lors d’une installation personnalisée de Visual Studio, il faut s’assurer qu’Entity Framework 6 Tools soit sélectionné, soit en étant inclus dans une charge de travail, ou en étant sélectionné individuellement.

- **Entity Framework Runtime** : Environnement d’exécution. Il est responsable de l’exécution de programmes dans un langage donné, tout en ayant des services tels que : les entrées-sorties, l’arrêt des processus, l’utilisation des services du système d’exploitation, la génération d’évènements, le débogage, le profilage, …

Depuis la version 4.1, les nouvelles versions du Runtime ont été publiées en tant que package NuGet : EntityFramework NuGet Package (compatible avec le .NET 4 et .NET 4.5. ).

Avant la 6ème version, les fonctionnalités principales d’Entity Framework faisaient partie du .NET Framework, mais désormais, elles font partie du package NuGet.

Nous pouvons tout de même installer une version spécifique du runtime en exécutant une commande dans la console du gestionnaire de package, la voici :



Cependant, les runtimes qui précédaient la version 4.1 ne peuvent pas être installées séparément car elles faisaient partie du .NET Framework.

On peut également installer le package en tapant la même commande sans spécifier la version, ce qui installera la toute dernière version disponible. (Annexe 2 – Figure 9)

Une autre possibilité est d’installer le package via la fenêtre de Gestionnaire des packages NuGet.

Pour cela, faites un clic-droit sur le projet dans l’explorateur de solutions, et cliquez sur « Gérer les packages NuGet » (Annexe 2 – Figure 10).

Une nouvelle fenêtre s’ouvre et vous pouvez maintenant rechercher le package et l’installer (Annexe 2 - Figure 11).

### 3.2.3 Ajout d’un modèle de données : exemple

Il faut maintenant créer une classe qui sera un modèle de données afin de créer une nouvelle table avec ses colonnes en base de données.

En annexe 7 se trouvent deux captures d’écrans de code effectués lors de mon stage, chacun étant des exemples de modèle de données.

Comme nous l’avons vu dans le point « DataAnnotations », il faut ajouter des configurations aux colonnes de la table, sur la capture d’écran Figure1 La table/classe Parametres aura comme clé primaire la propriété Id, grâce au mot-clé « Key ». Cette table sera liée à la table PriseEnChargePatient grâce au mot-clé « ForeignKey », qui indique que la propriété IdPriseEnCharge est une clé étrangère.

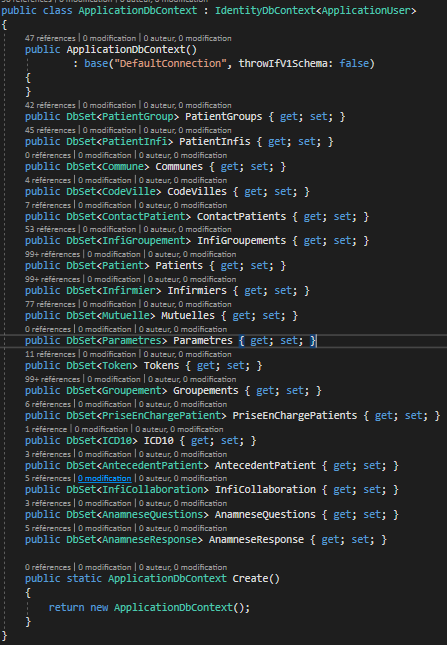
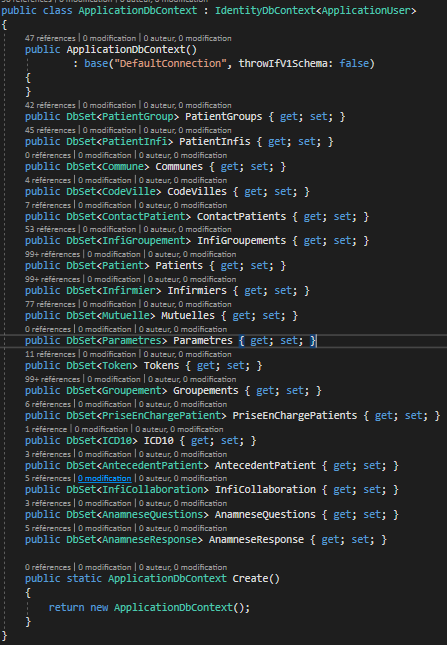
Un autre exemple, sur la capture d’écran Figure2, permet d’expliquer d’autres mot-clés, comme le mot-clé « Display », qui spécifie que la propriété Prenom sera affichée sur l’interface graphique avec la chaine de caractères « Prénom ». Le mot-clé « Required » peut quant à lui spécifier qu’un champ est requis, et une entité ne sera pas ajoutée si la valeur de la propriété configurée avec ce mot-clé est NULL. Il est également possible d’ajouter un message d’erreur, qui s’affichera par exemple dans un formulaire de contact, pour indiquer à l’utilisateur qu’il n’a pas indiqué son prénom et que celui-ci est obligatoire.

### 3.2.4 Définition et utilisation de DbContext

Il faut tout d’abord ajouter une classe dérivée de la classe DbContext, c’est la façon la plus recommandée de travailler.

Dans cette classe dérivée, il faut ajouter des DbSet, qui sont des collections des entités spécifiques du contexte. En travaillant avec le Code First, il faut écrire le contexte soi-même, mais il est également possible de travailler avec Entity Framework Designer. Dans ce cas, le contexte est généré automatiquement.

Lors de mon stage, j’ai également utilisé ASP .NET Identity, qui est un framework permettant de stocker toutes les informations de l’utilisateur dans la base de données. Il utilise le code first d’Entity Framework afin d’implémenter tous ses mécanismes de persistances.



Dans mon projet de stage, j’ai utilisé ce framework pour restreindre l’accès à certaines parties du site en fonction des rôles ayant été attribués à un utilisateur. On peut ajouter des rôles tels que Admin et ajouter des utilisateurs ayant ce rôle.



Il comprend également un système d’authentification basé sur les revendications, c’est-à-dire que l’identité d’un utilisateur est représentée par un ensemble de revendications.

L’authentification ASP .NET est basée sur OWIN, qui est un interlogiciel pouvant être utilisé sur n’importe quel hôte OWIN. ASP .NET Identity utilise ce mécanisme d’authentification pour la connexion ou la déconnexion d’un utilisateur sur le site web. Au lieu d’utiliser FormsAuthentication pour générer le cookie, OWIN CookieAuthentication le permet et est donc utilisé.

Voici maintenant les fonctionnalités reprises ci-dessus :

* Ajouter une nouvelle entité :

Pour créer une nouvelle entité, c’est comme construire une nouvelle instance d’un objet et l’enregistrer en utilisant la méthode Add du DbSet. Un exemple est disponible en Annexe 8 Figure1

* Modifier une entité existante :

Dans ce cas, c’est comme mettre à jour des valeurs assignées à une propriété, qui sont donc sauvegardées en utilisant la méthode SaveChanges. Un exemple est disponible en Annexe 8 Figure2

* Supprimer une entité existante :

Dans ce cas, il faut utiliser la méthode Remove sur le DbSet. Elle fonctionne tout aussi bien pour les entités existantes que celles qui ont été ajoutées récemment.

Si nous appelons cette méthode sur une entité qui n’a pas encore été ajoutée dans la base de données, l’ajout sera tout simplement annulé, car l’entité est supprimée du « tracker de changements » et n’est plus suivie par le DbContext.

En revanche, si l’entité qui est suivie par le « tracker » existe déjà , elle sera supprimée dès que la méthode SaveChanges sera appelée. Un exemple est disponible en Annexe 8 Figure3

* *Les transactions*

À chaque fois que la méthode SaveChanges() est appelée, l’opération associée (ajout, modification, suppression, …) sera enveloppée dans une transaction. Ces transactions sont totalement transparentes pour le développeur et se terminent une fois que l’opération est exécutée. À chaque nouvel appel de la méthode, une nouvelle transaction commence.

Il existe deux méthodes permettant de commencer ou d’utiliser une transaction :

* Database.BeginTransaction() : c’est une méthode simple permettant d’effectuer des transactions, elle permet également de combiner plusieurs opérations dans une seule et même transaction.
* Database.UseTransaction() : Permet au DbContext d’utiliser une transaction qui a débuté extérieurement à Entity Framework.

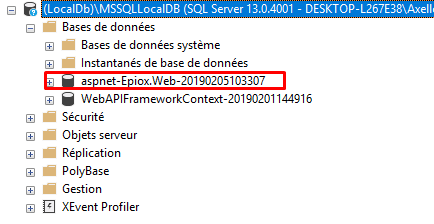
### 3.2.5 Ajout d’une chaine de connexion

Pour utiliser un service de base de données, il faut ajouter la une chaine de connexion dans le fichier Web.config, voici l’exemple de chaine reprise dans le projet que j’ai effectué lors de mon stage :

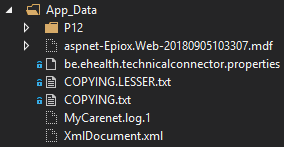
<connectionStrings>

<add name="DefaultConnection" connectionString="Data Source=(LocalDb)\MSSQLLocalDB;AttachDbFilename=|DataDirectory|\aspnet-Epiox.Web-20180905103307.mdf;Initial Catalog=aspnet-Epiox.Web-20190205103307;Integrated Security=True" providerName="System.Data.SqlClient" />

</connectionStrings>



Une fois que cette chaine est ajoutée et dès que le développeur effectuera une migration, la base de données est disponible dans le service de gestion de base de données, et un fichier mdf est ajouté dans le dossier App\_Data du projet :



### 3.2.6 Code-based migration : exemple

Le développeur peut dès à présent ajouter les vues et contrôleurs nécessaires au projet. Cependant, celui-ci devra effectuer une migration pour créer la base de données sur le système de gestion de base de données, ou y transmettre les modifications si le projet était déjà existant.

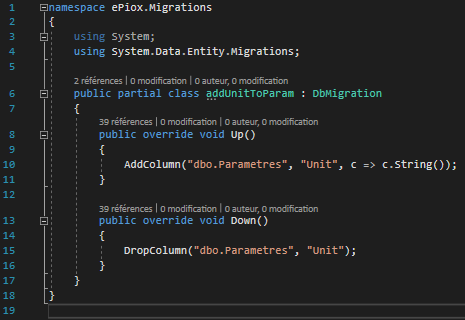
Si le développeur ne fait pas de migration après avoir modifié un modèle de données, alors une exception sera levée. Une capture d’écran illustrant cette exception est disponible en Annexe 9.

Comme cela a été expliqué dans la partie Migrations, si c’est la première fois que le développeur effectue une migration sur ce projet, il doit alors effectuer la commande « enable-migrations » dans la console de gestion de package. Si ce n’est pas le cas, alors il peut directement ajouter une nouvelle migration.

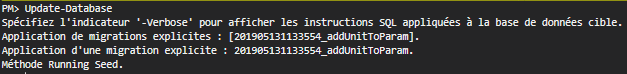
Après chaque modification d’un modèle de données, il faut mettre à jour le schéma de base de données en ajoutant une nouvelle migration, cela se fait grâce à la commande suivante (toujours dans la console de gestion de package) :



Une fois que cette commande est exécutée, une classe de migration est générée automatiquement :



Il faut maintenant effectuer la commande suivante qui exécutera la classe de migration pour mettre à jour la base de données :



La structure de la base de données étant mise à jour et correspondant aux classes du domaines, l’application est prête à s’exécuter.

# **Conclusions**

Ces 15 semaines de stage m’ont permis d’apprendre à m’intégrer dans un milieu professionnel, mais également à adopter une nouvelle cadence de travail ainsi qu’une toute autre méthode de développement.

J’ai été confrontée plusieurs fois à des méthodes ou technologies que je n’avais encore jamais rencontrées. Cela m’a permis d’acquérir de nouvelles connaissances et de m’améliorer dans le domaine du développement web. J’ai souvent dû faire des recherches pour me documenter à propos de techniques que je n’avais pas encore utilisées, ou tout simplement pour trouver une solution lorsque quelque chose ne fonctionnait pas. J’ai également dû apprendre à aller vers les autres pour leur faire part des problèmes rencontrés et leur demander de l’aide.

De plus, j’ai pu me rendre compte que la vision d’un développeur n’est pas toujours la vision idéale pour un client quelconque. En effet, un développeur peut être tellement plongé dans son code qu’il oublie parfois la facilité d’utilisation de l’application. J’ai donc eu l’occasion de travailler avec un web designer, qui pensait à l’ergonomie et au design de l’application que je devais ensuite reproduire.

Lors de mon stage, j’ai eu la chance d’utiliser la technologie Entity Framework, que je connaissais très peu. J’ai ainsi appris à la maitriser aussi bien grâce au développement effectué que grâce aux recherches faites pour la réalisation de mon travail de fin d’études. Je peux donc conclure que les avantages d’Entity Framework en tant que solution sont d’une part, sa facilité de mise en œuvre, le fait qu’il est customisable (différentes approches de développement, types de chargement d’entités, héritage de tables, …), et d’autre part, le fait qu’il est totalement approprié pour les gros projets. Il permet en effet une pérennité de la solution, une évolution rapide, et comme dit ci-dessus, une customisation poussée et une mise en œuvre aisée.

Grâce à ce stage au sein de la société Hygie-Care, j’ai gagné plus d’assurance en ce qui concerne mes capacités professionnelles et mes capacités d’apprentissage. De plus, j’ai ajouté de nouvelles compétences à celles que le cursus scolaire m’a données, et c’est totalement satisfaite que je termine cette expérience, prête à débuter ma vie professionnelle.

# **Bibliographie**

* **Documents PDF :** 
  + Langages Centrés Données, Cours 2 : Rappels JDBC Impedance Mismatch JPA & Hibernate, Université de Paris-Saclay (consulté le 20/03/2019) : <https://www.lri.fr/~kn/teaching/lcd/cours/pdf/02.pdf>
  + Apprenez Entity Framework par les contributeurs de Stack Overflow (eBook) (consulté la première fois le 10/03/2019)
* **Sites web :**
  + .NET et les ORM (consulté le 21/03/2019) : <https://blog.cellenza.com/publications/dotnet-et-les-orm/>
  + Mappage Objet Relationnel (consulté le 25/03/2019) : <https://www.editions-eni.fr/open/mediabook.aspx?idR=bf96af4804b1e94af3740097a6e45f18>
  + Entity Framework 6.2 Runtime Released (consulté le 26/03/2019) : <https://devblogs.microsoft.com/dotnet/entity-framework-6-2-runtime-released/>
  + Symboles et notation des diagrammes entité-association (consulté le 01/04/2019) : <https://www.lucidchart.com/pages/fr/symboles-et-notation-des-diagrammes-entite-association>
  + Comprendre Scrum en moins de 10 minutes (consulté le 29/01/2019) : <https://www.thierry-pigot.fr/scrum-en-moins-de-10-minutes/>
  + Introduction to ASP .NET Identity (consulté le 03/04/2019) : <https://docs.microsoft.com/en-us/aspnet/identity/overview/getting-started/introduction-to-aspnet-identity>
  + Entity Framework 6 (Documentation de Microsoft consultée la première fois le 15/03/2019) : <https://docs.microsoft.com/en-us/ef/>
  + Entity Framework Tutorial TutorialsPoint (consulté la première fois le 02/03/2019) : <https://www.tutorialspoint.com/entity_framework/entity_framework_overview.htm>
  + Entity Framework Tutorial : What is Code First ? (Consulté le 04/05/2019): <https://www.entityframeworktutorial.net/code-first/what-is-code-first.aspx>
  + 3 reasons to use code first design with Entity Framework (Consulté le 12/05/2019) : <https://www.itworld.com/article/2700195/3-reasons-to-use-code-first-design-with-entity-framework.html>
  + Définition de persistance (Consulté le 22/05/2019) : <http://gardeux-vincent.eu/Documents/ProjetJEE/LLM_Hibernate_JSF/Persistence.html>
  + Authentication Method in ASP.NET (consulté le 23/05/2019) : https://www.c-sharpcorner.com/UploadFile/1d42da/authentication-method-in-Asp-Net/

# **Annexes**

## 6.1 Annexe 1 : Schémas d’ORM



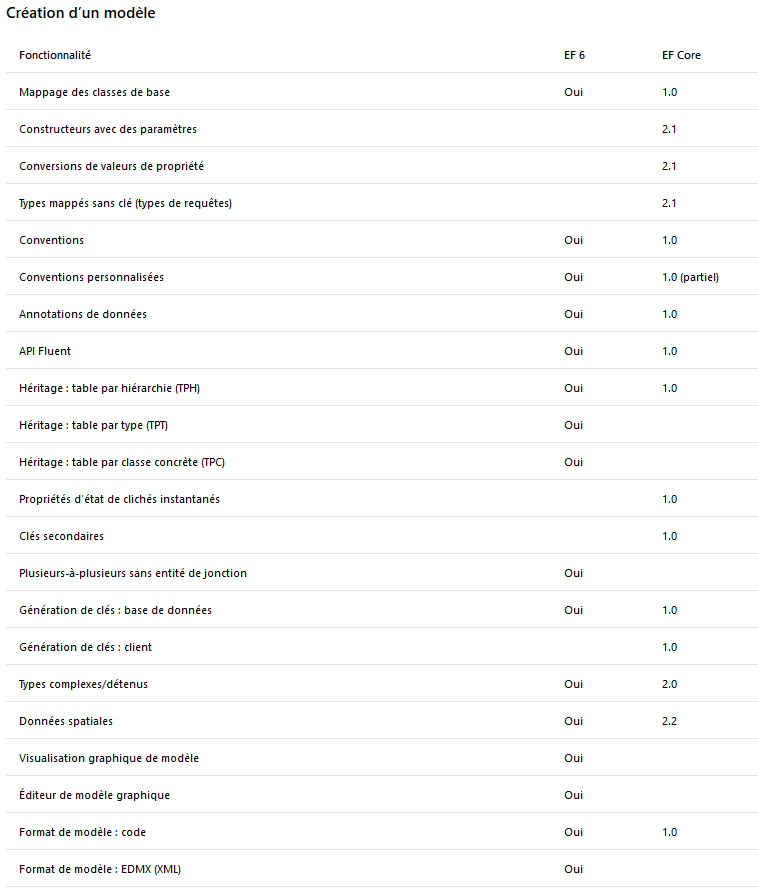
Figure 1 : Schéma d’un ORM typique (créé à partir de la documentation de TutorialsPoint)

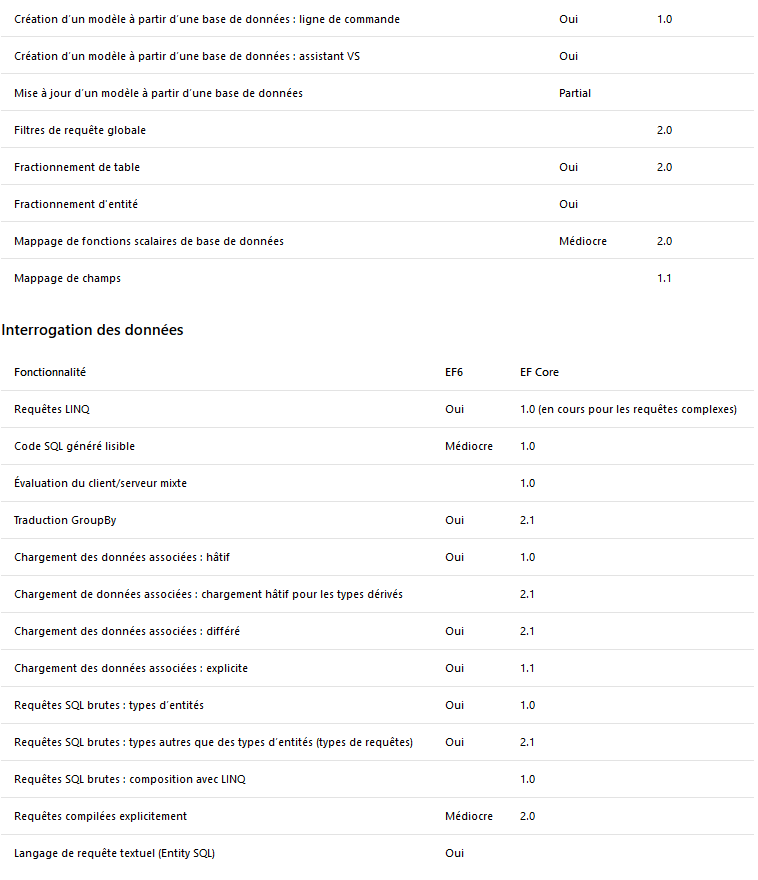


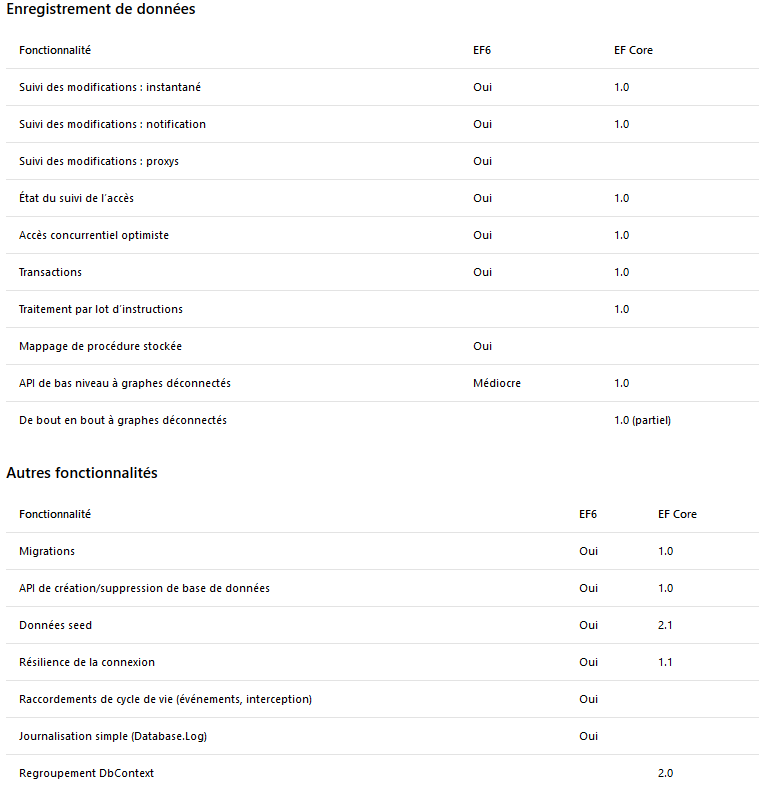
## 6.2 Annexe 2 : Fonctionnalités d’ORM implémentée par Entity Framework 6

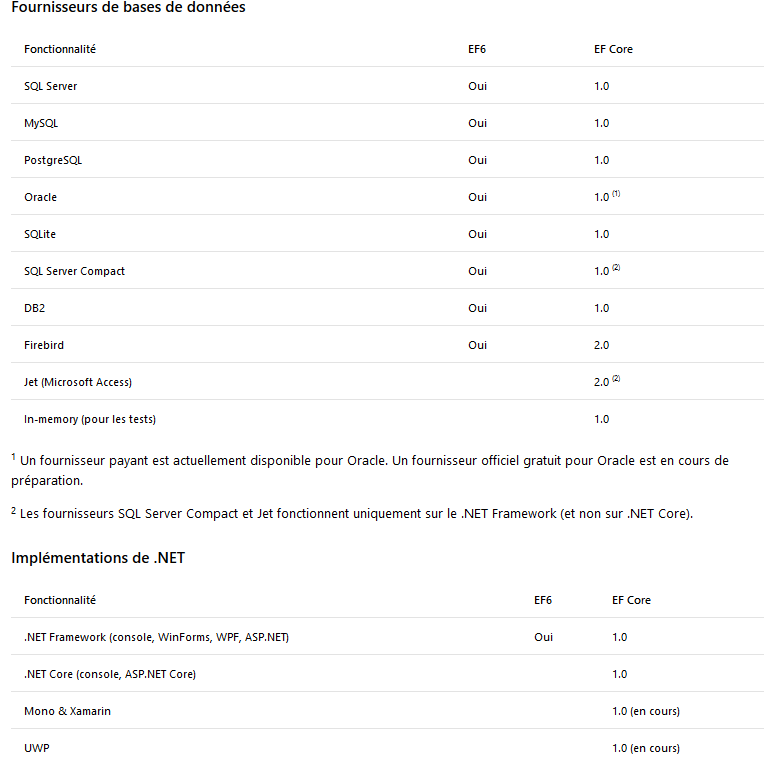
* *Mappage de classes d’entité OCT qui ne dépendent pas d’un type EF*
* *Détection automatique des changements*
* *Résolution d’identité et unité de travail*
* *Chargement hâtif, différé et explicite*
* *Traduction des requêtes fortement typées à l’aide de LINQ (Language Integrated Query)*
* *Fonctionnalités de mappage complètes, notamment la prise en charge des points suivants :* 
  + *Relations un-à-un, un-à-plusieurs et plusieurs-à-plusieurs*
  + *Héritage (table par hiérarchie, table par type et table par classe concrète)*
  + *Types complexes*
  + *Procédures stockées*
* *Un concepteur visuel pour créer des modèles d’entité.*
* *Une expérience « Code First » pour créer des modèles d’entité en écrivant du code.*
* *Les modèles peuvent être générés à partir de bases de données existantes et modifiés à la main, ou créés à partir de zéro et utilisés pour générer de nouvelles bases de données.*
* *Intégration aux modèles d’application .NET Framework, y compris ASP.NET, et au moyen de la liaison de données avec WPF et WinForms.*
* *Connectivité de base de données basée sur ADO.NET et nombreux fournisseurs disponibles pour établir la connexion à SQL Server, Oracle, MySQL, SQLite, PostgreSQL, DB2, etc*

## 6.3 Annexe 3 : Comparaison entre Entity Framework 6 et Entity Framework Core









## 6.4 Annexe 4 : Choix d’approche Entity Framework (schéma)

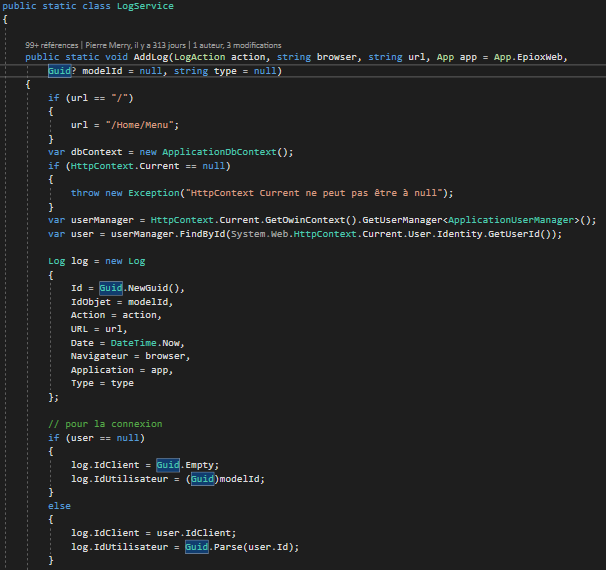


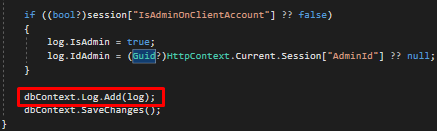
## 6.5 Annexe 5 : Exemple d’un service de log

Voici la ligne de code à ajouter pour créer une nouvelle entrée de log dans la table de la base de données :

****

La méthode AddLog est une méthode de la classe LogService, les variables passées en paramètres sont traitées et un nouvel objet Log est créé puis est ensuite ajouté à la base de données grâce à la méthode Add() du DbContext :

****



## 6.6 Annexe 6 : Ajout du package NuGet Entity Framework

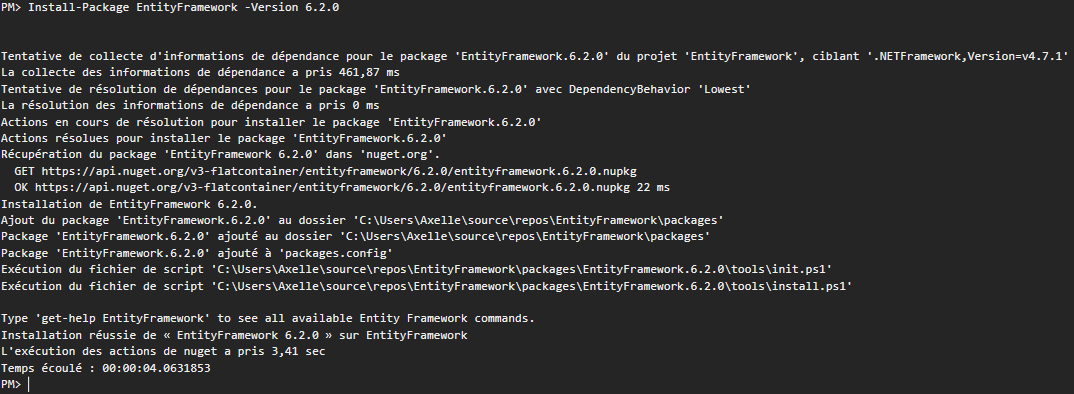


Figure 9 : Commande d’installation d’une version spécifique du package Entity Framework

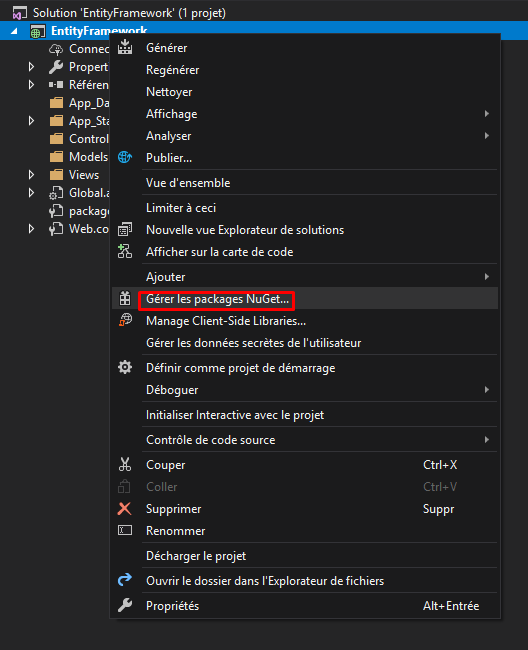


Figure 10 : Dans l’explorateur de solution, ouvrir le gestionnaire de package NuGet

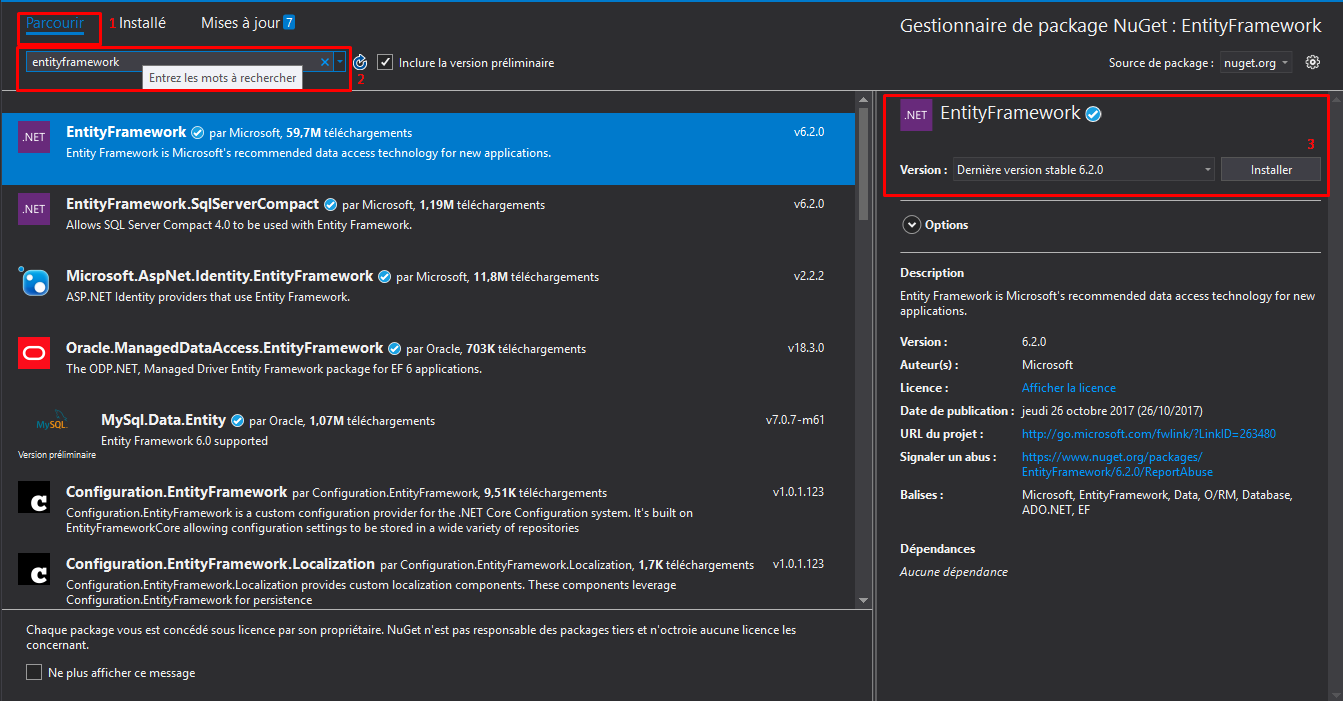
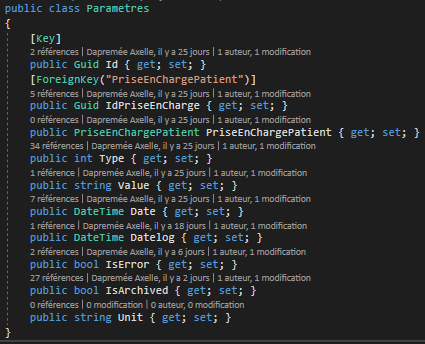


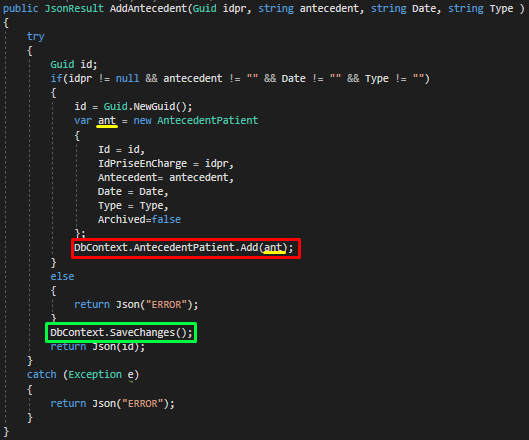
Figure 11 : Rechercher et installer le package NuGet Entity Framework

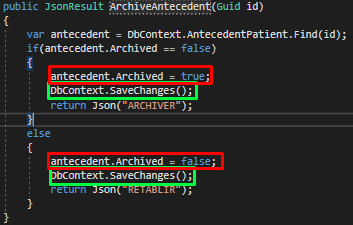
## 6.7 Annexe 7 : Exemples de modèles de données



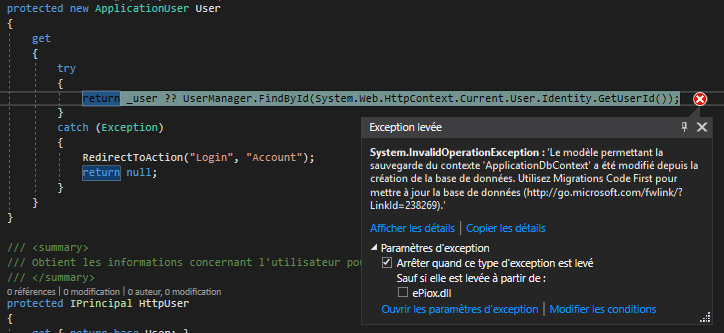


## 6.8 Annexe 8 : Ajout, Modification et Suppression d’entités



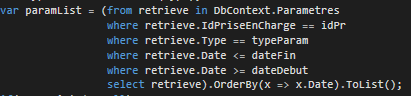


## 6.9 Annexe 9 : System.InvalidOperationException



## 6.10 Annexe 10 : Exemples de requêtes LINQ to SQL

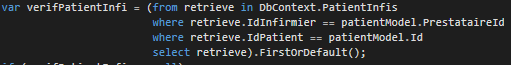
* **Exemple 1 :** Par cette requête, l’utilisateur recherche dans la table paramètre, avec plusieurs conditions spécifiées grâce à la clause « where », lorsque ceux-ci sont mis les uns à la suite des autres, cela équivaut à des opérations AND. Ensuite le développeur effectue un tri sur les entités récupérées grâce à la méthode « OrderBy », et il utilise ensuite la méthode « ToList » pour que la requête retourne une liste d’entités.



* **Exemple 2 :** Ceci n’est pas une requête LINQ to SQL, mais c’est ici la méthode « Find » du DbContext permettant de retrouver une entité précise grâce à sa clé primaire.



* **Exemple 3 :** Cette requête est semblable à celle du premier exemple, mais ici la méthode FirstOrDefault() est utilisée, retournant uniquement une seule entité et une valeur NULL si aucune entité n’est retrouvée.



* **Exemple 4 :** Avec cette requête, le développeur cherche à effectuer une jointure entre les tables Patients et PatientsInfis, avec une condition spécifiée avec les mot-clés « on » et « equals ». Il effectue également un tri mais cette fois avec un second niveau grâce à la méthode « ThenBy », si deux patients ont le même nom, alors un tri se fera avec leur prénom.



* **Exemple 5 :** Cette requête, toujours semblables aux précédentes, démontre un exemple de conditions les clauses « where » mais avec une opération || ou OR.

