Concepteur Développeur d’Applications

Dossier projet

Gestion du parc informatique

Jalon 2 : Développer une application

EPCF 2 du 15/02/2024

TORRENTI Sylvain

CDA 07

A picture containing text, circle, diagram, design

Description automatically generated

# Remerciements

Pour commencer, je souhaite remercier la Maison Départementale des Personnes Handicapées des Bouches du Rhône, Cap Emploi, CT conseil et le centre 2ISA pour m’avoir permis cette reconversion professionnelle.

Je remercie vivement l’équipe pédagogique. Que ce soient les formateurs techniques, **Serge BOISSEAU**, **Théo OLLIVIER-TRIQUET**, **Fabien BELUGOU** et **Philippe PALAU** qui m’ont permis de mieux appréhender les différentes problématiques auxquelles j’ai été confronté mais aussi les formateurs intervenus pour les compétences transverses, **Sarah KINSLEY, Annelies AANTJES, Dylan PEREZ** et **Alexandra TESTART.**

Je souhaite également remercier **Muriel ANDREO** et **Carole GALTIER** pour leur accueil au sein de l’établissement ainsi que tous mes collègues de la formation, qui ont permis un cadre propice à l’apprentissage.

Et enfin, merci à ma famille qui m’a soutenu et encouragé durant cette dernière étape de ma formation.

Table des matières

[Remerciements 2](#_Toc158232546)

[Liste des compétences couvertes par le projet 4](#_Toc158232547)

[Résumé 5](#_Toc158232548)

[Introduction 6](#_Toc158232549)

[Prérequis du projet 7](#_Toc158232550)

[Mise en place du projet 8](#_Toc158232551)

[Développement du projet 10](#_Toc158232552)

[Modification de la Base De Données 10](#_Toc158232553)

[Côté Client 12](#_Toc158232554)

[Windows Form 12](#_Toc158232555)

[Les onglets 15](#_Toc158232556)

[Connexion 15](#_Toc158232557)

[Consulter 16](#_Toc158232558)

[Création 17](#_Toc158232559)

[Modification 19](#_Toc158232560)

[Suppression 21](#_Toc158232561)

[Côté Serveur 22](#_Toc158232562)

[La structure 22](#_Toc158232563)

[L’API 23](#_Toc158232564)

[Les controller 23](#_Toc158232565)

[AccountController 25](#_Toc158232566)

[GestionMaterielServiceController 26](#_Toc158232567)

[BLL 27](#_Toc158232568)

[SecurityService 28](#_Toc158232569)

[GestionMaterielService 28](#_Toc158232570)

[DAL 29](#_Toc158232571)

[Le pattern Repositories 30](#_Toc158232572)

[Le repository Materiel 31](#_Toc158232573)

[GetAll 32](#_Toc158232574)

[GetSort 33](#_Toc158232575)

[Update 34](#_Toc158232576)

[Add 35](#_Toc158232577)

[Delete 36](#_Toc158232578)

[Les Test 36](#_Toc158232579)

[Les tests Unitaire 36](#_Toc158232580)

[Test GetSortByCategorieMaterielAsync 38](#_Toc158232581)

[Test 1 39](#_Toc158232582)

[Test 2 39](#_Toc158232583)

[Test 3 40](#_Toc158232584)

[Résultat des test 41](#_Toc158232585)

[Test d’Intégration 42](#_Toc158232586)

[Tests d’Acceptation 44](#_Toc158232587)

[Future évolution 45](#_Toc158232588)

[Conclusion 45](#_Toc158232589)

# Liste des compétences couvertes par le projet

* Maquetter une application.
* Développer des composants d’accès aux données.
* Concevoir une application.
* Concevoir une Base De Données.

# Résumé

Le projet consiste à créer une application Windows Form pour gérer la gestion du parc informatique.

Dans cette optique, je commence par analyser le **cahier des charges**. Je mets avant les fonctionnalités qui seront nécessaires à l’accomplissement du projet.

J’ai effectué ma **gestion de projet** grâce à l’outil en ligne [www.monday.com](http://www.monday.com) qui m’a permis de mieux gérer mon temps et de planifier les étapes qui me restaient à effectuer durant la mise en place de ce projet.

Une fois ces fonctionnalités listées, je produis une **maquette** en utilisant l’outil **FIGMA.** Cet outil permet de créer l’enchainement des écrans disponibles en indiquant les liens à suivre et peut se partager en ligne sans contrainte.

Une fois satisfait de la maquette, j’entreprends la **conception** du projet. J’utilise **Visual Studio** comme **IDE** et **.Net** comme **Framework**.

J’ai créé une **bibliothèque de classe** ou j’ai réuni les **DTO** (Data Transfert Object), les **entités** et les **exceptions** pour bien séparer les différents éléments composant l’ensemble du projet.

Ensuite j’utilise une structure **multicouches** que j’utilise coté serveur pour récupérer les données dans le SGBD **MYSQL** créé à l’aide de **DBeaver** et les retourne à l’API.

L’utilisateur utilise le logiciel créé grâce à la maquette pour interroger l’API et obtenir les données voulues ou effectuer les opérations souhaitées selon son rôle.

Mots Clés : cahier des charges, gestion de projet, maquette, FIGMA, conception, Visual Studio, IDE, .Net, Framework, bibliothèque de classe, DTO, entités, exceptions, multicouches, MYSQL, DBeaver

# Introduction

Je m’appelle Sylvain TORRENTI, j’ai 32 ans. J’étais, auparavant, employé polyvalent dans l’hôtellerie restauration et j’ai décidé de me reconvertir dans l’informatique. Ce secteur m’attire depuis toujours.

L’association 2ISA a été créée le 23 juin 2010. Elle propose des formations pour les personnes en situation de handicap mais possède également un organisme qui forme toutes personnes possédant le niveau Bac aux métiers du numérique.

Le concepteur Développeur d’Applications conçoit et développe des services numériques à destination des utilisateurs. Il intervient dans des projets visant à automatiser des processus de l'entreprise. Ces projets font suite à des demandes formulées directement par un client, par une maîtrise d’ouvrage ou par l’intermédiaire d’un chef de projet.

Durant la seconde partie de ma formation qualifiante Concepteur Développeur d’Application au sein de l’institut 2ISA, j’ai pu mettre en œuvre mes compétences techniques lors de la création d’un projet pour l’EPCF qui se déroule le 15/02/2024.

Pour ce projet, qui se déroule en deux étapes importantes, j’ai dû mettre en œuvre une application Windows Form pour permettre la gestion du parc informatique. Pendant la seconde étape, j’ai dû permettre la consultation et le filtrage/triage des informations par tous les utilisateurs mais aussi de permettre aux administrateurs de pouvoir enregistrer, allouer, supprimer… différents matériels.

# Prérequis du projet

Les prérequis techniques présents dans le projet ont été étudiés avec les différents formateurs techniques. Ils ont été présentés de façon à pouvoir effectuer le travail en continu. Ce projet a permis de mettre en pratique toutes les notions que nous avions abordées depuis le début de la formation.

Pour ce projet, et pour toute la formation, les éléments présents sur le site AMIO-FIT permettent de trouver les informations recherchées qu’elles soient transverses ou techniques.

Certaines informations ont nécessité des recherches internet que j’ai effectuées en anglais car les résultats sont plus nombreux et mieux documentés.

# Mise en place du projet

Pour commencer j’ai utilisé le langage UML pour dégager les comportements nécessaires. J’ai tout d’abord fait un Use Case Diagram comme le montre la figure 1.

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

Description générée automatiquement

Figure 1 : Use Case Diagram

J’ai également produit un Class Diagram (figure2).

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

Description générée automatiquement

Figure 2 : Class Diagram

Puis je me suis servi de l’outil de maquettage en ligne FIGMA comme le montre la figure 3.

Une image contenant texte, capture d’écran, logiciel, Logiciel multimédia

Description générée automatiquement

Figure 3: Illustration des onglets sous FIGMA

Le maquettage m’a permis d’avoir une vision globale de mon projet et ainsi de réfléchir en amont aux problématiques possibles.

# Développement du projet

## Modification de la Base De Données

Dans un premier temps, j’ai modifié la Base De Données utilisée lors de la première étape du projet fil rouge. Un exemple d’un script SQL pour créer une nouvelle table dans la BDD (figure 4).

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

Description générée automatiquement

Figure 4 : Exemple de SCRIPT

Comme nous pouvons le voir dans la figure 4, j’ai choisi de ne pas créer les clés étrangères pendant la création des tables pour éviter les éventuels conflits.

J’ai donc, par la suite, utilisé un autre script (figure 5).

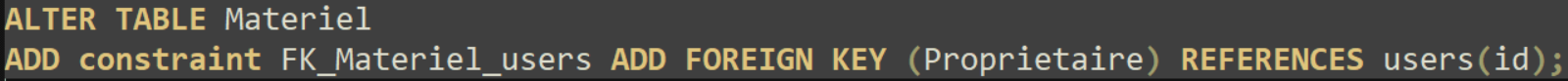


Figure 5 : Script Alter Table

Ce script permet de modifier les tables existantes et d’y ajouter les clés étrangères.

Par la suite, pour toutes les requêtes du projet, j’ai d’abord utilisé DBeaver et son outil de Script SQL pour ainsi pouvoir vérifier l’exactitude de la requête. Je l’ai ensuite incorporée dans mon code en y effectuant les modifications nécessaires pour son bon fonctionnement.

Grâce à DBeaver nous pouvons également voir les liens entre nos tables. Comme le montre la figure 6, voici les tables rajoutées pour le bon fonctionnement du projet.

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

Description générée automatiquement

Figure 6 : Diagramme BDD

La figure 6 nous montre que les tables **Materiel**, **CatMat** et **Catégorie** ont été ajoutées. La table **Materiel** et la table **users** sont reliées par la présence de la clé étrangère Proprietaire dans la table **Materiel** car un Materiel ne peut avoir qu’un seul propriétaire mais un Propriétaire peut avoir plusieurs Materiel.

En ce qui concerne la table **Categorie**, j’ai dû créer un table d’association **CatMat** qui relie la table **Materiel** et la table **Categorie** car un Materiel peut appartenir à plusieurs Categorie et une Categorie peut avoir plusieurs Materiel qui y font référence.

# Côté Client

## Windows Form

Pour cette deuxième étape du projet, j’ai utilisé une IHM réalisée grâce à Windows Form. Cette technologie permet de créer des fenêtres qui permettent d’afficher les données que nous désirons.

Il y avait plusieurs possibilités et j’ai opté pour une approche avec une seule fenêtre contenant les différents onglets nécessaires (figure7).

Une image contenant texte, capture d’écran, logiciel, ordinateur

Description générée automatiquement

Figure 7 : Fenêtre avec onglets

La figure 7 représente la fenêtre avec le premier onglet d’affiché.

Un WinForm est composé de plusieurs « control ». La bonne pratique veut que nous renommions ces « control » pour faciliter leur lecture et l’implémentation d’un futur code.

Une image contenant texte, capture d’écran, Police

Description générée automatiquement

Figure 8 : Structure du document

La figure 8 représente une partie de la structure du document. Pour le nommage je prends les initiales du « control » et j’ajoute ce à quoi il fait référence avec le nom de l’onglet dans lequel il est incorporé car plusieurs « control » concerne les s actions mais sur des onglets différents.

Je n’ai pas renommé les Label car ils ne changent pas durant l’exécution et ne sont reliés à aucun code.

L’affichage change si l’utilisateur est authentifié ou non mais également s’il s’agit d’un administrateur ou d’un simple utilisateur. Son rôle est vérifié lors du login de l’utilisateur. Un token JWT (JSON Web Token) est créé lors du login et permet d’obtenir les informations concernant l’utilisateur présentent dans la BDD.

Une image contenant capture d’écran, texte, logiciel, Icône d’ordinateur

Description générée automatiquement

Figure 9 : Fenêtre avant authentification

La figure 9 représente l’affichage lorsque l’application Windows Form est lancée. Nous pouvons ainsi voir que seul l’onglet Connexion est présent.

Une image contenant capture d’écran, ordinateur, texte, logiciel

Description générée automatiquement

Figure 10 : Fenêtre après authentification

La figure 10 représente l’authentification d’un administrateur. Nous pouvons voir qu’il a accès à tous les onglets permettant de faire des opérations sur le Materiel et que l’onglet de connexion n’est plus présent, il sera disponible si l’utilisateur utilise le bouton de Déconnexion.

## Les onglets

Ils permettent la navigation de l’administrateur selon l’opération voulue (figure 11).



Figure 11 : Onglets

### Connexion

Le premier onglet est très simple, il permet la connexion. Il est accessible dès l’ouverture de l’application. Nous pouvons voir sa structure grâce à la figure 12 et aussi ce que l’utilisateur peut voir grâce à la figure 13.

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

Description générée automatiquement

Figure 12 : Structure de l'onglet Connexion

Une image contenant capture d’écran, texte, logiciel, ordinateur

Description générée automatiquement

Figure 13 : onglet Connexion

### Consulter

L’onglet consulter est disponible par tous les utilisateurs quel que soit leur rôle. Sa structure est simple comme le montre la figure14. L’affichage est lui aussi très simple.

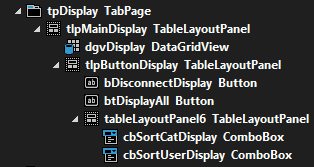


Figure 14 : Structure de l'onglet Consulter

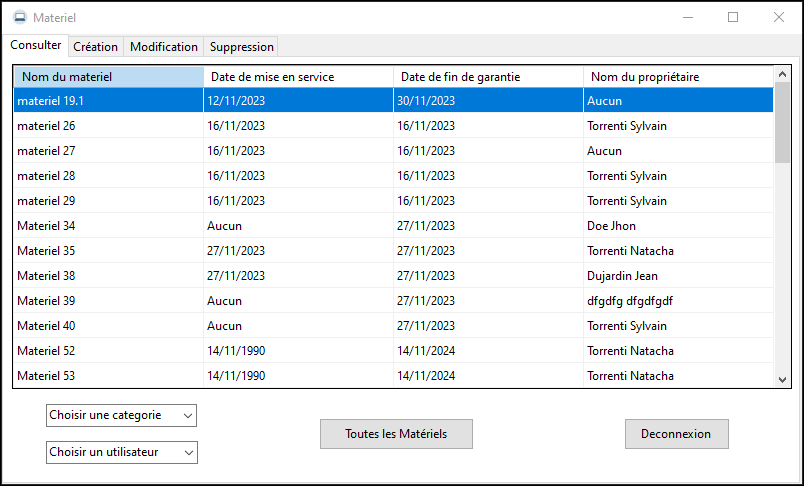


Figure 15 : onglet Consulter

Cet onglet permet aussi à l’utilisateur de filtrer les résultats qu’il souhaite afficher en fonction d’une catégorie ou d’un utilisateur demandés (figure 15).

### Création

L’onglet création n’est disponible qu’aux utilisateurs qui ont le rôle administrateur. Il permet d’enregistrer un nouveau materiel et aussi de créer une nouvelle catégorie (figure 16).

Une image contenant texte, capture d’écran, Police

Description générée automatiquement

Figure 16 : Structure de l'onglet Creation

Une image contenant texte, capture d’écran, affichage, logiciel

Description générée automatiquement

Figure 17 : onglet Creation

Cet onglet permet à l’administrateur de renseigner toutes les informations nécessaire à l’enregistrement d’un nouveau materiel. Si la/les catégories n’existent pas il est possible d’en créer de nouvelles (figure 17).

### Modification

L’onglet modification est également disponible seulement aux utilisateurs qui ont le rôle administrateur. Il permet de modifier un materiel déjà présent dans le parc informatique. Si la catégorie désirée n’existe pas, l’administrateur a également la possibilité de la créer via cet onglet. Sa structure est bien plus complète car il faut avoir les informations déjà présentes et pouvoir modifier n’importe laquelle (figure 18).



Figure 18 : Structure de l'onglet Modification

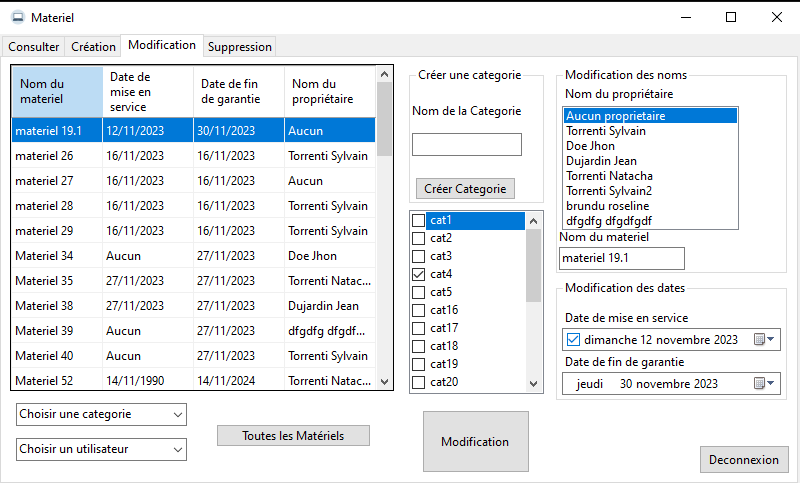


Figure 19 : onglet Modification

Comme nous pouvons le voir grâce à la figure 19, l’administrateur peut modifier toutes les informations concernant un materiel. Il peut également, pour faciliter sa recherche, les trier selon une catégorie spécifique ou un utilisateur.

En ce qui concerne la date de mise en service j’ai mis en place un DateTimePicker avec une CheckBox qui permet d’indiquer si le materiel est mis en service ou non.

J’ai également rajouté la possibilité de ne pas allouer de propriétaire à un materiel en sélectionnant « Aucun propriétaire » dans la sélection du propriétaire.

### Suppression

L’onglet suppression est également seulement disponible aux administrateurs. Sa structure est très proche de celle de l’onglet de consultation (figure 20).

Une image contenant texte, capture d’écran, Police

Description générée automatiquement

Figure 20 : Structure de l'onglet Suppression

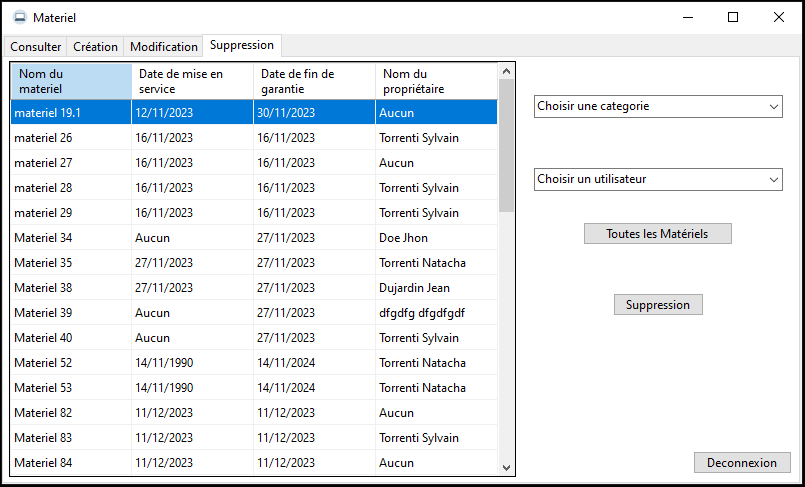


Figure 21 : Liste Tickets (Utilisateur)

Dans la figure 21 nous pouvons voir que l’administrateur peut également trier les matériels comme il le souhaite et ainsi trouver plus facilement ce qu’il cherche. Quand il décide de supprimer un materiel les relations présentent dans la BDD qui lui sont liées sont également supprimées.

# Côté Serveur

## La structure

Du côté serveur j’ai mis en place plusieurs couches pour faciliter l’organisation, la séparation des responsabilités et la maintenabilité.

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

Description générée automatiquement

Figure 22 : Structure côté serveur

Comme nous le montre la figure 22, plusieurs projets sont présents dans la solution. L’utilisateur ne communique qu’avec l’API. C’est elle qui fait les requêtes permettant de récupérer les données dans la BDD.

* L’utilisateur via l’IHM fait sa demande à l’API.
* L’API fait suivre cette demande à la BLL (procède aux vérifications qui lui sont allouées).
* La BLL interroge la DAL qui va faire les requêtes auprès de la BDD pour récupérer les données qui sont demandées.
* Les données font le chemin inverse pour arriver à l’utilisateur.

Durant toute la procédure des vérifications sont faites en fonction de chaque couche et de leurs fonctions.

## L’API

Une image contenant texte, capture d’écran, logiciel, Icône d’ordinateur

Description générée automatiquement

Figure 23 : Structure de l'API

L’API est la porte d’entrée pour permettre à l’utilisateur d’accéder aux données. A chaque fois que l’utilisateur fait une demande, celle-ci passe par le controller adéquat qui fait suivre cette demande à la BLL.

### Les controller

J’ai commencé par créer un BaseController qui hérite de la classe ControllerBase déjà présente dans le Framework. Tous mes futurs controller hériteront de ce controller ce qui facilitera de potentielles futures modifications et me permet d’avoir une configuration par défaut que je ne serai pas obliger de reproduire à chaque controller créés.

Une image contenant texte, capture d’écran, logiciel, Logiciel multimédia

Description générée automatiquement

Figure 24 : BaseController

Comme nous le montre la figure 24, il y a plusieurs décorateurs différents :

* [ApiController] : Qui permet d’indiquer que ce fichier est un Controller.
* [Route("api/")] : Qui définit le début des adresses pour accéder aux différents services que fournit l’API.
* [Authorize (Roles = "Utilisateur")] : Qui limite l’accès aux utilisateurs ayant le rôle utilisateur. Ce qui permet d’obliger l’accès via une authentification. Par la suite je pourrai indiquer que certains services ne sont accessibles que pour les utilisateurs ayant le rôle d’administrateur.

Il y a ensuite les controller pour accéder aux différents services fournis. Il faut créer un controller pour un ensemble de services. Dans mon projet je n’ai eu à créer seulement que deux controller. Un premier qui gère l’authentification et un autre qui gère la gestion du parc informatique.

### AccountController

Une image contenant texte, capture d’écran, Police

Description générée automatiquement

Figure 25 : AccountController

Comme nous le montre la figure 25, il y a la présence de deux décorateurs. Le premier indique qu’il répond à une requête http Post. Il attend donc que l’utilisateur fournisse les éléments nécessaires. Le deuxième indique qu’il n’est pas nécessaire d’être authentifié, ce qui est cohérent pour la page l’authentification *(tu en penses quoi ? tu avais mis logique au lieu de cohérent)*.

Nous pouvons aussi voir que cette requête récupère les informations dans le Body. Également qu’elle est asynchrone ce qui permet d’éviter que l’application se fige pendant le déroulement de la requête.

Si l’authentification échoue une exception BadRequest est retournée. Si elle réussit un token JWT est créé avec les informations de l’utilisateur telles que son rôle pour permettre la navigation qui lui est autorisée.

### GestionMaterielServiceController

Vient ensuite le controller qui gère notre service de gestion de materiel. Je vais vous montrer un exemple d’une fonctionnalité qui est celle de créer un materiel (figure 26).

Une image contenant texte, capture d’écran, Police

Description générée automatiquement

Figure 26 : Gestion Materiel Service Controller

Comme nous le montre cette image, l’utilisateur doit avoir le rôle Administrateur pour accéder à cette fonctionnalité. Même si pour avoir accès à l’onglet de création il faut en premier lieu avoir le rôle administrateur, rajouter cette sécurité évite le cas où une personne qui utilise un logiciel tier (tel que Postman) puisse avoir accès directement à cette fonctionnalité.

Nous pouvons également voir qu’accéder à cette fonctionnalité l’utilisateur doit forcément procéder à une requête http POST en utilisant la route définie par le Base Controller et celle indiquée qui est donc « api/materiels ». Chaque fonctionnalité a sa propre route et son propre mode de requête http. Une même route peut permettre d’accéder à plusieurs services qui ont un mode http différent.

Il y a également l’utilisation de DTO (Data Transfer Object) qui permet le transport de données entre les processus. Cela rend plus facile la communication et évite le risque d’exposition d’informations sensibles. Ainsi pour la création et tous autres services qui nécessitent une modification des données présentes en BDD il y aura l’utilisation de DTO.

## BLL

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

Description générée automatiquement

Figure 27 : structure de la BLL

La BLL sert d’intermédiaire entre l’API et la DAL (Data Acess Layer) et compile les règles métier de l’application. Comme nous le montre la figure 27, la présence d’interface pour chaque service permet de faciliter de futures modifications.

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

Description générée automatiquement

Figure 28 : BLL UOW

A l’intérieur de chaque implémentation de notre BLL nous devons définir le pattern Unit Of Work qui permet d’orchestrer les opérations sous forme de transaction et de coordonner le travail des différents Repository présents dans la DAL en un seul endroit.

### SecurityService

Une image contenant capture d’écran, texte, Police

Description générée automatiquement

Figure 29 : Login

Dans cet extrait (figure 29), nous pouvons voir comment le Login se déroule. L’utilisateur fournit l’email et le password. Le SecurityService fait une demande à la DAL pour obtenir les informations présente dans la BDD, ensuite vérifie leurs exactitudes. Si elles ne correspondent pas il retourne null et si elles sont bonnes retourne un JwtToken contenant les informations de l’utilisateur lui permettant d’accéder aux différents services autorisés.

### GestionMaterielService

Une image contenant texte, capture d’écran, Police

Description générée automatiquement

Figure 30 : Create Materiel

La figure 30 nous montre comment la requête pour la création d’un materiel est effectuée. L’API fournit les informations nécessaires à la création d’un materiel.

Une transaction avec la commande « BeginTransaction » est lancée car comme nous le voyons plusieurs requêtes vont être effectuées. Si toutes les requêtes réussissent alors la transaction est exécutée grâce à la commande « Commit ». Si une des requêtes présentent dans la transaction échoue alors un « Rollback » est effectuée ce qui permet de garder la BDD dans son état initiale.

La première opération est l’ajout du materiel dans la BDD grâce à la ligne « \_dbContext.Materiels.AddAsync(materiel) ; » qui indique que nous utilisons la méthode AddAsync présente dans le repository Materiel qui est lui-même présent dans la DAL en utilisant l’Unit Of Work créé en amont. Nous récupérons ensuite l’Id du materiel nouvellement créé pour l’associer à la liste des catégories fournies. L’ordre des opérations est important car si le materiel n’est pas créé nous ne pouvons pas encore créer les associations avec les catégories (enlever ça : avant que le Materiel soit créé. )

## DAL

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

Description générée automatiquement

Figure 31 : Structure de la DAL

La DAL permet d’avoir accès à la BDD. Dans toute l’application c’est le seul module qui a accès directement à la BDD.

### Le pattern Repositories

Une image contenant texte, capture d’écran, Police

Description générée automatiquement

Figure 32 : Structure du Pattern Repositories

Le pattern Repositories permet de réunir les méthodes propres à chaque entité. Il permet ainsi de rendre de futures modifications plus simples. En plus d’une interface propre à chaque entité nous pouvons voir qu’il existe des interfaces dites génériques. Ces interfaces regroupent des méthodes qui sont communes à toutes les entités.

Une image contenant texte, capture d’écran, Police

Description générée automatiquement

Figure 33 : Repositories interface de lecture générique

Comme nous le montre la figure 33, nous pouvons également définir des paramètres génériques qui peuvent, eux aussi, avoir des paramètres. Le fait d’inscrire « where T : IEntity » signifie que le paramètre T doit avoir « l’étiquette » IEntity. Ainsi nous pourrons retrouver ces méthodes dans les implémentations des repositories de nos entités en plus de celles qui leurs sont spécifiques.

#### Le repository Materiel

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

Description générée automatiquement

Figure 34 : Repositories interface Materiel

Nous pouvons voir dans la figure 34 l’interface qui concerne les Materiel. Celle-ci hérite des Interface génériques et ajoute également des méthodes propres seulement aux Materiel.

##### GetAll

Une image contenant texte, capture d’écran

Description générée automatiquement

Figure 35 : Repositories Materiel GetAll

La figure 35 représente l’implémentation du repositories Materiel pour le système de gestion de base de données MariaDB. Comme nous pouvons le voir, cette classe implémente l’interface IMaterielRepository qui regroupe donc les méthodes génériques et celles qui sont spécifiques aux Materiel.

Comme exemple j’ai choisi la méthode utilisée pour la récupération et l’affichage des materiels. J’utilise cette méthode pour afficher l’ensemble des materiels avec les informations de leur catégories et de leurs propriétaires quand ils en ont un.

Dans cette méthode, la requête récupère toutes les données nécessaires concernant les Materiels, les Catégories et les Utilisateurs. Il faut donc lui indiquer les entités correspondantes pour qu’il puisse mapper convenablement.

Pour que les bonnes données soient correctement enregistrées il faut lui indiquer le bon endroit. Ainsi je lui indique qu’il faut rajouter les « categorie » à la liste des categories du materiel mais également le propriétaire du materiel. Il faut également lui indiquer sur quel « champ » il doit faire un split avec : splitOn : « IdCategorie, id » qui correspond au nom des colonnes des différentes entités présentes en BDD. Grâce à cela il connait les délimitations des entités dans les données récupérées.

##### GetSort

Une image contenant texte, capture d’écran

Description générée automatiquement

Figure 36 : Repositories Materiel GetSort

Dans la figure 36 nous pouvons voir les différentes méthodes pour obtenir les Materiels selon les choix de filtres de l’utilisateur. La requête est sensiblement la même que celle vu précédemment il y a juste un rajout d’une clause « WHERE u.id = @idUser ». Cette clause récupère le paramètre passé dans la signature de la méthode et récupère ainsi seulement les Materiels concernés par cette recherche.

Comme nous pouvons le voir les méthodes pour récupérer les Materiels selon la categorie ou s’ils n’ont pas de propriétaire étaient différentes.

##### Update

Une image contenant texte, capture d’écran, Police

Description générée automatiquement

Figure 37 : Repositories Materiel Update

Quand l’utilisateur veut mettre à jour un Materiel, il le sélectionne, modifie les données voulues puis envoie toutes les informations concernant ce Materiel. Il ne saisit pas directement l’Id du Materiel mais l’Id est obtenu en sélectionnant le Materiel sur l’application.

La requête effectuer va donc modifier les informations avec celle fournie par l’utilisateur pour le Materiel dont l’Id est fourni.

En ce qui concerne les catégories liées à ce Materiel j’ai opté pour la stratégie de supprimer toutes les relations existantes de ce Materiel et de créer de nouvelles avec les catégories listées dans les informations données.

##### Add

Une image contenant texte, capture d’écran, logiciel

Description générée automatiquement

Figure 38 : Repositories Materiel Add

Pour la création de materiel j’effectue un insert dans la table Materiel avec les information fournit par l’utilisateur. Je récupère également l’Id du materiel créé qui me permet d’effectuer l’association entre le Materiel nouvellement créé et sa liste de catégories.

##### Delete

Une image contenant texte, capture d’écran, Police

Description générée automatiquement

Figure 39 : Repositories Materiel Delete

La méthode delete, comme son nom l’indique, sert à supprimer un Materiel. Nous pouvons voir qu’elle supprime en premier lieu les enregistrements dans la table d’association CatMat concernant ce materiel et seulement ensuite elle supprime le Materiel. L’ordre est important car il est impossible de supprimer un Materiel, et donc son Id, s’il sert de clef étrangère dans une autre table.

## Les Test

### Les tests Unitaire

Les tests unitaires permettent de vérifier le bon fonctionnement d’une partie précise d’un logiciel. Ils permettent également de mieux comprendre comment utiliser une méthode et permettent de passer outre une potentielle obsolescence de la documentation.

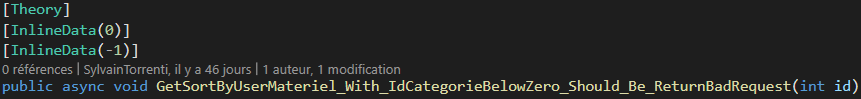


Figure 40 : Test Unitaire nom

Le nom de chaque test doit reprendre le nom de la méthode testée « **GetSortByUserMateriel** », les conditions « **with\_IdCategorieBelowZero** » et enfin les résultats que nous sommes sensé obtenir « **Should\_Be\_ReurnBadRequest** ». Chaque élément doit être séparé par des « **\_** ».

Il existe diffèrent tag à apposer aux tests. Dans la figure 40 il s’agit d’un test avec le tag **[Theory]** ce qui permet de rajouter des paramètres dans la signature du test. Suivi des décorateurs **[InlineData(0)]**et **[InlineData(-1)]** qui indiquent que les valeurs 0 et -1 seront prisent comme paramètres.

Les test unitaires sont séparés en trois parties. Ces trois parties sont :

* **Arrange** : Lors de cette étapes le programmeur doit faire en sorte que l’environnement

obtienne les caractéristiques voulues par le test. Pour cela des Mock (se sont des simulacres qui reproduisent le comportement d’objets réels de manière contrôlée) des éléments externes à la méthode testée sont créés et configurés pour correspondre au résultat souhaité.

* **Act**: Fait appel à la méthode testée.
* **Assert**: Lors de cette étapes, les résultats obtenus sont évalués. Cette ultime étape définit si le test réussit ou échoue.

Pour exécuter mes tests j’ai analysé mon code pour détecter les possibles issues de la fonction

testée et ainsi savoir combien de test je devais pratiquer pour cette fonction. Comme nous le montre la figure 41, je suis donc parti de la méthode que j’avais déjà codée pour savoir quels tests je devais pratiquer. Il est également possible de faire l’inverse avec le TDD (Test Driven Development) qui , lui, part des test pour coder la fonction.

#### Test GetSortByCategorieMaterielAsync

Une image contenant texte, capture d’écran, logiciel

Description générée automatiquement

Figure 41 : Analyse de la fonction pour les tests

Une fois les test dégagés, il faut créer du code pour chaque situation. Pour continuer sur l’exemple de notre méthode de filtrage par les catégories. (il manque un truc dans ta phrase…ou alors dire « nous continuons avec l’exemple…)

#### Test 1

Une image contenant texte, capture d’écran, logiciel, Logiciel multimédia

Description générée automatiquement

Figure 42 : 1er test

Lors de ce premier test (figure 42) j’utilise le décorateur **[Theory]** car le but de ce test est de vérifier le comportement de la fonction si l’Id de la categorie donnée est 0 ou -1.

Pour la partie **Arrange** il faut créer un **Mock** du service présent dans la BLL car nous testons la méthode et non ce qui est externe à celle-ci.

Pour la partie **Act** j’exécute la fonction. Les paramètres sont définis par **[InlineData(0)]** et **[InlineData(-1)]** ce qui signifie que deux tests seront faits. Le premier avec l’Id qui sera égal à 0 et le suivant l’Id sera égal à -1.

La dernière partie **Assert** permet de vérifier l’exactitude des résultats. Dans notre cas le résultat doit être une **BadRequest**.

#### Test 2

Une image contenant texte, capture d’écran, logiciel

Description générée automatiquement

Figure 43 : 2eme test

Pour ce deuxième test (figure 43), le décorateur **[Fact]** est utilisé.

Pour la partie **Arrange** un **Mock** et créé et est paramétré pour s’accorder avec les demandes du test. Il est donc configuré pour quelle que soit l’Id fournie rien ne sera retourné.

Pour la partie **Act** la fonction est exécutée.

La partie **Assert** vérifie l’exactitude du résultat.

#### Test 3

Une image contenant texte, capture d’écran, logiciel

Description générée automatiquement

Figure 44 : 3eme test

Ce troisième test (figure 44)permet de vérifier le bon fonctionnement de la fonction.

Dans la partie **Arrange** il y a toujours la création d’un **Mock** mais également de deux **Materiel** car nous devons paramétrer notre **Mock** pour qu’il retourne ces deux **Materiel**. Il faut également le paramétrer pour que n’importe quel Id soit donné.

La partie **Act** exécute la fonction.

Enfin la partie **Assert** vérifie l’exactitude des résultats.

#### Résultat des test

Une image contenant texte, capture d’écran, logiciel, Système d’exploitation

Description générée automatiquement

Figure 45 : Résultat des test

Comme nous le montre la figure 45, toutes les fonctions se trouvant dans le controller sont testées. Tout les tests sont valides. Nous pouvons également remarquer que chaque fonction n’a pas le même nombre de test.

### Test d’Intégration

*(futur capture d’écran pas encore faites car test pas encore effectuer pour donner à corriger le plus vite possible)*

Les tests d’intégrations permettent de vérifier le bon fonctionnement entre les différentes couches de l’application. Il faut fixer l’environnement de test pour simuler nos appels de requêtes.

Il faut également créer une base de données de test pour éviter de remplir notre base de données de production avec les différents tests. Pour cela j’ai créé un script qui copie la base de données utilisé par l’application. J’ai également récupéré les données présentes dans la base de données grâce au SGBDR DBeaver qui m’a permis de directement récupérer les données avec les commandes SQL nécessaires pour les insérer dans la base de données de test. (figure 47 et 48)

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

Description générée automatiquement

Figure 46 : Script Création de table

Une image contenant texte, capture d’écran, Police

Description générée automatiquement

Figure 47 : Script jeu de données

L’ordre d’exécution des différentes requêtes est important car il faut prendre en compte les contrainte de la Base de Données. Ce script sera lancé lors de l’exécution de chaque test.

### Tests d’Acceptation

Pour effectuer les différents d’acceptations tests j’ai fait appel à des personnes extérieures que je souhaite grandement remercier, **BRUNDU Roseline et BRUNDU Nicolas.**

Ils m’ont permis de mettre en évidence des problématiques que je n’avais pas envisagées et m’ont aussi prodigué des conseils au niveau du design et de l’ergonomie.

Pour que les tests soient le plus fidèles à la réalité, je ne leur ai fait qu’un rapide résumé de la plateforme sans donner d’indications sur la marche à suivre. Ils ont ainsi utilisé leurs comptes crées lors de la première étapes du projet et je leur ai également fournit un compte administrateur pour qu’il puisse tester toutes les fonctionnalités. Grâce à cela, j’ai pu mettre en lumière des dysfonctionnements de quelques fonctionnalités sous certaines conditions qui m’avaient échappées car je connaissais le code.

# Future évolution

Ce projet pourra, dans le futur, être relié au premier projet réaliser lors du premier EPCF. Par la suite il y a aussi la possibilité de rajouter une gestion de contrat de maintenance et une gestion des entreprises externes qui gèreront ces contrats de maintenances. Pour cela le choix de travailler avec des onglets facilitera l’implémentation de futures fonctionnalités.

# Conclusion

En conclusion, ce projet m’a permis de connaitre et d’apprendre à utiliser la conception en multicouche et comment bien séparer les différentes problématiques.

Il est également possible que des modifications futures y soient apportées. Ces modifications peuvent porter sur la technique mais aussi sur le design.