

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



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
Liste des compétences couvertes par le projet	4
Résumé	5
Introduction	6
Prérequis du projet	7
Mise en place du projet	8
Développement du projet.....	11
Modification de la Base De Données	11
La lisibilité et la documentation	13
Côté Client	14
Windows Form	14
Les onglets.....	17
Connexion	17
Consulter	18
Création	19
Modification	21
Suppression	23
Côté Serveur	24
La structure.....	24
L'API	25
Les controller.....	25
AccountController.....	27
GestionMaterielServiceController.....	28
BLL.....	29
SecurityService.....	30
GestionMaterielService	30
DAL	31
Le pattern Repositories	32
Le repository Matériel	33
GetAll	34
GetSort.....	35
Update	36
Add	37
Delete.....	38
Les Test.....	38
Les tests Unitaire	38

Test GetSortByCategorieMaterielAsync	40
Test 1	41
Test 2	41
Test 3	42
Résultat des test	43
Test d'Intégration	44
Tests d'Acceptation	49
Future évolution	50
Conclusion	50

Liste des compétences couvertes par le projet

- Maquetter une application.
- Analyser les besoins.
- Concevoir une Base De Données.
- Développer des composants d'accès aux données.
- Définir l'architecture logicielle d'une application.
- Concevoir une application sécurisée organisée en couches.
- Contribuer à la mise en production dans une démarche DevOps

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 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'enchaînement 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 reconverter 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 avons 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.

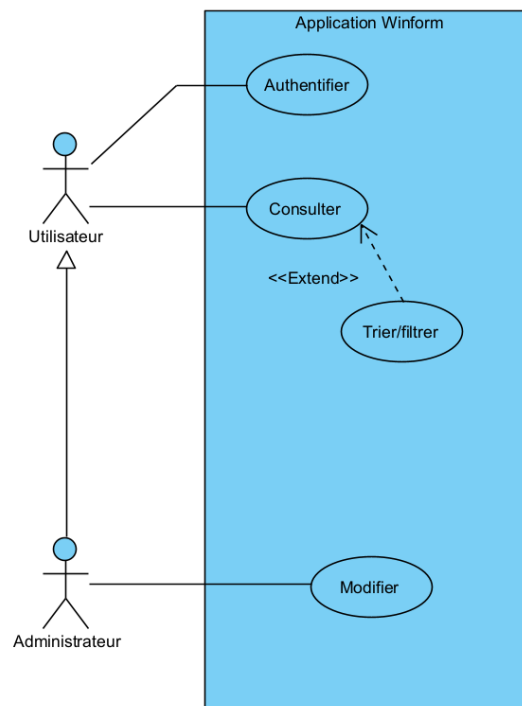


Figure 1 : Use Case Diagram

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

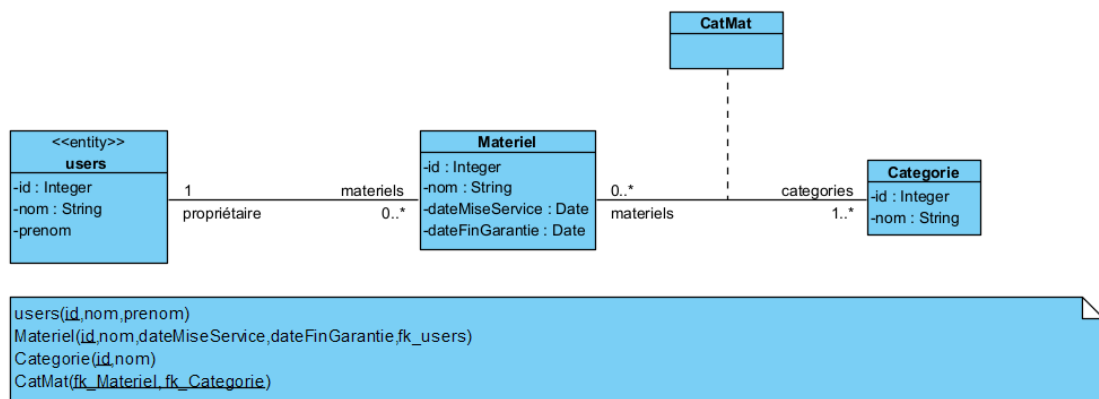


Figure 2 : Class Diagram

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

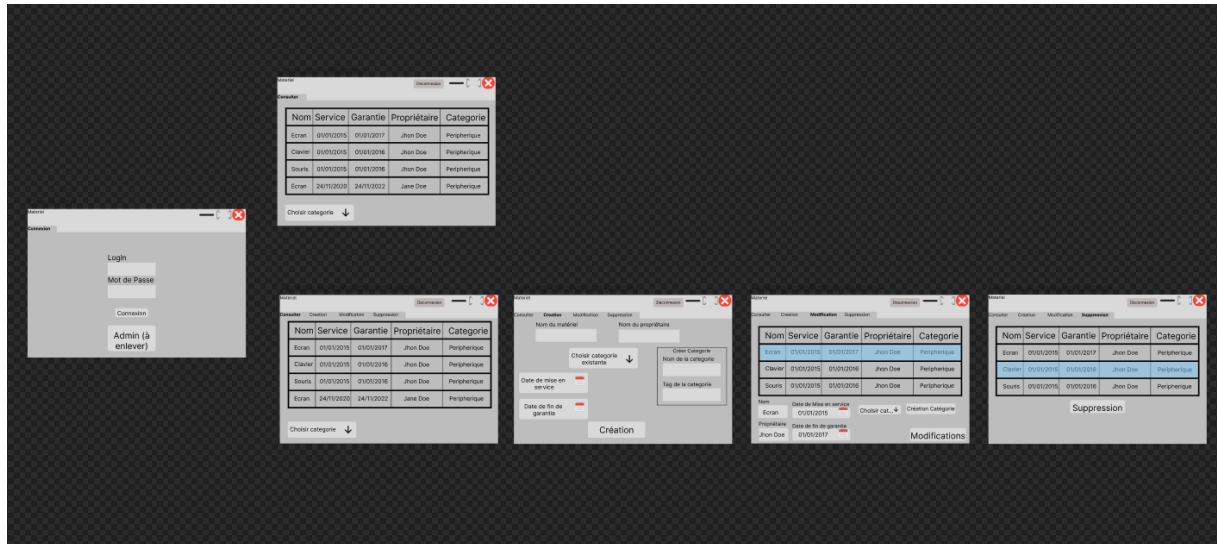


Figure 3: Illustration des onglets sous FIGMA

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).

```
create table Materiel(  
  Id int ,  
  Nom VARCHAR(90),  
  DateMiseEnService DATE,  
  DateFinGarantie DATE,  
  Propriétaire int,  
  constraint Message_PK primary key(Id)  
) ENGINE = InnoDB;
```

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).

```
ALTER TABLE Materiel  
ADD constraint FK_Materiel_users ADD FOREIGN KEY (Proprietaire) REFERENCES users(id);
```

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.

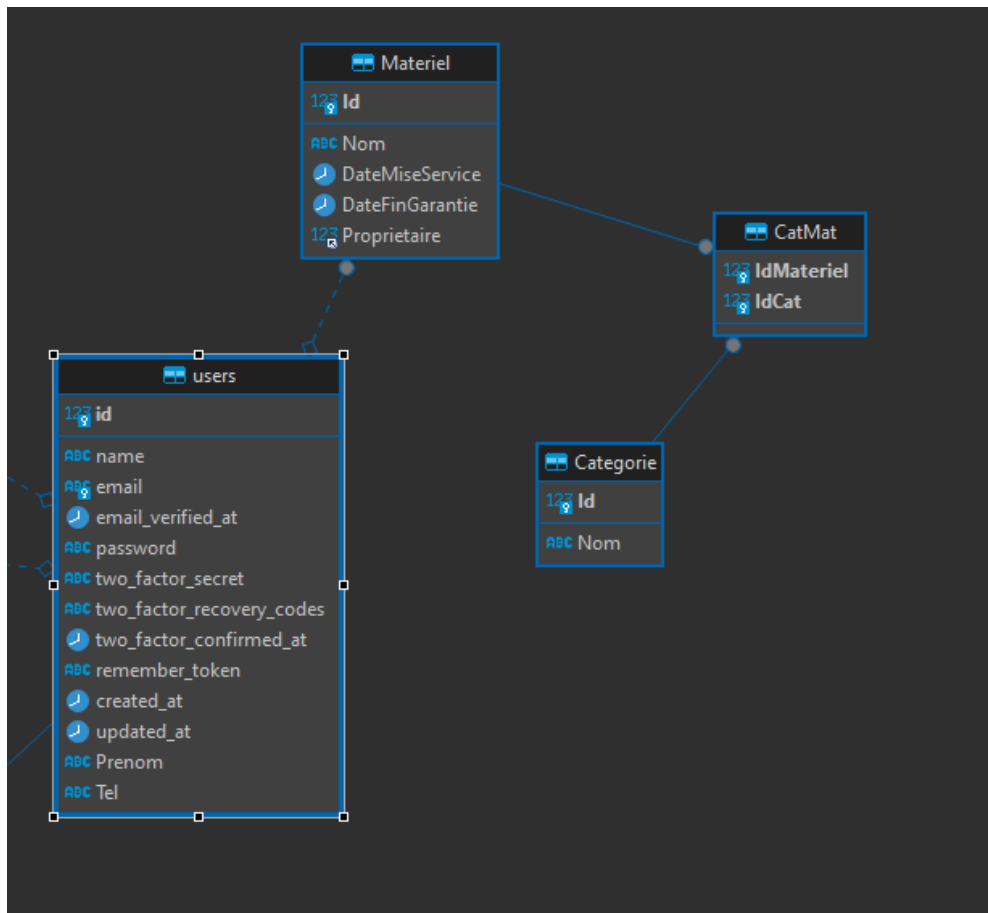


Figure 6 : Diagramme BDD

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 une 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.

La lisibilité et la documentation

Tout au long du projet, j'ai effectué différentes opérations pour faciliter la maintenabilité et la lecture de mon code.

En ce qui concerne la lecture du code et la facilité pour trouver une méthode précise, j'ai séparé mon code en Region. Cela n'influe aucunement sur l'exécution du code mais permet une visibilité bien meilleure car nous pouvons personnaliser les différentes Region comme nous le montre la figure 7. J'ai décidé de séparer mon code en fonction du CRUD et à l'intérieur de chaque Region j'ai décidé d'encore effectuer une séparation en fonctions de l'objet traité.

```
#region Get
Materiel

Categorie

Utilisateur
#endregion

#region Post
Materiel

Categorie
#endregion

Put

Delete
```

Figure 7 Region

En plus des Region, j'ai également utilisé les Summary. Cette fonction propre au C# permet d'entrer les informations et le compilateur les copie directement dans le document XML de sortie. Dès que le Summary est créé pour un élément, les commentaires sont visibles dans tout le projet.

En plus de permettre une meilleure vision à l'intérieur du projet, cette pratique est utile pour une meilleure vision avec Swagger qui est un langage de description d'interface qui permet de décrire des API à l'aide de JSON. La figure 8 nous montre que ces commentaires sont directement visibles.

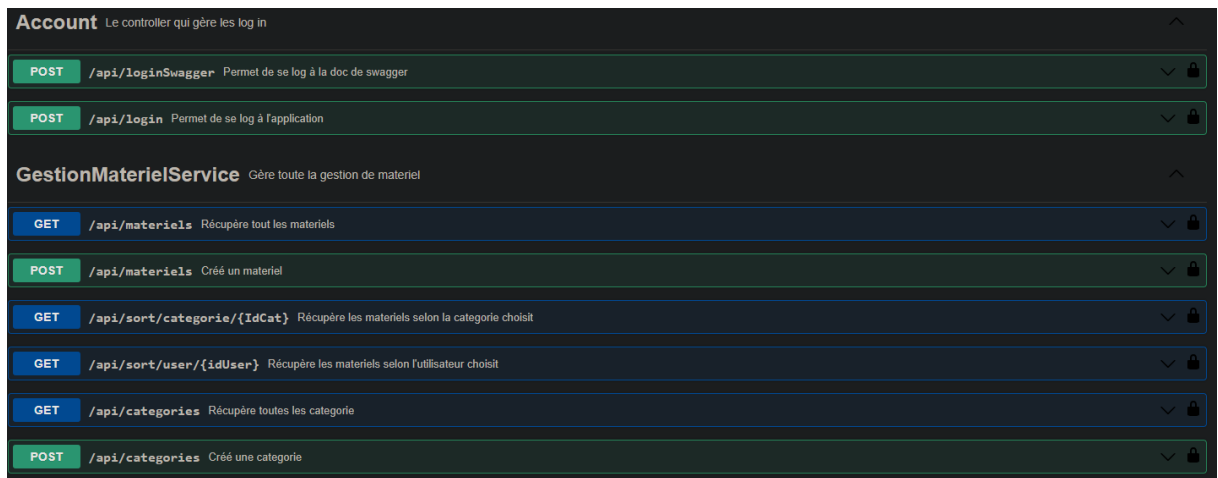


Figure 8 Swagger

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 (figure9).

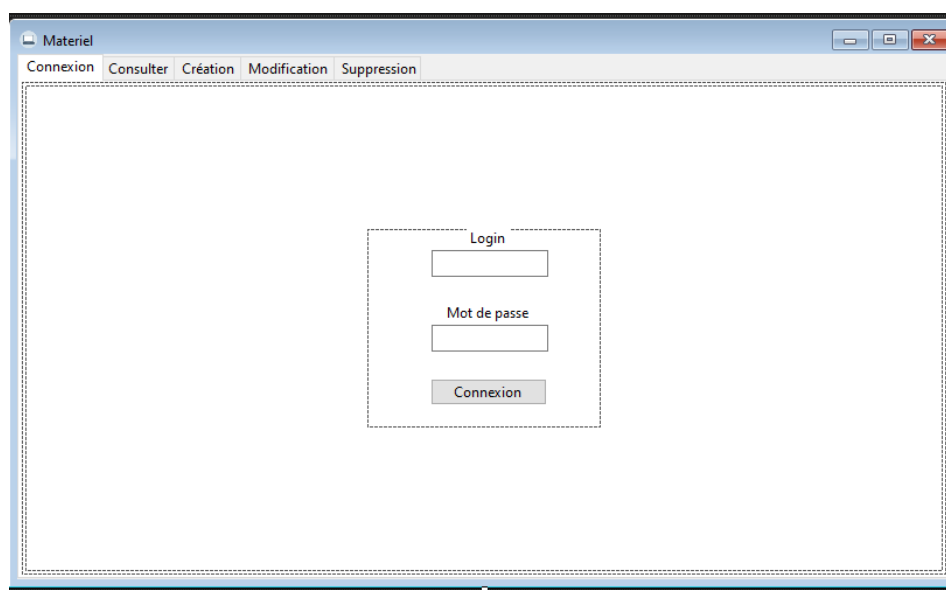


Figure 9 : Fenêtre avec onglets

La figure 9 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.

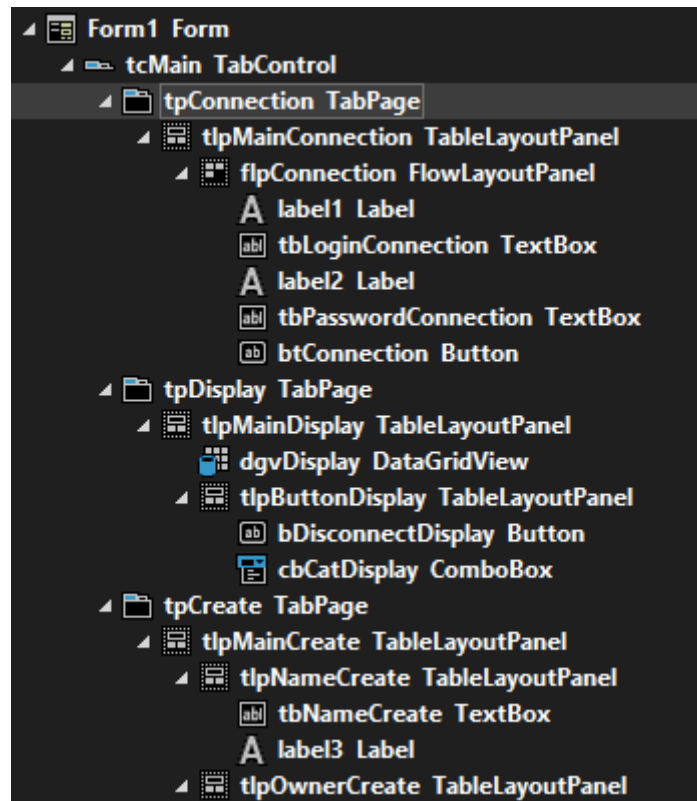


Figure 10 : Structure du document

La figure 10 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.

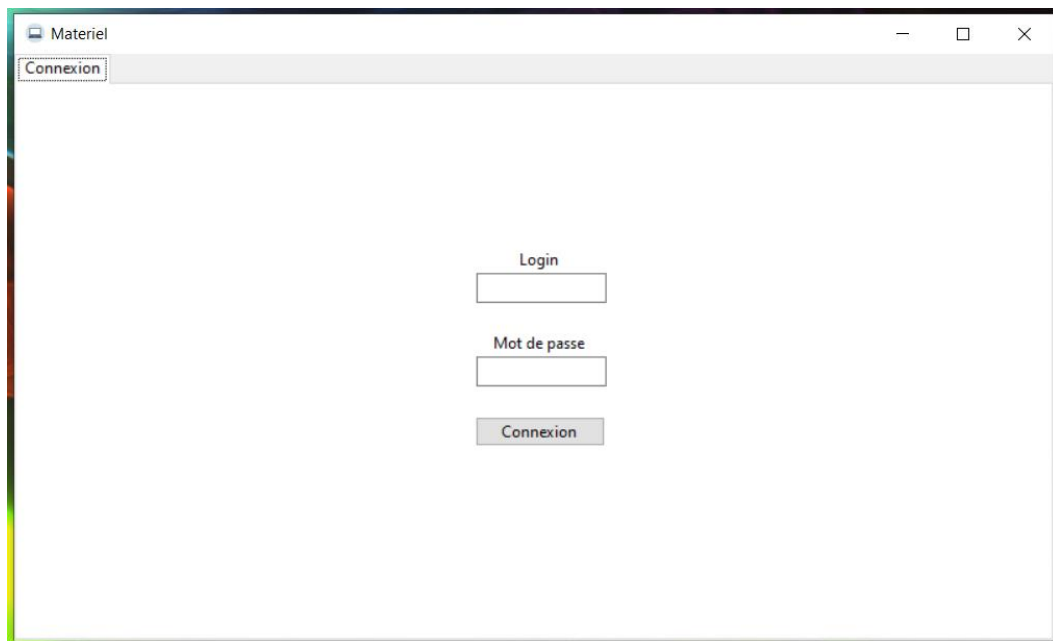


Figure 11 : Fenêtre avant authentification

La figure 11 représente l’affichage lorsque l’application Windows Form est lancée. Nous pouvons ainsi voir que seul l’onglet Connexion est présent.

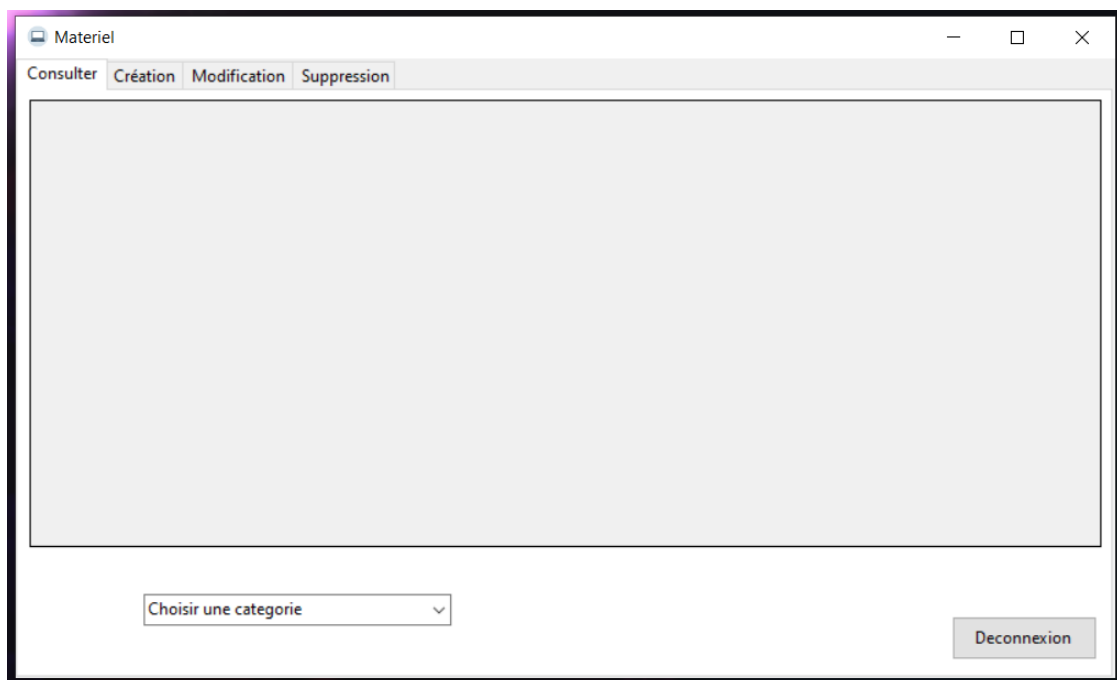


Figure 12 : Fenêtre après authentification

La figure 12 représente l'authentification d'un administrateur. Nous pouvons voir qu'il a accès à tous les onglets permettant de faire des opérations sur le Matériel 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 13).

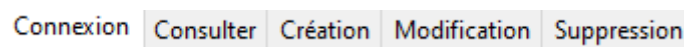


Figure 13 : Onglets

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 14 et aussi ce que l'utilisateur peut voir grâce à la figure 15.

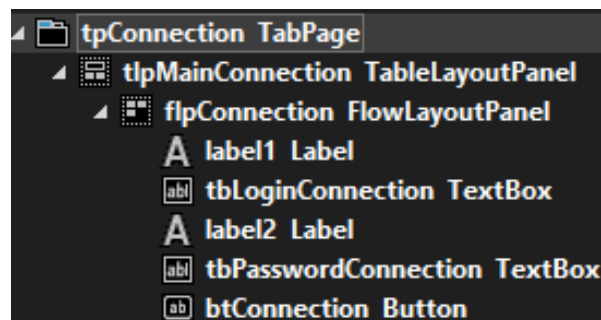


Figure 14 : Structure de l'onglet Connexion

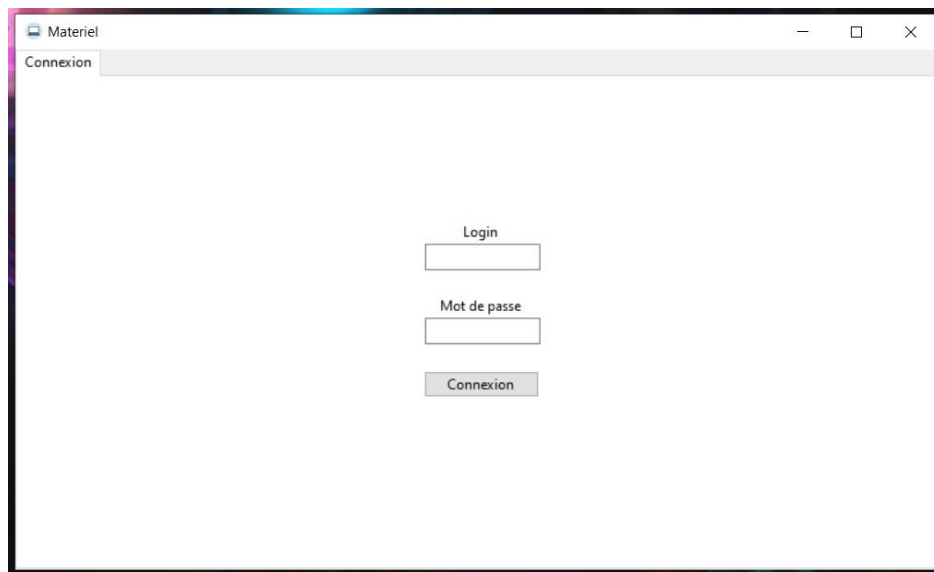


Figure 15 : onglet Connexion

Consulter

L'onglet consulter est disponible par tous les utilisateurs quel que soit leur rôle. Sa structure est simple comme le montre la figure16. L'affichage est lui aussi très simple.

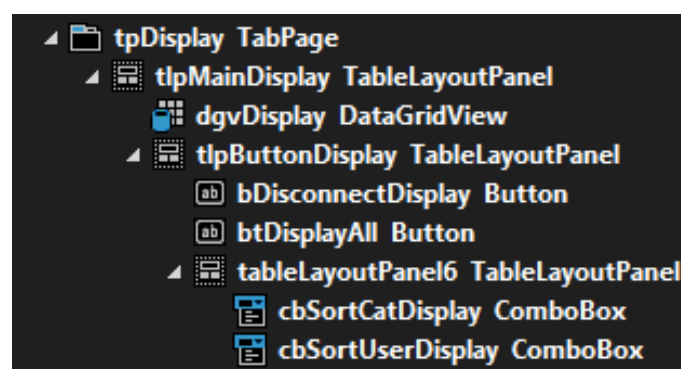
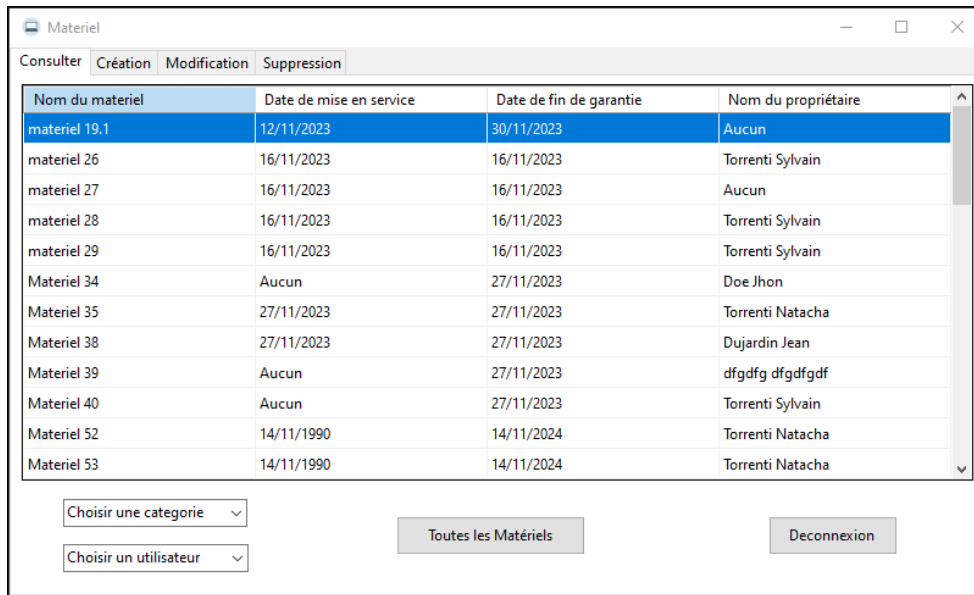


Figure 16 : Structure de l'onglet Consulter



Nom du materiel	Date de mise en service	Date de fin de garantie	Nom du propriétaire
materiel 19.1	12/11/2023	30/11/2023	Aucun
materiel 26	16/11/2023	16/11/2023	Torrenti Sylvain
materiel 27	16/11/2023	16/11/2023	Aucun
materiel 28	16/11/2023	16/11/2023	Torrenti Sylvain
materiel 29	16/11/2023	16/11/2023	Torrenti Sylvain
Materiel 34	Aucun	27/11/2023	Doe Jhon
Materiel 35	27/11/2023	27/11/2023	Torrenti Natacha
Materiel 38	27/11/2023	27/11/2023	Dujardin Jean
Materiel 39	Aucun	27/11/2023	dfgdfg dfgdfgdf
Materiel 40	Aucun	27/11/2023	Torrenti Sylvain
Materiel 52	14/11/1990	14/11/2024	Torrenti Natacha
Materiel 53	14/11/1990	14/11/2024	Torrenti Natacha

Figure 17 : 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 17).

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 18).

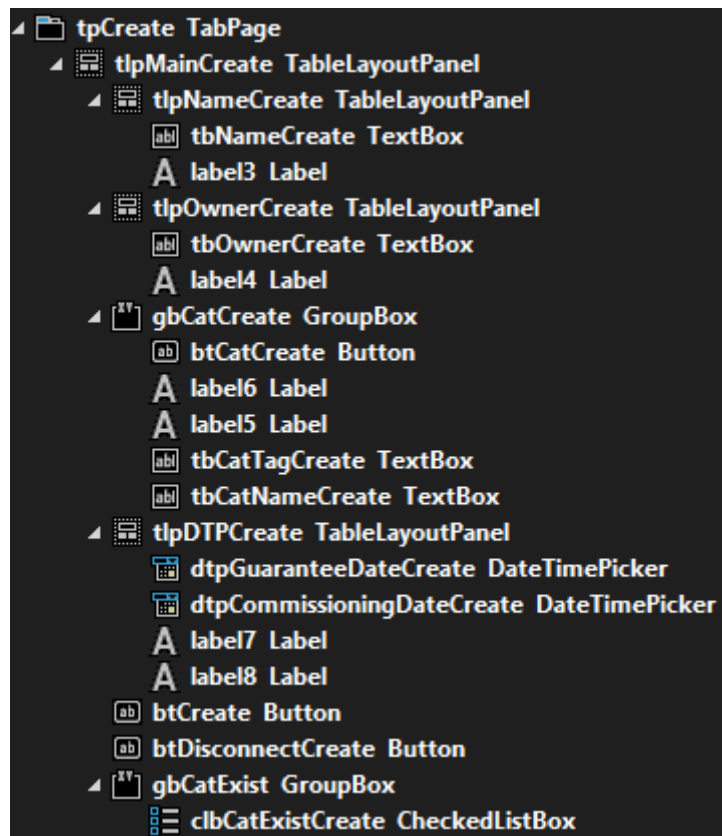
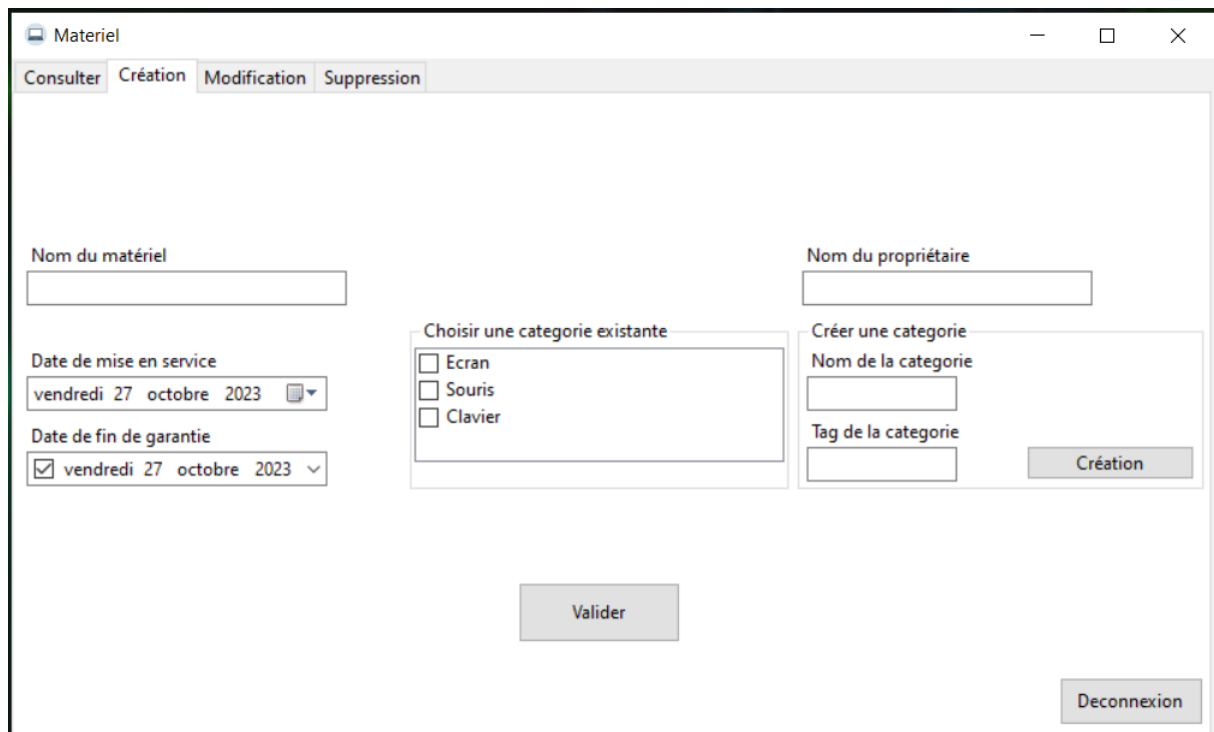


Figure 18 : Structure de l'onglet Creation



Materiel [Consulter] [Création] [Modification] [Suppression]

Nom du matériel:

Nom du propriétaire:

Date de mise en service: vendredi 27 octobre 2023

Date de fin de garantie: ☒ vendredi 27 octobre 2023

Choisir une categorie existante:

- ☐ Ecran
- ☐ Souris
- ☐ Clavier

Créer une categorie:

Nom de la categorie:

Tag de la categorie:

Figure 19 : onglet Creation

Cet onglet permet à l'administrateur de renseigner toutes les informations nécessaires à l'enregistrement d'un nouveau matériel. Si la/les catégories n'existent pas il est possible d'en créer de nouvelles (figure 19).

Modification

L'onglet modification est également disponible seulement aux utilisateurs qui ont le rôle administrateur. Il permet de modifier un matériel 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 20).

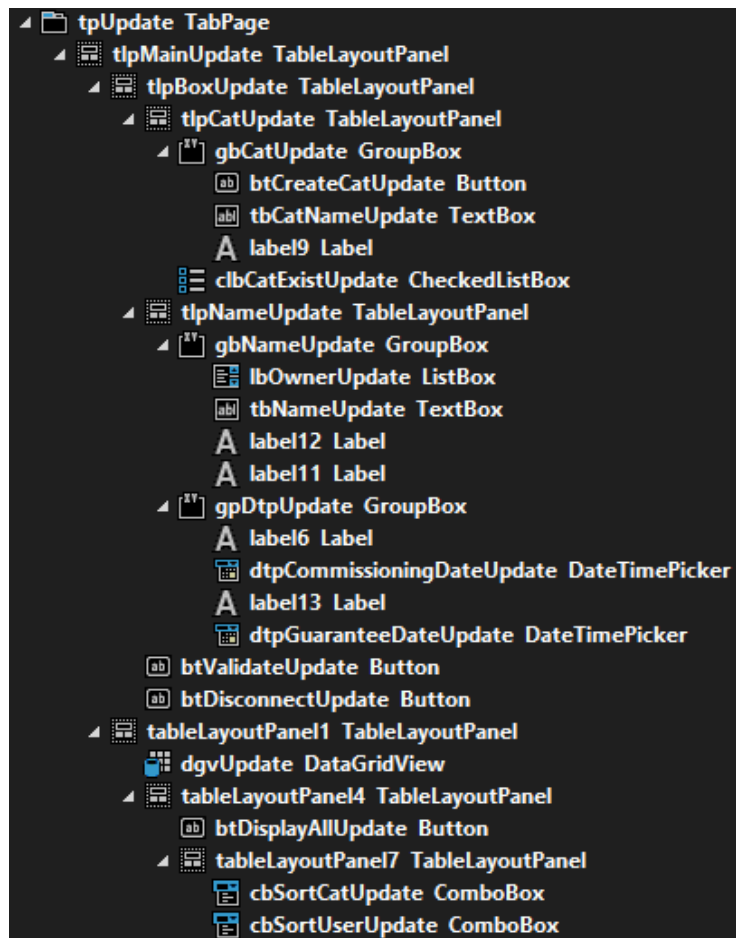
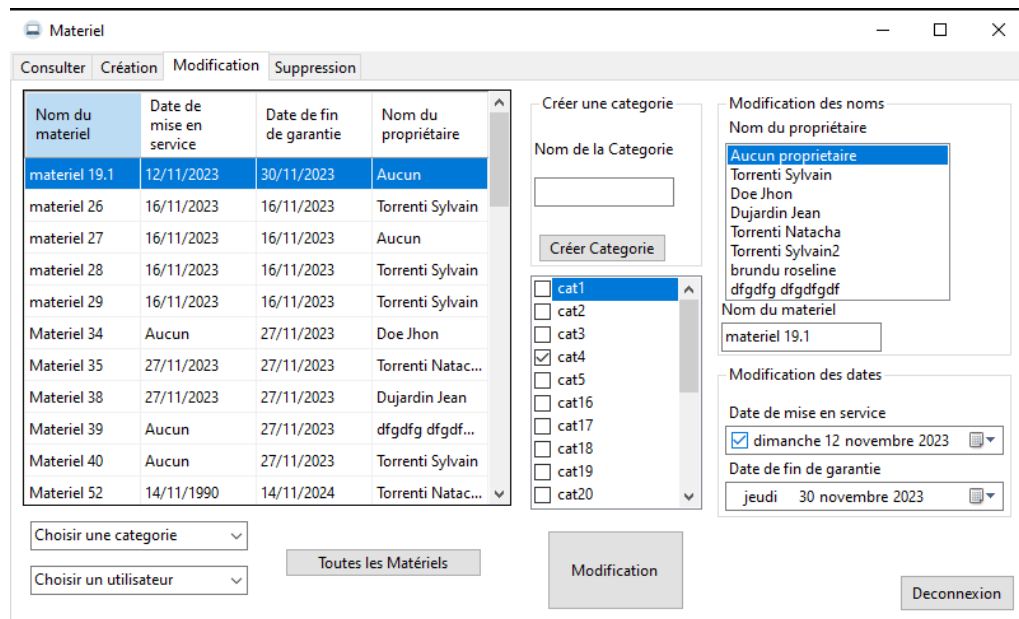


Figure 20 : Structure de l'onglet Modification



Materiel

Consulter Création **Modification** Suppression

Nom du materiel	Date de mise en service	Date de fin de garantie	Nom du propriétaire
materiel 19.1	12/11/2023	30/11/2023	Aucun
materiel 26	16/11/2023	16/11/2023	Torrenti Sylvain
materiel 27	16/11/2023	16/11/2023	Aucun
materiel 28	16/11/2023	16/11/2023	Torrenti Sylvain
materiel 29	16/11/2023	16/11/2023	Torrenti Sylvain
Materiel 34	Aucun	27/11/2023	Doe Jhon
Materiel 35	27/11/2023	27/11/2023	Torrenti Natac...
Materiel 38	27/11/2023	27/11/2023	Dujardin Jean
Materiel 39	Aucun	27/11/2023	dfgdfg dfgdf...
Materiel 40	Aucun	27/11/2023	Torrenti Sylvain
Materiel 52	14/11/1990	14/11/2024	Torrenti Natac...

Choisir une categorie

Choisir un utilisateur

Toutes les Matériels

Modification

Deconnexion

Créer une categorie

Nom de la Categorie

Créer Categorie

☐ cat1
☐ cat2
☐ cat3
☒ cat4
☐ cat5
☐ cat16
☐ cat17
☐ cat18
☐ cat19
☐ cat20

Modification des noms

Nom du propriétaire

Aucun propriétaire
Torrenti Sylvain
Doe Jhon
Dujardin Jean
Torrenti Natacha
Torrenti Sylvain2
brundu roseline
dfgdfg dfgdfgdf

Nom du materiel

materiel 19.1

Modification des dates

Date de mise en service

☒ dimanche 12 novembre 2023

Date de fin de garantie

jeudi 30 novembre 2023

Figure 21 : onglet Modification

Comme nous pouvons le voir grâce à la figure 21, 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 22).

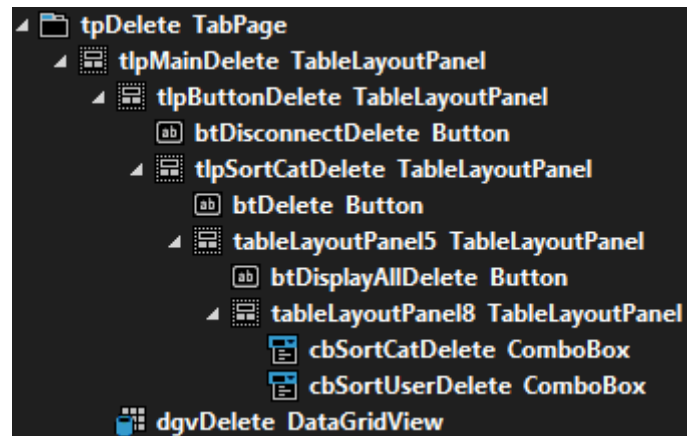


Figure 22 : Structure de l'onglet Suppression

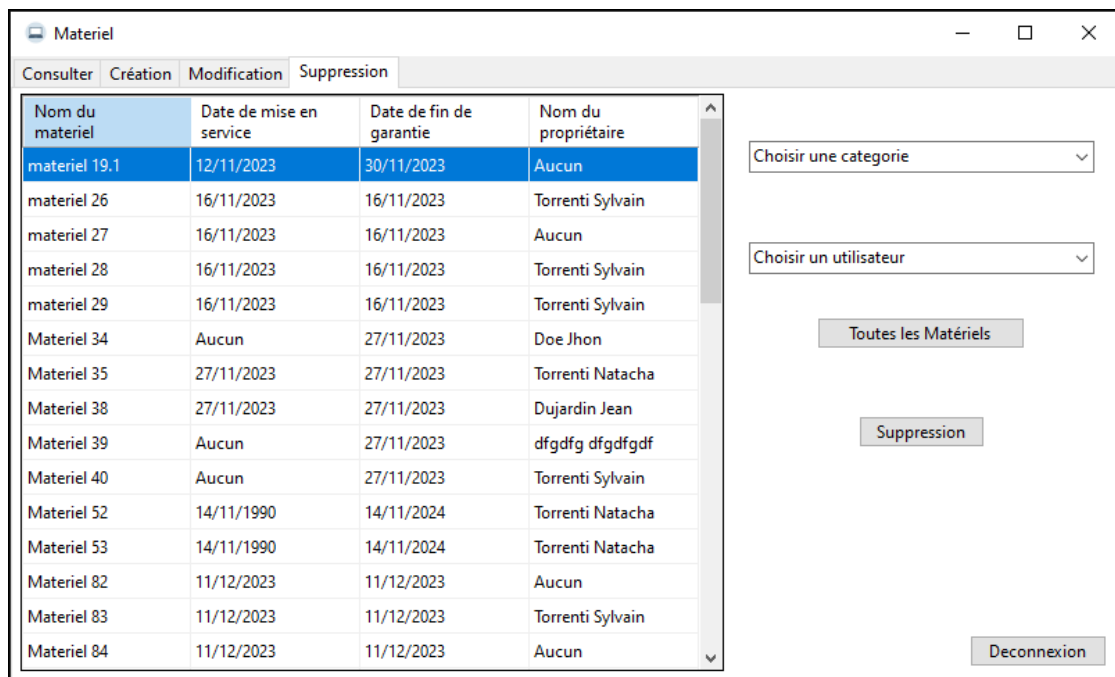


Figure 23 : Liste Tickets (Utilisateur). L'interface affiche une liste de matériels avec les colonnes : Nom du materiel, Date de mise en service, Date de fin de garantie, et Nom du propriétaire. À droite, il y a des filtres pour 'Choisir une categorie' et 'Choisir un utilisateur', ainsi que des boutons 'Toutes les Matériels', 'Suppression', et 'Deconnexion'.

Nom du materiel	Date de mise en service	Date de fin de garantie	Nom du propriétaire
materiel 19.1	12/11/2023	30/11/2023	Aucun
materiel 26	16/11/2023	16/11/2023	Torrenti Sylvain
materiel 27	16/11/2023	16/11/2023	Aucun
materiel 28	16/11/2023	16/11/2023	Torrenti Sylvain
materiel 29	16/11/2023	16/11/2023	Torrenti Sylvain
Materiel 34	Aucun	27/11/2023	Doe Jhon
Materiel 35	27/11/2023	27/11/2023	Torrenti Natacha
Materiel 38	27/11/2023	27/11/2023	Dujardin Jean
Materiel 39	Aucun	27/11/2023	dfgdfg dfgdfgdf
Materiel 40	Aucun	27/11/2023	Torrenti Sylvain
Materiel 52	14/11/1990	14/11/2024	Torrenti Natacha
Materiel 53	14/11/1990	14/11/2024	Torrenti Natacha
Materiel 82	11/12/2023	11/12/2023	Aucun
Materiel 83	11/12/2023	11/12/2023	Torrenti Sylvain
Materiel 84	11/12/2023	11/12/2023	Aucun

Figure 23 : Liste Tickets (Utilisateur)

Dans la figure 23 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.

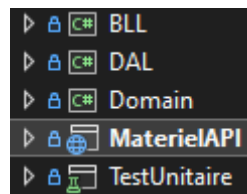


Figure 24 : Structure côté serveur

Comme nous le montre la figure 24, 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

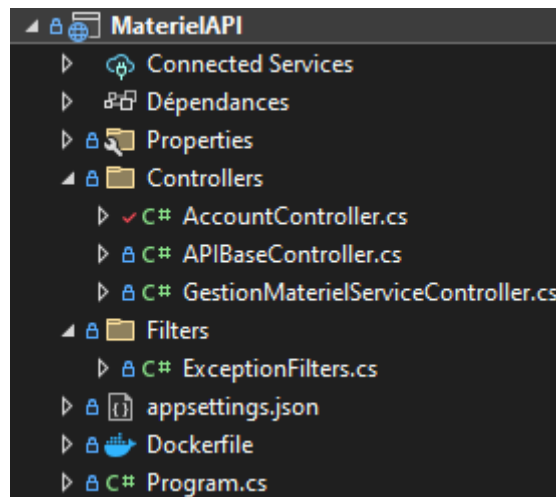


Figure 25 : Structure de l'API

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 obligé de reproduire à chaque controller créés.

```
1 using Microsoft.AspNetCore.Authorization;
2 using Microsoft.AspNetCore.Mvc;
3
4 namespace MaterielAPI.Controllers
5 {
6     /// <summary>
7     /// le controller qui sert de base à tout les controller que nous ferons par la suite
8     /// </summary>
9     [ApiController]
10    [Route("api/")]
11    [Authorize(Roles = "Utilisateur")]
12    public abstract class APIBaseController : ControllerBase
13    {
14    }
15 }
```

Figure 26 : BaseController

Comme nous le montre la figure 26, 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

```
[HttpPost("login")]
[AllowAnonymous]
0 références | SylvainTorrenti, Il y a 1 jour | 1 auteur, 4 modifications
public async Task<IActionResult> Login([FromBody] LoginRequest loginRequest)
{
    var result = await _securityService.SignIn(loginRequest.eMail, loginRequest.password);

    if (result is null) return BadRequest();
    else return Ok(new LoginResponse { AccessToken = result });
}
```

Figure 27 : AccountController

Comme nous le montre la figure 27, 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.

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 28).

```
#region Post
#region Materiel
/// <summary>
/// Créé un materiel
/// </summary>
/// <param name="createMaterielDTORequest"></param>
/// <returns></returns>
[Authorize(Roles = "Administrateur")]
[HttpPost("materiels")]
[ProducesResponseType(StatusCodes.Status201Created)]
[ProducesResponseType(StatusCodes.Status400BadRequest)]
0 références | SylvainTorrenti, il y a 5 jours | 1 auteur, 6 modifications
public async Task<IActionResult> CreateMaterielAsync([FromBody] CreateMaterielDTORequest createMaterielDTORequest)
{
    var materiel = new Materiel()
    {
        NomMat = createMaterielDTORequest.NomMat,
        DateMiseService = createMaterielDTORequest?.DateMiseService,
        DateFinGarantie = createMaterielDTORequest.DateFinGarantie,
        IdProprietaire = createMaterielDTORequest.IdProprietaire,
        Categories = createMaterielDTORequest.Categories
    };

    var result = await _gestionMaterielService.CreateMaterielAsync(materiel);

    var reponse = new CreateMaterielDTOResponse
    {
        IdMateriel = result.IdMateriel,
        NomMat = result.NomMat,
        DateMiseService = result.DateMiseService,
        DateFinGarantie = result.DateFinGarantie,
        IdProprietaire = result.IdProprietaire,
        Categories = result.Categories
    };
}
```

Figure 28 : 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

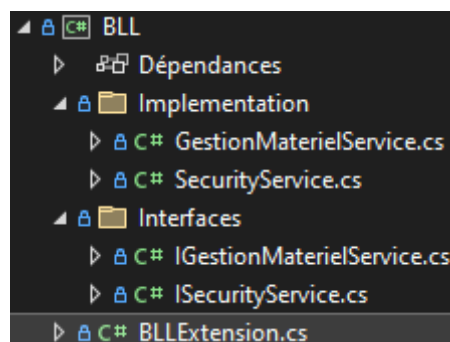


Figure 29 : structure de la BLL

La BLL sert d'intermédiaire entre l'API et la DAL (Data Access Layer) et compile les règles métier de l'application. Comme nous le montre la figure 29, la présence d'interface pour chaque service permet de faciliter de futures modifications.

```
internal class GestionMaterielService : IGestionMaterielService
{
    private readonly IUOW _dbContext;

    2 références | SylvainTorrenti, il y a 61 jours | 1 auteur, 1 modification
    public GestionMaterielService(IUOW dbContext)
    {
        _dbContext = dbContext;
    }
}
```

Figure 30 : 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

```
public async Task<string> SignIn(string eMail, string password)
{
    var utilisateur = await _dbContext.Users.GetProfileAsync(eMail);
    if (!(eMail == utilisateur.email) || !(BC.Verify(password, utilisateur.password))) return null;

    return await Task.FromResult(GenerateJwtToken(eMail, new List<string>() { $"{utilisateur.role}", "Utilisateur" }));
}
```

Figure 31 : Login

Dans cet extrait (figure 31), 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

```
public async Task<Materiel> CreateMaterielAsync(Materiel materiel)
{
    _dbContext.BeginTransaction();
    var resultMateriel = await _dbContext.Materiels.AddAsync(materiel);
    var mat = await _dbContext.Materiels.GetByIdAsync(resultMateriel.IdMateriel);
    foreach (Categorie categorie in materiel.Categories)
    {
        await _dbContext.Materiels.AddMatCatAsso(resultMateriel.IdMateriel, categorie.IdCategorie);
    }
    _dbContext.Commit();
    return mat;
}
```

Figure 32 : Create Materiel

La figure 32 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 matériel dans la BDD grâce à la ligne « `_dbContext.Materiels.AddAsync(materiel);` » qui indique que nous utilisons la méthode AddAsync présente dans le repository Matériel qui est lui-même présent dans la DAL en utilisant l'Unit Of Work créé en amont. Nous récupérons ensuite l'Id du matériel nouvellement créé pour l'associer à la liste des catégories fournies. L'ordre des opérations est important car si le matériel n'est pas créé nous ne pouvons pas encore créer les associations avec les catégories.

DAL

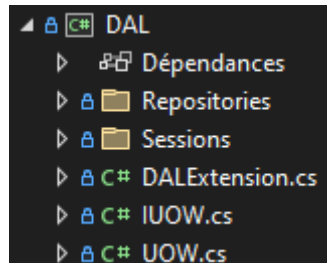


Figure 33 : Structure de la DAL

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

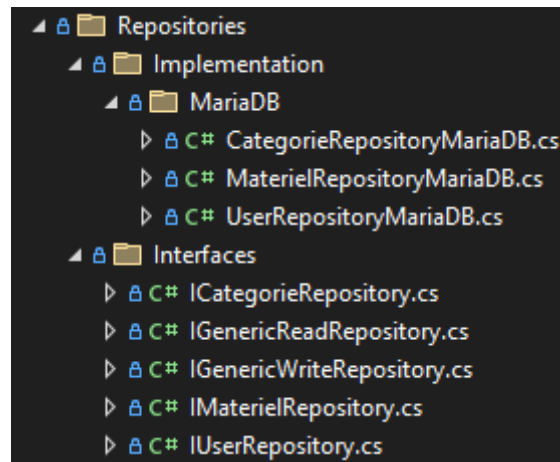


Figure 34 : Structure du Pattern Repositories

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.

```
/// <summary> Interface de lecture générique
3 références | SylvainTorrenti, il y a 61 jours | 1 auteur, 1 modification
public interface IGenericReadRepository<U, T> where T : IEntity
{
    /// <summary> Récupère la liste des T
    8 références | SylvainTorrenti, il y a 61 jours | 1 auteur, 1 modification
    public Task<IEnumerable<T>> GetAllAsync();

    /// <summary> Récupère la liste des T via son Id
    9 références | SylvainTorrenti, il y a 61 jours | 1 auteur, 1 modification
    public Task<T> GetByIdAsync(U id);
}
```

Figure 35 : Repositories interface de lecture générique

Comme nous le montre la figure 35, 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

```
public interface IMaterielRepository : IGenericReadRepository<int, Materiel>, IGenericWriteRepository<int, Materiel>
{
    /// <summary> Récupère la liste des materiels selon la categorie choisit
    2 références | SylvainTorrenti, il y a 46 jours | 1 auteur, 1 modification
    public Task<IEnumerable<Materiel>> GetSortMaterielByCategorieAsync(int idCat);

    /// <summary> Récupère la liste des materiels selon l'utilisateur choisit
    2 références | SylvainTorrenti, il y a 46 jours | 1 auteur, 1 modification
    public Task<IEnumerable<Materiel>> GetSortMaterielByUserAsync(int idUser);

    /// <summary> Récupère la liste des materiels qui n'ont pas de propriétaire
    2 références | SylvainTorrenti, il y a 4 jours | 1 auteur, 1 modification
    public Task<IEnumerable<Materiel>> GetSortMaterielNoOwnerAsync();

    /// <summary> Ajoute l'association du materiel et des categories
    2 références | SylvainTorrenti, il y a 46 jours | 1 auteur, 1 modification
    public Task<int> AddMatCatAsso(int idMat, int idCat);
}
```

Figure 36 : Repositories interface Materiel

Nous pouvons voir dans la figure 36 l'interface qui concerne les Materiel. Celle-ci hérite des Interface génériques et ajoute également des méthodes propres seulement aux Materiel.

GetAll

```
internal class MaterielRepositoryMariaDB : IMaterielRepository
{
    private readonly IDBSession db;

    1 référence | SylvainTorrenti, il y a 61 jours | 1 auteur, 1 modification
    public MaterielRepositoryMariaDB(IDBSession dBSession) {...}

    6 références | SylvainTorrenti, il y a 61 jours | 1 auteur, 1 modification
    public async Task<IEnumerable<Materiel>> GetAllAsync()
    {
        var SQL = @"SELECT m.IdMateriel,m.NomMat , m.DateMiseService, m.DateFinGarantie, c.*, u.id, u.nameOwner, u.PrenomOwner
                    FROM Materiel as m
                    LEFT JOIN CatMat as cm on m.IdMateriel = cm.IdMateriel
                    LEFT JOIN Categorie as c on cm.IdCat = c.IdCategorie
                    LEFT JOIN users as u on m.IdProprietaire = u.id
                    order by m.IdMateriel asc;";

        var materiels = await db.Connection.QueryAsync<Materiel, Categorie, users, Materiel>(SQL, (materiel, categorie, user) =>
        {
            materiel.Categories.Add(categorie);
            materiel.ProprietaireObj = user;
            return materiel;
        }, splitOn: "IdCategorie, id", transaction: db.Transaction);
    }
}
```

Figure 37 : Repositories Materiel GetAll

La figure 37 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 matériels. J'utilise cette méthode pour afficher l'ensemble des matériels 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 Matériels, 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 connaît les délimitations des entités dans les données récupérées.

GetSort

```
public async Task<IEnumerable<Matériel>> GetSortMatérielByCategorieAsync(int idCat)[...]

2 références | SylvainTorrenti, il y a 46 jours | 1 auteur, 1 modification
public async Task<IEnumerable<Matériel>> GetSortMatérielByUserAsync(int idUser)
{
    var SQL = @"SELECT m.IdMatériel, m.NomMat, m.DateMiseService, m.DateFinGarantie,c.* , u.id, u.nameOwner, u.PrenomOwner
                FROM Matériel as m
                LEFT JOIN CatMat as cm on m.IdMatériel = cm.IdMatériel
                LEFT JOIN Categorie as c on cm.IdCat = c.IdCategorie
                LEFT JOIN users as u on m.IdProprietaire = u.id
                WHERE u.id = @idUser
                order by m.IdMatériel asc;";
    var materiels = await db.Connection.QueryAsync<Matériel, Categorie, users, Matériel>(SQL, (matériel, categorie, user) =>
    {
        matériel.Categories.Add(categorie);
        matériel.ProprietaireObj = user;
        return matériel;
    }, splitOn: "IdCategorie, id", param: new { idUser }, transaction: db.Transaction);

    var result = materiels.GroupBy(m => m.IdMatériel).Select(groupe =>
    {
        var firstMatériel = groupe.First();
        firstMatériel.Categories = groupe.Select(m => m.Categories.Single()).ToList();
        return firstMatériel;
    });

    return result;
}

2 références | SylvainTorrenti, il y a 4 jours | 1 auteur, 1 modification
public async Task<IEnumerable<Matériel>> GetSortMatérielNoOwnerAsync()[...]
```

Figure 38 : Repositories Matériel GetSort

Dans la figure 38 nous pouvons voir les différentes méthodes pour obtenir les Matériels 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 Matériels concernés par cette recherche.

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

Update

```
public async Task<Materiel> UpdateAsync(Materiel materiel)
{
    var sql = @" UPDATE Materiel
                SET NomMat = @NomMat,
                    DateMiseService = @DateMiseService,
                    DateFinGarantie = @DateFinGarantie,
                    IdProprietaire = @IdProprietaire
                WHERE IdMateriel = @IdMateriel";
    await db.Connection.ExecuteScalarAsync<int>(sql,
        new
        {
            NomMat = materiel.NomMat,
            DateMiseService = materiel.DateMiseService,
            DateFinGarantie = materiel.DateFinGarantie,
            IdProprietaire = materiel.IdProprietaire,
            IdMateriel = materiel.IdMateriel,
        }, transaction: db.Transaction);

    var mat = await GetByIdAsync(materiel.IdMateriel);
    var sql2 = @"DELETE FROM CatMat
                WHERE CatMat.IdMateriel = @IdMateriel";

    await db.Connection.ExecuteScalarAsync(sql2,
        new
        {
            IdMateriel = materiel.IdMateriel,
        }, transaction: db.Transaction);
    foreach (Categorie categorie in materiel.Categories)
    {
        var sql3 = @"INSERT IGNORE INTO CatMat (IdMateriel,IdCat) VALUES (@IdMateriel,@IdCat)";
        await db.Connection.ExecuteScalarAsync(sql3,
            new
            {
                IdMateriel = materiel.IdMateriel,
                IdCat = categorie.IdCategorie
            }, transaction: db.Transaction);
    }
    return mat;
}
```

Figure 39 : 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 effectuée 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 Matériel j'ai opté pour la stratégie de supprimer toutes les relations existantes de ce Matériel et de créer de nouvelles avec les catégories listées dans les informations données.

Add

```
public async Task<Materiel> AddAsync(Materiel materiel)
{
    var sql = @"INSERT INTO Materiel (NomMat, DateMiseService , DateFinGarantie, IdProprietaire)
                VALUES (@NomMat, @DateMiseService,@DateFinGarantie,@IdProprietaire);
                SELECT LAST_INSERT_ID() ";

    var id = await db.Connection.ExecuteScalarAsync<int>(sql,
        new
        {
            NomMat = materiel.NomMat,
            DateMiseService = materiel.DateMiseService,
            DateFinGarantie = materiel.DateFinGarantie,
            IdProprietaire = materiel.IdProprietaire
        }, transaction: db.Transaction);
    var mat = await GetByIdAsync(id);
    return mat;
}

2 références | SylvainTorrenti, il y a 47 jours | 1 auteur, 1 modification
public async Task<int> AddMatCatAsso(int idMat, int idCat)
{
    var sql = @"INSERT INTO CatMat (IdMateriel , IdCat)
                VALUES (@IdMateriel, @IdCat)";
    var result = await db.Connection.ExecuteAsync(sql,
        new
        {
            IdMateriel = idMat,
            IdCat = idCat
        }, transaction: db.Transaction);
    return result;
}
```

Figure 40 : 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

```
public async Task<int> DeleteAsync(int id)
{
    var sql = @"DELETE FROM CatMat
                WHERE IdMateriel = @id";
    await db.Connection.ExecuteAsync(sql, new { id }, transaction: db.Transaction);
    var sql2 = @"DELETE FROM Materiel
                WHERE IdMateriel = @id";
    var result2 = await db.Connection.ExecuteAsync(sql2, new { id }, transaction: db.Transaction);
    return result2;
}
```

Figure 41 : 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.

```
[Theory]
[InlineData(0)]
[InlineData(-1)]
0 références | SylvainTorrenti, il y a 46 jours | 1 auteur, 1 modification
public async void GetSortByUserMateriel_With_IdCategorieBelowZero_Should_Be_ReturnBadRequest(int id)
```

Figure 42 : 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 différents tags à apposer aux tests. Dans la figure 42 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 prises comme paramètres.

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

- **Arrange** : Lors de cette étape 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 étape, 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 43, 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

```
/// <summary> Récupère les materiels selon la categorie choisit
[HttpGet("sort/categorie/{IdCat}")]
3 références | 4/4 ayant réussi | SylvainTorrenti, il y a 46 jours | 1 auteur, 1 modification
public async Task<IActionResult> GetSortByCategorieMaterielAsync(int IdCat)
{
    // test 1
    if (IdCat <= 0)
    {
        return BadRequest();
    }
    var materiels = await _gestionMaterielService.GetSortMaterielByCategorieAsync(IdCat);

    //test 2
    if (materiels is null)
    {
        return NotFound();
    }

    //test 3
    var reponse = new List<MaterielReponse>();

    foreach (var materiel in materiels)
    {
        reponse.Add(new MaterielReponse
        {
            Categories = materiel.Categories,
            DateFinGarantie = materiel.DateFinGarantie,
            DateMiseService = materiel.DateMiseService,
            IdMateriel = materiel.IdMateriel,
            IdProprietaire = materiel.IdProprietaire,
            NomMat = materiel.NomMat,
            ProprietaireObj = materiel.ProprietaireObj
        });
    }
    return Ok(reponse);
}
```

Figure 43 : Analyse de la fonction pour les tests

Une fois les test dégagés, il faut créer du code pour chaque situation. Nous continuons avec l'exemple de notre méthode de filtrage par les catégories.

Test 1

```
/// <summary> 1er test sur la méthode GetSortMaterielByCategorie()
[Theory]
[InlineData(0)]
[InlineData(-1)]
0 références | SylvainTorrenti, il y a 46 jours | 1 auteur, 1 modification
public async void GetSortMaterielByCategorie_With_IdCategorieBelowZero_Should_Be_ReturnBadRequest(int id)
{
    //Arrange
    IGestionMaterielService gestionMaterielService = Mock.Of<IGestionMaterielService>();
    GestionMaterielServiceController gestionMaterielController = new GestionMaterielServiceController(gestionMaterielService);

    //Act
    var result = await gestionMaterielController.GetSortByCategorieMaterielAsync(id);

    //Assert
    Assert.NotNull(result);
    Assert.NotNull(result as BadRequestResult);
}
```

Figure 44 : 1er test

Lors de ce premier test (figure 44) 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

```
/// <summary> 2eme test sur la méthode GetSortMaterielByCategorie()
[Fact]
0 références | SylvainTorrenti, il y a 46 jours | 1 auteur, 1 modification
public async void GetSortMaterielByCategorie_With_AnyIdAndServiceNotFoundMateriel_Should_Be_ReturnNotFound()
{
    //Arrange
    IGestionMaterielService gestionMaterielService = Mock.Of<IGestionMaterielService>();

    Mock.Get(gestionMaterielService)
        .Setup(gestionMaterielService => gestionMaterielService.GetSortMaterielByCategorieAsync(It.IsAny<int>()))
        .ReturnsAsync(null as List<Materiel>);

    GestionMaterielServiceController gestionMaterielServiceController = new GestionMaterielServiceController(gestionMaterielService);

    //Act
    var result = await gestionMaterielServiceController.GetSortByCategorieMaterielAsync(1);

    //Assert
    Assert.NotNull(result);
    Assert.NotNull(result as NotFoundResult);
}
```

Figure 45 : 2eme test

Pour ce deuxième test (figure 45), le décorateur **[Fact]** est utilisé.

Pour la partie **Arrange** un **Mock** est 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

```
/// <summary> 3eme test sur la méthode GetSortMaterielByCategorie()
[Fact]
0 références | SylvainTorrenti, il y a 46 jours | 1 auteur, 1 modification
public async void GetSortMaterielByCategorie_With_AnyIdAndServiceFoundMateriel_Should_Be_ReturnMaterielResponse()
{
    //Arrange
    IGestionMaterielService gestionMaterielService = Mock.Of<IGestionMaterielService>();
    Materiel materiel1 = new();
    Materiel materiel2 = new();

    Mock.Get(gestionMaterielService)
        .Setup(gestionMaterielService => gestionMaterielService.GetSortMaterielByCategorieAsync(It.IsAny<int>()))
        .ReturnsAsync(new List<Materiel>() { materiel1, materiel2 });

    GestionMaterielServiceController gestionMaterielServiceController = new GestionMaterielServiceController(gestionMaterielService);

    //Act
    var result = await gestionMaterielServiceController.GetSortByCategorieMaterielAsync(1);

    //Assert
    Assert.NotNull(result as OkObjectResult); //SatutsCode = 200
    var materielResponse = (result as OkObjectResult).Value;
    Assert.NotNull(materielResponse); //materielResponse not null
}
```

Figure 46 : 3eme test

Ce troisième test (figure 46) 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

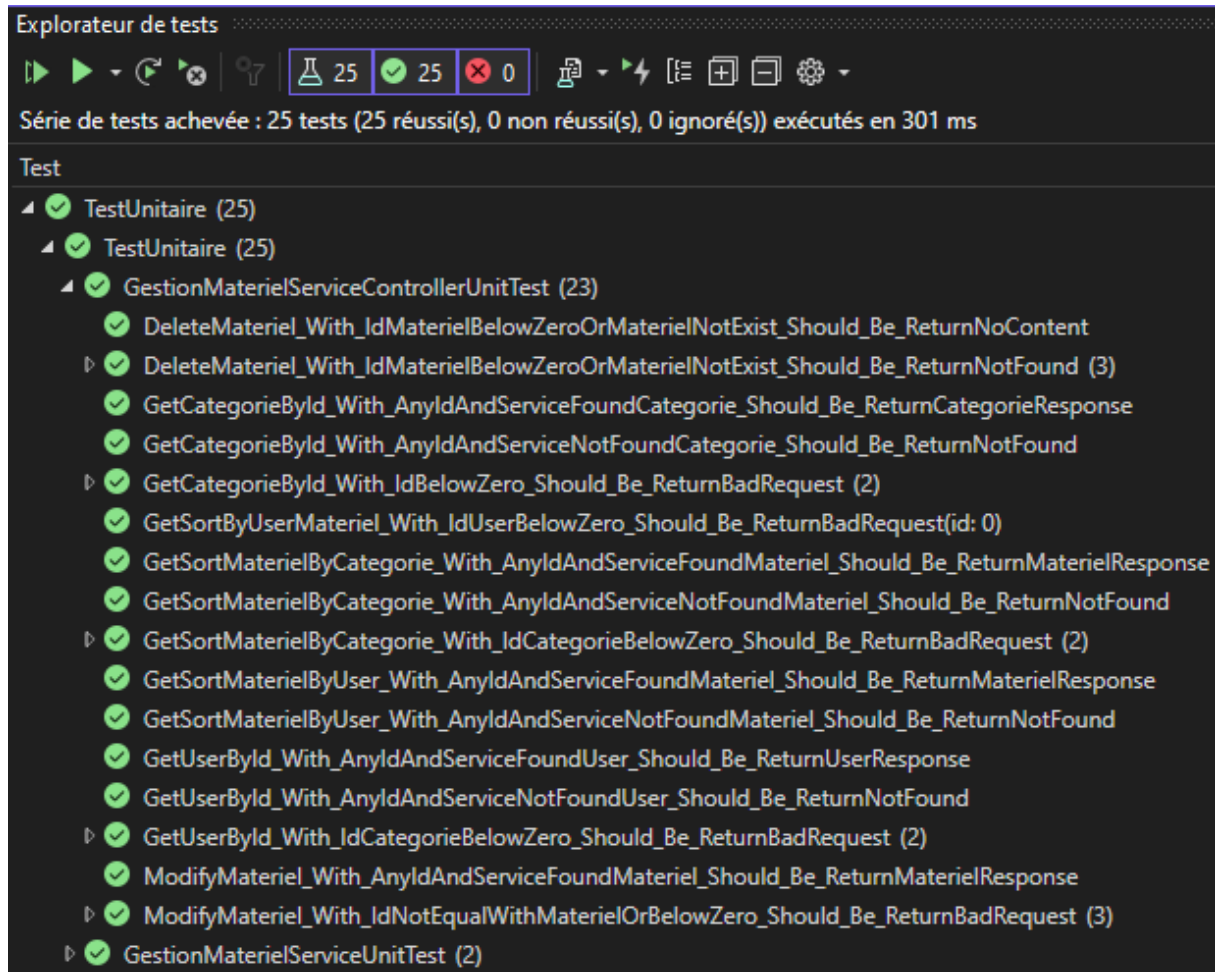


Figure 47 : Résultat des test

Comme nous le montre la figure 47, toutes les fonctions se trouvant dans le controller sont testées. Tous les tests sont valides. Nous pouvons également remarquer que chaque fonction n'a pas le même nombre de test.

Test d'Intégration

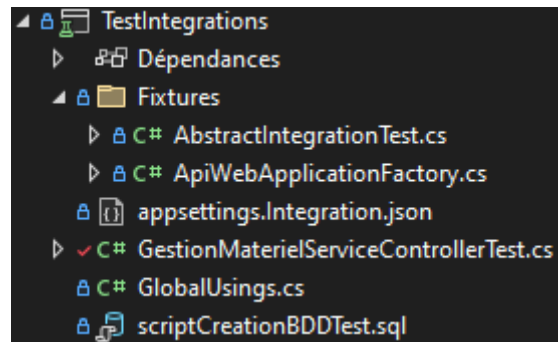


Figure 48 Test d'intégration Structure

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 grâce à une fixture.(figure 49)

```
public class AbstractIntegrationTest : ClassFixture<ApiWebApplicationFactory>
{
    protected readonly HttpClient _client;
    //référence | SylvainTorrenti, il y a 22 heures | 1 auteur, 1 modification
    public AbstractIntegrationTest(ApiWebApplicationFactory fixture)
    {
        _client = fixture.CreateClient();

        //fixer la BDD
        StreamReader create = new StreamReader("scriptCreationBDDTest.sql");

        string query = create.ReadToEnd();

        string connection = fixture.Configuration.GetSection("DBConnectionStrings").Value;

        var db = new DBSessionMariaDB(connection);

        var tr = db.Connection.BeginTransaction();

        db.Connection.Execute(query, transaction : tr);

        tr.Commit();
    }
}

//5 références | 5/5 ayant réussi | SylvainTorrenti, il y a 22 heures | 1 auteur, 1 modification
public void Login()
{
    _client.DefaultRequestHeaders.Authorization = new AuthenticationHeaderValue("bearer",
        "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIiOiI0bzYyZjEwNjllLTlhMmWlTnDk4Yy1iMDA5LTBhYmQwZjk5MWt0OSIsImh0dHA6Ly9zY2htbmFzLnhtbHNvVXXAub3lnL3dzLzIWMduVmDUvaWRlbnpRpdHkvY2xhak1zL25hbWVpZGVudGlmYWVjaWoidG9ycnN5bEBnbWFpbC5jb20ilClodHRwOih8vc2NoZW1hcY5taaNyb3NvZnQuY29tL3dzLzIWMduVmDUvaWRlbnpRpdHkvY2xhak1zL3dvbGUqOlslQWRtanAw5pc3RyYXRkdXI1LCVldGlscXNhdkVlcidjdClleHoAIeJoE3MDk4OTAtNjNsImJsbnVkbGVuc3Q6NTExMyIsImFIZICI6Imh0dHA6Ly9sb2hhbkdhcnV3Q6NTExMyFe-jzGjfzc3Hu27EFca4CAraipr4NYRJolfyBKVPjP2IT")";
}
```

Figure 49 Fixture

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 utilisée par l'application. J'ai également récupéré quelques 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 50 et 51)

```
USE TestIntegrationProjetFilRouge2;

drop table if exists CatMat;
drop table if exists Materiel;
drop table if exists Categorie;
drop table if exists UserRole;
drop table if exists Role;
drop table if exists users;

create table Role(
    Id int NOT NULL AUTO_INCREMENT,
    Label VARCHAR(90),
    PRIMARY KEY(Id)
) ENGINE = InnoDB;

CREATE TABLE users(
    Id bigint(20) unsigned NOT NULL AUTO_INCREMENT,
    email VARCHAR(255) UNIQUE,
    password VARCHAR(255),
    nameOwner VARCHAR(255),
    PrenomOwner VARCHAR(25),
    PRIMARY KEY(Id)
) ENGINE=InnoDB;

create table UserRole(
    IdUser bigint(20) unsigned,
    IdRole int,
    PRIMARY KEY(IdUser, IdRole)
```

Figure 50 : Script Création de table

```
INSERT INTO users (Id,email,password,nameOwner,PrenomOwner) VALUES
('1','torrsyl@gmail.com','$2a$10$8a6S3gfS1nWlk2/W7NCxquGnUMNkKhviIXKKWVa5cs9zEmbuZNhL6','Torrenti','Sylvain'),
('2','Doe@Doe.com','$2a$10$8a6S3gfS1nWlk2/W7NCxquGnUMNkKhviIXKKWVa5cs9zEmbuZNhL6','Doe','Jhon');

INSERT INTO Categorie (IdCategorie,NomCat) VALUES
('1','cat1'),
('2','cat2'),
('3','cat3'),
('4','cat4'),
('5','cat5');

INSERT INTO Materiel (NomMat,DateMiseService,DateFinGarantie,IdProprietaire) VALUES
('materiel 1','2023-11-12','2023-11-30',NULL),
('materiel 2','2023-11-16','2023-11-16',1),
('materiel 3','2023-11-16','2023-11-16',NULL),
('materiel 4','2023-11-16','2023-11-16',1),
('materiel 5','2023-11-16','2023-11-16',2);

INSERT INTO Role (Id,Label) VALUES
(1,'Administrateur'),
(2,'Utilisateur');

INSERT INTO UserRole (IdUser,IdRole) VALUES
(1,1),
(2,2);

INSERT INTO CatMat (IdMateriel,IdCat) VALUES
(1,2),
(1,3),
(2,3),
(2,4),
(3,4),
(4,5),
(5,1),
(5,2),
(5,3),
(5,4),
(5,5);
```

Figure 51 : Script jeu de données

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

Le fichier contenant ce scripts est lu à chaque exécution d'un test ce qui permet d'avoir une BDD dans un état similaire lors de chaque tests.

J'ai également créé une méthode qui permet de se loguer avec le renseignements d'un Bearer Token d'un compte administrateur pour avoir accès à toutes les fonctionnalités. (figure 52)

```
public void Login()
{
    _client.DefaultRequestHeaders.Authorization = new AuthenticationHeaderValue("bearer",
    "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIiOiJ0b3Jyc3lsQGdtYWlsZmVhbnVbSIsImp0aSI6ImY4ZjI0NjI1LTUyMTU0NDk4Yy1iMDA5LTBhYmQwZjZk5mMTU0OSIsImh0dHA6Ly9zY2h1bWZlbnRlbnRlbnRpdHkvY2xhaW1zL25hbnVpZGVudG1maWVzIjoidG9ycnN5bEBnbWVpbC5jb20iLCJodHRwOi8vc2NoZm1hcy5taWVyb3NvZnQuY29tL3dzLzIwMjYvMDYvYmRlbnRpdHkvY2xhaW1zL3JvbGUiOiIsImRtaW5pc3RyYXRlZDIiLCJvdG1saXNhdGV1ciJldCJleHAiOiJlMjY3MDk4OTAzNjIsIm1zcyI6Imh0dHA6Ly9sb2NhbGhvc3Q6NTE3MyIsImF1ZCI6Imh0dHA6Ly9sb2NhbGhvc3Q6NTE3MyJ9.Fe-jzGJfzc1Hu2L7EFca4CRAipR4nYRjof1fyBKvjP21");
}
```

Figure 52 Test d'intégration Login

Cette méthode sera appelée lors de chaque tests.

Je n'ai malheureusement pas eu le temps de faire tous les tests d'intégrations. Voici une capture d'écran des tests réalisés. (Figure 53 et figure 54)

```

public class GestionMaterielServiceControllerTest : AbstractIntegrationTest
{
    0 références | SylvainTorrenti, Il y a 22 heures | 1 auteur, 1 modification
    public GestionMaterielServiceControllerTest(ApiWebApplicationFactory fixture) : base(fixture)
    {
    }

    [Fact]
    0 références | SylvainTorrenti, Il y a 22 heures | 1 auteur, 1 modification
    public async Task GetAllMaterielAsync_ShouldBeRetrieve_5Materiels()...

    [Theory]
    [InlineData(1)]
    0 références | SylvainTorrenti, Il y a 19 heures | 1 auteur, 1 modification
    public async Task GetSortByCategorieMaterielAsync_WithIdCat1_ShouldBeRetrieve_2Materiels(int IdCat)...

    [Theory]
    [InlineData(1)]
    0 références | SylvainTorrenti, Il y a 19 heures | 1 auteur, 1 modification
    public async Task GetSortByUserMaterielAsync_WithIdUser1_ShouldRetrieve_2Materiels(int IdUser)...

    [Fact]
    0 références | SylvainTorrenti, Il y a 19 heures | 1 auteur, 1 modification
    public async Task GetAllCategorieAsync_ShouldRetrieve_5Categories()...

    [Fact]
    0 références | SylvainTorrenti, Il y a 19 heures | 1 auteur, 1 modification
    public async Task GetAllUser_ShouldRetrieve_2Users()...
    
```

Figure 53 Tests d'intégrations effectués

▲ ✓ TestIntegrations (5)	38,9 s
▲ ✓ TestIntegrations (5)	38,9 s
▲ ✓ GestionMaterielServiceControllerTest	38,9 s
✓ GetAllCategorieAsync_ShouldRetr...	7,6 s
✓ GetAllMaterielAsync_ShouldBeRe...	8,4 s
✓ GetAllUser_ShouldRetrieve_2Users	7,9 s
✓ GetSortByCategorieMaterielAsyn...	8,3 s
✓ GetSortByUserMaterielAsync_Wit...	6,8 s

Figure 54 Test d'intégrations résultats

Pour entrer dans la détail inspectons de plus près un test effectuer que nous pouvons voir dans la figure 55.

```
[Fact]
✓ | 0 références | SylvainTorrenti, Il y a 22 heures | 1 auteur, 1 modification
public async Task GetAllMaterielAsync_ShouldBeRetrieve_5Materiels()
{
    //Arrange
    Login();

    //Act
    var result = await _client.GetFromJsonAsync<List<Materiel>>("api/materiels");

    //Assert
    Assert.Equal(5, result.Count);
}
```

Figure 55 Test GetAllMaterielAsync

Il y a toujours la présence de nos trois étapes, **Arrange**, **Act** et **Assert**. Dans la section **Arrange** nous pouvons voir qu'il n'y a pas de Mock contrairement aux tests unitaires. Comme nous le montre la figure 53 la fixture est créée dans le constructeur. J'ai donc simplement mis ma méthode pour me Login avec un compte administrateur.

Les deux autres étapes sont similaires aux tests unitaires. L'étape **Act** exécute ce que demande la méthode en lui indiquant la route qu'elle doit emprunter. L'étape **Assert** vérifie que les résultats obtenus sont ceux désirés.

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 créés 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 réalisé 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 connaître 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.