Dossier de Projet

Table des matières

[I. Liste des compétences couvertes par ce dossier. 3](#_Toc143098719)

[1. Concevoir et développer des composants d'interface utilisateur en intégrant les recommandations de sécurité 3](#_Toc143098720)

[2. Concevoir et développer la persistance des données en intégrant les recommandations de sécurité 3](#_Toc143098721)

[3. Concevoir et développer une application multicouche répartie en intégrant les recommandations de sécurité 3](#_Toc143098722)

[II.Resume 4](#_Toc143098723)

[III. Cahier des charges 5](#_Toc143098724)

[IV. Gestion de projet 8](#_Toc143098725)

[1.Environnement Humain 8](#_Toc143098726)

[2.Planning et suivi 8](#_Toc143098727)

[3.Sauvegardes 8](#_Toc143098728)

[4.Convention de nommages et règles globales 8](#_Toc143098729)

[5.Environement technique 9](#_Toc143098730)

[V. Spécification fonctionnelle 10](#_Toc143098731)

[VI. Spécifications Technique 16](#_Toc143098732)

[VII. Réalisations 20](#_Toc143098733)

[1.Back-End 20](#_Toc143098734)

[a. Base de données 20](#_Toc143098735)

[b. Api et utilisation du design pattern strategy. 21](#_Toc143098736)

[2.Le front End 31](#_Toc143098737)

[a. La création de l’application mobile 31](#_Toc143098738)

[VIII. Jeu d’essai 38](#_Toc143098739)

[XI. Veille de sécurité 42](#_Toc143098740)

[1.L’absence de support à long terme 42](#_Toc143098741)

[2.La mise à jour du projet 42](#_Toc143098742)

[X. Recherches nécessaires 43](#_Toc143098743)

[1.Origine du problème 43](#_Toc143098744)

[2. Premières solutions 43](#_Toc143098745)

[3.L’injection de dépendance 45](#_Toc143098746)

[XI. Conclusion 47](#_Toc143098747)

# I. Liste des compétences couvertes par ce dossier.

### 1. Concevoir et développer des composants d'interface utilisateur en intégrant les recommandations de sécurité

\*Maquetter une application

\*Développer des composants d’accès aux données

\*Développer la partie back-end d’une interface utilisateur web

### 2. Concevoir et développer la persistance des données en intégrant les recommandations de sécurité

\*Concevoir une base de données

\*Mettre en place une base de données

\*Développer des composants dans le langage d’une base de données

### 3. Concevoir et développer une application multicouche répartie en intégrant les recommandations de sécurité

\*Concevoir une application

\*Développer des composants métier

\*Construire une application organisée en couches

\*Développer une application mobile

\*Préparer et exécuter les plans de tests d’une application

# II.Resume

This project follows a previous one where my employer received an application used to help him manage is sellout. Due to limitations in the previous project, mainly the modularity of the accessible data, he wanted to expand the fonctionnality of the application to make it more versatile. My goal was to add fonctionnality to the previous project, add a lot of modularity to it and increase the overall security of it regarding to RGPD, excpecially its API security through the addition of CSRF token. All of this must take the form of a mobile application with a distant database accessible through the previously mentionned API. I was alone on this project so all the conception, from the database to the app maquette, as to be done by me. The whole project was supervised by Loic Creusot the store owner and Christophe Vaucourt its informatical referent and was done mainly in my home where i was remote-working in a almost complete autonomy.

# III. Cahier des charges

1. Introduction

L'objectif de ce projet est d'améliorer une application Android existante pour rendre l'expérience utilisateur plus efficace. L'application actuelle a été développée avec MIT App Inventor. Les fonctionnalités existantes comprennent l'enregistrement des coordonnées des clients, la sélection de produits par catégories, la gestion des quantités, la sélection de la date de commande, l'ajout de produits au panier, le calcul du total du panier, la génération d'un récapitulatif de commande avec un numéro unique, l'impression des bons de commande sur une imprimante thermique Bluetooth, la génération de rapports de commande sur une page web et l'envoi de mails de confirmation de commande. Ces fonctionnalités existantes doivent être améliorées pour rendre l'expérience utilisateur plus fluide et conviviale.

2. Améliorations souhaitées

Les améliorations suivantes doivent être apportées à l'application existante :

2.1 Interface utilisateur conviviale

- Rendre l'interface utilisateur plus intuitive, attrayante et conviviale en améliorant la disposition des éléments, les icônes, les couleurs et les animations.

2.2 Optimisation des performances

- Optimiser les performances de l'application pour garantir une expérience utilisateur fluide et réactive. Cela inclut la réduction des temps de chargement, l'optimisation de l'utilisation des ressources système et la gestion efficace de la mémoire.

2.3 Amélioration de la gestion des coordonnées clients

- Permettre aux utilisateurs de modifier facilement leurs coordonnées enregistrées, ainsi que la possibilité de supprimer leur compte si nécessaire.

2.4 Amélioration de la sélection de produits par catégories

- Améliorer la fonctionnalité de sélection des produits par catégories en rendant la navigation plus rapide, plus intuitive et offrant des options de recherche et de filtrage avancées.

2.5 Amélioration de la gestion des quantités

- Permettre aux utilisateurs de gérer plus efficacement les quantités de produits, en offrant des options pour modifier les quantités directement dans le panier, utiliser des boutons +/- pour ajuster les quantités, ou utiliser des gestes intuitifs.

2.6 Amélioration de la génération du récapitulatif de commande

- Améliorer la présentation et la lisibilité du récapitulatif de commande en y incluant toutes les informations pertinentes, telles que la date de commande, les coordonnées du client, le détail des produits commandés et le montant total.

2.7 Amélioration de l'impression des bons de commande

- Optimiser le processus d'impression des bons de commande sur l'imprimante thermique Bluetooth, en veillant à ce que l'interface soit conviviale, que l'impression soit fiable et de haute qualité, et en offrant des options de personnalisation du format et du contenu des bons de commande.

2.8 Amélioration de la génération des rapports de commande

- Améliorer la génération des rapports de commande sur une page web en utilisant des technologies web modernes, en ajoutant des fonctionnalités de recherche, de filtrage, de tri, de pagination et en améliorant la visualisation des données.

2.9 Amélioration de l'envoi de mails de confirmation de commande

- Rendre l'envoi de mails de confirmation de commande plus fiable, rapide et personnalisable, en incluant toutes les informations pertinentes et en permettant une personnalisation avancée du contenu du mail.

3. Contraintes techniques

Le projet n'est pas limité à une technologie spécifique et les développeurs sont libres d'utiliser les langages de programmation, les Framework et les outils appropriés pour améliorer l'application Android existante.

4. Livrables attendus

Les livrables attendus pour ce projet sont les suivants :

- L'application Android améliorée, avec toutes les fonctionnalités et les améliorations spécifiées dans ce cahier des charges.

- Tout code source, fichiers de projet ou documentation pertinents.

5. Échéancier

L'échéancier du projet sera défini en concertation avec Loïc CREUSOT et l'équipe de développement, en tenant compte de la complexité des améliorations et des ressources disponibles.

# IV. Gestion de projet

## 1.Environnement Humain

Comme mentionné dans mon résumé j’ai travaillé sur mon projet seul, mon suivi étant assuré par un non informaticien. Comme mon projet était quasi intégralement en télétravail j’ai décidé avec mon tuteur de faire un rapport de ma progression tous les vendredis au minimum, avec des échanges régulier le reste de la semaine quand cela s’avère nécessaire.

## 2.Planning et suivi

J’ai également créé un planning afin d’anticiper au mieux les durées et de savoir quand je devais accepter de faire des concessions pour atteindre l’objectif d’avoir une application viable avant la fin du stage :

Une image contenant texte, capture d’écran, Caractère coloré, Parallèle

Description générée automatiquement

## 3.Sauvegardes

Afin de sécuriser la progression du projet j’ai également créé un repo git pour pouvoir garder une trace de l’avancement et avoir une sauvegarde de secours, à lequel s’est ajouté une sauvegarde physique sur un clé, afin de respecter la règle des 3 sauvegardes, 2 supports différents, 1 hors site.

## 4.Convention de nommages et règles globales

J’ai choisi de suivre scrupuleusement les conventions de nommage recommandée par .net pour mon code, le PascalCase pour les attributs de classe et les fonctions, le camelCase pour les propriétés afin que mon code soit aisément compréhensible pour tout futur développeur c# qui serait amener à travailler dessus.

J’ai également réalisé un plan de tests pour l’application mobile et systématisé le test d’une fonctionnalité après son ajout dans l’application.

Enfin j’ai rencontré à plusieurs reprise mon tuteur afin de lui montrer l’avancement de mon travail quand celui-ci avait atteint des paliers significatifs, notamment sur l’avancée de l’application mobile afin de valider les choix que j’ai effectué sur l’implémentation concrète de la maquette que j’avais construit au début du stage.

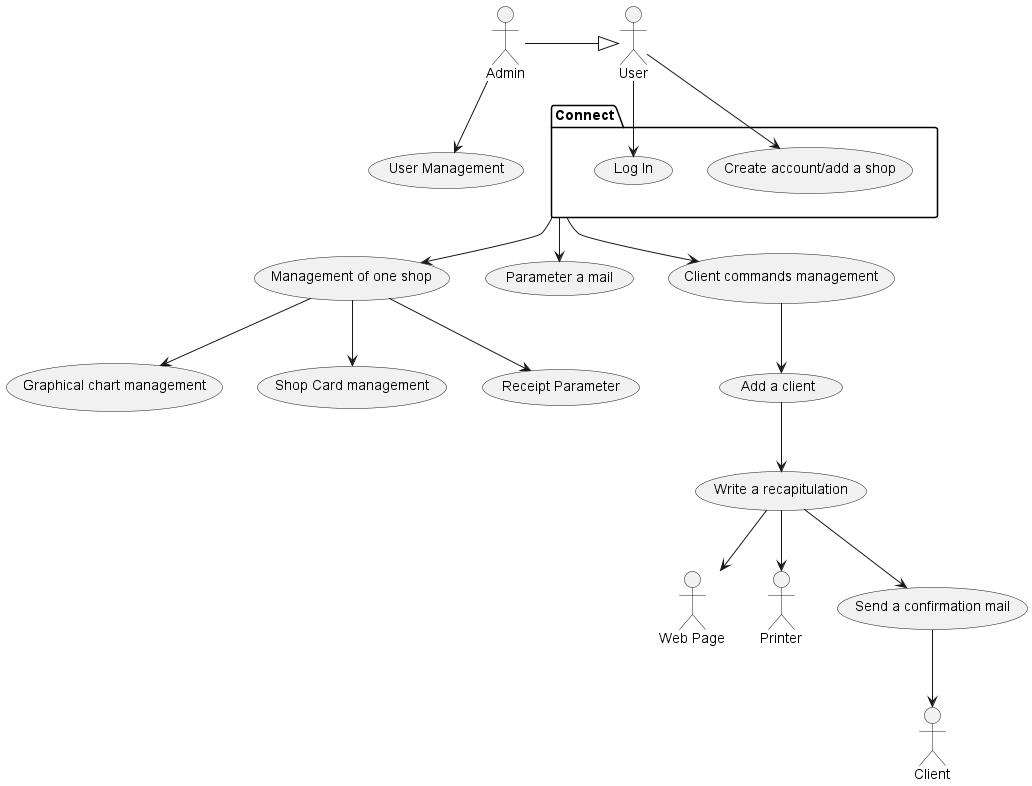
## 5.Environement technique

Le travail a été réalisé sur mon ordinateur personnel, avec dans un premier temps une base de données locale PostgreSQL, puis MySQL. J'ai utilisé un dépôt GitHub personnel pour le versionnage de l'application, et j'ai travaillé avec une imprimante Bluetooth fournie par mon tuteur pour tester les fonctionnalités liées à l'impression.

# V. Spécification fonctionnelle

1.Use case

La première étape à été de réaliser un diagramme Use Case de l’application :



2. Gestion des coordonnées clients

- Les utilisateurs doivent pouvoir modifier facilement leurs coordonnées enregistrées, telles que le numéro de téléphone, le nom et l'adresse e-mail.

- Une fonctionnalité de suppression de compte doit être disponible.

3. Sélection de produits par catégories améliorée

- La fonctionnalité de sélection de produits par catégories doit être améliorée pour faciliter la recherche et la sélection des produits.

- Des options de recherche et de filtrage doivent être fournies pour une meilleure expérience utilisateur.

4. Gestion des quantités optimisée

- Les utilisateurs doivent pouvoir modifier les quantités de produits dans le panier de manière facile et intuitive.

- Les modifications de quantités doivent être mises à jour en temps réel dans le panier et le montant total doit être recalculé automatiquement.

5. Génération du récapitulatif de commande améliorée

- Le récapitulatif de commande doit inclure toutes les informations pertinentes, telles que la date de commande, les coordonnées du client, le détail des produits commandés avec les quantités et les prix unitaires, ainsi que le montant total de la commande.

- Le récapitulatif doit être clair et facilement compréhensible pour les utilisateurs.

6. Impression des bons de commande optimisée

- L'interface d'impression des bons de commande sur l'imprimante thermique Bluetooth doit être conviviale et facile à utiliser.

- Les bons de commande imprimés doivent être bien formatés et inclure toutes les informations essentielles, telles que le numéro de commande, les coordonnées du client et le détail des produits commandés.

7. Génération des rapports de commande améliorée

- Les rapports de commande générés sur une page web doivent offrir des fonctionnalités avancées, telles que la recherche, le filtrage, le tri et la pagination des données.

- Les rapports doivent être clairs et faciles à interpréter, avec des graphiques ou des tableaux si approprié, pour faciliter l'analyse des données.

8. Envoi de mails de confirmation de commande amélioré

- L'envoi de mails de confirmation de commande doit être fiable, rapide et personnalisable.

- Les mails doivent inclure toutes les informations essentielles, telles que le numéro de commande, le détail des produits commandés, le montant total et les coordonnées du client.

-Les utilisateurs doivent avoir la possibilité de personnaliser le contenu du mail, en ajoutant des messages spécifiques ou des offres promotionnelles.

9.Maquette de l'application

Afin de pouvoir travailler le plus efficacement possible sur la partie mobile, j'ai réalisé une maquette de l'application sur Figma. Mon objectif premier était de me concentrer sur l'expérience utilisateur et de pouvoir discuter avec mon maître de stage de ma vision du fonctionnement de l'application en pouvant m'appuyer sur des démonstrations visuelles, les détails de l'apparence des objets n'étant à ce stade pas considérés.

Voici quelques exemples des pages de l'application réalisées avec cet outil :

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

Description générée automatiquement\*Le menu de navigation

\*L’écran de gestion d’une commande

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

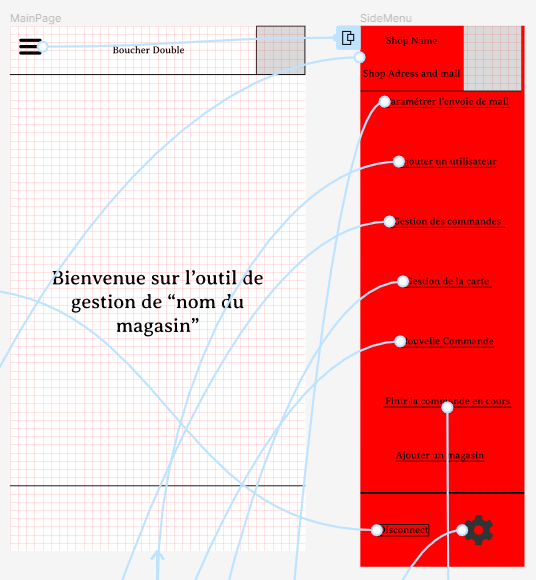
Description générée automatiquement

Une vue d’ensemble de la maquette :



J’ai également utilisé les fonctionnalités de Figma pour simuler la navigation dans la maquette en reliant les différents éléments.

Un exemple de lien Figma :



Cette étape m’a permis, après plusieurs allez retour par mail ou téléphone et une rencontre avec mon tuteur afin de montrer l’utilisation pratique de l’application de préciser plusieurs choses :

\*la nécessité de pouvoir rester connecter, un client pouvant être amener à prendre la commande lui-même

\* l’importance de pouvoir mettre à jour facilement le catalogue et surtout les commandes.

\*L’organisation des catégories de produits

# VI. Spécifications Technique

1. Technologie utilisée

-C# comme langage de programmation

-Asp.Net comme Framework pour le développement du back end.

- Utilisation de Xamarin Forms puis de .Net Mauie comme Framework de développement pour créer l'application Android, permettant ainsi une approche de développement multiplateforme.

- Utilisation de PostgreSQL puis de MySQL comme système de gestion de base de données pour stocker les données de l'application.

- Utilisation de Visual Studio comme IDE principal pour le développement de l'application.

2. Maquettage et conception de l'interface utilisateur

- Utilisation de Figma comme outil de conception pour créer les maquettes et les wireframes de l'interface utilisateur de l'application Android.

- Les maquettes serviront de référence visuelle lors du développement de l'interface utilisateur, assurant une cohérence visuelle et une expérience utilisateur optimale.

3. Interface de paramétrage

- Développement d'une interface permettant aux utilisateurs de configurer les paramètres de l'application, tels que l'identité de l'entreprise, les informations de connexion au serveur de base de données, les identifiants, etc.

4. Interface de gestion de l'imprimante

- Développement d'une interface permettant la gestion de l'imprimante thermique Bluetooth, facilitant l'impression des bons de commande sur des supports physiques.

5. Interface de gestion du choix des cartes

- Développement d'une interface utilisateur intuitive pour faciliter la gestion des cartes disponibles dans l'application, offrant aux utilisateurs la possibilité de faire des choix de cartes en fonction de leurs préférences.

6. Amélioration du système d'ajout/suppression du panier

- Amélioration du système d'ajout et de suppression d'articles dans le panier pour offrir une expérience utilisateur plus fluide.

- Mise à jour en temps réel du panier et du montant total lors de l'ajout ou de la suppression d'articles.

7. Amélioration de la gestion des quantités

- Optimisation de la gestion des quantités de produits dans le panier, permettant aux utilisateurs de modifier facilement les quantités.

- Mise à jour automatique du panier et du montant total en fonction des modifications de quantités.

8. Interface de gestion des identifiants mails et du corps du mail

- Développement d'une interface conviviale pour la gestion des identifiants mails, tels que les adresses e-mail de l'expéditeur, les informations de connexion au serveur SMTP, etc.

- Possibilité de personnaliser le contenu du mail de confirmation de commande.

9. Interface de gestion du ticket

- Développement d'une interface utilisateur pour la gestion des paramètres du ticket, tels que le logo, les mentions légales, le message de pied de page, les mentions spéciales, les exigences RGPD, etc.

10. Sécurité de l'application

- Ajout de mesures de sécurité, notamment l'utilisation de tokens CSRF (Cross-Site Request Forgery) pour l'accès à l'API. Cela permettra de prévenir les attaques CSRF en vérifiant l'origine des requêtes et en s'assurant que seules les requêtes légitimes provenant de l'application sont autorisées.

-Les mots de passe seront hachés avant d’être enregistrer dans la base de données

-La base de données et le back end sont conçus pour compartimenter l’accès aux informations en fonction du magasin de l’utilisateur et son rôle.

11. Développement de la base de données et du backend

- Modification et extension de la base de données existante pour prendre en charge les nouvelles fonctionnalités et la gestion multi-boutiques.

- Développement d'un backend avec le Framework l'API ASP.NET avec Visual Studio comme IDE principal pour la gestion des entités, y compris les utilisateurs, les cartes, les catégories, les clients, les commandes, les bas de page du ticket et le système d'envoi d'e-mails.

La base de données et le modèle ont été définit à cette étape :

J’ai commencé par modéliser la base de données via un diagramme MCD. Le principal défi a été d’assurer l’étanchéité des données entre les Magasins, ce qui a nécessité plusieurs réécritures. Pour sécuriser l’accès aux données j’ai fini par arriver à la structure de base de données suivante :

\*MCD :

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

Description générée automatiquement

\*MLD :

Une image contenant diagramme, texte, Plan, Dessin technique

Description générée automatiquement

Les classes métiers, qui sont communes au Back End et au Front end ont aussi été modélisé à ce moment :

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

Description générée automatiquement

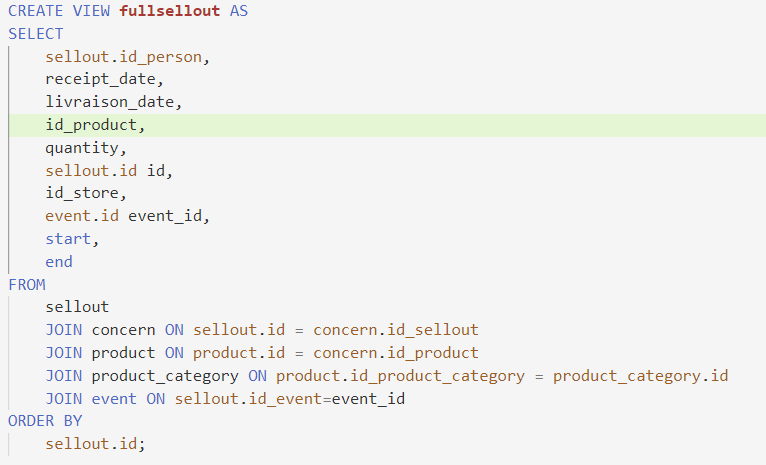
# VII. Réalisations

## 1.Back-End

Une fois le modèle conçus j’ai choisi de réaliser pendant mon stage est la partie « backend » de l’application qui consiste en une Api permettant de réaliser un CRUD de la base de données en étant contrôlée à distance par la partie client. Afin d’atteindre cet objectif je me suis d’abord concentré sur la base de données puis j’ai d’abord réalisé une partie métier avec toutes les classes associées avant de créer une DAO permettant l’échange avec la base de données et enfin des contrôleurs gérant les requêtes HTTP.

### Base de données

Afin de simplifier l’utilisation de la base de donnée dans le code j’ai choisit de réaliser plusieurs view regroupant les informations nécessaire pour extraitre les informations nécessaire pour l’instantiation des objets du modèle dont voici un des exemples :



Le but de cette vue est de récupérer les informations d’une vente en prenant en compte le magasin qu’elle concerne afin de filtrer correctement les résultats des requêtes.

J’ai également implémenté des triggers afin d’améliorer la robustesse des données et d’assurer que le paramétrage du mail et du ticket de caisse est par défaut le nom du magasin qui les a créés, si il est disponible, voici un exemple avec la facturation :

CREATE OR REPLACE TRIGGER **update\_bill\_parameter\_name**

BEFORE INSERT ON bill\_parameter

FOR EACH ROW

BEGIN

    IF NEW.name IS NULL THEN

        SET NEW.name = (SELECT name FROM store WHERE id = NEW.id\_store);

    END IF;

END //

Enfin, j'ai créé des fonctions permettant d'obtenir des informations non triviales dans la base de données. L'exemple le plus marquant est cette fonction permettant de déterminer si un produit correspond à un menu ou non. Cela facilite sa distinction par rapport aux autres types de produits :

CREATE OR REPLACE FUNCTION **is\_menu**(id INT) RETURNS BOOLEAN AS $$

BEGIN

    SELECT \* FROM contain WHERE id\_menu=id;

    RETURN FOUND;

END;

$$ LANGUAGE plpgsql;

### b. Api et utilisation du design pattern strategy.

#### b.1 DAO

Afin d’interagir avec la base de données j’ai décidé d’utiliser un design pattern DAO, nécessaire pour simplifier la création des API. L’interface générique de la DAO est la suivante :



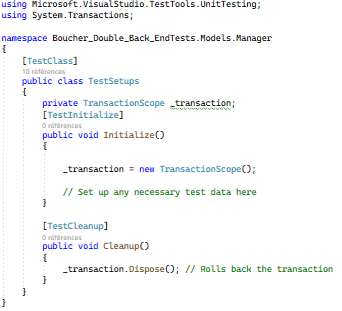
Une fois cette interface créée le principal défi a été de travailler avec les classes du modèle incluant d’autres classes. Un des exemples le plus parlant est la fonction GetAllAsync de la classe SelloutDAO :



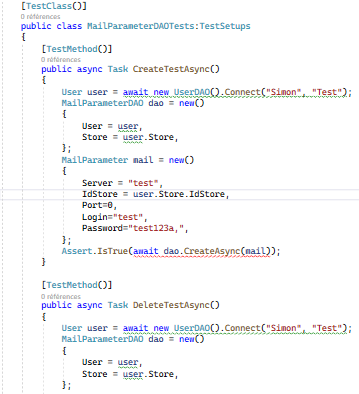
Une image contenant texte, capture d’écran

Description générée automatiquement

Une fois les classes de la DAO créées, l'étape suivante a été de mettre en place un plan de test afin de m'assurer du bon fonctionnement de cette partie centrale du code. Comme toutes les fonctions utilisent la base de données, j'ai dû trouver une solution pour que celle-ci reste identique au fur et à mesure des tests. Afin de réaliser cet objectif, j'ai initialisé chacun de mes tests en commençant une transaction sur la base de données et j'ai terminé chacun de mes tests en effectuant un Rollback. Ainsi, les modifications sur la base de données n'étaient pas persistées en dehors de la portée des tests, assurant une stabilité des résultats. L'implémentation concrète de cette solution est la suivante :



Pour que tous mes tests soient initialisés correctement, je définis chaque classe de test comme une sous-classe de la classe "TestSetups". Grâce à cela, je peux manipuler la base de données sans compromettre son intégrité. Chaque série de tests suit une trame similaire en instanciant la classe DAO associée avec un utilisateur et un magasin, ce qui permet d'interagir correctement avec la base de données. Voici un exemple :





La réalisation et la validation de tous ces tests étaient des conditions indispensables pour poursuivre mon travail vers la prochaine étape : la création des contrôleurs de l'API.

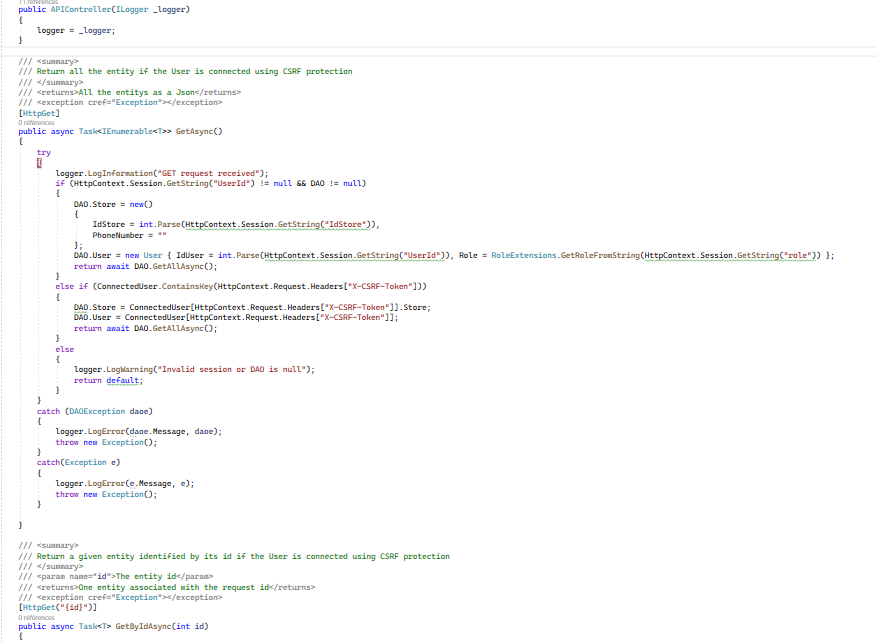
b.2 Les contrôleurs

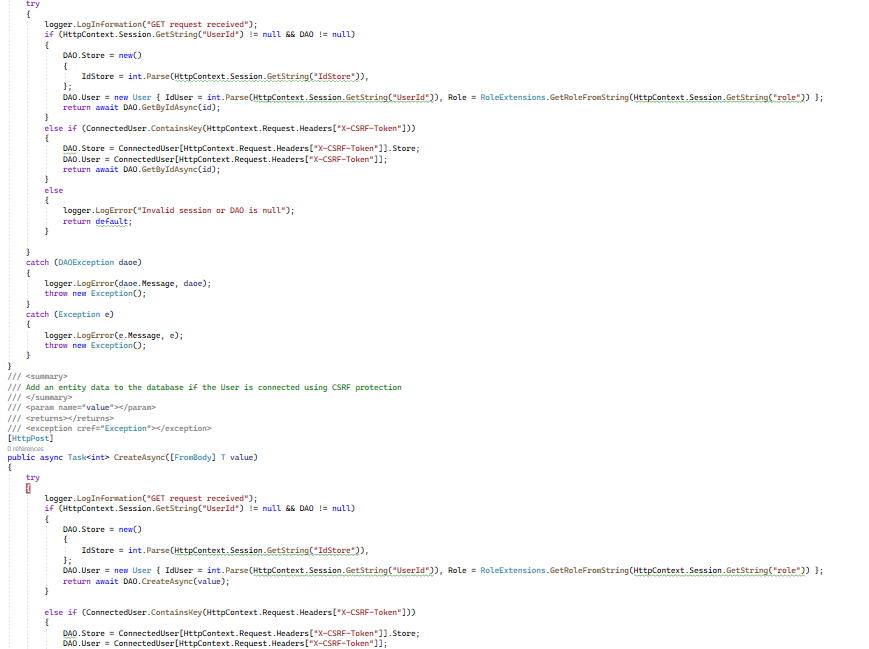
Afin de limiter au maximum la répétition, j'ai décidé d'utiliser la généricité de la classe DAO et d'implémenter un design pattern Strategy pour instancier la bonne classe DAO en fonction de l'objet que l'on essaie de récupérer via l'API. L'implémentation concrète du design pattern a pris la forme suivante :

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

Description générée automatiquement

En fonction de la classe avec laquelle on cherche à interagir la bonne DAO est instancié de manière anonyme, permettant de définir toutes les API une seule fois :





Une image contenant texte, capture d’écran

Description générée automatiquement

Une image contenant texte, capture d’écran

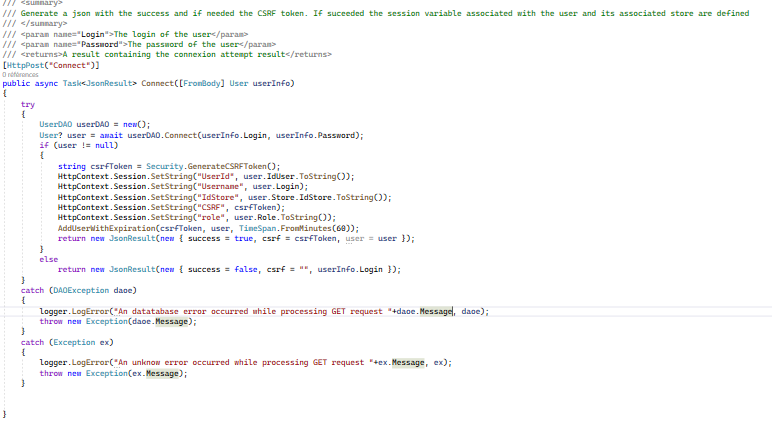
Description générée automatiquement

Pour obtenir toutes les APIs nécessaires, il me suffit maintenant de les définir comme des implémentations concrètes de la classe "APIController", comme ceci :

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

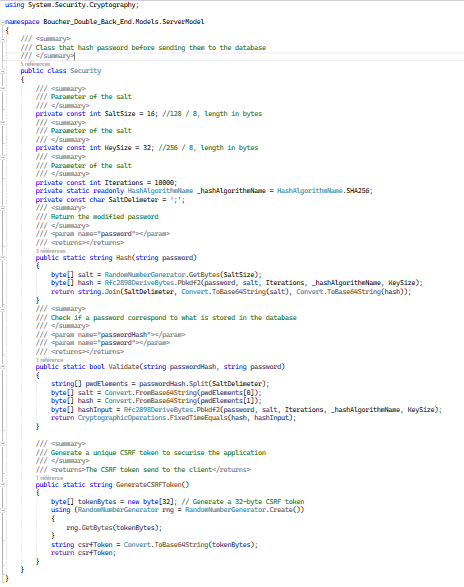
Description générée automatiquement

Chacune de ces API interagit avec l’utilisateur, le magasin et le token CSRF stocké dans les variables de session afin d’assurer la sécurisation de l’API. Ces variables sont définies lors de la connexion de l’utilisateur dans la classe UserController :



#### b.3 Sécurisation

De plus afin de rajouter une couche de sécurité supplémentaire les mots de passe des utilisateurs sont hachés en utilisant les méthodes de la classe « Security », qui gère aussi la création des tokens CSRF :



Les tokens CSRF mentionnés précédemment jouent un rôle central dans l’application, en effet dans le cadre d’une application mobile l’authentification via des variables de session est impossible, c’est là qu'interviennent les tokens CSRF, envoyer dans le header de chaque requête à l’Api, en dehors bien évidemment de la requête de connexion, ils permettent d’identifier un utilisateur à l’aide d’une table Clé-Valeurs des utilisateurs connectés.

#### c. Le déploiement

Maintenant que mon API était construite et testée, il était temps de la déployer pour pouvoir l'utiliser. En accord avec mon tuteur de stage, nous avons choisi d'utiliser un hébergement à distance. Après avoir consulté plusieurs offres d'hébergement, nous avons opté pour l'hébergement ASP Starter de l'hébergeur LWS, car il inclut Asp.Net. Pour le déploiement, j'ai envoyé les fichiers du build via le protocole TCP et j'ai reconstruit la base de données sur le serveur distant. Étant donné que l'hébergement ne prenait pas en charge PostgreSQL, j'ai dû changer la technologie de la base de données et migrer vers MySQL. Après avoir réécrit le script pour le rendre compatible avec ce SGBD, j'ai également apporté quelques modifications mineures au code pour qu'il puisse fonctionner avec ce type de SGBD. Maintenant que j'avais une adresse web sur laquelle tester mes appels à l'API, je pouvais passer à la suite du travail : la création de l'application mobile.

## 2.Le front End

### a. La création de l’application mobile

#### a.1 Le choix d’une navigation à onglet

Afin de me rapprocher au mieux de la maquette que j'ai réalisée, j'ai choisi de développer une application à onglets. À mon sens, c'est la forme d'application qui permet d'intégrer au mieux la navigation par menu que j'avais en tête. Toutes les pages de l'application seront ainsi contenues dans une coquille (Shell) qui englobe à la fois la logique de navigation dans son ViewModel et le menu de navigation dans sa vue.

#### a.2. Le modèle MVVM

Le code de l'application mobile a été organisé selon le modèle MVVM : Modèle-Vue-Modèle de Vue, qui est composé des trois parties suivantes :

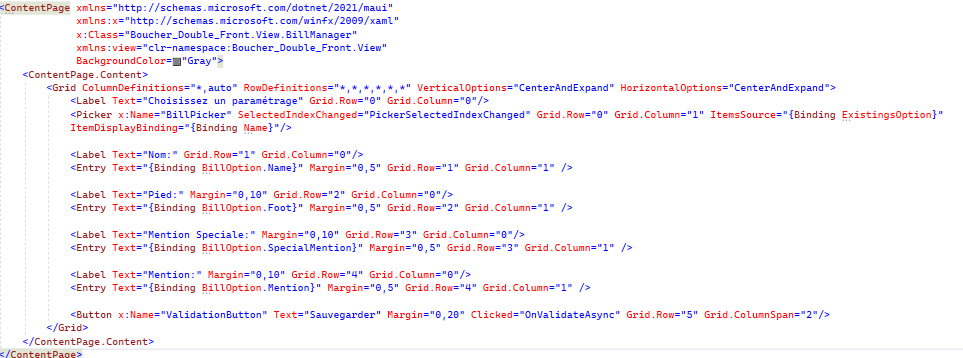
**Le Modèle** : Il définit les types qui représentent vos données métier. Cela englobe tout ce qui est nécessaire pour modéliser le domaine d'application principal, y compris souvent la logique d'application principale. Dans le cas de mon application, il s'agit du modèle commun mentionné précédemment, auquel s'ajoutent quelques classes utilitaires.

**La Vue** : La couche de vue définit l'interface utilisateur en utilisant le balisage XAML. Le balisage comprend des expressions de liaison de données qui établissent la connexion entre des composants spécifiques de l'interface utilisateur et différents membres du modèle et du modèle de vue. La vue comprend également un fichier de code, permettant de gérer les effets des différentes actions et de manipuler l'interface en conséquence. Par analogie, cela remplit un rôle similaire à ce qu'un script JavaScript ferait sur une page web.

**Le ViewModel** : La couche du modèle de vue fournit des cibles de liaison de données pour la vue. Dans de nombreux cas, le modèle de vue expose directement le modèle ou fournit des membres qui encapsulent des membres spécifiques du modèle. Le modèle de vue peut également définir des membres pour suivre les données liées à l'interface utilisateur, mais pas au modèle lui-même, comme l'ordre d'affichage d'une liste d'éléments. Le modèle de vue sert également de point d'intégration à d'autres services, tels que le code d'accès aux bases de données.

Pour illustrer ces différents concepts voici quelques exemples de codes :

Une View :



Et une partie du code associé, ici l’event listener du bouton de validation :



Et la ViewModel associée :



On voit dans ces deux exemples la séparation claire des rôles, bien que le bouton de validation manipule des données l’organisation de celles-ci et la logique de leurs récupération lui est totalement étranger, permettant une meilleure lisibilité et une délimitation plus claire des rôles de chaque composant du code de l’application.

#### a.3 Gestion de l’accès au backend

Comme mentionné précédemment il a été nécessaire, pour des questions de sécurité, d’inclure une authentification par token CSRF. Afin de confirmer l’identité de l’utilisateur de l’application celui-ci doit être inclut dans chacune des requêtes envoyées à l’API, en dehors de la requête de connexion.

La première étape est de récupérer le token lors de la connexion :

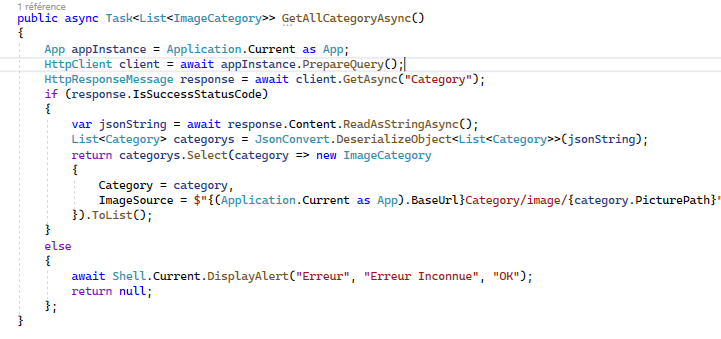


Une fois cette étape effectuée le token sera inclut dans toutes les requêtes suivantes permettant l’identification :



En exportant la préparation du client et l’ajout du token dans une fonction distincte je limite le risque d’oubli et simplifie la création des nombreuses requêtes nécessaires au bon fonctionnement de l’application. Cette fonction test également si l’utilisateur est toujours connecté à l’api, celle-ci supprimant les connexions automatiquement après un certain laps de temps, et l’y reconnecte le cas échéant.

#### a.4 Les appels à l’api

Comme précédemment mentionné, le fonctionnement de l'application mobile nécessite de nombreux appels à l'API décrite dans une section antérieure du rapport. Tous les appels à l'API en dehors de la connexion suivent une structure similaire. Voici une illustration de cette structure avec l'appel permettant d'obtenir toutes les catégories de produits enregistrées pour un magasin :

De nombreux appels à l'API sont également liés à des écouteurs d'événements (event listeners), généralement pour la création ou la mise à jour d'une entité. Voici, par exemple, l'écouteur d'événements associé au bouton de validation de la page de gestion d'une catégorie unique :

\*La création



* La mise à jour :



On vérifie lors de la mise à jour et de la création que le nom de la catégorie est valide. Comme la fonction « PrepareQuery » définit déjà l'URL commune à toutes les requêtes, il ne reste plus qu'à la compléter en fonction de l'entité concernée et du type de requête.

#### a.5 Gestion du style de l’application

Pour gérer le style d'une application, la méthode habituelle consiste à définir un dictionnaire de ressources qui va contenir différents styles et couleurs, référencés dans l'application via un système de clé-valeur.

Voici la définition des paires clé-valeur :

Le référencement d’une des clés, ici pour la couleur de l’arrière-plan :

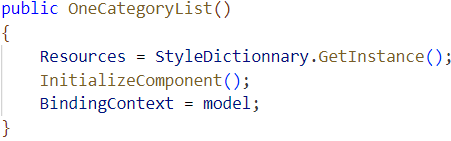


Afin de restreindre l'utilisation des ressources et de simplifier la modification du style, j'ai décidé de faire du dictionnaire de données un singleton. De cette manière, je pourrai centraliser aisément la gestion du style de l'application et diffuser toute modification de manière simple à l'ensemble de l'application.

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

Description générée automatiquement

Puis il reste à référencer le dictionnaire de ressource dans l’initialisation des pages de l’application :

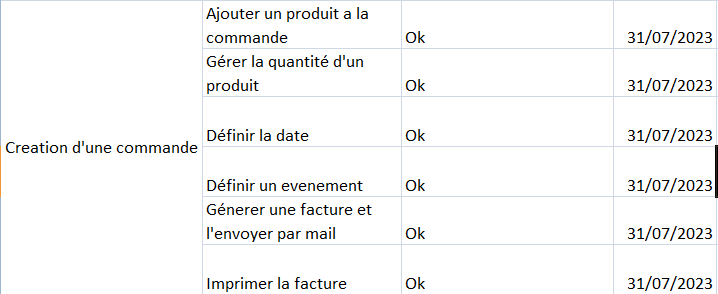


Afin de rendre l’application utilisable par différent commerce j’ai rendu les couleurs définit précédemment paramétrables :



# VIII. Jeu d’essai

La fonctionnalité centrale de l'application est la gestion des commandes des clients, notamment la création d'une commande. Voici le plan de test que j'avais prévu pour cette fonctionnalité :



Pour que la création d'une commande soit un succès, il faut :

**\* Pouvoir définir un client.**

**\* Définir l'événement associé à la commande, s'il y en a un.**

**\* Définir les produits et leurs quantités.**

**\* Créer une facture, l'envoyer par e-mail et éventuellement l'imprimer.**

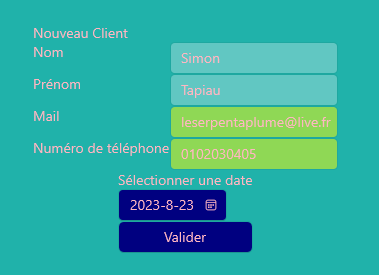
**\* Enregistrer la vente dans la base de données.**

**\* Définir les paramètres de la facture.**

**\* Définir les paramètres de l'e-mail envoyé au client.**

Données en entrée :

\*définir un client



\*Définir les produits et leurs quantités



\*Définir les paramètres de la facture

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

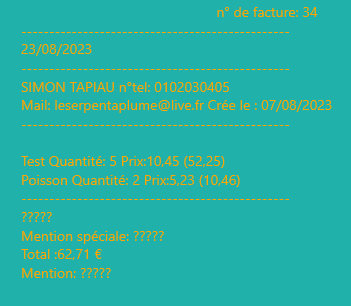
Description générée automatiquement

\*Définir les paramètres du mail

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

Description générée automatiquement

\*Envoyer la facture par mail



Données en sortie :

Mail :

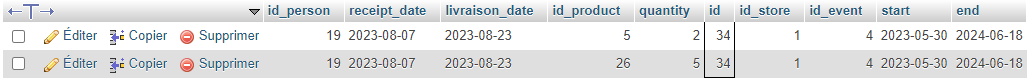
Une image contenant texte, capture d’écran, Police

Description générée automatiquement

Ici les lettres T correspondent à l’entête et au pied de mail choisit.

Base de données :

-La vue FullSellout



-La Table Product

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

Description générée automatiquement

-La vue client

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

Description générée automatiquement

-La table event



On a bien réussi à définir une vente, envoyer la facture par mail avec les paramètres sélectionné et enregistrer la vente dans la base de données.

# XI. Veille de sécurité

## 1.L’absence de support à long terme

Comme mentionné au début de ce document, j'ai commencé la création de mon projet à l'aide de Xamarin.Forms, un framework de C# permettant de développer des applications mobiles multiplateformes. Cependant, j'ai réalisé après avoir modifié certaines classes du modèle que certaines bibliothèques de mon modèle étaient incompatibles avec le framework. Cela était dû à la différence de versions de C# utilisées : Xamarin.Forms utilisait la version 4.17 tandis que mon modèle était écrit en version 6.0.

Après avoir cherché comment mettre à jour la version de C# utilisée par mon framework, j'ai compris que le problème était plus profond. En effet, j'ai découvert que Xamarin.Forms est un framework obsolète et que le support de Microsoft pour celui-ci prendrait fin en 2024. Cette absence de support à long terme pour la technologie utilisée met en péril la sécurité de mon application à long terme, rend le projet difficilement maintenable et expose potentiellement à des failles de sécurité permanentes découvertes ultérieurement. De plus, cela pourrait empêcher les mises à jour nécessaires si l'écart technologique avec le reste de l'environnement de l'application devient trop important.

Ces informations m'ont poussé à chercher une alternative vers laquelle migrer mon projet. Après quelques recherches, j'ai découvert que cette alternative est le framework .NET MAUI (Multi-platform App UI).

## 2.La mise à jour du projet

Pour mettre à jour mon projet, je me suis basé sur la documentation de .NET MAUI qui explique étape par étape comment réaliser la mise à jour. La majeure partie de cette mise à jour a consisté à identifier les références aux bibliothèques de Xamarin.Forms et à les remplacer par des bibliothèques .NET MAUI. Une fois cette étape réalisée correctement, il ne restait plus qu'à mettre à jour les bibliothèques utilisées pour qu'elles correspondent à la version plus moderne de C# que j'utilisais alors.

# X. Recherches nécessaires

À plusieurs moments durant mon stage, j'ai été confronté à des problèmes qui ont nécessité de mettre en pause mon travail afin d'effectuer des recherches. Le problème le plus significatif de ces situations a été la mise en place d'un logger pour le backend de mon application.

### 1.Origine du problème

L'utilisation d'un serveur web distant pour tester les appels à mon API a rapidement engendré un problème : l'accès aux messages d'erreur. Pour être efficace dans mon travail, il est devenu évident que je devais trouver un moyen d'accéder aux erreurs renvoyées par mon backend, afin de pouvoir rapidement et efficacement déterminer les causes des erreurs d'accès aux données. J'ai donc dû construire un journal (logger) pour stocker les messages d'erreur renvoyés et pouvoir les consulter.

### 2. Premières solutions

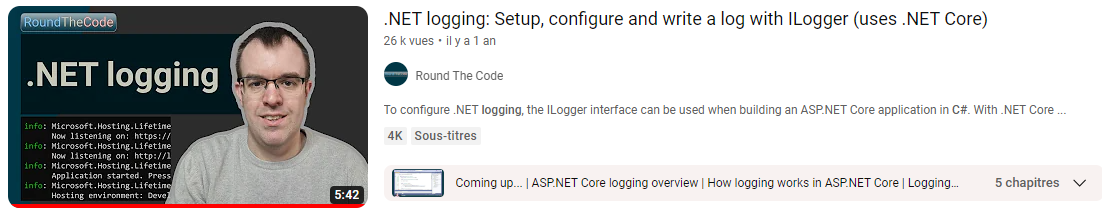
Ma première approche pour résoudre ce problème a été de rechercher comment créer des logs en C#. Cependant, je me suis rapidement heurté à une difficulté : la plupart des ressources sur ce sujet en C# concernaient la création de journaux dans la console, tandis que je m'intéressais à la création de journaux dans un fichier. J'ai donc dû affiner ma recherche en utilisant une requête plus spécifique : « Create a log in a file in c# ».

Voici les résultats de cette recherche :

Une image contenant texte, Appareils électroniques, capture d’écran, logiciel

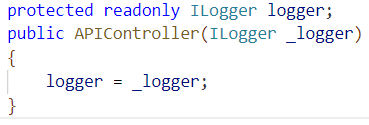
Description générée automatiquement

Parmi toutes les ressources que j’ai trouvé la plus explicite sur la création de se logger était une vidéo présentant étape par étape la création d’une implémentation concrète de l’interface de logging ILOGGER :



Cependant, l'une des étapes restait incompréhensible pour moi, même après avoir visionné et suivi ce tutoriel. À plusieurs reprises, l'interface ILOGGER était utilisée comme un paramètre de constructeur sans que cela ne soit jamais explicitement appelé, ce qui me laissait perplexe quant à la façon dont mon implémentation concrète était réellement utilisée en pratique :

Un des constructeurs :

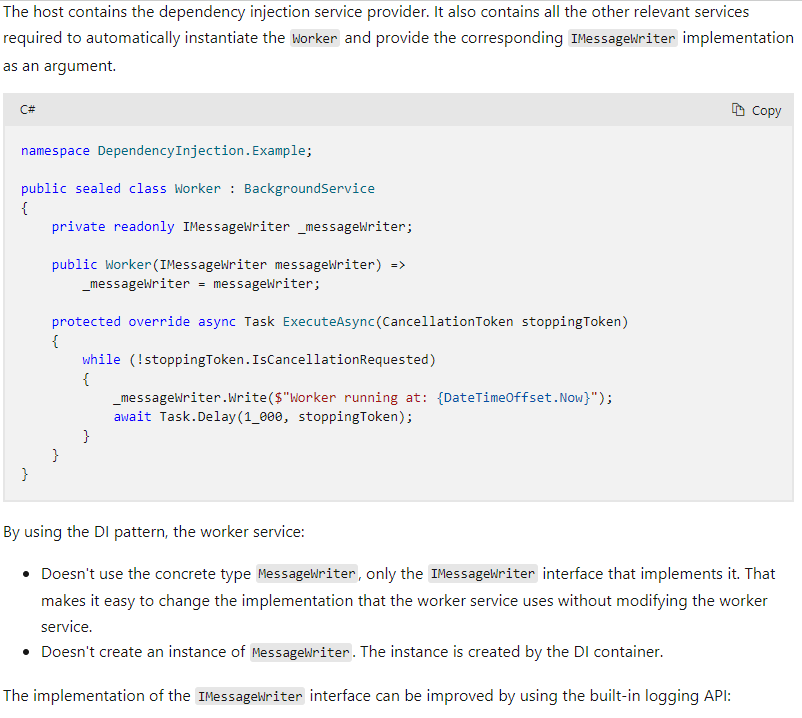


Pour comprendre le fonctionnement de ce que j'avais créé et l'utiliser correctement, j'ai dû me plonger dans l'un des concepts fondamentaux de .NET et de la programmation en général : l'injection de dépendances.

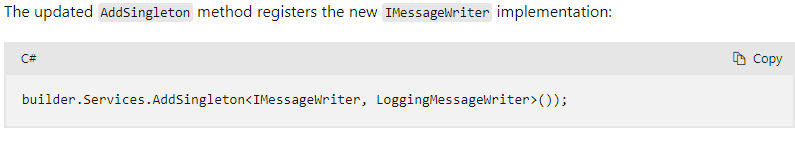
### 3.L’injection de dépendance

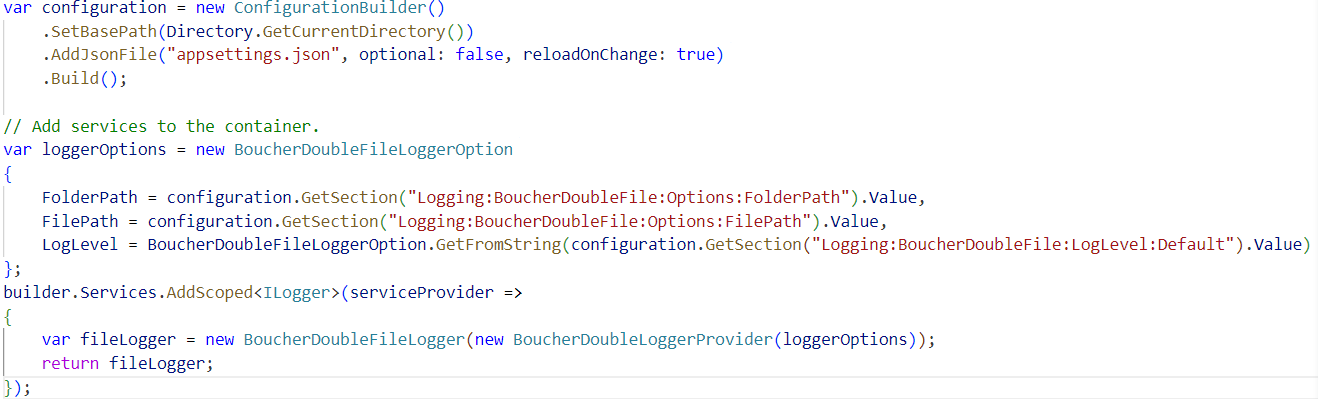
La première étape a été de comprendre à quelle particularité du langage C# j'étais confronté. Pour cela, j'ai sollicité l'intelligence artificielle ChatGPT en lui soumettant un extrait de code concerné et en lui demandant le nom de cette « technique » de programmation. Cela m'a permis de découvrir le nom de ce que je cherchais à comprendre : l'injection de dépendances.

Une fois que j'ai su ce que je devais rechercher, je me suis tourné vers la documentation de .NET, ce qui m'a aidé à mieux comprendre le fonctionnement de ce que j'avais créé.

Voici un extrait de la documentation de .NET sur l'injection de dépendances :

Je comprenais alors que le Framework instancie automatiquement une implémentation concrète de l’interface ILOGGER que je lui aurais précisé, pour faire cela la réponse se trouvait une fois encore dans la documentation de .NET :



A l’aide de ces informations j’ai pu correctement définir mon logger dans le fichier d’initialisation de mon API : 

J’avais maintenant un logger fonctionnel dont l’ajout dans une nouvelle classe de mon nécessitait peu de code et était cohérent dans toute mon application.

# XI. Conclusion

Ce stage m'a offert l'occasion pour la première fois de faire face à des problématiques liées au développement et à la conception professionnelle d'applications. J'ai pu découvrir le déploiement d'un backend, travailler pour la première fois avec un framework, et me familiariser avec un nouveau langage. De plus, j'ai eu l'opportunité de travailler sur une application mobile pour la première fois, ce qui a élargi ma compréhension du domaine et m'a permis de confronter des idées que j'avais préconçues sur le fonctionnement des applications web, notamment en ce qui concerne la notion de session.

En outre, j'ai eu l'occasion de travailler avec un client, ce qui m'a permis de réellement expérimenter l'importance de la conception, notamment du maquettage, pour éviter de multiples révisions dues à des malentendus. Ce stage a confirmé mon intérêt pour ce domaine et m'a permis d'explorer de nombreux aspects passionnants que je n'avais pas encore eu l'occasion de découvrir pendant ma formation. Je ressors de cette expérience avec une forte envie d'approfondir les concepts que j'ai déjà abordés et d'apprendre de nouveaux concepts, notamment ceux de niveau plus avancé.