

Compte rendu professionnel

FabLabea

Base de données et application S.A.G.A

Sommaire

Mise en situation :	3
À propo de FabLabea :	3
Mon rôle :	3
Les systèmes applicatifs :	3
Outils et logiciels :	3
Organisation :	3
Organisation du projet :	4
L'élaboration du cahier des charges :	4
Contrôle et présentation des productions :	4
La base de données :	5
Les préparatifs :	5
Génération du SCD et du scripte SQL:	5
L'intégration au SGBDR et création des profils utilisateurs :	6
Mise en place des Triggers:	8
Correction et modification :	11
Modification pendant le développement :	11
Le maquetage de l'application :	12
Les préparatifs - Cas utilisations :	12
Le maquetage :	14
Présentation orale de la maquette :	15
Le développement de l'application :	16
Les préparatifs :	16
Le développement - Création du package DAL:	16
Le développement - Mise en place des classes métiers:	18
Le développement - Création des formulaires et UserControl:	18
Le développement - Mise en place de l'authentification:	20
Le développement - Gestions de adhérents - l'ajout d'un adhérent:	22
Le développement - Gestions de adhérents - Recherche d'un adhérent:	24
Le développement - Gestions d' adhérents - Nouvelle Prestation:	28
Le développement - Gestions d' adhérents - Consulter l'historique:	30
Le développement - Modification d'un consommables :	32
Le développement - Le déploiement :	34
Le développement - Présentation de l'application devant l'assemblée :	34
Le développement - Conclusion du développement :	34
Conclusion:	35
Projet d'évolution :	35
Gain de compétences et remerciement :	35

Mise en situation :

À propos de FabLabea :

J'ai été recruté au sein du FabLabea d'Hasparren en tant que stagiaire pour ma deuxième année de BTS SIO SLAM, pour une durée effective de 7 semaines.

FabLabea est une association qui a pour projet de mettre à disposition des outils technologiques pour tous, aussi bien pour des particuliers que des professionnels. Le principe est que n'importe qui puisse venir au FabLab avec une idée de projet ou un besoin de production ; être accompagné dans son projet, modélisation 3D, impression 3D, calcul des coûts, etc... ; et ressortir avec ce dont il avait besoin. Ça peut aller de la réparation d'une poignée de porte d'un particulier à du maquettage de prototype pour un professionnel.

Mon rôle :

Le FabLabea n'est pas encore ouvert et les lieux sont encore en travaux. Par anticipation de la prochaine ouverture qui aura lieu en Mars 2022, il faut d'ores et déjà commencer le développement du système informatique et plus particulièrement celui de l'application principale qui permettra de gérer les adhérents.

Les systèmes applicatifs :

Ma contribution au sein du système applicatif s'arrête au développement de l'application principale et tout ce dont elle a besoin pour fonctionner. Mon projet va être de développer une base de données ainsi que le système applicatif qui communiquera avec cette dernière. A terme, une application Android et un portail Web seront implémentés pour les adhérents, afin qu'ils puissent à distance accéder à leurs données personnelles, les modifier, ainsi que recharger leur compte en crédits.

Outils et logiciels :

La base de données sera faite sous MySQL avec PHPMyAdmin et l'application sera développée sur l'IDE Visual Studio Community en C#.

WinDesign sera également utilisé pour faire le schéma conceptuel de données (SCD) ainsi que Pencil pour réaliser le maquettage de l'application de gestion des adhérents.

Organisation :

Afin de m'organiser au mieux j'utiliserai un tableur Excel comme planning ainsi qu'un Trello pour garder une trace de ce qui a été fait et surtout pour me fixer des dates limite de réalisation. J'ai également un développeur qui pourra m'aider ponctuellement sur le code de l'application.

Organisation du projet :

L'élaboration du cahier des charges :

L'élaboration du cahier des charges s'est faite à l'oral lors d'une première réunion composée du président de l'association, du trésorier (mon responsable pendant ce stage) et de moi-même. Nous avons échangé sur leurs besoins ce qui m'a permis de rédiger une synthèse de ce qu'ils recherchaient.

Cette synthèse fait mention de 3 modules distincts : une base de données, une application client lourd de gestion des adhérents (S.A.G.A) ainsi que d'une application mobile sous Android pour un accès plus pratique pour les adhérents à leur compte client. Je ne travaillerai que sur les deux premiers modules lors de ce stage, à savoir sur la base de données ainsi que sur la conception de l'application S.A.G.A.

Contrôle et présentation des productions :

Afin de contrôler mon avancée, des présentations régulières de mes productions seront faites aux principaux intéressés. Le planning fait sous Excel me permet de prévoir les points à chaque changement de tâche qui nécessitent une présentation et/ou un contrôle.

Voici un extrait de ce planning :

Planning prévisionnel						
	Légende	La base de donnée	Maquettage	Developpement	Jours de repos	
	Code couleur	1 semaine (5 jours)	2 semaines (10 jours) Tache 1 Tableau cas utilisateur: 5 jours Tache 2 Maquette : 5 jours	4 semaines (21 jours)	14 jours	
ine 1		mercredi 15 septembre 2021 - Définir les besoins de la BDD	jeudi 16 septembre 2021 - Création du SCD/MCD	vendredi 17 septembre 2021 - Intégration du scripte SQL - Création des utilisateurs	samedi 18 septembre 2021 - Création des triggers	dimanche 19 septembre 2021
	lundi 20 septembre 2021	mardi 21 septembre 2021 -Création des triggers	mercredi 22 septembre 2021 -Schéma des cas d'utilisateurs	jeudi 23 septembre 2021 - Création des tableaux de cas d'utilisation	vendredi 24 septembre 2021 -Suite création des tableaux de cas d'utilisation	samedi 25 septembre 2021 - Suite création des tableaux de cas
						dimanche 26 septembre 2021

La base de données :

Les préparatifs :

L'objectif était d'avoir une base de données fonctionnelle prête à être remplie est exportée en une semaine, terme auquel on ferait un point dessus.

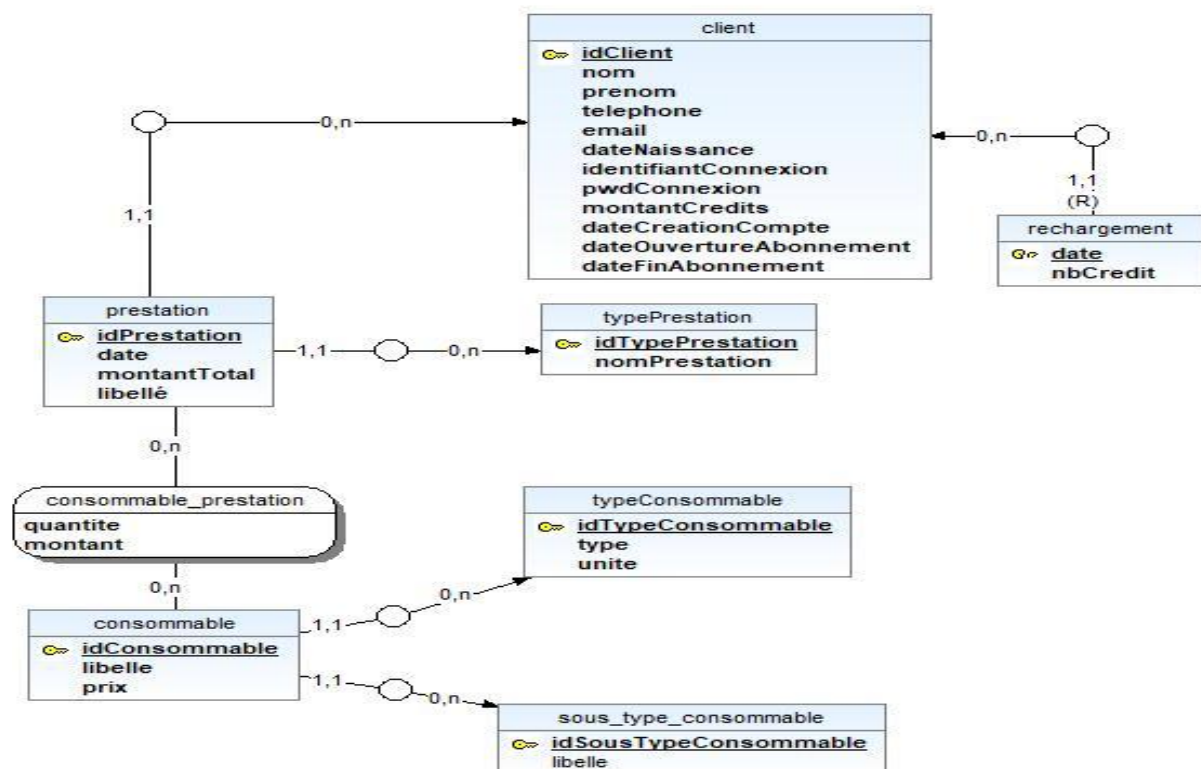
Avant toute chose, il a fallu déterminer les besoins de la base de données. Pour cela nous avons fait plusieurs réunions afin de bien déterminer les besoins du projet global du système applicatif pour en déduire les besoins de la base de données.

À partir de ces réunions, j'ai pu produire une synthèse de ce que la base de données à besoins.

Génération du SCD et du scripte SQL :

J'utilise WinDesign pour réaliser le SCD (Schéma Conceptuel de Données). WinDesign est un logiciel permettant de faire de la modélisation à but entreprise de tout genre, pour ma part je ne m'intéresse qu'à la partie base de données et plus précisément à la partie MCD.

Voici le SCD/MCD :



Ce schéma reflète les mécanismes et interactions de la base de données avec chacune des entités.

Pour générer le script avec les outils fournis par WinDesign, il faut d'abord générer un Modèle Logique de Donnée (MLD) en veillant bien à sélectionner le SGBD MySQL lors de sa création.

À partir de ce MLD, je génère le scripte SQL qui servira à importer cette base de données dans un SGBDR.

L'intégration au SGBDR et création des profils utilisateurs :

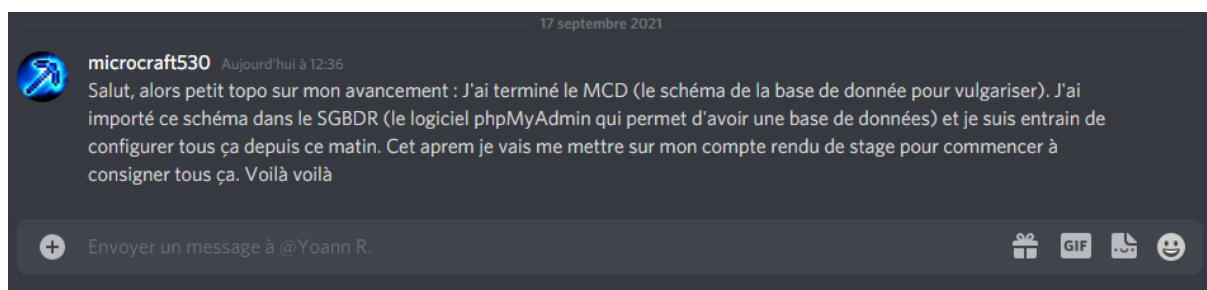
Avant toute chose, je mets à jour mon Trello sur mon avancé, et je me crée une ToDoList détaillé de ce que je dois produire sur la base de données



Une fois le script SQL terminé, je l'importe dans PHPMyAdmin par l'onglet prévu à cet effet en veillant bien à choisir le jeu de caractère utf-8.

Nous retrouvons bien toutes nos tables qui reflètent nos entités que nous avons dans le MCD

J'en profite pour avertir mon responsable de mon avancée



La première chose à faire est de créer les droits d'accès à la base de données. Je commence par créer un profil Opérateur, au sein du FabLab, pour respecter au mieux le principe de moindre privilège, il y aura deux types de profils :

- Le profil Opérateur qui pourra créer et modifier les données client ainsi que d'enregistrer des prestations, ainsi que d'uniquement lire les données plus sensibles à travers l'application qui sera en place.
- Le profil Administrateur qui pourra quant à lui modifier des données plus sensibles comme les tarifs des produits ou encore pouvoir ajouter ou supprimer ces derniers.

Je créer tout d'abord un compte au sein de PHPMyAdmin :

Ajouter un compte d'utilisateur

Informations pour la connexion

Nom d'utilisateur : Saisir une valeur:

Nom d'hôte : Tout hôte 

Mot de passe : Saisir une valeur: Strength: Fort

Saisir à nouveau :

Extension d'authentification


Générer un mot de passe:


Base de données pour ce compte d'utilisateur

☐ Créer une base portant son nom et donner à cet utilisateur tous les privilèges sur cette base.

Après quoi je lui confère des privilèges uniquement sur la base de données fablabsage (notre base de données). Par la suite, je lui confère sur cette BDD uniquement le droit d'usage sur seulement quelques tables, les seules tables qui lui sont pertinente pour son rôle.

Privilèges spécifiques à une base de données

Base de données	Privilèges	« Grant »	Privilèges spécifiques à une table	Action
fablabsaga	USAGE	Non	Oui	 Éditer les privilèges

Ajouter des privilèges sur ces bases de données : 

Privilèges spécifiques à une table					
Table	Privilèges	« Grant »	Privilèges de colonnes	Action	
client	SELECT, INSERT, UPDATE, DELETE, REFERENCES	Non	Non	Éditer les privilèges	Révoquer
consommable_prestation	SELECT, INSERT, UPDATE, DELETE, REFERENCES	Non	Non	Éditer les privilèges	Révoquer
prestation	SELECT, INSERT, UPDATE, DELETE, REFERENCES	Non	Non	Éditer les privilèges	Révoquer
rechargement	SELECT, INSERT, UPDATE, DELETE, REFERENCES	Non	Non	Éditer les privilèges	Révoquer

A présent je fais la même chose pour l'administrateur, je lui autorise toujours uniquement des droits sur fablabsaga à ceci près que ces derniers, ont tous les privilèges sur l'ensemble de la base de données et pas seulement sur certaines tables.

Privilèges spécifiques à une base de données				
Base de données	Privilèges	« Grant »	Privilèges spécifiques à une table	Action
fablabsaga	ALL PRIVILEGES	Non	Non	Éditer les privilèges Révoquer

mysql
sys

À présent je me connecte avec le compte administrateur pour vérifier que les droits ont correctement été appliqués. À partir de là je n'utiliserai plus que ce compte pour configurer le reste de la BDD.

Mise en place des Triggers :

Afin d'automatiser certaines tâches et de rendre plus fluide et plus faciles la création et la maintenance des futurs systèmes applicatifs qui utiliserons cette base de données, la décision a été prise d'intégrer le maximum de triggers entre les tables. Comme déjà cité, les triggers ont des avantages multiples en plus de pouvoir offrir des possibilités très intéressantes.

Ici je vais automatiser la mise à jour des crédits que possède un client dès qu'un rechargement lui est appliqué. Autrement dit du point de vue du SGBDR, dès qu'une ligne dans rechargement est créée, il va lire cette ligne, récupérer le client qui y est affilié, récupérer le montant de crédit que possède ce client pour en faire la somme avec le montant inscrit dans la nouvelle ligne. Voici l'architecture de ce trigger :

Exécuter une ou des requêtes SQL sur la base de données « fablabsaga »:	
1	DELIMITER
2	create TRIGGER after_insert_rechargement after INSERT
3	on rechargement FOR EACH ROW
4	BEGIN
5	UPDATE client SET montantCredits = montantCredits + new.nbCredit WHERE IDCLIENT = new.idClient;
6	END
7	DELIMITER ;

Ce trigger est très simple, étant donné que l'on veut effectuer cette démarche lors de l'ajout d'une ligne, il s'agit d'un trigger sur l'INSERT, le AFTER signifie que l'on veut exécuter le trigger après l'ajout de la ligne dans rechargement. Après quoi on précise sur quelle table ce

trigger doit écouter les actions, ici il s'agit de rechargement. Tout ce qui se trouve après le BEGIN jusqu'au END sont les instructions que le trigger doit exécuter.

Dans les instructions nous avons qu'une seule requête UPDATE sur la table client, le "new." devant nbCredit correspond à la nouvelle valeur de la ligne créer dans rechargement, idem pour le client.

A présent, il s'agit de faire deux autres triggers sur les événements UPDATE (si on met à jour une ligne rechargement) et DELETE (si on supprime une ligne).

Exécuter une ou des requêtes SQL sur la table « fablabsaga.rechargement »:

```

1 DELIMITER |
2 create TRIGGER after_update_rechargement after update
3 on rechargement FOR EACH ROW
4 BEGIN
5 UPDATE client SET montantCredits = montantCredits - old.nbCredit WHERE IDCLIENT = old.idClient;
6 UPDATE client SET montantCredits = montantCredits + new.nbCredit WHERE IDCLIENT = new.idClient;
7 END |
8 DELIMITER ;

```

SELECT* SELECT INSERT UPDATE DELETE Effacer Format Récupérer la requête auto-se

Exécuter une ou des requêtes SQL sur la table « fablabsaga.rechargement »:

```

1 DELIMITER |
2 create TRIGGER after_delete_rechargement after delete
3 on rechargement FOR EACH ROW
4 BEGIN
5 UPDATE client SET montantCredits = montantCredits - old.nbCredit WHERE IDCLIENT = old.idClient;
6 END |
7 DELIMITER ;

```

Pour le trigger UPDATE, je soustraire la précédente somme de la ligne avec le "old." ; puis je réexécute la même chose que pour l'INSERT.

Pour le DELETE, il suffit de soustraire la somme.

A présent je mets en place la même chose pour l'exacte opposé, c'est-à-dire lors de la création d'une ligne de Consommable_prestation, mettre à jour le solde du client concerné.

Exécuter une ou des requêtes SQL sur la table « fablabsaga.consommable_prestation »:

```

1 DELIMITER |
2 CREATE TRIGGER `before_insert_consommable_prestation` BEFORE INSERT ON `consommable_prestation` FOR EACH ROW BEGIN
3 UPDATE prestation SET montanttotal = montanttotal + (new.quantite * (select prix from consommable where IDCONSOmmABLE = new.idconsommable)) WHERE IDPRESTATION = new.idprestation;
4 update client set montantCredits = montantcredits - (new.quantite * (select prix from consommable where idconsommable = new.idconsommable)) where (SELECT DISTINCT prestation.idclient from prestation inner join consommable_prestation on prestation.IDPRESTATION = consommable_prestation.IDPRESTATION where consommable_prestation.IDPRESTATION = new.idprestation);
5 SET new.montant = new.quantite * (select prix from consommable where idconsommable = new.idconsommable);
6 END |
7 DELIMITER ;

```

Ici le principe est le même que pour la table rechargement, à la seule différence, que les commandes SQL sont un peu plus complexes avec des sous requêtes.
Pour les triggers de l'UPDATE et du DELETE, les structures sont identiques à celle du rechargement.

Extrait du trigger sur l'Update :

Exécuter une ou des requêtes SQL sur la table « fablabsaga.consomnable_prestation »:

```

1
2 DELIMITER |
3 CREATE TRIGGER `before_update_consomnable_prestation` BEFORE UPDATE ON `consomnable_prestation` FOR EACH ROW BEGIN
4 UPDATE prestation SET montanttotal = montanttotal - (old.quantite * (select prix from consomnable where idconsomnable = old.idconsomnable)) WHERE
  IDPRESTATION = old.idprestation;
5 update client set montantCredits = montantCredits + (old.quantite * (select prix from consomnable where idconsomnable = old.idconsomnable)) where
  (SELECT DISTINCT prestation.idclient from prestation inner join consomnable_prestation on prestation.IDPRESTATION =
  consomnable_prestation.IDPRESTATION where consomnable_prestation.IDPRESTATION = old.idprestation);
6 UPDATE prestation SET montanttotal = montanttotal + (new.quantite * (select prix from consomnable where idconsomnable = new.idconsomnable)) WHERE
  IDPRESTATION = new.idprestation;
7 update client set montantCredits = montantCredits - (new.quantite * (select prix from consomnable where idconsomnable = new.idconsomnable)) where
  (SELECT DISTINCT prestation.idclient from prestation inner join consomnable_prestation on prestation.IDPRESTATION =
  consomnable_prestation.IDPRESTATION where consomnable_prestation.IDPRESTATION = new.idprestation);
8 SET new.montant = new.quantite * (select prix from consomnable where idconsomnable = new.idconsomnable);
9 END|
10 DELIMITER ;

```

A présent, il ne nous reste qu'un trigger à faire, celui de l'adhérent. Pour précisions, lorsqu'un adhérent est créé, on part du principe qu'il souscrit à un abonnement automatiquement. Ce dernier (l'abonnement) dure 1 an, il y a donc une date de commencement et une date de fin. Également chaque adhérent à une date de création de son compte (cette donnée est utile à des fins analytique).

L'identifiant de connexion doit être également créé dynamiquement avec le nom, le prénom et une partie de la date de naissance afin de s'assurer qu'il soit unique. Le mot de passe quant à lui est par défaut identique à l'identifiant. L'adhérent le changera lors de sa première connexion.

Lors de la création d'un adhérent, le système applicatif, n'enverra aucune date et aucun identifiant, par conséquent les identifiants et toutes les dates seront gérés par le trigger.

Exécuter une ou des requêtes SQL sur la table « fablabsaga.adherent »:

```

1 DELIMITER $$
2 CREATE TRIGGER `before_insert_adherent` BEFORE INSERT ON `adherent` FOR EACH ROW BEGIN
3 set new.identifiantconnexion = concat(new.nom, new.prenom, month(new.DATENAISSANCE), year(new.DATENAISSANCE));
4 IF new.pwdconnexion IS null
5 THEN set new.pwdconnexion = SHA2(new.identifiantconnexion, '256');
6 END IF;
7 set new.date_creation_compte = now();
8 set new.date_ouverture_abonnement = now();
9 set new.DATE_FIN_ABONNEMENT = new.DATE_OUVERTURE_ABONNEMENT + interval '1' year;
10 END
11 $$
12 DELIMITER ;

```

SELECT * SELECT INSERT UPDATE DELETE Effacer Format Récupérer la requête auto-sauvegardée

Notez que le mot de passe est identique à l'identifiant uniquement si ce dernier est NULL. Et cela même si nous savons que l'application n'enverra jamais de mot de passe. La raison est que cet IF anticipe la future application mobile, où les adhérents pourront créer un compte depuis leur smartphone et renseigneront directement leur mot de passe. Le tout est bien évidemment haché avec un algorithme de hachage en 256 bits (SHA2_256).

La base de données étant terminée, je mets à jour le Trello et j'organise un entretien pour présenter mon avancée.

Correction et modification :

Après avoir terminé la base de données dans les temps, un point a été fait avec mon responsable. S'il était globalement très satisfait de ma production, certains points demandaient à être revus.

- Initialement on ne mémorise pas les date de création d'adhérent ainsi que celles de l'abonnement, il fallait donc les rajouter
- Également l'entité sous_type_consommable n'existait pas non plus, il m'a été demandé de le rajouter pour faciliter les recherches de consommable.
- Certains champs, ainsi que certaines tables ont dû être renommés. Par exemple, la table adhérent s'appelait initialement client ou encore les consommables s'appelaient matériels.

Tous ces points ont été revus en 2 jours. Les changements ont également été mis à jour dynamiquement dans ce compte rendu.

Modification pendant le développement :

Également suite au développement de l'application, je me suis rendu compte que les droits accordés à l'opérateur étaient erronés. Les droits déjà octroyés sont corrects, toutefois il manque les droits de SELECT sûr toutes les autres tables. Ces droits de SELECT vont permettre à l'opérateur de pouvoir choisi parmi les différents consommables qui seront proposé dans l'application (étant donné que l'application utilisera les identifiants rentrés pour générer la chaîne de connexion, il est impératif qu'il puisse avoir accès au SELECT pour remplir les ComboBox).

Le maquetage de l'application :

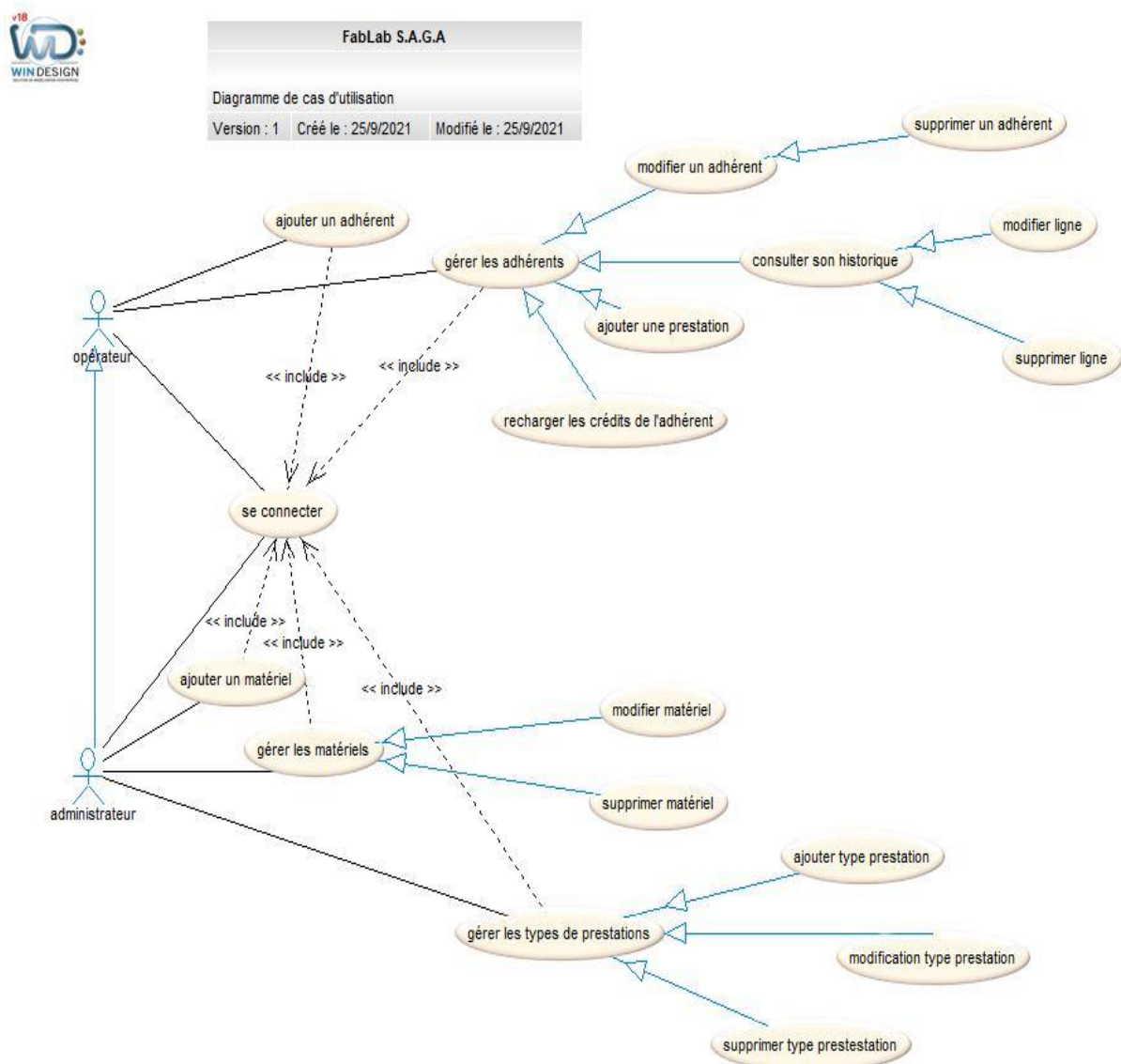
Pour mettre en place une maquette qui servira de base pour le développement de l'application, un délai de 2 semaines m'a été suggéré.

Les préparatifs - Cas utilisations :

Avant de commencer toute production, il faut commencer par la base, le schéma de cas utilisateur. Ce schéma permet de correctement cibler les droits des différents acteurs ainsi que définir la portée de chacun dans les différentes actions possibles. De plus, ce schéma permet de pouvoir appliquer au mieux le principe de moindres privilèges.

Ce schéma se fait à partir du cahier des charges qui a été élaboré à l'oral lors d'une réunion.

Voici ce schéma qui a été réalisé sous WinDesign



A partir de ce schéma, je créer les cas d'utilisations des procédures pour chaque action que peut mener chaque acteur. Dans chaque procédure, je précise l'auteur concerné, l'événement déclencheur, l'intérêt de cette procédure, la précondition pour que celle-ci

s'exécute, puis le scénario nominal avec éventuellement le scénario alternatif en détaillant point par point chaque étape.

Voici un extrait de ces cas d'utilisation

Cas d'utilisation	Authentification
Acteur	Opérateur/Administrateur
Événement déclencheur	Exécution du démarrage de l'application
Intérêt	Se connecter à l'application afin d'avoir les privilèges requis
Précondition	Aucune
Scénario nominal	1. Le système affiche une boîte de dialogue invitant l'utilisateur à saisir ses identifiants. 2. L'utilisateur saisie ses identifiants puis clique sur « Connexion »
Scénario alternatif	2a. les identifiants sont erronés ; retour au point 2

Cas d'utilisation	Menu
Acteur	Opérateur/Administrateur
Événement déclencheur	Clique sur connexion depuis le portail d'authentification
Intérêt	Avoir accès aux différentes fonctionnalités des opérations courantes
Précondition	S'être authentifié
Scénario nominal	1. Le système affiche une fenêtre présentant la possibilité de se déconnecter, d'ajouter un adhérent, rechercher un adhérent où d'accéder aux fonctionnalités avancées de l'administrateur. 2. L'utilisateur peut se déconnecter : retour au cas

Afin de ne pas me perdre dans la quantité de cas d'utilisation à créer, j'utilise la ToDoList sur Trello. Cela me permet également de pouvoir visualiser mon avancée.

The screenshot shows a Trello board titled "Cas d'utilisation". At the top, there are buttons for "Masquer les tâches cochées" and "Supprimer". A progress bar indicates 27% completion. The list of tasks includes:

- ☒ Authentification
- ☒ Menu
- ☒ Ajout d'un adhérent
- ☒ Rechercher un adhérent
- ☐ Consulter un adhérent
- ☐ Modifier un adhérent
- ☐ Nouvelle prestation
- ☐ Historique
- ☐ Modifier prestation
- ☐ Recharger crédits
- ☐ Espace administrateur

On the right side, there are sections for "PIECE JOINTE" (Image de couvert...), "CHAMPS PERSONNALISÉS" (Ajoutez des menus déroulants, des champs de texte, des dates et plus encore à vos cartes. Commencer l'essai gratuit), "POWER-UPS" (+ Ajouter des Power...), "AUTOMATISATION" (+ Ajouter un bouton), and "ACTIONS" (→ Déplacer, Copier).

Une fois les cas d'utilisations terminés, un rapide contrôle a été fait avec mon responsable qui m'a confirmé que ma vision des choses correspondait aux besoins et que je pouvais commencer la maquette.

Le maquettage :

En possession du cas utilisateur définitif, je peux commencer la maquette. La maquette est réalisée sous Pencil, un outil très polyvalent dans la conception de maquettes dynamiques permettant de simuler la navigation entre les pages.

Une semaine s'est déjà écoulée, il ne me reste plus qu'une semaine pour terminer cette étape du projet.

Aucun style graphique ne m'a été imposé, je suis parti sur un style sobre avec des couleurs douces pas trop prononcé en utilisant au maximum des icônes au style graphique simplifié très arrondi pour faire les boutons. Afin de coller au mieux avec les systèmes d'exploitation moderne qui prônent un mélange arrondi et rigide (Windows 11), les boutons qui ne sont pas des icônes ainsi que les champs de saisie, prennent une architecture rigide en angle à 90°.

Voici quelques exemples du rendu de la maquette



Nouvelle prestation | FabLab S.A.G.A

Connecté en tant que : Opérateur Quentin Adhérent : Jack SPARROW

Type de prestation : Impression 3D

Ajouter une ligne

Choisir un type de consommable

Type consommable : Filament Sous type consommable : PLA

Consommable : PLA Bois (15%) PLA Classique (Blanc) PLA Classique (Noir)

Libellé :

Quantité : 300 Gramme

Coût en crédits : 90

Détail des prestations	Quantité	Unité	Prix unitaire	Montant (Crédits)	Libellé
PLA Classique (Blanc)	450	Gramme	0,05	22,5	Pièce de 8

Coût total en crédits : 22.5

Présentation orale de la maquette :

Une fois la maquette terminée je l'ai présentée devant les principaux intéressés lors d'une réunion sur vidéo projecteur en expliquant chaque fonctionnalité et en démontrant l'ergonomie de la navigabilité de la future application. Ils ont été très satisfaits du résultat mais ont néanmoins citer quelques détails à modifier.

Le fait de pouvoir choisir un consommable par également son sous type était l'une de leurs idées par exemple. Les autres modifications étaient plus d'ordre ergonomique comme déplacer un bouton, ce genre de chose.

Après les quelques modifications appliquées, la maquette est prête à servir de support pour le développement de l'application. Cette maquette a pu être quasiment réalisée dans les temps, seuls deux jours supplémentaires ont été nécessaires pour terminer le rendu ainsi que les modifications suite au contrôle.

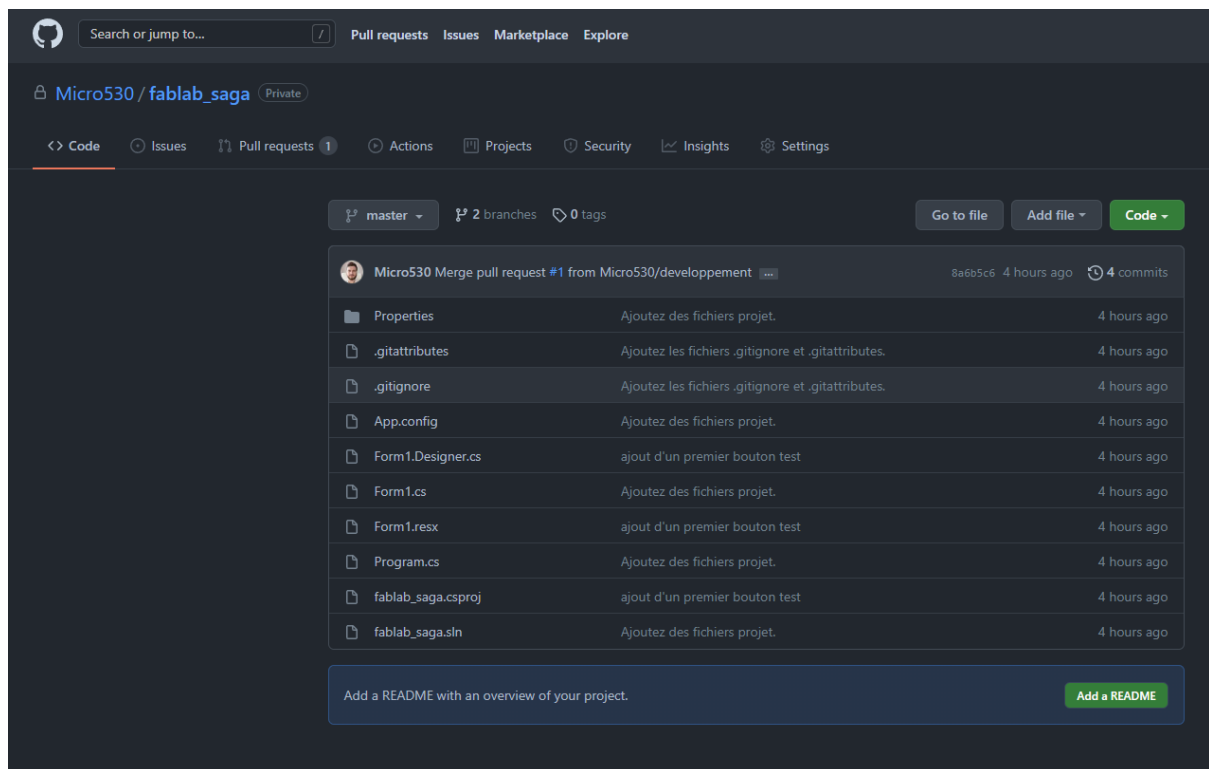
Le développement de l'application :

Pour le développement de l'application et son exportation, je me suis fixé un délai de 4 semaines. J'avais également à mes côtés un développeur plus expérimenté qui m'a ponctuellement aidé et conseillé.

Les préparatifs :

Avant de commencer tout développement, je commence par créer un dépôt privé GitHub pour assurer le versionning du projet et également pour permettre et faciliter le travail collaboratif avec les autres développeurs.

À partir de là, je crée un nouveau projet dans Visual Studio en MVC et je fais mon premier commit and push. Puis je crée deux branches, une branche développement qui servira au développement "général" et une autre, développementFormulaire, qui servira à isoler les développeurs travaillant sur la création des formulaires.



Le développement - Création du package DAL :

Afin de respecter au mieux le design pattern MVC, la Classe Accès données est placée dans un dossier DAL (Data Access Layer) qui signifie "La couche d'accès données". Le principe du DAL est qu'il fait abstraction du type de support utilisé pour avoir accès aux données (SQL, fichier XML, ...), ce qui le rend très polyvalent. Dans notre cas, il s'agit du SQL.

La classe Accès données est vide pour l'instant, et accueillera au fur et à mesure des besoins, les différentes requêtes SQL nécessaires. Cette classe Accès données va devoir communiquer avec une autre classe : la classe ConnexionBDD. Cette dernière est celle qui va gérer la connexion avec la base de données et qui va assurer l'envoi des requêtes et la réception des réponses. Cette classe ConnexionBDD à l'avantage de ne presque pas

changer d'une application à l'autre, en effet, grâce à la polyvalence du MVC, cette classe est totalement indépendante du reste du programme.

Cette classe a la particularité d'avoir son constructeur en privé (Singleton), ce qui lui permet de ne pas être construit de l'extérieur. Ici l'intérêt est que la classe Accès données va simplement créer un objet ConnexionBDD en appelant la méthode GetInstance, avec comme paramètre, la chaîne de connexion.

Extrait de la classe Accès données

```
/// <summary>
/// permet de construire le chaîne de connexion avec les identifiants de connexion reçu en paramètres
/// </summary>
/// <param name="identifiant"></param>
/// <param name="pwd"></param>
1 référence
public static bool Authentification(string identifiant, string pwd)
{
    chaîneConnexion = "Server=" + adresseServeur + "; Port=" + port + "; user id=" + identifiant + ";password=" + pwd + "; " +
        "persistsecurityinfo=True; database=" + nomBDD + ";SSL Mode=None";
    ConnexionBDD bdd = ConnexionBDD.GetInstance(chaîneConnexion);
    return bdd.IsRunning();
}
```

Ensuite, ConnexionBDD vérifie si une instance n'a pas déjà été créée en utilisant la variable locale static isRunning, qui est initialisée à True lors de la création de l'instance. Si cette instance n'existe pas, le constructeur privé est appelé en envoyant la chaîne de connexion. Dans tous les cas, cette méthode GetInstance retourne l'instance de connexion.

```
/// <summary>
/// création de l'instance si elle n'est pas déjà existante
/// </summary>
/// <param name="chaîneConnection">la chaîne de connexion à la base de données</param>
/// <returns>l'instance de la connexion</returns>
1 référence
public static ConnexionBDD getInstance(string chaîneConnection)
{
    // contrôle si l'instance existe déjà
    if (isRunning is false )
    {
        instance = new ConnexionBDD(chaîneConnection);
    }
    // la méthode retourne l'instance créée ou celle déjà existante
    return instance;
}
```

```
/// <summary>
/// constructeur privé appelé uniquement par GetInstance
/// </summary>
/// <param name="chaîneConnection">la chaîne de connexion à la base de données</param>
1 référence
private ConnexionBDD(string chaîneConnection)
{
    try
    {
        this.connection = new MySqlConnection(chaîneConnection);
        this.connection.Open();
        Console.WriteLine("Connexion réussie à la base de données");
        isRunning = true;
    }
    catch (Exception e)
    {
        Console.WriteLine(e.Message);
        MessageBox.Show("Impossible de vous connecter à la base de données, vérifiez vos identifiants.\nSi le problème persiste, " +
            "contactez un administrateur", "Authentification impossible", MessageBoxButtons.OK, MessageBoxIcon.Warning);
    }
}
```

L'intérêt de tout ça c'est de s'assurer de créer qu'une seule connexion à la base de données afin d'optimiser les ressources.

ConnexionBDD possède de ce fait le package MySQL.data ce qui lui permet d'avoir les propriétés nécessaires comme MySqlConnection qui permet d'établir la connexion ou encore MySqlCommand qui permet de préparer une commande.

Le développement - Mise en place des classes métiers :

Avant de commencer le développement des UserControl, je crée toutes les classes métier du package Model, correspondantes aux tables de la base de données afin de pouvoir les utiliser comme des objets, facilitant grandement leurs utilisations.

Les propriétés de ces objets sont encapsulées avec un {get; set} afin de pouvoir les utiliser librement depuis l'extérieur.

```
public class Adherents
{
    private int idAdherent;
    private string nomAdherent;
    private string prenomAdherent;
    private string telAdherent;
    private string emailAdherent;
    private DateTime dateNaissanceAdherent;
    private float montantCreditsAdherent;
    private DateTime dateOuvertureAbonnement;
    private DateTime dateFinAbonnement;

    2 références
    public Adherents(int idAdherent, string nomAdherent, string prenomAdherent, string telAdherent, string emailAdherent, DateTime dateNaissanceAdherent,
        float montantCreditsAdherent, DateTime dateOuvertureAbonnement, DateTime dateFinAbonnement)
    {
        this.IdAdherent = idAdherent;
        this.NomAdherent = nomAdherent;
        this.PrenomAdherent = prenomAdherent;
        this.TelAdherent = telAdherent;
        this.EmailAdherent = emailAdherent;
        this.DateNaissanceAdherent = dateNaissanceAdherent;
        this.MontantCreditsAdherent = montantCreditsAdherent;
        this.DateOuvertureAbonnement = dateOuvertureAbonnement;
        this.DateFinAbonnement = dateFinAbonnement;
    }

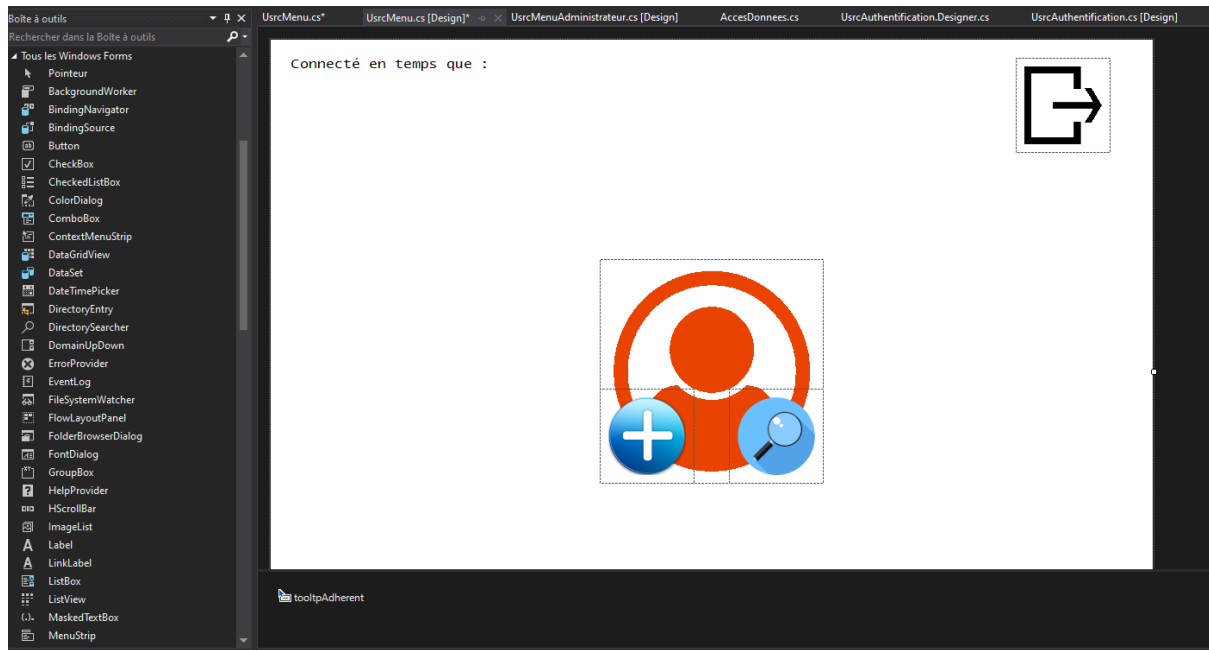
    11 références
    public int IdAdherent { get => idAdherent; set => idAdherent = value; }
    13 références
    public string NomAdherent { get => nomAdherent; set => nomAdherent = value; }
    12 références
    public string PrenomAdherent { get => prenomAdherent; set => prenomAdherent = value; }
    5 références
    public string TelAdherent { get => telAdherent; set => telAdherent = value; }
```

Le développement - Création des formulaires et UserControl :

Le projet initial comptait utiliser des formulaires différents pour chaque élément de la maquette, toutefois le collaborateur m'aidant sur la création des formulaires m'a conseillé d'utiliser les UserControl. Ces derniers se comportent comme des calques qui peuvent être appelés et retirés à tout moment sur un formulaire. L'énorme avantage d'utiliser les UserControl à la place des formulaires est l'optimisation.

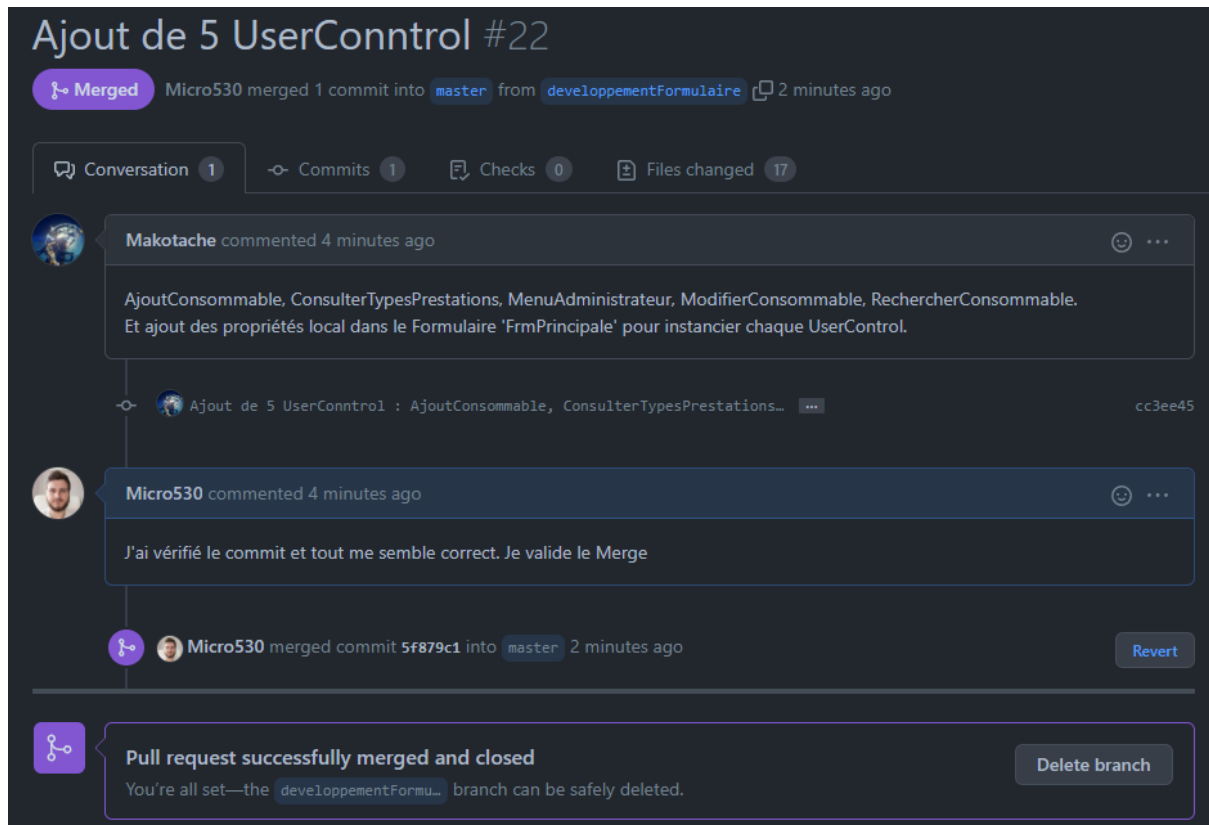
Ainsi, à la place d'avoir des multiples formulaires à créer et fermer, l'application n'aura qu'un seul formulaire sur lequel on appelle les UserControls nécessaire.

La création de ces derniers est très similaire à la création d'un formulaire. Il suffit de glisser et déposer les objets souhaités depuis la boîte à outils sur le UserControl.



Après quoi je récupère la création des UserControl par le collaborateur m'aidant sur le projet via GitHub.

Les UserControl créés sont fait depuis le maquettage que je lui ai transmis, il ne fait aucun code, il se contente de placer les objets graphiques de la même manière que la maquette. Cette aide m'est précieuse puisqu'elle me permet de gagner beaucoup de temps puisque, en plus de n'avoir qu'à renommer les objets graphiques avant de faire le code, je peux d'ores et déjà appelé les UserControl étant donné que les objets existent déjà



Le développement - Mise en place de l'authentification :

Je crée tout d'abord le formulaire FrmPrincipal, c'est ce formulaire qui gèrera tous les UserControl comme un contrôleur. Toujours dans un souci de respecter le MVC les UserControl ne pourront communiquer qu'avec ce formulaire.

Ensuite, je configure programme.cs pour qu'il démarre sur le contrôle, contrôle qui dans son constructeur instancie frmPrincipal de type FrmPrincipal, en envoyant son instance.

```
/// <summary>
/// Constructeur de controle qui instancie et affiche le formulaire principal
/// </summary>
1 référence
public Controle()
{
    frmPrincipal = new FrmPrincipal(this);
    frmPrincipal.ShowDialog();
}
```

Le formulaire principal n'est qu'une fenêtre vide avec seulement un fond d'image. Ce formulaire garde en mémoire dans une propriété privée l'instance du contrôle qui l'a créée. Puis lors du chargement du formulaire, le UserControl usrcAuthentification est instancié et ajouté au formulaire.

```
/// <summary>
/// Instanciation du usrcAuthentification au chargement du fomulaire
/// </summary>
/// <param name="sender"></param>
/// <param name="e"></param>
1 référence
private void FrmPrincipal_Load(object sender, EventArgs e)
{
    ConsUsrcAuthentification();
}
```

```
/// <summary>
/// Construction du UsrcAuthentification
/// </summary>
1 référence
private void ConsUsrcAuthentification()
{
    usrcAuthentification = new UsrcAuthentification(this);
    usrcAuthentification.Location = new Point(0, 0);
    this.Controls.Add(usrcAuthentification);
    usrcAuthentification.Show();
}
```

Lors d'une tentative de connexion, usrcAuthentification appelle la méthode Authentification en envoyant en paramètre les identifiants de connexion au frmPrincipal grâce à l'instance qui avait reçu en paramètre lors de sa création. Le frmPrincipal appelle à son tour dans un "if" la fonction Authentification du contrôleur. Si ce dernier retourne Vrai, le usrcAuthentification se ferme et celui du menu est créé.

La fonction Authentification du contrôleur se contente d'appeler la fonction de même nom dans Accès données, toujours en lui envoyant les paramètres reçus

```

/// <summary>
/// La méthode qui permet de lancer la procédure d'authentification
/// </summary>
/// <param name="identifiant">l'identifiant de l'utilisateur</param>
/// <param name="pwd">le mot de passe de l'utilisateur</param>
1 référence
public bool Authentification(string identifiant, string pwd)
{
    return AccesDonnees.Authentification(identifiant, pwd);
}

```

La fonction d' Accès données va terminer de créer la chaîne de connexion avec les identifiants reçus en paramètre puis va demander de récupérer l'instance de connexion avec cette chaîne. Si la fonction IsRunning retour vrai, c'est que la connexion est réussie, par conséquent, la chaîne est correcte et les identifiants le sont aussi.

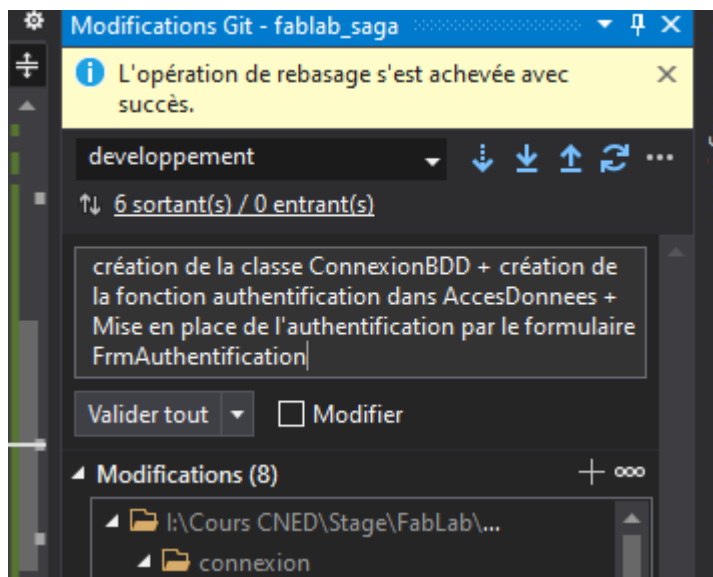
Créer la chaîne de cette manière permet de rendre l'accès à la base de données totalement sécurisé. En cas de hackage de l'application, l'individu malveillant n'aura pas de quoi se connecter à la base de données

```

/// <summary>
/// permet de construire le chaine de connexion avec les identifiants de connexion reçu en paramètres
/// </summary>
/// <param name="identifiant"></param>
/// <param name="pwd"></param>
1 référence
public static bool Authentification(string identifiant, string pwd)
{
    chaineConnexion = "Server=" + adresseServeur + "; Port=" + port + "; user id=" + identifiant + ";password=" + pwd + "; " +
        "persistsecurityinfo=True; database=" + nomBDD + ";SSL Mode=None";
    ConnexionBDD bdd = ConnexionBDD.GetInstance(chaineConnexion);
    return bdd.IsRunning();
}

```

Également, de cette manière, l'utilisateur ne pourra affecter la base de données qu'avec les droits qui lui ont été attribués et il ne pourra pas accéder à certaine zone de l'application puisqu'il ne sera pas en mesure de faire une requête SELECT sur certaines parties sensibles si il n'est pas autorisé.



Ne pas oublier après tout ça de commit et Push sur le dépôt distant GitHub

Le développement - Gestions de adhérents - l'ajout d'un adhérent :

L'ajout d'un adhérent est assez simple, seuls 5 coordonnées sont utilisées pour en créer un.

Connecté en temps que : <nom utilisateur>

Nom :

Prénom :

Téléphone :

E-mail :

Date de naissance :

Jour

Mois

Année

1

Janvier

2021

Le 1 Janvier 2021

L'identifiant et le mot de passe sont générés automatiquement puis envoyés par E-mail

Le calendrier utilisé est de ma création, je n'aimais pas les calendriers proposés par Form donc j'ai choisi d'en faire un personnalisé. Ce calendrier est un UserControl également qui est appelé comme n'importe quel objet. Celui-ci possède une méthode publique "GetDateNaissance", qui permet de retourner un DateTime en fonction des sélections faite dans les ComboBox.

```

/// <summary>
/// Permet de remplir les comboBox
/// </summary>
1 référence
private void RemplissageCombo()
{
    List<int> jour = new List<int> { };
    List<string> mois = new List<string> { "Janvier", "Fevrier", "Mars", "Avril", "Mai", "Juin",
        "Juillet", "Août", "Septembre", "Octobre", "Novembre", "Décembre" };
    List<int> annee = new List<int> { };
    for (int i = 1; i < 32; i++)
    {
        jour.Add(i);
    }
    for (int i = DateTime.Now.Year; i > 1899; i--)
    {
        annee.Add(i);
    }
    combJour.DataSource = jour;
    combMois.DataSource = mois;
    combAnnee.DataSource = annee;
}

```

```

/// <summary>
/// Retourne la date choisie
/// </summary>
/// <returns>le dateTime contenant la date sélectionné</returns>
3 références
public DateTime GetDateNaissance()
{
    DateTime date = new DateTime(int.Parse(combAnnee.SelectedItem.ToString()), combMois.SelectedIndex + 1, int.Parse(combJour.SelectedItem.ToString()));
    return date;
}

```

Suite au bon remplissage des champs, le bouton sauvegarder contrôle qu'ils sont tous bien remplis avant d'appeler la méthode du frmPrincipal AjoutAdherent

```

/// <summary>
/// Événement clique sur le bouton btnSauvegarder, vérifie les champs puis envoie les info du nouvel adherent
/// </summary>
/// <param name="sender"></param>
/// <param name="e"></param>
1 référence
private void pïctbSauvegarder_Click(object sender, EventArgs e)
{
    if(! (txtNomAdherent.Text.Equals("") || txtPrenomAdherent.Text.Equals("") || txtTelephoneAdherent.Text.Equals("") ||
        txtEmailAdherent.Text.Equals("") || !calendrier.SelectionDateNaissance()))
    {
        frmPrincipal.AjoutAdherent(txtNomAdherent.Text, txtPrenomAdherent.Text,
            txtTelephoneAdherent.Text, txtEmailAdherent.Text, calendrier.GetDateNaissance());
        MessageBox.Show("Adhèrent ajouter avec succès", "Confirmation d'ajouter d'un adherent", MessageBoxButtons.OK, MessageBoxIcon.Information);
        ViderChamps();
    }
    else
    {
        MessageBox.Show("Tous les champs doivent être remplis pour continuer", "Champ(s) vide(s)", MessageBoxButtons.OK, MessageBoxIcon.Error);
    }
}

```

Notez que ce dernier n'envoie pas un objet Adhérent mais directement les informations saisies. Ceci est normal, parce qu'un adhérent possède beaucoup plus d'informations que les 5 renseignées, toutefois dans une optique d'optimisation et de simplicité, tout ce qui pouvait être fait par la base de données est géré par elle. Le reste des informations sont principalement des dates de création de compte et la création des identifiant de connexion, tout ceci est créé grâce aux triggers dans la base de données.

Le FrmPrincipal (qui fait office de contrôleur pour les UserControl) se contente d'envoyer les informations reçues en paramètre à la méthode du même nom du contrôleur de l'application.

```

#region Méthode d'ajouts/Modifications
/// <summary>
/// Événement d'ajout Adherent depuis le usrcAjoutAdherent
/// </summary>
/// <param name="nom">nom de l'adherent</param>
/// <param name="prenom">prenom de l'adherent</param>
/// <param name="telephone">telephone de l'adherent</param>
/// <param name="email">email de l'adherent</param>
/// <param name="dateNaissance">deta de naissance de l'adherent</param>
1 référence
public void AjoutAdherent(string nom, string prenom, string telephone, string email, DateTime dateNaissance)
{
    controle.AjoutAdherent(nom, prenom, telephone, email, dateNaissance);
}

```

Le contrôleur fait exactement la même chose, sauf qu'il appelle la méthode dans AccesDonnees.

Ici AccesDonnees créer la requête SQL, et l'envoie à ReqUpdate de ConnexionBDD avec le Dictionnaire contenant les paramètres de la requête (permettant d'éviter les injections SQL pour plus de sécurité)

```

Requetes de récupération
/// <summary>
/// Événement d'ajout d'un nouvel adherent
/// </summary>
/// <param name="nom">nom de l'adherent</param>
/// <param name="prenom">prenom de l'adherent</param>
/// <param name="telephone">telephone de l'adherent</param>
/// <param name="email">email de l'adherent</param>
/// <param name="dateNaissance">deta de naissance de l'adherent</param>
1 référence
public static void AjoutAdherent(string nom, string prenom, string telephone, string email, DateTime dateNaissance)
{
    ConnexionBDD bdd = ConnexionBDD.GetInstance(chaineConnexion);
    Dictionary<string, object> parameters = new Dictionary<string, object>();
    parameters.Add("@nom", nom);
    parameters.Add("@prenom", prenom);
    parameters.Add("@telephone", telephone);
    parameters.Add("@email", email);
    parameters.Add("@dateNaissance", dateNaissance);
    bdd.ReqUpdate("insert into adherent (NOM, PRENOM, TELEPHONE, EMAIL, DATENAISSANCE) " +
        "VALUES (@nom, @prenom, @telephone, @email, @dateNaissance);", parameters);
}

```

ReqUpdate est une méthode qui prépare les commandes qui ne sont pas destinées à avoir un retour (comme un SELECT). Cette dernière appelle la méthode creationCommand qui permet de "mixer" la chaineRequete et les paramètres.

```

/// <summary>
/// execution requete de type update (insert, delete, update, ...)
/// </summary>
/// <param name="chaineRequete">la requete</param>
/// <param name="parameters">les paramètres de la requete</param>
16 références
public void ReqUpdate(string chaineRequete, Dictionary<string, object> parameters)
{
    try
    {
        creationCommand(chaineRequete, parameters);
        this.command.ExecuteNonQuery();
    }
    catch (Exception e)
    {
        Console.WriteLine(e.Message);
    }
}

```

```

/// <summary>
/// permet de créer la commande avec la ChaineRequete et le dictionnaire de paramètres
/// </summary>
/// <param name="chaineRequete">la requete</param>
/// <param name="parameters">les paramètres de la requete</param>
2 références
private void creationCommand(String chaineRequete, Dictionary<string, object> parameters)
{
    this.command = new MySqlCommand(chaineRequete, this.connection);
    foreach (KeyValuePair<string, object> parameter in parameters)
    {
        command.Parameters.Add(new MySqlParameter(parameter.Key, parameter.Value));
    }
    command.Prepare();
}

```

Le développement - Gestions de adhérents - Recherche d'un adhérent:

The image shows a user interface for searching members. It features three dropdown menus for 'Nom', 'Prénom', and 'Date de naissance'. Below these are two buttons: a red circular button with a white 'X' and a blue rectangular button with a white right-pointing arrow.

Ce UserControl UsrcRechercheAdherent ne possède que 3 ComboBox permettant de faire une recherche dans la base de données. Au chargement du UserControl, la fonction GetLesAdherent du frmPrincipal est appelée. Cette méthode retourne une liste de type Adherent, la liste locale lesAdherents permet de garder en mémoire ce retour. Également les comboBox combPrenom et combDateNaissance sont désactivé afin de forcer l'utilisateur à commencer par un nom.


```

/// <summary>
/// Au chargement du formulaire
/// </summary>
/// <param name="sender"></param>
/// <param name="e"></param>
1 référence
private void UsrcRechercheAdherent_Load(object sender, EventArgs e)
{
    lesAdherents = frmPrincipal.GetLesAdherent();
    combPrenom.Enabled = false;
    combDateNaissance.Enabled = false;
    RechargeCombNom();
}

```

Extrait du formulaire principal

```

/// <summary>
/// Permet de récupérer et d'envoyer la liste des adhérents
/// </summary>
1 référence
public List<Adherents> GetLesAdherent()
{
    return controle.GetLesAdherents();
}

```

Cette fonction appelle la fonction du même nom du contrôleur, qui à son tour appelle la même de AccesDonnees. Dans AccesDonnees, la requête permettant de récupérer tous les adhérents est préparée en triant le retour sur le nom par ordre alphabétique. Cette fois il s'agit de la méthode ReqSelect qui est appelée depuis ConnexionBDD.

```

/// <summary>
/// Événement de récupération des adhérents
/// </summary>
/// <returns>la liste des adherents</returns>
1 référence
public static List<Adherents> GetLesAdherents()
{
    List<Adherents> lesAdherents = new List<Adherents>();
    ConnexionBDD bdd = ConnexionBDD.GetInstance(chaineConnexion);
    Dictionary<string, object> parameters = new Dictionary<string, object>();
    bdd ReqSelect("SELECT * FROM adherent order by NOM;", parameters);
    while (bdd.Read())
    {
        lesAdherents.Add(new Adherents(int.Parse(bdd.Field("IDADHERENT").ToString()), (string)bdd.Field("NOM"), (string)bdd.Field("PRENOM"),
            (string)bdd.Field("TELEPHONE"), (string)bdd.Field("EMAIL"), (DateTime)bdd.Field("DATENAISSANCE"), (float)bdd.Field("MONTANTCREDITS"),
            (DateTime)bdd.Field("DATE_OUVERTURE_ABONNEMENT"), (DateTime)bdd.Field("DATE_FIN_ABONNEMENT")));
    }
    bdd.close();
    return lesAdherents;
}

```

La méthode ReqSelect fonctionne de la même manière que ReqUpdate, à ceci près qu'elle initialise le curseur de type MySqlDataReader avec la commande préparée. Ceci permet de récupérer le retour de la requête dans ce curseur.

```

/// <summary>
/// execution du requete de type SELECT
/// </summary>
/// <param name="chaineRequete">la requete</param>
/// <param name="parameters">les paramètres de la requete</param>
13 références
public void ReqSelect(String chaineRequete, Dictionary<string, object> parameters)
{
    try
    {
        creationCommand(chaineRequete, parameters);
        this.curseur = this.command.ExecuteReader();
    }
    catch (Exception e)
    {
        Console.WriteLine(e.Message);
    }
}

```

Une fois le curseur rempli avec notre résultat de requête, il suffit de lire ce curseur lignes après lignes avec la fonction Read() de ConnexionBDD. Cette fonction retourne la ligne suivante (il commence à -1 donc le premier Read() permet d'avoir la première ligne) pour en récupérer les informations qui nous intéressent. Dès qu'il n'y a plus de ligne, ce dernier retourne false, arrêtant la boucle. Ici nous remplissons une liste d'Adherent avec une boucle While dans lesquels nous récupérons chaque valeur de chaque colonne de la ligne actuelle afin de valoriser les propriétés privées. La fonction Field() permet de retourner la valeur de la colonne passée en paramètre.

```

while (bdd.Read())
{
    lesAdherents.Add(new Adherents(int.Parse(bdd.Field("IDADHERENT").ToString()), (string)bdd.Field("NOM"), (string)bdd.Field("PRENOM"),
    (string)bdd.Field("TELEPHONE"), (string)bdd.Field("EMAIL"), (DateTime)bdd.Field("DATENAISSANCE"), (float)bdd.Field("MONTANTCREDITS"),
    (DateTime)bdd.Field("DATE_OUVERTURE_ABONNEMENT"), (DateTime)bdd.Field("DATE_FIN_ABONNEMENT")));
}
bdd.close();
return lesAdherents;

```

Après quoi la liste est retournée permettant de valoriser notre liste locale d'Adherent dans leUsrcRechercheAdherent.

Après la valorisation de notre liste locale, la méthode RechargeCombNom() est appelée, cette méthode permet de remplir le comboBox combNom avec uniquement les noms des adhérent de la liste lesAdherents.

```

/// <summary>
/// Permet de charger/recharger le comboBox combNom
/// </summary>
1 référence
private void RechargeCombNom()
{
    foreach (Adherents unAdherent in lesAdherents)
    {
        if (!combNom.Items.Contains(unAdherent.NomAdherent))
        {
            combNom.Items.Add(unAdherent.NomAdherent);
        }
    }
}

```

A partir de maintenant il suffit de placer un événement sur le changement d'index sur le comboBob combNom afin qu'il s'occupe de remplir le combPrenom avec les prénom qui correspondent au nom choisi, et ainsi de suite avec la date de naissance.

```

/// <summary>
/// Événement lors d'une sélection dans le comboBox
/// </summary>
/// <param name="sender"></param>
/// <param name="e"></param>
1 référence
private void combNom_SelectedIndexChanged(object sender, EventArgs e)
{
    RechargeCombPrenom(combNom.SelectedItem.ToString());
    combPrenom.Enabled = true;
}

```

Ici le RechargeCombPrenom attend un nom pour filtrer les prénoms, après quoi dans un foreach il parcourt tous les adhérents et ajoute au combPrenom uniquement ceux qui ont un nom identique à filtreNom.

Pour la date de naissance, le principe est identique, toutefois il faut un filtreNom et un filtrePrenom pour ce dernier

```

/// <summary>
/// Permet de charger/recharger le comboBox combPrenom
/// </summary>
/// <param name="filtreNom"></param>
1 référence
private void RechargeCombPrenom(string filtreNom)
{
    combPrenom.Items.Clear();
    foreach (Adherents unAdherent in lesAdherents)
    {
        if (!combPrenom.Items.Contains(unAdherent.PrenomAdherent) && unAdherent.NomAdherent == filtreNom)
        {
            combPrenom.Items.Add(unAdherent.PrenomAdherent);
        }
    }
    combPrenom.SelectedIndex = 0;
}

/// <summary>
/// Permet de charger/recharger le comboBox combDateNaissance
/// </summary>
/// <param name="filtreNom"></param>
/// <param name="filtrePrenom"></param>
1 référence
private void RechargeCombDateNaissance(string filtreNom, string filtrePrenom)
{
    combDateNaissance.Items.Clear();
    foreach (Adherents unAdherent in lesAdherents)
    {
        if (!combDateNaissance.Items.Contains(unAdherent.DateNaissanceAdherent) &&
            unAdherent.NomAdherent == filtreNom && unAdherent.PrenomAdherent == filtrePrenom)
        {
            combDateNaissance.Items.Add(unAdherent.DateNaissanceAdherent);
        }
    }
    combDateNaissance.SelectedIndex = 0;
}

```

Après quoi lors du clique sur le bouton "suivant", le système récupère l'adhérent à partir des critères sélectionnés, puis il est envoyé en paramètre dans la méthode du frmPrincipal "CreationConsultationAdherent(Adherents unAdherent)".

```

/// <summary>
/// Événement du bouton sauvegarder
/// </summary>
/// <param name="sender"></param>
/// <param name="e"></param>
1 référence
private void pictbSuivant_Click(object sender, EventArgs e)
{
    if(!combNom.SelectedIndex.Equals(-1) && !combPrenom.SelectedIndex.Equals(-1) &&
        !combDateNaissance.SelectedIndex.Equals(-1))
    {
        foreach (Adherents unAdherents in lesAdherents)
        {
            if (unAdherents.NomAdherent.Equals(combNom.SelectedItem.ToString()) &&
                unAdherents.PrenomAdherent.Equals(combPrenom.SelectedItem.ToString()) &&
                unAdherents.DateNaissanceAdherent.Equals((DateTime)combDateNaissance.SelectedItem))
            {
                frmPrincipal.CreationConsultationAdherent(unAdherents);
                break;
            }
        }
    }
    else
    {
        MessageBox.Show("Tous les champs doivent être rempli pour continuer",
            "Champs vide(s)", MessageBoxButtons.OK, MessageBoxIcon.Information);
    }
}

```

Cette méthode appelle la méthode de construction de la même manière que pour les autres UserControl, à ceci près qu'elle envoie l'adhérent qu'elle a reçu en paramètre. Adherent nécessaire pour ouvrir le UserControl de la consultation d'adhérent qui attend un adhérent dans ses paramètres

```

/// <summary>
/// Permet de fermer le UserControl ayant initialisé et de lancer la méthode d'ouverture du usrcConsultationAdherent
/// </summary>
/// <param name="unAdherent"></param>
4 références
public void CreationConsultationAdherent(Adherents unAdherent)
{
    DisposeAll();
    ConsUsrcConsultationAdherent(unAdherent);
}

```

Le développement - Gestions d' adhérents - Nouvelle Prestation:

Connecté en temps que : <nom utilisateur>
Adhérent : <nom utilisateur>

Type prestation :

Ajouter une ligne


Choisir un type de consommable

Type consommable :
Sous type consommable :

Choisir consommable

Consommable :
Quantité : Grammes

Libellé :
Coût en crédits : 0



	Consommable	Libellé	Quantité	Montant
*				

Ici le but recherché est d'assister le plus possible l'utilisateur dans sa démarche. Le système se veut comme un entonnoir, chaque sélection de comboBox va réduire le choix des possibilités du suivant afin de trouver facilement et rapidement le consommable recherché. Une fois le consommable recherché sélectionné, l'utilisateur saisit la quantité utilisée de ce dernier ainsi que la raison de son utilisation (le nom de l'objet fabriqué avec), puis l'ajoute. Le DataGridView en dessous liste les différents consommables utilisés pour cette prestation.

Cette fois, les comboBox ne sont pas remplis avec des boucles foreach, mais plutôt en utilisant leur DataSource. Les listes récupérées depuis la base de données sont valorisées dans des DataSource de BindingSource. Après quoi, ce BindingSource est lié avec le comboBox. La méthode RemplissageComboBox sert justement à les lier, en affichant que la propriété qui nous intéresse sur l'entièreté de l'objet.

```

/// <summary>
/// Récupération des listes au sein de la base de donnée, et affectation au BingdingSource correspondant
/// </summary>
1 référence
private void RecuperationListes()
{
    lesTypesPrestations.DataSource = frmPrincipal.GetLesTypesPrestations();
    lesTypesConsommables.DataSource = frmPrincipal.GetLesTypesConsommables();
    lesSousTypesConsommables.DataSource = frmPrincipal.GetLesSousTypesConsommables();
    lesConsommables.DataSource = frmPrincipal.GetLesConsommables();
}
#region Remplissage dynamique des ComboBox
/// <summary>
/// Permet de remplir un comboBox avec n'importe BingdingSource
/// </summary>
/// <param name="unCombo">le comboBox à remplir</param>
/// <param name="uneBdgs">le BingdingSource à utiliser pour le remplir</param>
/// <param name="nomPropriete">la propriété à afficher dans le comboBox</param>
4 références
private void RemplissageComboBox(ComboBox unCombo, BindingSource uneBdgs, string nomPropriete)
{
    unCombo.DataSource = uneBdgs;
    unCombo.DisplayMember = nomPropriete;
    unCombo.ValueMember = nomPropriete;
    unCombo.SelectedIndex = -1;
}

```

La façon dont les comboBox se mettent à jour est très similaire à la recherche d'un adhérent vu plutôt.

Une fois tous les champs remplis, la liste lesConsommablesPrestations de type ConsommablePrestation (objet qui représente une ligne de prestation), se voit ajouter un nouvel élément à partir des sélections faites. En parallèle, le dataGridView est également rempli mais pas avec les mêmes informations. En effet, un objet ConsommablePrestations comporte beaucoup de propriétés non pertinentes comme par exemple des ID qui ne vont pas parler à l'utilisateur, ce pourquoi certains ID sont directement remplacés par du texte dans ce DGV.

```

/// <summary>
/// Événement lors du clique sur le bouton pictAjouterLigne
/// </summary>
/// <param name="sender"></param>
/// <param name="e"></param>
1 référence
private void pictAjouterLigne_Click(object sender, EventArgs e)
{
    if (combTypePrestation.SelectedIndex != -1 && combTypeConsommable.SelectedIndex != -1 &&
        combSousTypeConsommable.SelectedIndex != -1 && combConsommable.SelectedIndex != -1 &&
        txtQuantite.Text != "" && txtQuantite.Text != "0" && txtLibelle.Text != "")
    {
        //ajout des paramètres dans le bindingSource afin de mémoriser les objets ConsommablePrestation
        lesConsommablesPrestations.Add(new ConsommablePrestation(1, ((Consommable)combConsommable.SelectedItem).IdConsommable, txtLibelle.Text,
            int.Parse(txtQuantite.Text), int.Parse(txtQuantite.Text) * ((Consommable)combConsommable.SelectedItem).PrixConsommable));
        //ajout des paramètres de façon claire et lisible dans le dataGridView
        dgvDetailPrestation.Rows.Add(((Consommable)combConsommable.SelectedItem).LibelleConsommable, txtLibelle.Text,
            int.Parse(txtQuantite.Text), int.Parse(txtQuantite.Text) * ((Consommable)combConsommable.SelectedItem).PrixConsommable);
        ResetFormulaire();
    }
    else
    {
        MessageBox.Show("Tous les champs doivent être rempli pour pouvoir continuer", "Champs vide(s)",
            MessageBoxButtons.OK, MessageBoxIcon.Information);
    }
}

```

Une fois que l'utilisateur clique sur le bouton sauvegarder, le système crée dans la base de données une nouvelle Prestation dans laquelle on retrouvera notre liste de ConsommablePrestation.

```

/// <summary>
/// Événement lors du clique sur le bouton pictSauvegarder permettant de valider toutes les lignes et d'envoyer les données pour les requetes
/// </summary>
/// <param name="sender"></param>
/// <param name="e"></param>
/// </param>
private void pictSauvegarder_Click(object sender, EventArgs e)
{
    //la variable est déjà instancié avec ctte chaine, aucas ou l'utilisateur clique sur le bouton annuler
    string libelle = "SansNom";
    libelle = Interaction.InputBox("Voulez vous donner un nom à votre projet ?", "Nom de projet", "SansNom");

    //Permet de mémoriser la date utiliser pour être certain d'utiliser la même à chaque fois
    DateTime datePrestation = DateTime.Now;
    frmPrincipal.AjoutNouvellePrestation(((TypePrestation)combTypePrestation.SelectedItem).IdTypePrestation,
        unAdherent.IdAdherent, datePrestation, libelle);

    //Une sécurité permettant de laisser le temps à la base de données de créer la requete précédente en cas de lag
    System.Threading.Thread.Sleep(2000);

    //Permet de contrer un défaut d'arrondi au supérieur des secondes de la date dans la base de données
    //si les milliseconde s'out supérieur ou égale à 500
    if (datePrestation.Millisecond >= 500)
    {
        datePrestation.AddSeconds(1);
    }
    Prestation unePrestation = frmPrincipal.RecupUnePrestation(unAdherent.IdAdherent, ConversionDateBdd(datePrestation));
    MessageBox.Show(ConversionDateBdd(datePrestation));
    ActualisationIdPrestation(unePrestation.IdPrestation);
    frmPrincipal.AjoutLignesPrestation(lesConsommablesPrestations);
    pictAnnuler_Click(null, null);
}

```

Le développement - Gestions d'adhérents - Consulter l'historique:

Connecté en temps que : <nom utilisateur>

Adhérent : <nom utilisateur>

Événement	Date	Type de prestation	Libellé	Montant

Supprimer

Trier depuis :

Revenir à l'adhérent

Consommable utilisé	Quantité	Unité	Prix unitaire	Montant

Consulter l'historique permet d'avoir une vue sur la consommation ou l'ajout de crédits d'un adhérent. Le DGV du haut permet de lister, trier par date de la plus récente à la plus éloignée, les prestations et les rechargements. Lors de la sélection d'une ligne, le DGV du bas affiche les détails de cette prestation (les fameux ConsommablePrestation). Notez le combTrie qui permet d'afficher les prestations uniquement sur une certaine plage.

Avant toute chose, je commence par gérer le combTrie puisque qu'il va être lu avant l'affichage des prestations. Dans les propriétés, je crée une liste avec les différentes possibilités de tries.

```

/// <summary>
/// Liste des différents modes de tries
/// </summary>
private List<string> listeCombTrie = new List<string> { "Dernière semaine", "Dernier mois", "3 derniers mois", "6 derniers mois", "Toujours"};

```

Ensuite, lors du remplissage du DGV, le contenu du combTrie est interrogé dans switch, chaque case appelant la méthode permettant de générer la date avec des paramètres différents, date qui sera utilisée plus tard dans la requête de la base de données.

Notez l'utilisation de Concat() sur les listes de prestations et de rechargements afin de les fusionner pour les afficher dans le DGV.

```

/// <summary>
/// Permet de récupérer les listes avec le filtre sur la date
/// </summary>
/// <param name="modeTrie">filtre sur le date</param>
2 références
private void RecupLesListesEtRemplissageDgvListesPrestation(string modeTrie)
{
    string dateTrie = null;
    switch (modeTrie)
    {
        case "Dernière semaine":
            dateTrie = ConversionDateBdd(DateTime.Now.AddDays(-7));
            break;
        case "Dernier mois":
            dateTrie = ConversionDateBdd(DateTime.Now.AddMonths(-1));
            break;
        case "3 derniers mois":
            dateTrie = ConversionDateBdd(DateTime.Now.AddMonths(-3));
            break;
        case "6 derniers mois":
            dateTrie = ConversionDateBdd(DateTime.Now.AddMonths(-6));
            break;
        case "Toujours":
            dateTrie = "2021-01-01 00:00:00";
            break;
    }
    //Concat(<UneListe>) permet d'assembler deux listes
    toutesLesPrestations = (List<object>)frmPrincipal.GetLesRechargements(unAdherent.IdAdherent, dateTrie).Concat(
        (List<object>)frmPrincipal.GetLesPrestations(unAdherent.IdAdherent, dateTrie)).ToList();
    lesTypesPrestations = frmPrincipal.GetLesTypesPrestations();
    lesConsommables.DataSource = frmPrincipal.GetLesConsommables();
    RemplirDgvListePrestation();
}

```

Afin de gérer le visuel du détail d'une prestation, à chaque changement de sélection d'une des prestations, une multitude de traitements est nécessaire.

Tout d'abord, étant donné que les chargements de possèdent pas d'informations plus détaillées à afficher, il faut distinguer si la sélection en cours est une prestation ou un rechargement. Ensuite, l'id de la prestation est comparé avec les ConsommablePrestation afin de récupérer toutes les lignes correspondantes. Il y a également du traitement de mise en forme pour un affichage plus parlant envers l'utilisateur.

```

/// <summary>
/// Événement lors du changement de selection dans le dgvListePrestation pour generer l'affichage du dgvDetailPrestation
/// </summary>
/// <param name="sender"></param>
/// <param name="e"></param>
1 référence
private void dgvListePrestation_SelectionChanged(object sender, EventArgs e)
{
    dgvDetailPrestation.Rows.Clear();
    foreach(object unObject in toutesLesPrestations)
    {
        if(unObject is Prestation)
        {
            if (((Prestation)unObject).DatePrestation.Equals(dgvListePrestation.CurrentRow.Cells["date"].Value))
            {
                idPrestationSelectionnee = ((Prestation)unObject).IdPrestation;
                lesConsommablesPrestation = frmPrincipal.GetLesConsommablePrestation(((Prestation)unObject).IdPrestation);
                for (int i = 0; i < lesConsommablesPrestation.Count; i++)
                {
                    foreach(Consommable unConsommable in lesConsommables)
                    {
                        if (lesConsommablesPrestation[i].IdConsommable.Equals(unConsommable.IdConsommable))
                        {
                            List<TypeConsommable> lesTypesConsommables = new List<TypeConsommable>(frmPrincipal.GetLesTypesConsommables());
                            foreach(TypeConsommable unTypeConsommable in lesTypesConsommables)
                            {
                                if (unConsommable.IdTypeConsommable.Equals(unTypeConsommable.IdTypeConsommable))
                                {
                                    dgvDetailPrestation.Rows.Add(unConsommable.LibelleConsommable, lesConsommablesPrestation[i].QuantiteConsommable,
                                        unTypeConsommable.UniteConsommable, unConsommable.PrixConsommable, lesConsommablesPrestation[i].MontantConsommable);
                                    break;
                                }
                            }
                        }
                    }
                }
            }
        }
    }
}

```

Cette suite de boucles et de if, manque d'optimisation, j'ai déjà quelques idées pour rendre la chose plus légère, toutefois il ne me reste plus que quelques jours pour présenter une version stable. Cette partie-là sera référencée dans la documentation “**Projet d'évolution**” qui accompagnera l'application pour les prochains développeurs après mon stage.

Le développement - Modification d'un consommables :

Ici, la modification d'un consommable, utilise le même UserControl que pour son ajout (l'ajout quant à lui fonctionne plus ou moins comme l'ajout d'un adhérent).

La partie intéressante sur la modification est l'utilisation d'une surcharge (ou overloading) sur le constructeur du UserControl d'ajout/modification.

Le premier constructeur est réservé à l'ajout d'un consommable, il attend uniquement l'instance du formulaire principal ainsi que l'identifiant de connexion. Cependant, le deuxième constructeur utilise un paramètre de plus dans sa signature, ce dernier attend un consommable, consommable qu'ilinstanciera dans la propriété privé unConsommable.

```

/// <summary>
/// Constructeur
/// </summary>
/// <param name="frmPrincipal">L'instance du formulaire parent</param>
/// <param name="identifiant">l'identifiant utilisé lors de la connexion</param>
1 référence
public UsrcAjoutModificationConsommable(FrmPrincipal frmPrincipal, string identifiant)
{
    this.frmPrincipale = frmPrincipal;
    this.identifiant = identifiant;
    InitializeComponent();
}

/// <summary>
/// surcharge du Constructeur pour une modification
/// </summary>
/// <param name="frmPrincipal">L'instance du formulaire parent</param>
/// <param name="identifiant">l'identifiant utilisé lors de la connexion</param>
1 référence
public UsrcAjoutModificationConsommable(FrmPrincipal frmPrincipal, string identifiant, Consommable unConsommable)
{
    this.unConsommable = unConsommable;
    this.frmPrincipale = frmPrincipal;
    this.identifiant = identifiant;
    InitializeComponent();
}

```


Plus tard, lors du chargement du UserControl, la première partie du chargement ne change pas par rapport à un ajout. En revanche, par la suite, un if vérifie si unConsommable est instancié ou pas, étant donné que celui l'est uniquement si le deuxième constructeur est appelé, cela signifie que le UserControl soit passer en mode modification si unConsommable n'est pas null.

La deuxième partie (celle du if) permet de présélectionner les informations du consommable unConsommable tout en ajustant la mise en page pour plus de cohérence.

```

/// <summary>
/// Initialisation du formulaire
/// </summary>
/// <param name="sender"></param>
/// <param name="e"></param>
1 référence
private void UsrAjoutConsommable_Load(object sender, EventArgs e)
{
    lblNomIdentifiant.Text = identifiant;
    lesTypesConsommables.DataSource = frmPrincipale.GetLesTypesConsommables();
    lesSousTypesConsommables.DataSource = frmPrincipale.GetLesSousTypesConsommables();
    RemplissageAllComboBox();
    List<string> lesUnites = new List<string>();
    foreach (TypeConsommable unTypeConsommable in lesTypesConsommables)
    {
        lesUnites.Add(unTypeConsommable.UniteConsommable);
    }
    this.lesUnites.DataSource = lesUnites;
    combUnite.SelectedIndex = -1;

    //Permet de remplir le UserControl avec un consommable en mode modification
    if(unConsommable != null)
    {
        btnSupprimerConsommable.Visible = true;
        lblTitre.Text = "Mode : Modifier un consommable";
        foreach (TypeConsommable unType in lesTypesConsommables)
        {
            if (unConsommable.IdTypeConsommable.Equals(unType.IdTypeConsommable))
            {
                combTypeConsommable.SelectedItem = unType;
            }
        }
        foreach (SousTypeConsommable unSousType in lesSousTypesConsommables)
        {
            if (unConsommable.IdSousTypeConsommable.Equals(unSousType.IdSousTypeConsommable))
            {
                combSousType.SelectedItem = unSousType;
            }
        }
        txtLibelle.Text = unConsommable.LibelleConsommable;
        txtPrix.Text = unConsommable.PrixConsommable.ToString();
    }
}

```

Lors de l'enregistrement, après toute la partie permettant de créer tous les objets dont je vous épargne, un contrôle est fait pour savoir si les objets créés sont nouveaux ou s'il s'agit de modification. Auquel cas (s'il s'agit d'une modification), le nouveau consommable est comparé avec l'ancien pour vérifier qu'il y a bien des modifications afin d'éviter d'envoyer des requêtes update inutiles à la BDD. Si modification il y a, la requête update part mettre à jour la BDD avec les nouvelles sélections faites par l'utilisateur.

```

//Permet de controler si on est en mode modification ou ajout
if(this.unConsommable is null)
{
    frmPrincipale.AjouterConsommable(unNewConsommable);
    MessageBox.Show("Nouveau consommable ajouter avec succès !", "Confirmation d'ajout d'un nouveau consommable", MessageBoxButtons.OK, MessageBoxIcon.Information);
    ResetUserControl();
}
else
{
    if(unNewConsommable.IdTypeConsommable.Equals(unConsommable.IdTypeConsommable) &&
        unNewConsommable.IdSousTypeConsommable.Equals(unConsommable.IdSousTypeConsommable) &&
        unNewConsommable.LibelleConsommable.Equals(unConsommable.LibelleConsommable) && unNewConsommable.PrixConsommable.Equals(unConsommable.PrixConsommable))
    {
        MessageBox.Show("Tous les champs sont identique à l'objet initiale, faite des modifications afin de sauvegarder", "Champs Identiques", MessageBoxButtons.OK, MessageBoxIcon.Information);
    }
    else
    {
        unNewConsommable.IdConsommable = unConsommable.IdConsommable;
        frmPrincipale.ModificationConsommable(unNewConsommable);
        MessageBox.Show("Consommable modifié avec succès !", "Confirmation de modification d'un consommable", MessageBoxButtons.OK, MessageBoxIcon.Information);
    }
}
}

```

Le développement - Le déploiement :

Étant donné que les locaux n'existent pas encore, le déploiement a pu être effectué que sur des infrastructures de test. La base de données a été exportée sur un serveur distant avec le fichier SQL que PhpMyAdmin est capable de générer pour une exportation. Après l'exécution du fichier, je me suis connecté à ce serveur pour configurer les profils nécessaires à l'utilisation de cette base de données.

En ce qui concerne l'application, j'ai fait générer par Visual studio une publication en fichier exécutable installable sur n'importe quelle machine de type PC. La base de données et l'exécutable fonctionnent tous deux parfaitement bien.

À noter que le déploiement a été assez rapide (fait dans la journée) grâce au fait que les vrais locaux n'existent pas encore.

Le développement - Présentation de l'application devant l'assemblée :

Au dernier jour de mon stage, j'ai pu présenter mon application stable utilisant la base de données distante devant les principaux acteurs de l'association (une dizaine de personnes). La présentation s'est faite à l'oral avec un vidéo projecteur où j'expliquais point par point toutes les fonctionnalités avec l'ensemble des possibilités. Ils ont tous été satisfaits du rendu ainsi que de l'ergonomie.

Le développement - Conclusion du développement :

La version sortie en exécutable fonctionne dans l'ensemble mais il lui manque plusieurs fonctionnalités que je ne savais pas possible de terminer dans les temps impartis. Fonctionnalités qui seront détaillées dans la documentation "Projet d'évolution". Toutefois il est important de préciser que cette version possède des soucis de stabilité et d'optimisation. Soucis qui devront être impérativement résolus (en plus de l'ajout des fonctionnalités manquantes) avant le vrai déploiement dans les locaux finaux.

Conclusion:

Projet d'évolution :

Une documentation Projet d'évolution a été créée (au côté de la documentation technique) pour les futurs développeurs afin d'apporter des précisions sur certaines parties du code qui sont prévues pour d'autres fonctionnalités. Également cette documentation pointe des fonctions et des méthodes qui posent problème où qui manque d'optimisation.

Gain de compétences et remerciement :

Ce stage m'a été très lucratif aussi bien humainement que techniquement.

Sur la technique, j'ai pu perfectionner mon apprentissage avec des enjeux concrets. Cela m'a permis d'être dans une vraie situation avec des problématiques et des besoins divers, sur lesquels, il m'a fallu adapter ma logique d'approche en continue. Également, la mise en place du travail collaboratif a pris tout son sens sur ce stage car certaines personnes dépendaient de mes productions, tout comme moi, je dépendais parfois des leurs.

J'ai dû aussi me mettre à la place de l'utilisateur dans la création des interfaces, car, contrairement aux exercices et aux TP, ces dernières vont réellement être utilisées et leurs ergonomies deviennent un enjeu crucial.

Sur le plan humain j'ai beaucoup appris aussi. Comprendre les besoins, questionner la pertinence de telle ou telle chose, savoir se faire comprendre par des personnes qui ne sont pas forcément du milieu par des explications simplifiées. Également, devoir présenter devant plusieurs personnes des avancements de projet, cela m'a permis d'avoir plus confiance en moi.

Ce stage a été très riche sur tous ces plans, et j'en remercie l'intégralité l'équipe pour leur bienveillance et leur gentillesse, ils m'ont beaucoup apporté.

Suite aux résultats jugés satisfaisants par mes responsables, ces derniers ont décidé de m'embaucher pour continuer le développement de l'application ainsi que pour le développement de la version mobile et Web pour les adhérents.