

**Documentation technique**

Aide aux diabétiques

Lucas Pighini

cENTRE DE FORMATION PROFESSIONNELLE TECHNIQUE

|  |  |  |
| --- | --- | --- |
| **Date** | **Description** | **Version** |
| 22.05 | Page de garde | 1.0.0 |
| 22.05 | Versionning | 1.0.1 |
| 22.05 | Introduction | 1.1.1 |
| 22.05 | Fonctionnalités | 1.2.1 |
| 22.05 | Recommandations | 1.2.2 |
|  |  |  |

# Versionning

Table des matières

[Versionning 1](#_Toc516748279)

[Introduction 4](#_Toc516748280)

[Motivation 4](#_Toc516748281)

[Cahier des charges 4](#_Toc516748282)

[Objectif 4](#_Toc516748283)

[Fonctionnalités 4](#_Toc516748284)

[Analyse de l’existant 5](#_Toc516748285)

[Analyse fonctionnelle 5](#_Toc516748286)

[Connexion 5](#_Toc516748287)

[Mesures glycémie 5](#_Toc516748288)

[Graphiques de mesure 6](#_Toc516748289)

[Tableau de mesures 6](#_Toc516748290)

[Tableau d’insuline 7](#_Toc516748291)

[Ajout d’un patient 7](#_Toc516748292)

[Analyse organique 8](#_Toc516748293)

[Framework 8](#_Toc516748294)

[Base de données 8](#_Toc516748295)

[Users 8](#_Toc516748296)

[Measures : 9](#_Toc516748297)

[InsulinTable : 9](#_Toc516748298)

[Types : 9](#_Toc516748299)

[HasType 9](#_Toc516748300)

[Modélisation 10](#_Toc516748301)

[frmLogin 11](#_Toc516748302)

[frmMain 11](#_Toc516748303)

[boxMeasure 11](#_Toc516748304)

[boxGraphique 11](#_Toc516748305)

[boxTableau 11](#_Toc516748306)

[boxRecommandation 11](#_Toc516748307)

[boxPatient 11](#_Toc516748308)

[btnAccueil 11](#_Toc516748309)

[vRecommandation 12](#_Toc516748310)

[ModelMeasure 12](#_Toc516748311)

[ModelUser 12](#_Toc516748312)

[ModelMyInsulin 12](#_Toc516748313)

[User 12](#_Toc516748314)

[Measure 12](#_Toc516748315)

[InsulinTable 13](#_Toc516748316)

[Connection 13](#_Toc516748317)

[Fonctions importantes 13](#_Toc516748318)

[Affichage des mesures : 13](#_Toc516748319)

# Introduction

Etant actuellement en 3ème et dernière année accélérée de ma formation en informatique au sein du Centre de Formation Professionnel Technique, j’ai pour but de réaliser un travail pratique individuelle afin de confirmer l’obtention de mon CFC. J’ai donc choisi de développer une application C# qui permettra aux patients diabétiques et à leur médecin d’avoir un suivi informatisé et simplifié de leurs mesures de glycémie.

# Motivation

Au milieu de ma 3ème année, j’ai dû décidé de ce que je voulais faire comme projet pour mon TPI. D’abord j’ai dû choisir entre une application Web ou C#, ayant découvert Bunifu qui est une librairie de composant graphique pour Visual Studio, j’ai donc pris le choix de développer en C#. Ensuite il a fallu que je réalise un cahier des charges de ce que je voulais faire comme application, n’ayant pas vraiment d’idée je me suis donc décidé à faire une application qui pourrait se rendre utile pour un membre de ma famille. Ma grand-mère étant diabétique, j’ai donc pu voir plusieurs fois ma mère s’occuper des mesures de la glycémie et j’ai pu remarquer que tout est noté à l’écrit dans un carnet. C’est pourquoi j’ai décidé de simplifier les mesures et de les rendre informatisées.

# Cahier des charges

## Objectif

L’objectif de cette application Windows est de permettre à un patient d’entrer ses mesures quotidiennes de glycémie et ensuite de pouvoir visualiser les données.

## Fonctionnalités

Les utilisateurs connecté en tant que médecin auront les fonctionnalités suivantes :

* Consulter sous forme tabulaire les mesures d’un patient
* Sur le graphique des mesures, il peut choisir la date de début et de fin et les types
* Afficher sur un graphique la tendance des mesures de glycémie
* Affichage tabulaire des mesures
* Mesure de glycémie en lecture seul
* Gestion complète du tableau de recommandation d’insuline

Pour ceux qui sont connectés en tant que patient :

* Gestion complète des mesures de glycémie
* Sur le graphique des mesures, il peut choisir la date de début et de fin et les types
* Afficher sur un graphique la tendance des mesures de glycémie
* Affichage tabulaire des mesures et de la tendance
* Gestion complète du tableau de recommandation d’insuline

# Analyse de l’existant

# Analyse fonctionnelle

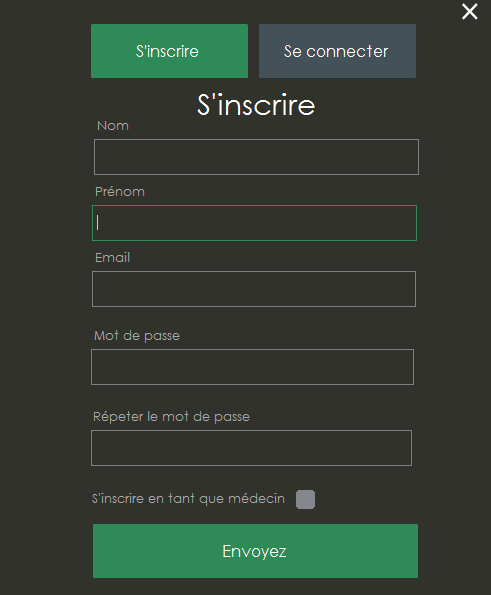
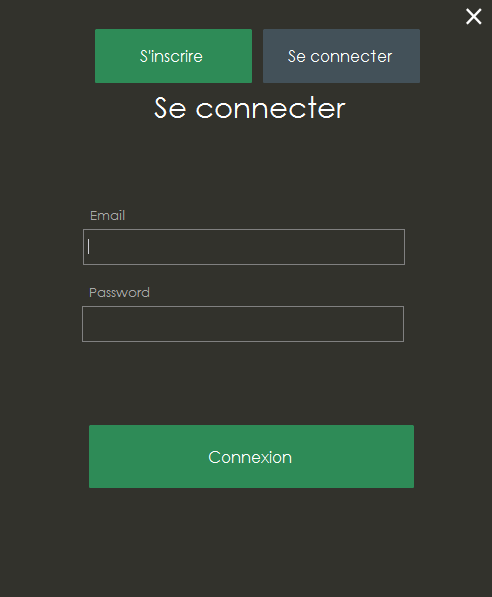
Ce programme va contenir plusieurs fonctionnalités qui offriront à l’utilisateur une aide et un suivi sur l’évolution de son diabète. Les fonctionnalités seront différentes entre un utilisateur identifié en tant que patient qu’un médecin.

## Connexion

Cela sera la première page à s’afficher lors du démarrage de l’application

L’utilisateur aura la possibilité de se créer un compte

Après avoir créé son compte l’utilisateur pourra se rendre sur « Se connecter » et se connecter à l’application

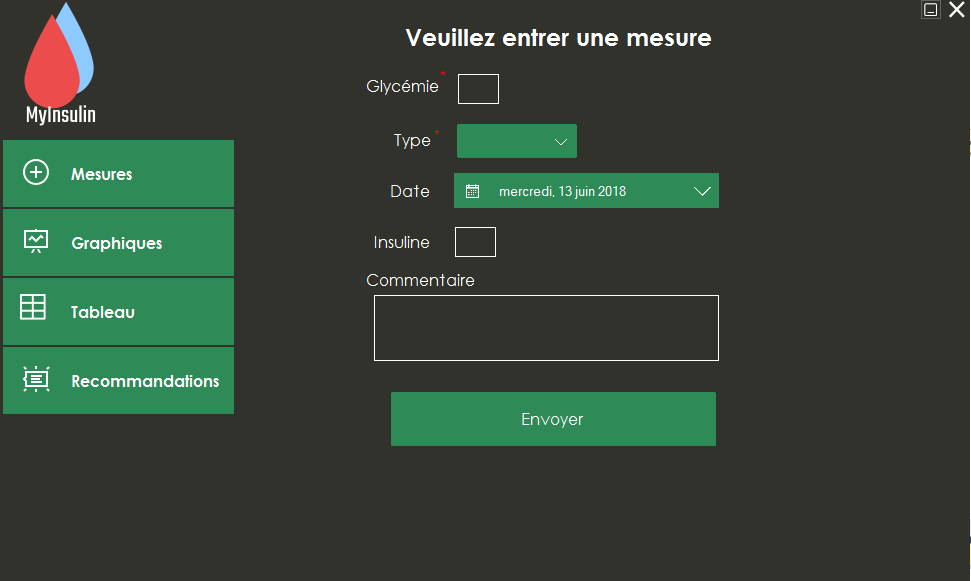


## Mesures glycémie

L’utilisateur aura la possibilité sur cette page de créer de nouvelles mesures de glycémie.

S’il valide et envois la mesure à la base de donnée, il recevra une recommandation pour savoir quel niveau d’insuline il doit s’injecter.

Dans la page « Tableau de Mesure », quand l’utilisateur cliquera sur une donnée il sera redirigée sur cette page pour modifier la mesure déjà existante.

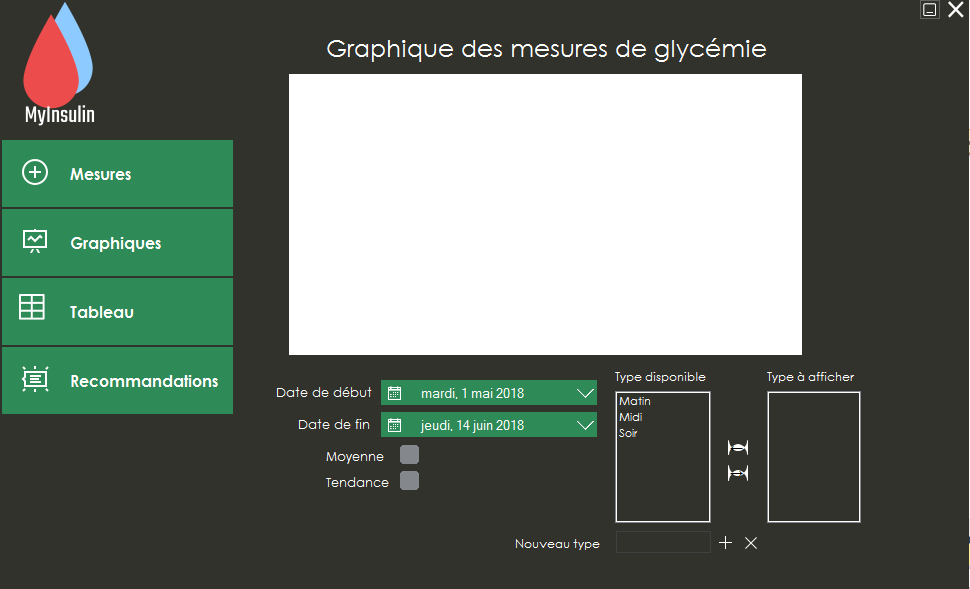


## Graphiques de mesure

Sur cette fenêtre, il sera possible de consulter ses mesures à partir d’un graphique. L’utilisateur peut choisir la date de début et de fin.

Il faut aussi laisser à l’utilisateur le choix de trier les différents types de courbe à afficher

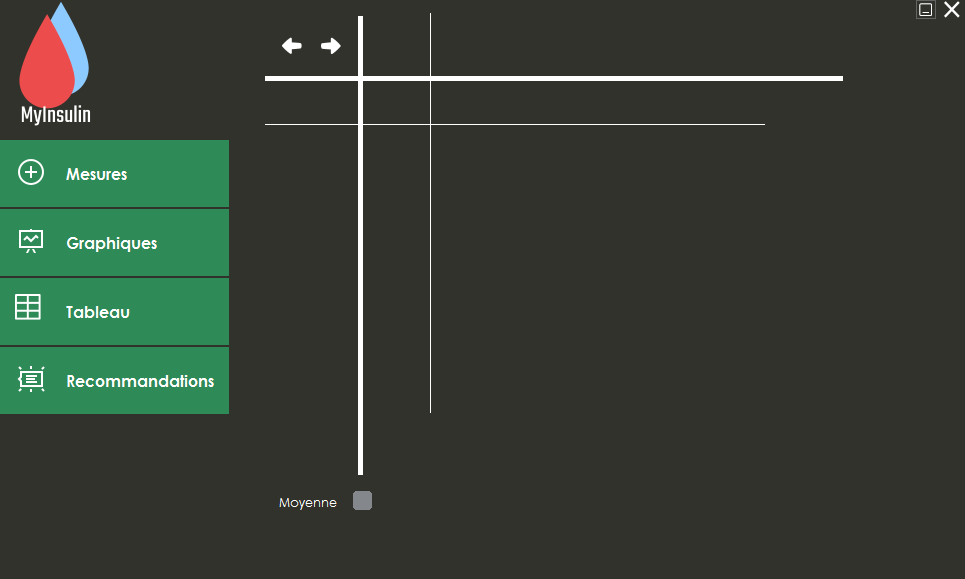
Et pour finir il peut aussi afficher ses données sous la forme d’une courbe de tendance



## Tableau de mesures

L’utilisateur aura aussi la possibilité de consulter ses mesures sous forme tabulaire.

Les données qui seront affiché seront toutes celles qui remonte à 7 jours avant la dernière mesure faites. Il aura la possibilité de se promener de semaine en semaine.



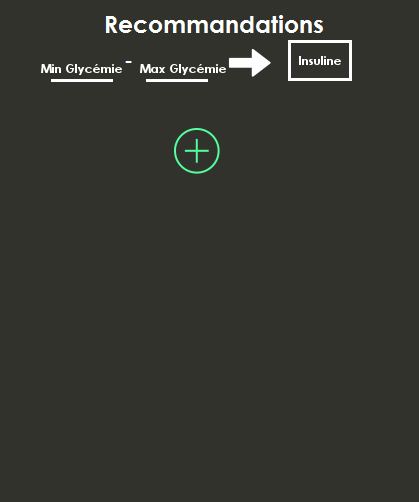
Si une valeur doit être changé, il suffit de cliquer sur la mesure et l’utilisateur sera redirigé sur la page de « Gestion de mesure ».

## Tableau d’insuline

Cette page permettra de changer le tableau de recommandation d’insuline d’un patient.

Il pourra ajouter, modifier et supprimer de nouvelles recommandations.

Ce tableau servira à recommander correctement l’utilisateur lorsqu’il entrera une nouvelle donnée.



## Ajout d’un patient

L’utilisateur aura un accès à cette page uniquement s’il s’est connecté en tant que médecin.

Cette page lui permettre d’ajouter à un patient et de pouvoir accéder à son suivi.



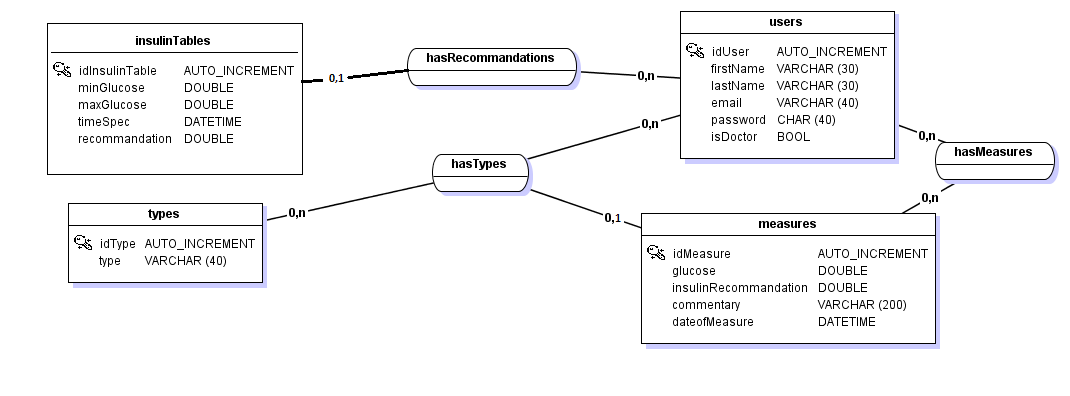
# Analyse organique

## Framework

Pour réaliser l’interface de cette application je me suis servie d’une librairie de composant graphique C# nommée Bunifu. Cela me permet de créer un interface plus ergonomique et plus design que ce qu’il y a de base sur Visual Studio. Pour utiliser cette bibliothèque il faudra payer une somme de 250$ pour une année d’utilisation sur Visual Studio, ayant découvert cette bibliothèque il y a de ça 7 mois je l’ai acheté quand il ne coutait encore que 50$.

## Base de données

Pour réaliser cette base de données je me suis inspiré du MCD fournies dans le cahier des charges et j’y ai ajouté une relation entre la tables `users`, `measures` et `types`



En développant le MCD nous aurons les 5 tables suivantes :

### Users

La table « users » contient les informations personnelles de l’utilisateur : son nom (`lastName`), son prénom (`firstName`) et son `email`, ces trois champs sont en Varchar. Elle contient aussi le mot de passe qui est de type de char et d’une longueur de 40 car le mot de passe est haché en sh1 pour éviter de laisser paraitre en clair dans la base de données. Ensuite vient `isDoctor` qui est de type boolean et permet de déterminer si l’utilisateur est un médecin ou un patient. Et pour finir il y aura le champ `idMedecin` qui sera remplie par l’id du médecin d’un utilisateur. Cela restreint un médecin par utilisateur.

### Measures :

Cette table est composée des différentes valeurs nécessaire qu’un patient diabétique se doit de remplir lorsque qu’il prend ses mesures. Il y a comme champ `glucose`, qui représente le niveau de glucose dans le sang d’une personne et `recommandationInsulin` qui donne la quantité d’insuline par rapport à un taux de glycémie. Ensuite vient le champ `dateofMeasure` pour fixer une date à la mesure. Et pour finir un champ `commentaire ` si l’utilisateur voudrait laisser une information complémentaire.

### InsulinTable :

La table représente les recommandations d’insuline d’un patient. Elle se compose des champs suivants. Il y’a d’abord `minGlucose` qui est le minimum de glycémie, puis `maxGluxose` qui lui est le max de glycémie et qui à eux deux forment un interval. Et ensuite il y a `recommandation` qui donne la recommandation d’insuline par rapport aux deux valeurs définies précédemment. Ainsi que `timeSpec` qui donne la date à la quel elle a été créé ou modifié et pour l’utilisateur qui a fait cette recommandation.

### Types :

Cette table permet de stocker les différents types de période horaire qu’un utilisateur souhaite ajouter à ses mesures Elle se compose d’un champ `type` qui représente le nom du de la période. Cela permet d’avoir plusieurs types différents et non pas des valeurs fixées

### HasType

Cette table est créé par la relation « 0,n/0,n » en la table `users` et `types`, elle est composée des id des deux tables. Cette table permet de définir des types à un utilisateur et qu’il puisse gérer ses propres types.

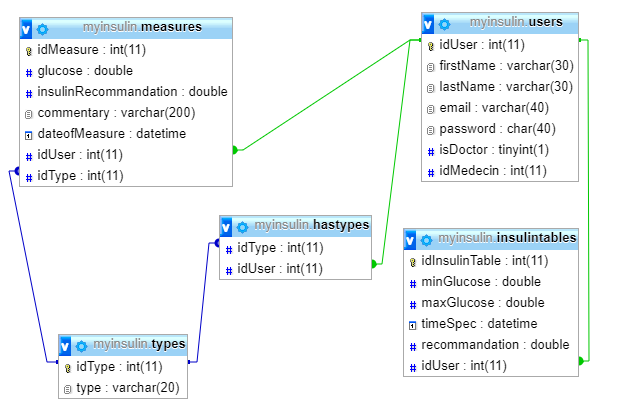
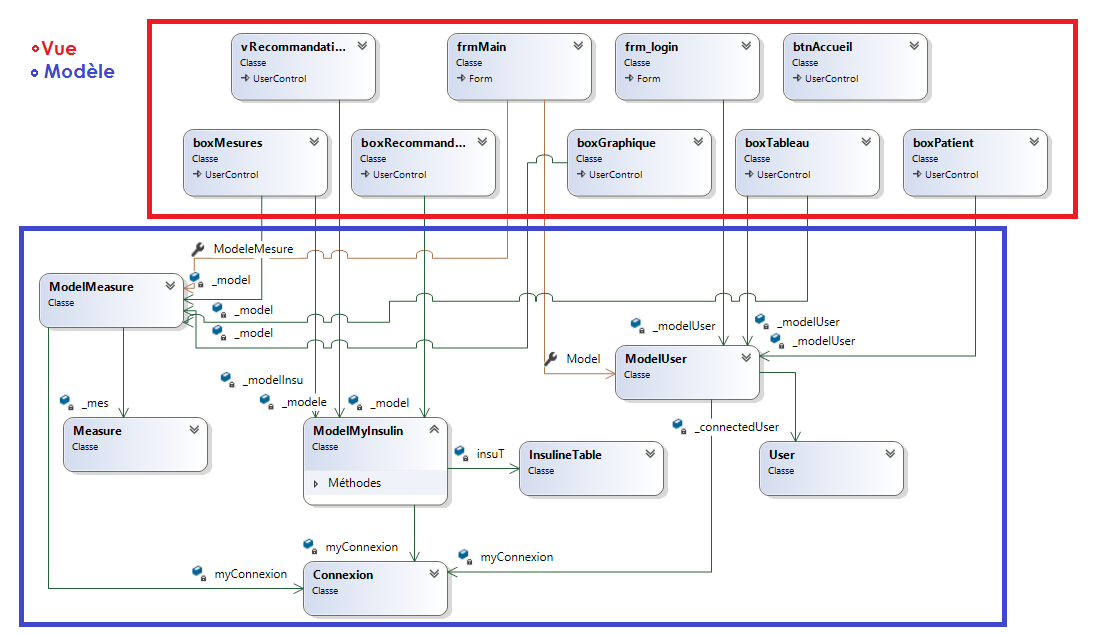


Figure 1 Vue depuis le concepteur de PhpMyAdmin

## Modélisation

Pour le développement j’ai opté pour une architecture Modèle-Vue, le modèle ne s’occupant uniquement des données et la vue doit uniquement afficher correctement les données fournies par le modèle. Voici un diagramme de classe avec une représentation visuelle du MV que j’ai mis en place pour ce projet :



## frmLogin

La classe « frmLogin » hérite du type « Form »et apparatient donc à la vue car elle va s’occuper d’afficher la fenêtre d’inscription ou de connexion à l’utilisateur. Elle va instancier le « ModelUser » pour lui envoyer les données des formulaires, lorsque l’utilisateur se connecte il va passer son « ModelUser » en paramètre au « frmMain ».

### frmMain

Cette classe hérite aussi du type « Form », elle est la fenêtre principale et va s’occuper d’afficher les différentes vues. Les fenêtres qu’elle doit afficher sont des contrôles utilisateurs de Visual Studio que j’ai créé moi-même qu’elle va ajouter ou enlever de son panel principal. Lorsque « frmMain » est instancié, il reçoit en paramètre le « ModelUser » et instancie un nouveau « ModelMeasure ». La page se doit de gérer un affichage différent lorsque l’utilisateur est un médecin ou bien un patient d’où l’utilité d’un modèle gérant les utilisateurs.

### boxMeasure

Ceci est un contrôle utilisateurs que j’ai créé afin de simplifier l’affichage sur la page principale. Cette classe permet à un utilisateur de saisir les différentes données qui compose une mesure de glycémie d’un diabétique. Elle recevra en paramètre le « ModelMeasure » instancié dans le « frmMain » ainsi qu’un boolean qui permet de savoir si la page est en modification pour l’utilisateur ou non.

### boxGraphique

Cette vue est encore un contrôle utilisateur qui va afficher les différentes courbes des mesures d’un utilisateur. Elle instancié dans la fenêtre principale et recevra en paramètre le ModelMeasure de la vue.

### boxTableau

Cette classe est un contrôle utilisateur que j’ai créé afin d’afficher à un utilisateur ses mesures de glycémie de la semaine sous formes tabulaire. Lorsqu’elle sera instanciée par le « frmMain » elle recevra en paramètre le ModelMeasure pour les données à afficher sur le tableau et le ModelUser pour savoir si l’utilisateur est un médecin ou patient afin de bloquer ou la redirection sur la « boxMesure ».

### boxRecommandation

La fenêtre « box recommandation » permet d’afficher à un utilisateur ses recommandations d’insuline par rapport à un intervalle de glycémie. Elle va instancier le « ModelMyInsulin » à sa création afin de permettre à l’utilisateur de gérer complétement son tableau d’insuline

### boxPatient

Cette vue va s’afficher uniquement si l’utilisateur connecté est médecin car cette page va permettre d’ajouter un patient à sa liste. Elle va être appelé par le « frmMain » et aura comme paramètre un ModelUser qu’il recevra de la fenêtre principale afin d’y ajouter le patient.

### btnAccueil

Ce composant qui est un contrôle utilisateur que j’ai créé afin de simplifier le dynamisme de la fenêtre principale et qui font office de boutons de navigation pour l’application. Il est composé de 3 propriétés, premièrement il y le nom affiché du bouton, deuxièmement vient l’image qui sert d’icone au bouton et finalement il y a sa position qui permet de positionner le bouton par rapport au nombre à afficher.

### vRecommandation

Ce contrôle utilisateur permet de représenter une recommandation d’insuline avec son intervalle de glycémie. Elle sera générée par la page « boxRecommandation » et récupérera le « ModelMyInsulin » de la box. Elle est composée par 7 propriétés, il y a d’abord Le Model des recommandations récupérer au moment de l’initialisation ensuite vient l’id de la recommandation qui sera utile lors de la modification ou suppression. Ensuite vient l’intervalle qui sera représenté deux valeurs, une pour le glycémie minimum une autre pour le maximum. Apres l’intervalle nous aurons la recommandation d’insuline pour un patient ainsi que l’id de l’user pour récupérer les valeurs du tableau de l’user. Et pour finir un booléen qui permet de déterminer s’il faut modifier ou ajouter le composant à la base.

### ModelMeasure

Cette classe est le modèle qui fait le lien entre la vue et les table des mesures et des types. Elle comprend toutes les différentes fonctions pour gérer l’ajout, la suppression, l’affichage ou la modification les mesures ou les types d’un utilisateur. La classe contient deux propriétés, la première étant un objet de type « Measure » afin récupérer les données de la base et de les remettre à la vue plus facilement. Ensuite vient un objet de type « User » qui sera instancié dans le constructeur, il permettra de savoir l’id de l’utilisateur connecté mais aussi son rôle(médecin/patient). Mais cette identifiant peut varier de celui présent dans le modèle pour les utilisateurs quand il s’agit d’un médecin de connecter. Il faut donc stocker l’identifiant du medecin dans le « ModelUser » et celui du patient dans celui-ci.

### ModelUser

Ce « ModelUser » est la classe qui va lier l’affichage et la base de données. Il contient toutes les requêtes pour ajouter, afficher, modifier ou bien même supprimer un utilisateur. Il est le premier modèle à être instancié car il est utilisé dès que l’application se lance sur la page de connexion et d’inscription. Il n’aura qu’une propriété de type User permettant de déterminer l’identifiant et le rôle d’un utilisateur. Cette propriété ne changera jamais après la connexion d’un utilisateur contrairement au modèle des mesures.

### ModelMyInsulin

Cette classe est le lien entre la table des recommandations d’insuline et la vue. Elle est composée d’une propriété, une de type « InsulinTable » qui va récupérer les données de la table spécifique. Il sera instancié lorsque l’utilisateur voudra entrer une donnée afin de savoir quel est la recommandation d’insuline par rapport à son niveau d’insuline. Le dernier cas est lorsque l’utilisateur voudra modifier son tableau de recommandation, la vue appellera donc ce modèle pour gérer les valeurs du tableau.

### User

Cette classe permet au modèle de stocker un enregistrement de la table `users` dans une seule variable alors que la table contient plusieurs valeurs de types différents. Cette classe possède 5 propriétés. La première est l’identifiant de l’utilisateur, toutes les requêtes qui fonctionnent par rapport à un utilisateur utilise l’id pour chercher les données. La seconde est le prénom, suivie du nom et de l’email qui sont utiles lors de la connexion ou de l’inscription. Et pour finir la valeur booléenne qui détermine si l’utilisateur est un médecin ou un patient.

### Measure

La classe « Measure » va servir comme celle précédente à stocker un enregistrement mais cette fois ci ce sera de la table `measures`. Elle est composée de 6 propriétés, la première est l’identifiant de la mesure qui est utilisé pour toutes les fonctions en rapport avec les utilisateurs sauf au niveau de l’ajout de nouvelle mesure. Ensuite vient le niveau de glycémie, la recommandation d’insuline et un commentaire. Ensuite viens la date qui va être très importante car à chaque fois que l’on va devoir afficher une mesure il va falloir utiliser les dates. Et pour finir il y’aura le type de la mesure qui représente la période horaire de la mesure qui sera utile lorsqu’il l’utilisateur voudra une vue graphique ou tabulaire de ses données.

### InsulinTable

Cette classe va servir à simplifier l’affichage des recommandations et représente la table `insulintable`. Elle sera utilisée dans le modèle des recommandations, lorsqu’elle est instanciée dans le modèle, elle a 4 paramètre qui vont remplir les différentes propriétés de la classe. Il y a en premier l’identifiant comme sur les classes précédentes, il sera utile lorsque l’utilisateur voudra modifier ou supprimer une recommandation. Ensuite vient l’intervalle de glycémie représenté par une valeur minimale et une valeur maximale et pour finir vient le niveau de recommandation qui donne le niveau d’insuline qu’un patient doit prendre sur un certain intervalle.

### Connection

La connexion à la base de donnée depuis l’application se fait par cette classe. Elle possède deux propriétés, la première est de type « MySqlConnection » et elle permet de créer le lien entre la base et la classe. Et la dernière est une chaîne de caractère qui dit à l’utilisateur s’il y a eu une erreur au niveau de la connexion à la base.

## Fonctions importantes

### Affichage des mesures :

Pour l’affichage il y’aura plusieurs méthodes qui vont être utilisée pour récupérer les bonnes données ainsi que les afficher d’une façon cohérente.

#### GetMesureDate()

Elle recevra en paramètre une chaine de caractère qui représentera la date. Cette méthode a pour but d’exécuter la requête Sql suivante :

SELECT avg(`glucose`) as glucose FROM `measures` WHERE `dateofMeasure` = @date

Elle va essayer de récupérer la moyenne dans un try catch car dans le cas où il n’y aurait pas de moyenne sur une date donnée, il retournera une moyenne de zéro et la vue saura interpréter cette valeur correctement. Cette méthode sera utilisée dans la page des graphiques pour afficher la tendance et la moyenne.

#### GetMesureByType()

Cette méthode va recevoir trois paramètres, deux seront une chaîne de caractère, une pour la date de début et l’autre pour la date de fin. Le dernier paramètre est l’identifiant du type à afficher. Elle va exécuter la requête suivante :

SELECT \* FROM `measures` WHERE `idUser` = @idUser AND `dateofMeasure` >= @dateDeb AND `dateofMeasure` <= @dateFin AND `idType`= @type

Elle va retourner une liste de d’objet de type «Measure», cette liste sera l’équivalent des mesures du type choisi par l’utilisateur qui se trouve entre les deux dates sélectionnés. Cette méthode sera utilisée dans « boxGraphique », elle permet de récupérer les données à afficher à l’user.

#### GetMesureWeekly()

Dans cette méthode nous aurons trois paramètres, le premier est la date de départ ensuite vient l’intervalle qui va être ajouté ou soustrait à la date et le dernier est l’unité que vous allez fixer à votre intervalle (DAY/MONTH/YEAR). Tous ces paramètres seront inclus dans la requête suivante :

SELECT \* FROM `measures` WHERE `dateofMeasure` < @date and `dateofMeasure` > DATE\_SUB( @date , INTERVAL @interval " + unit + ") and idUser = @idUser ORDER BY `dateofMeasure` ASC

Le résultat qui est retourné est à nouveau de liste de mesure, il sera utile dans la « boxTableau » car nous aurons toutes les mesures de la semaine sur la quel l’utilisateur se trouve dans son tableau.

#### GetMesureById()

La méthode ne reçoit qu’un paramètre qui est l’identifiant d’un type et exécutera la requête suivante :

SELECT \* FROM `measures` WHERE idMeasure = @idMesure

Elle va nous retourner toutes les valeurs qui compose une mesure avec comme identifiant celui passé en paramètre. Cette méthode est utilisée lorsqu’un patient qui se trouve sur la « boxTableau » clique sur une des valeurs du tableau, il sera redirigé sur la page « boxMesure » avec les données de la requête.

### Ajout, modification et suppression de mesure :

Pour l’ajout de mesures il va y’avoir deux méthode importante.

#### VerifMeasure()

Dans cette méthode il y aura deux paramètres. Le premier une chaîne de caractère qui représentera la date dans la requête et le deuxième est le nom de la période de la mesure. Ces paramètres seront ajoutés à la requête suivante :

SELECT idMeasure FROM `measures` WHERE `dateofMeasure` = '" + date + "' AND `idType` ='" + GetTypeByName(type) + "

Dans cette requête nous allons récupérer un certain nombre d’identifiant par rapport à la date et à leur type. Par la suite la méthode va vérifier le nombre d’enregistrement retourné et dès qu’il y a un enregistrement la méthode nous retournera une valeur booléenne à « false ». Alors que si la requête ne retourne aucun enregistrement, elle retourna un booléen à « true ». Cette méthode servira à vérifier lors de l’insertion ou modification s’il n’y pas déjà une valeur dans la table. La méthode GetTypeByName() permet de récupérer l’id du type en donnant une string et étant donné que la table demande l’id il faut appeler cette méthode.

#### ChangeMesure()

Pour cette méthode il y aura 6 paramètres. En premier la glycémie puis l’insuline ainsi que le commentaire et viens ensuite le type et la date qui sont des chaînes de caractère et pour finir l’identifiant de la mesure à modifier. Les paramètres cités iront tous dans la requête suivante :

UPDATE `measures` SET `glucose`=" + glucose + " , `insulinRecommandation`=" + insu + " , `commentary`='" + comment + "' , `idType`='" + GetTypeByName(type) + "',`dateofMeasure`='" + dateMesure + "' WHERE `idMeasure`=" + idMesure + "

Cette fois ci la méthode ne va rien retourner, elle va juste aller modifier dans la base l’enregistrement équivalent à l’identifiant fournies en paramètres.

#### AddMesure()

Pour cette méthode il y aura 5 paramètres. A nouveau la glycémie, l’insuline, le commentaire et viens ensuite le type et la date de la nouvelle mesure. Cette méthode va exécuter la requête suivante : suivante :

INSERT INTO `measures`(`glucose`, `insulinRecommandation`, `commentary`, `dateofMeasure`, `idUser`, `idType`) VALUES (@glucose , @insulinRecommandation, @commentary , @dateMesure , @idUser, @type )

Cette fois ci la méthode ne va rien retourner à nouveau, elle va insérer dans la table `measures` un nouvel enregistrement de ce qui a été passé en paramètre.

#### DelMeasure()

Pour cette méthode le seul paramètre sera l’identifiant de la mesure que l’utilisateur souhaite effacer. Pour ce faire la méthode va exécuter la requête suivante :

DELETE FROM `measures` WHERE `idMeasure` = @idMeasure

La requête ne va rien retourner mais elle va aller dans la table `measures` pour supprimer l’enregistrement avec l’identifiant passé en paramètre.

### Gestion complète des utilisateurs

La gestion des utilisateurs, tel que l’ajout de médecin ou bien même l’inscription demande des méthodes assez importantes.

#### LoginUser()

Cette fonction est composé deux paramètres, le premier étant l’email de connexion et le deuxième étant le mot de passe. La requête exécuté est la suivante :

SELECT \* FROM `users` WHERE `email` = @email AND `password` = @password

La méthode va juste s’occuper de vérifier qu’elle récupère bien un enregistrement mais elle ne va pas retourner, elle va juste rendre une valeur booléenne à 1 lorsqu’il y a un enregistrement et 0 lorsqu’il n’y en a pas. La page de connexion va juste vérifier grâce à cette méthode si la valeur est vraie ou fausse pour connecter l’utilisateur.

#### UserExist()

Il y aura deux méthode très semblables mais chacune aura des paramètres différents. La première aura qu’un paramètre qui est l’email et la deuxième en aura deux, le nom et le prénom. Les requêtes seront les suivantes :

SELECT \* FROM `users` WHERE `email` = @email

SELECT \* FROM `users` WHERE `lastName` = @lName AND `firstName` = @fName

Les deux méthodes vont traiter la requête de la même façon, elles vont compter les enregistrements et retourner une valeur booléenne à 1 s’il n’y a aucun résultat et à 0 s’il y a un enregistrement. Cela va permettre de savoir si les paramètres sont disponibles.

#### AddUser()

Cette méthode recevra cinq paramètres, le premier est le prénom ensuite vient le nom puis l’email ainsi que le mot de passe et pour finir un booléen qui fixe le rôle de l’utilisateur. Tous les paramètres iront dans la requête suivante :

INSERT INTO `users`(`firstName`,`lastName`,`email`, `password`, `isDoctor`) VALUES (@fName, @lName, @email, @password, @isDoctor)

La requête ne va rien retourné mais un nouvel enregistrement va s’ajouté à la table `users`, l’utilisateur est donc inscrit à l’application.

#### GetidByLogin()

Cette méthode reçoit 3 paramètres, les trois sont des chaines de caractères. Le premier est le prénom ensuite le nom et pour finir le mot de passe. Elle va exécuter la requête suivante :

SELECT `idUser` FROM `users` WHERE `firstName` =@fName AND `lastName` =@lName AND `password` =@pwd

La requête va récupérer l’id d’un patient lorsqu’un médecin va ajouter un nouveau patient à sa liste à partir des données passées en paramètre.

#### Gestion des recommandations()