Rapport DM BDD

1. Organisation de la Base de données

Nous avons créé une base de données nommée “cooking” constituées de 8 tables :
client, createur, fournisseur, produit, recette, listeIngredients, plat et commande.
Le détail des tables ainsi que leurs attributs sont indiqués dans notre modèle relationnel.

Il nous a semblé pertinent de séparer les tables client et createur car un créateur étant un client particulier, il aura accès à de nombreuses fonctionnalités qui lui sont propres. Dans tout notre projet, nous préférons donc bien distinguer ces deux types de personnes (client d’une part et createur de l’autre).

Nous avons créé deux tables distinctes plat et commande pour gérer les commandes d’un client. En effet, nous avions pour optique de donner la possibilité à un client de pouvoir commander plusieurs plats lors d’une seule commande - c’est à dire de pouvoir créer un “panier” constitué par plusieurs plats, avant de valider et payer sa commande - plutôt que de créer une nouvelle commande à chaque fois qu’il sélectionne une nouvelle recette.
Un plat diffère d’une recette car il possède un attribut “quantité” et il représente une réalisation de cette recette.
Une commande est quant à elle constituée de plusieurs plats.

Lors de la création de nos tables, nous avons hésité à utiliser la fonction AUTO_INCREMENT pour nos clés primaire, applicable uniquement à des attributs de type INT. Nous avons préféré utiliser des attributs de type VARCHAR pour nos clés primaires car nous voulions des syntaxes d’identifiants propres à chaque table : “PL1” pour les plats, “R1” pour les recettes, “C1” pour les clients… Cela nous a un petit peu handicapé par la suite lors de la création de nouveaux plats, recettes, clients… En effet, cela implique l’utilisation d’une requête de notre base de données afin de récupérer le dernier élément créé pour pouvoir en déduire ensuite le nouvel identifiant.

```
SELECT idRecette FROM recette where length(idRecette) IN (SELECT max(length(idRecette)) from recette) ORDER by idRecette desc limit 1;
```

2. Options de codage et développements complémentaires

Pour l’aspect graphique, nous avons choisi le WPF car il nous a semblé plus intéressant de pouvoir interagir avec une interface plutôt qu’un menu affiché sur la console.
Le fait de pouvoir naviguer aisément entre plusieurs pages (en particulier les pages Espace Client et Espace Créateur) est l’un des avantages apportés par l’interface.
Alban avait quelques notions de WPF, vues au premier semestre cette année. Cependant, Morgane n’avait aucune notion car elle a effectué son premier semestre à l’étranger, mais le fait de découvrir un nouveau langage l’a fortement motivée à partir sur un projet WPF, malgré son évolution en tant que fonctionnalité optionnelle du projet.

3. Organisation du code

Notre code est divisé en 19 classes : 9 classes .xaml.cs et 10 classes .cs.

- Les 9 classes .xaml.cs :

(MainWindow.xaml.cs, PageCdr.xaml.cs, PageClient.xaml.cs, PageConnexion.xaml.cs, PageCreationCdR.xaml.cs, PageCreationCompte.xaml.cs, PageCreationRecette.xaml.cs, PageDemo.xaml.cs et PageGestionnaire)

La création d'un nouvelle page WPF implique la génération de 2 fichiers, un .xaml ainsi qu'un .xaml.cs, ceux-ci permettent de faire le lien entre l'affichage et notre base de données. Le fichier .xaml comprend les balises nécessaires à l'organisation des éléments graphiques (nous avons surtout utilisé les widgets suivants : textblock, textbox, datagrid, button).
Le fichier .xaml.cs comprend un Main initialisant les différents widgets et leurs données ainsi que des fonctions définissant les actions à réaliser lors de nos interactions avec la page (clics sur des boutons ou écriture d'un texte dans nos textbox ou sélection d'éléments dans des combobox...). Dans un premier temps nous avons créé une Window pour contenir nos différentes pages, celle-ci navigue directement vers la page de connexion. Selon les choix de l'utilisateur, il est possible de se créer un compte, de se connecter en tant que Client ou Gestionnaire, ou encore d'accéder au mode démo.
Chaque client aura la possibilité de commander des produits ou d'accéder à l'espace Créateur. Si le client est déjà créateur, il verra son solde de Cook et un récapitulatif de ses recettes. Il aura également la possibilité de créer une nouvelle recette. Si le client n'est pas déjà créateur, il pourra le devenir en créant sa première recette.
Quant au gestionnaire de Cooking, il aura accès à diverses fonctionnalités telles que suppression de recette ou de créateur, réapprovisionnement des stocks...

- Les 10 classes .cs :

(RootClass.cs, IdUtilisateurActif.cs, Client.cs, Createur.cs, Produit.cs, Recette.cs, ListeIngredients.cs, Plat.cs, CréateurEtCommandes.cs et TypesRecettes.cs)

Nous avons créé **une** classe statique nommée RootClass.cs, contenant les fonctions liées aux requêtes SQL, qui pourront donc être utilisées à n'importe quel endroit du projet sans devoir réécrire leur implémentation à chaque fois.
Elle contient notamment les fonctions qui permettent d'accéder à notre base de données afin d'y sélectionner, insérer, modifier ou supprimer des données.
On trouve déjà une simple fonction ouvrant la connexion.
Il y a également des fonctions de style ExécutionRequête…() prenant en argument une commande SQL (SELECT, INSERT, DELETE,...).
Seules les fonctions prenant en paramètre des commandes de type SELECT (Langage de Requête) renvoient un Reader. La connexion est donc laissée ouverte à ce moment-là. Elle sera refermée après l'extraction des données du reader.
Les fonctions prenant en paramètre des commandes de type INSERT, UPDATE ou DELETE (Langage de Manipulation de Données) ouvrent la connexion, effectuent les modifications de la base de données nécessaires, puis ferment la connexion. Ce sont des fonctions de type "void", elles ne renvoient rien.

Nous avons également créé **une** classe statique nommée IdUtilisateurActif.cs qui permet d'avoir accès à chaque instant à l'idClient de l'utilisateur qui s'est connecté.
Par ailleurs, nous avons intégré à presque toutes nos pages un bouton "Déconnexion" qui permet de retourner à la page d'accueil. L'IdUtilisateurActif sera mis à jour dès qu'un autre client se connectera.

Nous avons créé des classes .cs pour chacune des tables de notre base de données dont les éléments devaient être affichés dans un tableau (datagrid) sur les pages .xaml.
En effet, pour ajouter des éléments dans un tableau, il faut utiliser un Binding particulier pour chaque colonne. Ce binding correspond à l'un des attributs de nos classes. C'est pourquoi les **6** classes : Client.cs, Createur.cs, Produit.cs, Recette.cs, ListeIngredients.cs et Plat.cs ont été créées, mais pas les 2 classes commande.cs et fournisseur.cs (car nous n'avons jamais eu besoin d'afficher un attribut de ces 2 classes dans un tableau).

Nous n'avons pas réussi à afficher des attributs de classe différente dans un même tableau.
Par exemple, pour le mode Démo, il fallait afficher le nom du créateur (nomClient, qui est un attribut de la table client) ainsi que son nombre total de commandes (sum(compteur), compteur étant un attribut de la table recette). Nous avons donc créé **une** classe supplémentaire, CréateurEtCommandes.cs, qui contient ces 2 attributs.

Nous avons voulu créer une comboBox pour afficher les types des recettes créées (Entrée, Plat, Dessert…). Les comboBox utilisant également des Binding, nous avons créé **une** classe TypesRecettes.cs.