Vue Tâche

Choix d’implémentations :

La vue tâche consiste à afficher les tâches créer par l’utilisateur. Afin de se faire nous avions pour but d’aller récupérer les différentes tâche stockées dans notre base de donnée à l’aide de requête SQL. Pour cela nous avons créé une classe sqlConnection qui permet de se connecter à la base de données mais également à réaliser certaine action telles que récupérer toute les tâches. Cette façon de procédée est simple d’implémentation mais demande beaucoup de ligne de code surtout lorsqu’il faut ensuite afficher ces données. C’est pour cette raison que nous avons décidé non pas de créer nous-même nos requêtes SQL mais d’utiliser les fonctionnalités offertes par le Framework Qt. En effet ce Framework nous met à disposition un outil très puissant qui est les classes QSqlTableModel ainsi que QSqlRelationTableModel.

QSqlTableModel

Cette classe nous permet de lié un model a une base de donnée. En d’autre mot nous lions chaque champ de notre model Task avec un champ dans notre base de donnée. Ceci a pour but que lorsque nous chargeons une tâche depuis la base de donnée, cette dernière est instanciée dans notre modèle. Il est ensuite extrêmement pratique de réaliser des actions sur notre modèle, actions qui seront répercutées directement sur notre base de donnée.

L’étape suivante est de pouvoir afficher ces données et une fois encore Qt nous met à disposition des méthodes permettant de lier une vue à un modèle (mapper). C’est de cette manière que nous pouvons réaliser des changements depuis l’interface de l’utilisateur (par exemple créer une nouvelle tâche) qui se répercuteront directement dans notre base de données. Cela sans devoir à créer chacune des commandes SQL requise pour ces opérations.

QSqlRelationTableModel

Ce deuxième outils nous a permis d’améliorer l’affichage de nos données en utilisant le moins possible de requête SQL et donc en gardant un code plus simple et plus lisible. QSqlRelationTableModel nous est utile lorsque nous avons une Foreign Key dans notre table SQL. En effet si nous utilisons seulement QSqlTableModel nous stockerions dans le model les informations qui sont dans une table de notre base de donnée et donc pour le modèle de tâche au lieu de nous d’obtenir le nom du cours auquel notre tâche est lié nous obtiendrons son identifiant. Il faudrait pouvoir ensuite rechercher cette id et récupérer le nom auquel elle correspond. C’est exactement à cela que sert QSqlRelationTableModel. Nous lions une colonne dans une table SQL à une autre colonne dans une autre table. Ceci a pour effet d’afficher le nom du cours plutôt que son identifiant.

Tri des tâches

Dans le but de pouvoir trier nos tâches par ordre d’échéance ou par ordre de priorité, nous avons utilisé une fonctionnalité qu’offre Qt et la classe QSqlTableModel est la méthode sortBy qui trie notre model, lié à une table SQL, en fonction de la colonne dans cette table (par exemple la priorité).

Problèmes rencontrés

Lors de notre implémentation de la vue tâche nous avons remarqué, que lors d’un tri des tâches, la tâche sélectionnée était différente avant et après le tri. Ceci est dû au fait que la sélection concerne la ligne sur laquelle se trouve la tâche. En raison d’un retard sur le développement de notre application nous n’avons pas eu le temps de régler ce problème, qui n’empêche en rien au bon fonctionnement de notre application. Cependant nous avons cherché un moyen de résoudre cela et avons trouvé que Qt offrait la possibilité de trier nos modèles en gardant la sélection. Ceci ce ferai à l’aide de la classe QSortFilterProxyModel cette couche se situe entre le model et la vue et nous permettrai de garder la sélection entre les différents tris et filtres. A cause de manque nous n’avons pu implémenter cette classe dans notre application et nous avons donc implémenté une solution non définitive mais fonctionnelle.