Record

Revenons à la BDD. Pour mieux faire, plutôt que de manipuler des indices de valeurs pour les requêtes, on peut utiliser des objets `QSqlRecord`.

Ces objets permettent de manipuler les champs d'un enregistrement, voire d'en ajouter ou d'en supprimer.

Ils permettent notamment d'obtenir un index de colonne en fonction d'un nom, et d'éviter le problème de `value()`.

Exemple sur notre base SQLite :

Listing 22 : Utilisation de `QRecord`

```
1   QSqlQuery query(chaineRequete);
2   QSqlRecord record = query.record();
3
4   qDebug() << "nombre de colonnes " << record.count();
5
6   int nameCol = record.indexOf("nom"); //index de la colonne "nom"
7   qDebug() << "indice de nom " << nameCol;
8
9   while (query.next())
10      qDebug() << query.value(nameCol).toString(); //affiche toutes les chaînes de la colonne "nom"
```


Plan intermédiaire

Introduction

Qt permet d'utiliser des fichiers structurés en lecture et écriture. Les types lus sont définis par Qt, ce qui facilite la manipulation des données.
JSON et XML font partie de ceux-là. XML est « un peu » verbeux, nous nous focaliserons sur le premier.

Au cas où :

- `https://fr.wikipedia.org/wiki/Extensible_Markup_Language`
- `https://fr.wikibooks.org/wiki/Programmation_XML/Introduction`

JSON

JSON (*JavaScript Object Notation*) est un format de données (pas forcément sous forme de fichier), structuré par des balises ({}, "" ou []).

Il est constitué de 4 types simples : `null`, chaînes de caractères, nombre, booléen. Et 2 types composés des précédents : objet (imbrication) ou liste (séquence).

Il est utilisé massivement pour l'échange des fichiers pour le web, la configuration de logiciels (« `prefs.json` » de thunderbird), la sérialisation de valeurs. C'est un format simple, lisible et « facile » à utiliser (liste, clé-valeur, structuration). Il n'a pas de contrainte sur la cohérence à part les blocs de balises.

Référence :
`https://fr.wikipedia.org/wiki/JavaScript_Object_Notation`
Le support dans différents langages :
`https://www.json.org/json-fr.html`

Avec Qt

Qt propose des `QJsonDocument` pour manipuler ces données. On pourra ensuite filtrer/analyser un document de ce type par champ par exemple.

Le format géré par Qt est une représentation du JSON (en `ByteArray` pour assurer le portage sur plusieurs encodages). Un `QByteArray` est le container standard d'une lecture de fichier par Qt.

Par exemple :

```
QByteArray donnees;
QJsonDocument doc = QJsonDocument::fromJson(donnees, &error);
```

Le `fromJson` s'assure donc que le balisage du texte est conforme, que des champs existent, avec des valeurs.

Référence : `https://doc.qt.io/qt-5/qjsondocument.html`

Extraire des valeurs ?

Une fois le `QJsonDocument` instancié, on peut lire les valeurs, en les liant avec leurs types. Par exemple pour une recette de cuisine, on trouvera un nom (`texte`), une description (`texte`), les ingrédients (`tableau de texte`) et les instructions (`tableau de texte`).

```
QJsonDocument doc = QJsonDocument::fromJson(donnees, &error);
qDebug() << doc.isObject();

QJsonObject obj=doc.object();

qDebug() << "Nom " << (obj.value("name")).toString();
qDebug() << "Description : " << (obj.value("description")).toString();

QJsonValue val = obj.value("recipeIngredient");
```

On voit que, là-aussi, comme pour la BDD, `QJsonObject` est un `QVariant`.

Un tableau de valeurs

Si on tombe sur un tableau de valeurs, il faut alors l'extraire pour l'analyse, c'est le cas de notre liste d'ingrédients :

```
QJsonValue val = obj.value("recipeIngredient");
qDebug() << val.isArray();

QJsonArray valArray = val.toArray();

//parcours
for (auto value : valArray)
    qDebug() << "ingredient " << value.toString();

val = obj.value("recipeInstructions");
valArray = val.toArray();

//parcours
for (auto value : valArray)
    qDebug() << "instruction " << value.toString();
```

Là-encore, les `QVariant` sont de mise.

Gestion des erreurs

`QJsonParseError` est utilisé à la création du `QJsonDocument`.

Constant	Value	Description
QJsonParseError::NoError	0	No error occurred
QJsonParseError::UnterminatedObject	1	An object is not correctly terminated with a closing curly bracket
QJsonParseError::MissingNameSeparator	2	A comma separating different items is missing
QJsonParseError::UnterminatedArray	3	The array is not correctly terminated with a closing square bracket
QJsonParseError::MissingValueSeparator	4	A colon separating keys from values inside objects is missing
QJsonParseError::IllegalValue	5	The value is illegal
QJsonParseError::TerminationByNumber	6	The input stream ended while parsing a number
QJsonParseError::IllegalNumber	7	The number is not well formed
QJsonParseError::IllegalEscapeSequence	8	An illegal escape sequence occurred in the input
QJsonParseError::IllegalUTF8String	9	An illegal UTF8 sequence occurred in the input
QJsonParseError::UnterminatedString	10	A string wasn't terminated with a quote
QJsonParseError::MissingObject	11	An object was expected but couldn't be found
QJsonParseError::DeepNesting	12	The JSON document is too deeply nested for the parser to parse it
QJsonParseError::DocumentTooLarge	13	The JSON document is too large for the parser to parse it
QJsonParseError::GarbageAtEnd	14	The parsed document contains additional garbage characters at the end

FIG. : valeurs de QJsonError

Référence : `https://doc.qt.io/qt-5/qjsonparseerror.html`

Un exemple

Un schéma de données existe donc pour les recettes de cuisine :
`https://schema.org/Recipe`

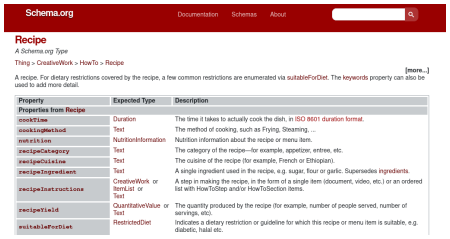

FIG. : Schéma de recette

Nextcloud Cookbook utilise notamment ce schéma :
`https://apps.nextcloud.com/apps/cookbook`
Un analyseur lit la page web d'une recette et extrait les mots-clés, les range ensuite dans un fichier JSON.

Les données enregistrées

Voilà par exemple une recette de Mayonnaise :

Listing 23 : Mayonnaise.json

```
1 {"name" :"Mayonnaise","description" :"Un classique pour accompagner vos plats froids ! Une recette inratable !","url" :"https
    :\/\/www.750g.com\/mayonnaise-r4143.htm","image" :"https :\/\/static.750g.com\/images\/600-600\/
    d77754742240fb4a00c7551215694 0ab\/gettyimages-623436518.jpg","prepTime" :"PT0H10M","cookTime" :"
    PT0H0M","totalTime" :"PT0H10M","recipeCategory" :"Sauces","keywords" :"vinaigrette \u00e0 la mayonnaise,
    mayonnaise sans moutarde,mayonnaise,sauces,\u0153ufs,moutarde,sauces froides,Chef Damien","recipeYield" :6,"
    recipeIngredient" :["3 jaunes d'\u0153ufs","1\/2 litre d'huile de tournesol","20 gr de moutarde","2 cl de vinaigre de vin
    rouge","Sel","Poivre blanc du moulin"],"recipeInstructions" :["\u00c9tape 1 :","Mettre les jaunes d'oeufs dans un petit
    saladier. Caler votre saladier avec un torchon. \u00c9tape 2 :","Saler et poivrer. Ajouter la moutarde et le vinaigre. \
    u00c9tape 3 :","Fouetter \u00e9n\u00e9rgiquement et ajouter tr\u00e8s progressivement la moiti\u00e9 de l'huile en
    filet. \u00c9tape 4 :","Ajouter plus rapidement le reste de l'huile en continuant de fouetter. \u00c9tape 5 :","Utiliser ou r\
    u00e9server au r\u00e9frig\u00e9rateur votre sauce mayonnaise."],"id" :"2147534","@context" :"http :\/\/schema.org","
    @type" :"Recipe","tool" :[]}
```

Application

Construisons une application QT qui lise et analyse ce type de fichiers.

Let's go...

Plan intermédiaire

Introduction

Doxygen est un logiciel libre, qui permet de documenter « facilement » tous types de projets de développement. Il repose sur une écriture des commentaires du code et sur l'analyse des sources : La documentation de ce système est disponible ici : `https://www.doxygen.nl/manual/index.html`

Doxygen vous permettra également de sorti les graphes d'héritage, d'appel, de dépendance entre entités de vos sources.

Attention, il ne remplace pas `javadoc` pour Java, les contraintes des projets sont parfois exclusivement des documentation issues de la `javadoc`.

Organisation

Il faut installer `Doxygen` sur les machines, plus `dot` et `graphviz` pour la génération des diagrammes, plus `texlive` (un environnement LaTeX) éventuellement pour générer un PDF.
Ensuite, en ligne de commande, dans le répertoire du projet :

```
1 doxygen
```

Cela ne va pas fonctionner... `Doxygen` a besoin d'un fichier de configuration pour spécifier les analyses à faire, les types de fichiers, les répertoires des sources, les sorties...

Une configuration par défaut est possible par :

```
1 doxygen -g <fichierConf>
```


Exécution

Si vous ne mettez pas de `fichierConf`, le nom par défaut est `Doxyfile`. Ensuite, à chaque fois que vous voulez lancer l'analyse et la création de la documentation :

```
1 doxygen <fichierConf>
```

Note : si vous ne donnez pas le fichier de configuration `doxygen` utilisera `Doxyfile`.

Doxywizard

La sortie par défaut, sans dépendance à d'autres bibliothèques est HTML. Si vous voulez modifier des paramètres, vous pouvez éditer le `Doxyfile`.

Cela peut être vite rébarbatif vu le nombre de lignes. L'outil `doxywizard` est un GUI à `doxygen`. Il permet de régler les paramètres, de créer/charger un fichier de configuration, de lancer la génération de la documentation et de l'afficher.

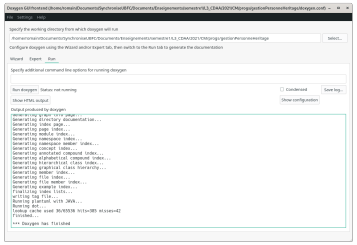

FIG. : Doxywizard

Syntaxe(s)

Plusieurs syntaxes sont possibles pour démarrer un commentaire « enrichi » :

- comme en Javadoc (**je préfère celle-là**) :

```
/**
 * ... text ...
 */
```

- comme ce que fait Qt

```
/*!
 * ... text ...
 */
```

- comme C++

```
///
/// ... text ...
///
```

ou

```
//!
//!... text ...
//!
```

- ou même !

```
/************//**
 *   ... text
 ***************/
```


Commentaires plus évolués

Les commentaires précédents vont être « vus » par doxygen et remplir la documentation pour chaque entité (source, classe, fonction) se trouvant immédiatement après dans le code.

Des *tags* sont également disponibles pour structurer la documentation, pour indexer les choses à faire, spécifier un auteur...

Exemple :

Listing 24 : Documentation d'une classe

```
1  /**
2   * @class Contact
3   * hérite de Personne (public)
4   * @brief Classe Contact
5   * @todo passer tout en Anglais
6   * @todo ajouter mail, photo
7   */
8  class Contact : public Personne
9  {
10     /**
```


Plus complexe

Listing 25 : Documentation d'un membre

```
1  /**
2   * (Commentaire général) IF (\f$ entreprise==c.entreprise \f$ AND \f$nom==c.nom\f$ AND \f$prenom==c.prenom\f$...)
3   * RETURN TRUE
4   * ELSE
5   * RETURN FALSE
6   * autre ligne
7   * – liste0
8   * – liste1
9   * – sous–liste1_0
10  * – sous–liste1_1
11  * – liste2
12  * @author R. Raffin
13  * @date 14/10/2021
14  * @brief test d'égalité entre 2 contacts
15  * fonction const, operator==(Contact)
16  * @return un booléen VRAI si égalité
17  * @see https ://www.cplusplus.com/doc/tutorial/operators/
18  * @todo utiliser l'héritage pour tester la partie Personne, cf #Personne
19  */
20 bool Contact : :operator==(const Contact & c) const
21 {
22     if ( getEntreprise() == c.getEntreprise())
```


Quelques tags

- `@class` pour documenter une classe,
- `@struct` pour documenter une structure C,
- `@fn` pour documenter une fonction,
- `@param` pour documenter un paramètre de fonction/méthode,
- `@return` pour documenter les valeurs de retour d'une méthode/fonction,
- `@var` pour documenter une variable/un `typedef`/un énuméré,
- `@def` pour documenter un `#define`,
- `@namespace` pour documenter un `namespace`,
- `@brief` pour donner une description courte,
- `@warning` pour attirer l'attention,
- `@author` pour donner le nom de l'auteur,
- `@see` pour renvoyer le lecteur vers quelque chose (une fonction, une classe, un fichier, une URL...),
- `@exception` pour documenter une exception,
- `@todo` pour indiquer une opération restant «< à faire »,
- `@fixme` pour indiquer un code défectueux, « à réparer »,
- ...

Source : `https://fr.wikipedia.org/wiki/Doxygen`

Application

Utilisons Doxygen sur le TD4 : classe `Contact`, `Personne` et `Gestion Personne` (sans Qt).

Let's go...

Commande dans QTCreator

Il y a possibilité dans QTCreator, de créer des commandes externes et d'y associer un contrôle clavier. Dans le menu : `Outils` -> `Externe` -> `Configure`, on rajoute une entrée :

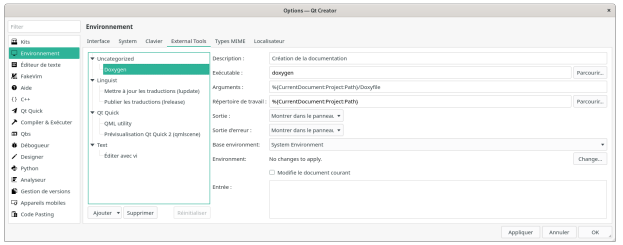

FIG. : Création d'une commande externe

Raccourci dans QTCreator

Il n'y a plus ensuite qu'à relier un raccourci clavier à cette commande, et la génération de la documentation est accessible depuis QTCreator.

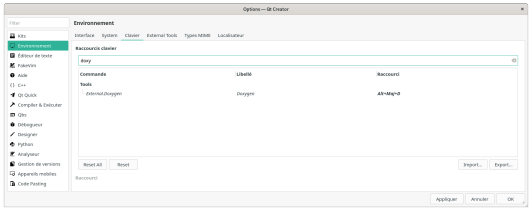

FIG. : Édition d'un raccourci clavier pour Doxygen

Note : on a généralement une cible des `Makefile` « `make doc` » qui sert à générer la documentation.