Практическое задание №5.

Расширение функционала MDI-приложения QTabelo для работы с таблицами.

Цель работы: Изучить архитектуру открытого MDI-приложения для работы с документами и расширить его функциональность, реализовав работу с таблицами.

Задание:

Склонировать репозиторий проекта QTabelo – шаблона приложения для работы с таблицами, написанного на C++ с использованием фреймворка Qt: https://github.com/beletalabs/qtabel . Изначально это приложение включает: поддержку нескольких документов (MDI — Multiple Document Interface) во вкладках, систему меню и настраиваемую панель инструментов, работу с файлами, настройками (QSettings), а также базовую структуру документа с поддержкой нескольких табличных листов. Но реальной функциональности в приложении пока нет.

- Реализуйте создание таблиц с помощью QTableWidget.
- Добавьте возможность редактирования данных таблицы пользователем, добавление и удаления строк и столбцов.

Порядок выполнения:

1) Откройте проект в Qt Creator и убедитесь, что он компилируется и запускается без ошибок. В данном проекте класс DocumentManager (наследник QMdiArea) управляет несколькими документами, а каждый документ представлен классом DocumentWindow, содержащим DocumentWidget, который в свою очередь наследуется от TableDocument , содержащий QTabWidget. То есть изначально реализована структура вкладок, но таблиц в них пока нет.
2) В методе slotAddTab(int count) класса TableDocument замените создание пустых виджетов на добавление таблицы, например:

```
if (m_tabs->count() == 0) {
        for (int i = 0; i < count; ++i) {
            QTableWidget* table = new QTableWidget(this);
            table->setRowCount(15);  // Стартовое количество строк
            table->setColumnCount(5); // Стартовое количество столбцов

            // Устанавливаем заголовки
            QStringList headers;
            for (int col = 0; col < 5; ++col) {
                headers << QString("Column %1").arg(col + 1);
            }
            table->setHorizontalHeaderLabels(headers);

            table->setAttribute(Qt::WA_DeleteOnClose);
            m_tabs->addTab(table, tr("Sheet %1").arg(i + 1));
        }
        m_tabs->setTabsClosable(m_tabs->count() > 1);
    }
```

Рисунок 28. Фрагмент слота для добавления таблицы

Здесь m_tabs является указателем на объект класса QTabWidget, который является членом класса TableDocument. Он хранит и управляет вкладками, каждая из которых содержит таблицу QTableWidget. Посмотрите конструктор класса TableDocument и обратите внимание на инициализацию этого указателя.

В цикле for (int i = 0; i < count; ++i) создаётся count новых таблиц (QTableWidget). Условие if (m_tabs->count() == 0) гарантирует, что таблицы создаются только при первом вызове метода (если вкладок ещё нет).

Далее установка атрибута Qt::WA_DeleteOnClose для таблицы обеспечит её автоматическое удаление при закрытии вкладки, а метод addTab() добавляет таблицы как вкладки в m_tabs (основной контейнер вкладок). Название вкладки генерируется автоматически на основе номера i.

Запустите приложение и убедитесь, что теперь при создании нового документа появляется созданная таблица:

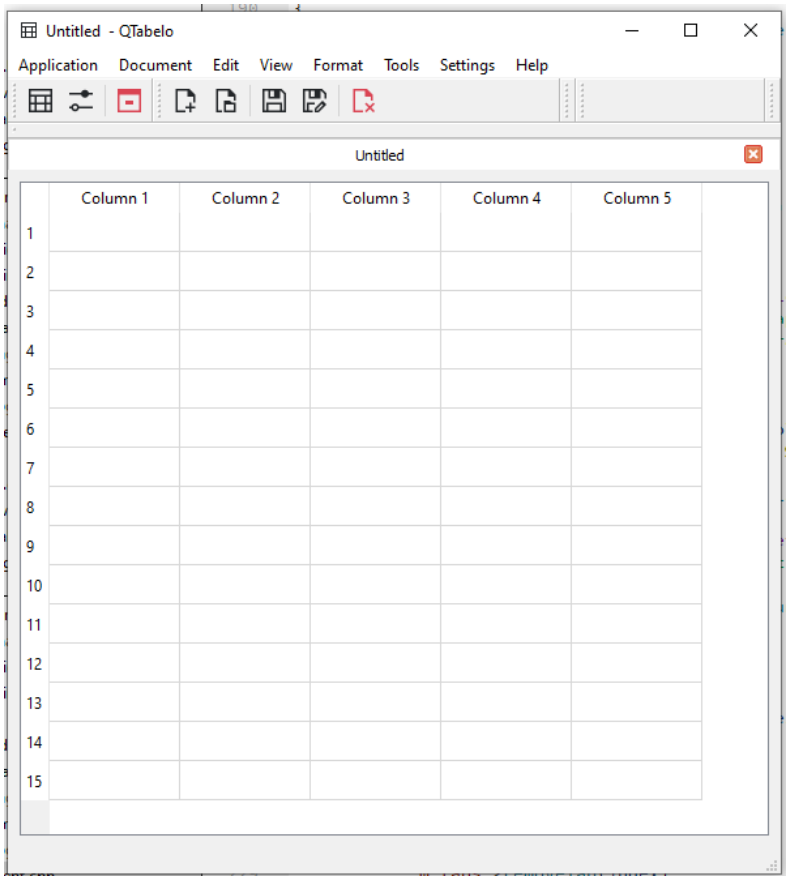

Рисунок 29. Новый документ с таблицей

3) Далее нужно реализовать возможность редактирования числа строк и столбцов таблицы. Сделать это лучше через контекстное меню, причем изменения должны влиять только на текущую таблицу. В заголовочный файл table_document.h добавьте слот:

   void showTableContextMenu(QTableWidget *table, const QPoint &pos) для реализации контекстного меню. В этом слоте нужно будет не только создать действия, но и предусмотреть то, что когда вы добавляете строки или столбцы через контекстное меню в TableDocument, нужно уведомлять DocumentWidget, что ваш документ изменился. Обратите внимание, что класс DocumentWidget имеет свойство modified (флаг изменения) и сигнал modifiedChanged(). Но так как TableDocument создаёт таблицы и не знает о свойстве modified, нужно добавить сигнал изменения содержимого, например, contentChanged (bool modified), в заголовочный файл table_document.h.

Тогда реализация контекстного меню будет выглядеть так:

```
//слот реализации контекстного меню
void TableDocument::showTableContextMenu(QTableWidget *table, const QPoint &pos)
{
    QMenu menu;

    QAction *actionAddRow = menu.addAction(tr("Add Row"));
    QAction *actionRemoveRow = menu.addAction(tr("Remove Last Row"));
    QAction *actionAddColumn = menu.addAction(tr("Add Column"));
    QAction *actionRemoveColumn = menu.addAction(tr("Remove Last Column"));

    QAction *selectedAction = menu.exec(table->mapToGlobal(pos));

    if (!selectedAction)
        return;

    if (selectedAction == actionAddRow) {
        table->setRowCount(table->rowCount() + 1);
        emit contentChanged(true); // Уведомляем, что документ модифицирован
    }
    else if (selectedAction == actionRemoveRow && table->rowCount() > 0) {
        table->setRowCount(table->rowCount() - 1);
        emit contentChanged(true);
    }
    else if (selectedAction == actionAddColumn) {
        int newColIndex = table->columnCount();
        table->setColumnCount(newColIndex + 1);
        table->setHorizontalHeaderItem(newColIndex, new QTableWidgetItem(QString("Column %1").arg(newColIndex + 1)));
        emit contentChanged(true);
    }
    else if (selectedAction == actionRemoveColumn && table->columnCount() > 0) {
        int colCount = table->columnCount();
        table->setColumnCount(colCount - 1);
        emit contentChanged(true);
    }
}
```

Рисунок 30. Слот для контекстного меню

Затем в конструкторе класса DocumentWidget нужно добавить эту связь между сигналом contentChanged из TableDocument и вызовом setModified для уведомления о любом изменении содержимого таблицы:

```
DocumentWidget::DocumentWidget(QWidget *parent)
    : TableDocument(parent), m_modified{false}, m_url{QUrl()}
{
    setAttribute(Qt::WA_DeleteOnClose);
    connect(this, &TableDocument::contentChanged, this, &DocumentWidget::setModified);
}
```

Рисунок 31. Конструктор класса DocumentWidget

Здесь используется this, потому что DocumentWidget наследуется от TableDocument.

Теперь в слот TableDocument::slotAddTab() нужно добавить подключение контекстного меню:

```
void TableDocument::slotAddTab(const int count)
{
    if (m_tabs->count() == 0) {
            for (int i = 0; i < count; ++i) {
                auto *table = new QTableWidget(15, 5); // Таблица 15x5
                table->setObjectName(QStringLiteral("Sheet%1").arg(i + 1));

                // Устанавливаем заголовки
                QStringList headers;
                for (int col = 0; col < 5; ++col) {
                    headers << QString("Column %1").arg(col + 1);
                }
                table->setHorizontalHeaderLabels(headers);

                table->setContextMenuPolicy(Qt::CustomContextMenu);

                // Подключаем контекстное меню
                connect(table, &QTableWidget::customContextMenuRequested,
                        this, [this, table](const QPoint &pos) {
                    showTableContextMenu(table, pos);
                });

                // Подключаем сигнал изменения ячейки
                connect(table, &QTableWidget::itemChanged,
                        this, [this](QTableWidgetItem*) {
                    DocumentWidget *doc = qobject_cast<DocumentWidget*>(this->parent());
                    if (doc) {
                        doc->setModified(true);
                    }
                });
                m_tabs->addTab(table, tr("Sheet %1").arg(i + 1));
            }
        }
}
```

Рисунок 32. Слот для добавления таблицы с контекстным меню

Этот метод является ключевой частью реализации таблицы, и теперь в нем подключается контекстное меню, открывающееся по правой кнопке мыши. То есть при вызове события customContextMenuRequested открывается меню через метод showTableContextMenu(), а при изменении любой ячейки генерируется сигнал itemChanged().Через qobject_cast<DocumentWidget*>(this->parent()) получаем родительский документ и вызываем setModified(true), то есть документ помечается как изменённый.

4) Протестируйте работу полученного приложения, проверив следующие случаи:
    - Убедитесь, что при создании нового документа появляется таблица заданного размера с заголовками столбцов.
    - Заполните ячейки данными.
    - Добавьте/удалите строку и столбец.

Дополнительное задание: добавить возможность удаления всех выделенных столбцов и строк.

Контрольные вопросы:

1) Что такое MDI и как он реализован в Qt?
2) Как реализуется контекстное меню в Qt?
3) Какие сигналы и слоты используются для отслеживания изменений в таблице?
4) Как работает механизм modifiedChanged и зачем он нужен?
5) Как реализуется связь между TableDocument и DocumentWidget?