

Практическое задание №6.

Расширение функционала MDI-приложения QTableo для работы с CSV.

Цель работы: Добавить в приложение QTableo функциональность для импорта/экспорта табличных данных в формате CSV.

Задание:

- Реализовать загрузку данных из CSV-файла в таблицу.
- Реализовать сохранение данных таблицы в CSV-файл.
- Интегрировать новые методы в существующие слоты loadDocument() и saveDocument()

Порядок выполнения:

- 1) Реализуйте сохранение данных:
 - В классе ApplicationWindow уже имеются слоты для сохранения и загрузки документа – loadDocument() и saveDocument(), но в них нет реализации. Также имеется слот importCsvToTable() для импорта табличных данных, без реализации. Нужно добавить слот exportTableToCsv() для экспорта таблиц, а также реализовать все необходимые действия для перечисленных методов.
 - Реализуйте экспорт таблицы из QTableWidget в CSV-файл. Для этого нужно открыть файл с переданным именем для записи, используя QFile и QTextStream, сохранить заголовки таблицы в список, затем сохранить данные строк таблицы и записать файл. Пример реализации данного слота:

```
bool ApplicationWindow::exportTableToCsv(QTableWidget *table, const QString &filename)
{
    QFile file(filename);
    if (!file.open(QIODevice::WriteOnly | QIODevice::Text))
        return false;

    QTextStream out(&file);

    // Сохраняем заголовки
    QStringList headers;
    for (int col = 0; col < table->columnCount(); ++col) {
        headers << table->horizontalHeaderItem(col)->text();
    }
    out << headers.join(",") + "\n";

    // Сохраняем строки
    for (int row = 0; row < table->rowCount(); ++row) {
        QStringList rowText;
        for (int col = 0; col < table->columnCount(); ++col) {
            QTableWidgetItem *item = table->item(row, col);
            rowText << (item ? item->text() : "");
        }
        out << rowText.join(",") + "\n";
    }

    file.close();
    return true;
}
```

Рисунок 33. Слот для экспорта таблицы

- Обновите слот saveDocument() в application_window.cpp. В нем можно добавить поиск активной таблицы через иерархию виджетов, используя метод findChild(), а также необходим вызов реализованного слота для экспорта таблицы. При успешном сохранении вызывайте метод resetModified(), который сбрасывает флаг «измененности» документа и указывает, что все изменения сохранены. Этот метод в проекте QTableo играет ключевую роль в управлении состоянием документа и вызывается при создании пустого документа, после успешного сохранения и при открытии нового документа. Пример реализации слота saveDocument():

```
bool ApplicationWindow::saveDocument(DocumentWidget *document, const QUrl &altUrl)
{
    Q_UNUSED(altUrl)
    document->resetModified();
    // Определяем URL для сохранения
    QUrl url = altUrl.isEmpty() ? document->url() : altUrl;

    if (!url.isValid()) {
        QMessageBox::critical(this, tr("Error"), tr("Invalid file location."));
        return false;
    }

    // Получаем текущую таблицу через безопасный метод
    QWidget *table = nullptr;
    if (auto tabs = document->findChild<QTabWidget*>()) {
        table = qobject_cast<QTableWidget*>(tabs->currentWidget());
    }

    if (!table) {
        QMessageBox::critical(this, tr("Error"), tr("No table data to save."));
        return false;
    }

    // Экспортируем данные
    if (!exportTableToCsv(table, url.toLocalFile())) {
        QMessageBox::critical(this, tr("Error"), tr("Could not save the file."));
        return false;
    }
    document->resetModified();
    document->setUrl(url);
    m_recentDocuments->addUrl(url);
    return true;
}
```

Рисунок 34. Слот сохранения таблицы

- 1) Реализуйте импорт таблицы из CSV-файла в соответствующем слоте importCsvToTable() файла application_window.cpp. Для этого откройте файл для чтения, пользуясь QFile и QTextStream. Чтобы загрузить новую структуру таблицы, сначала вызываются методы очистки ячеек, строк и столбцов. Затем происходит построчное считывание данных и результат записывается в список строк, где первая строка содержит

заголовки столбцов, а последующие строки – данные. Пример

реализации слота для импорта таблицы:

```
bool ApplicationWindow::importCsvToTable(const QString &filename, QTableWidget *table)
{
    QFile file(filename);
    if (!file.open(QIODevice::ReadOnly | QIODevice::Text)) {
        QMessageBox::critical(this, tr("Error"), tr("Could not open the file."));
        return false;
    }

    QTextStream in(&file);

    table->clearContents();
    table->setRowCount(0);
    table->setColumnCount(0);

    int row = 0;
    while (!in.atEnd()) {
        QString line = in.readLine();
        QStringList fields = line.split(',');

        if (row == 0) {
            // Устанавливаем заголовки
            table->setColumnCount(fields.size());
            table->setHorizontalHeaderLabels(fields);
        } else {
            table->insertRow(table->rowCount());
            for (int col = 0; col < fields.size(); ++col) {
                if (col >= table->columnCount())
                    table->insertColumn(table->columnCount());
                table->setItem(row - 1, col, new QTableWidgetItem(fields[col]));
            }
        }
        ++row;
    }

    file.close();
    return true;
}
```

Рисунок 35. Слот для импорта таблицы

- Обновите слот loadDocument() в application_window.cpp. В нем сначала создается новый документ, затем с помощью встроенного механизма documentCountChanged() создается новая вкладка и инициализируется виджет таблицы. Затем происходит поиск таблицы в иерархии виджетов, аналогично как было реализовано при сохранении, и вызывается метод импорта данных из CSV. Пример реализации слота loadDocument():

```

bool ApplicationWindow::loadDocument(const QUrl &url)
{
    DocumentWidget *document = createDocument();
    if (!document) return false;

    // Добавляем таблицу через стандартный механизм
    document->documentCountChanged(1); // Используем существующий метод

    // Получаем таблицу через доступ к m_tabs
    QTableWidget *table = nullptr;
    if (document->findChild<QTabWidget*>()) { // Проверяем наличие QTabWidget
        QTabWidget *tabs = document->findChild<QTabWidget*>();
        table = qobject_cast<QTableWidget*>(tabs->currentWidget());
    }

    if (!table) {
        document->close();
        return false;
    }

    //Загружаем данные
    if (!importCsvToTable(url.toLocalFile(), table)) {
        document->close();
        return false;
    }

    document->show();
    document->setUrl(url);
    m_recentDocuments->addUrl(url);
    documentCreated();

    return true;
}

```

Рисунок 35. Слот открытия таблицы

2) Протестируйте приложение, следуя плану:

- Создайте таблицу, заполните её данными и сохраните как CSV-файл.
- Закройте текущую таблицу.
- Проверьте загрузку через меню "Файл → Открыть".
- Измените данные в таблице.
- Сохраните таблицу через "Файл → Сохранить".
- Проверьте содержимое сохраненного файла.

Дополнительное задание: Перевести QTableWidget на QTableView с кастомной моделью, унаследованной от QAbstractTableModel.

Контрольные вопросы:

- 1) Какие этапы включает загрузка данных из CSV в QTableWidget?
- 2) Какие классы в QTableWidget отвечают за управление документами?
- 3) Как связаны между собой классы ApplicationWindow, DocumentManager и DocumentWidget при работе с CSV?
- 4) Какие сигналы и слоты задействованы при импорте/экспорте таблиц?
- 5) Какие улучшения можно добавить в функциональность работы с CSV?