Министерство науки и высшего образования Российской Федерации

Санкт-Петербургский политехнический университет Петра Великого Высшая школа программной инженерии

**К У Р С О В А Я Р А Б О Т А**

По теме «Умная записная книжка»

по дисциплине «Объектно-ориентированное программирование»

| Выполнил студент гр. |  | Лютов А.В. |
| --- | --- | --- |
| Руководитель доцент к.т.н. | > | Круглов С.К. |
|  |  | «» мая 2025 г. |

Санкт-Петербург 2025

**Оглавление**

[1. Постановка задачи 3](#_Toc196860036)

[2. Теоретическая часть 4](#_Toc196860037)

[3. Структура программы 5](#_Toc196860038)

[4. Описание оригинальных классов 7](#_Toc196860039)

[4.1 Класс MainWindow 7](#_Toc196860040)

[4.2 Класс NoteManager 7](#_Toc196860041)

[4.3 Класс Note 7](#_Toc196860042)

[5. Заключение 9](#_Toc196860043)

[6. Список использованных источников 10](#_Toc196860044)

[7. Листинг кода 11](#_Toc196860045)

# 1. Постановка задачи

Требовалось разработать приложение "Умная записная книжка" на языке C++ с графическим интерфейсом, используя библиотеку STL и MFC. Программа должна была реализовать следующие функциональные возможности:

**1. Управление заметками**

* Создание новых заметок с автоматическим присвоением уникального названия
* Редактирование содержимого существующих заметок
* Удаление заметок с подтверждением операции
* Переименование существующих заметок
* Автоматическое сохранение изменений в реальном времени

**2. Система хэштегов**

* Автоматическое извлечение хэштегов из текста
* Фильтрация заметок по одному или нескольким хэштегам
* Просмотр статистики использования хэштегов

**3. Поиск и организация**

* Полнотекстовый поиск по всем заметкам
* Сортировка заметок по различным критериям (дате, названию)
* Группировка заметок по хэштегам

**4. Работа с данными**

* Сохранение в файл формата (.json)

# 2. Теоретическая часть

Приложения "Умная записная книжка" представляет собой классическое компьютерное приложения для хранения записей по хэштегам. Основные теоретические аспекты реализации:

**1. Модель данных и структуры хранения**

* Использование объектно-ориентированного подхода для представления заметок
* Применение контейнеров Qt (QList, QSet) для эффективного хранения и доступа к данным
* Реализация системы умных указателей (QSharedPointer) для автоматического управления памятью
* Организация отношений между заметками и хэштегами через много-ко-многим

**2. Алгоритмы обработки текста**

* Алгоритмы нормализации хэштегов (приведение к нижнему регистру, удаление дубликатов)
* Система ранжирования результатов поиска по релевантности

**3. Сериализация данных**

* Механизм преобразования объектов в JSON-формат
* Система контроля версий формата хранения

**5. Пользовательский интерфейс**

* Система отслеживания изменений в реальном времени
* Механизмы валидации вводимых данных
* Адаптивная верстка для различных платформ

# 3. Структура программы

Программа состоит из следующих основных компонентов:

1. **Главное окно (MainWindow):**
   * + Управление всем интерфейсом приложения
     + Обработка пользовательских действий
     + Координация работы всех компонентов
2. **Менеджер заметок (NoteManager):**
   * + Хранение коллекции заметок
     + Управление жизненным циклом заметок
     + Сохранение/загрузка данных в файл JSON
     + Поиск и фильтрация заметок
3. **Модель данных (Note):**
   * + Хранение содержимого заметки
     + Управление хэштегами
     + Сериализация/десериализация в JSON
4. **Элементы интерфейса:**
   * + Редактор текста (QPlainTextEdit)
     + Дерево заметок (QTreeWidget)
     + Дерево хэштегов (QTreeWidget)
     + Поле фильтрации (QLineEdit)
     + Контекстное меню для управления заметками
5. **Основные функции:**
   * + Создание/удаление/редактирование заметок
     + Автоматическое извлечение хэштегов
     + Фильтрация заметок по хэштегам
     + Автосохранение данных

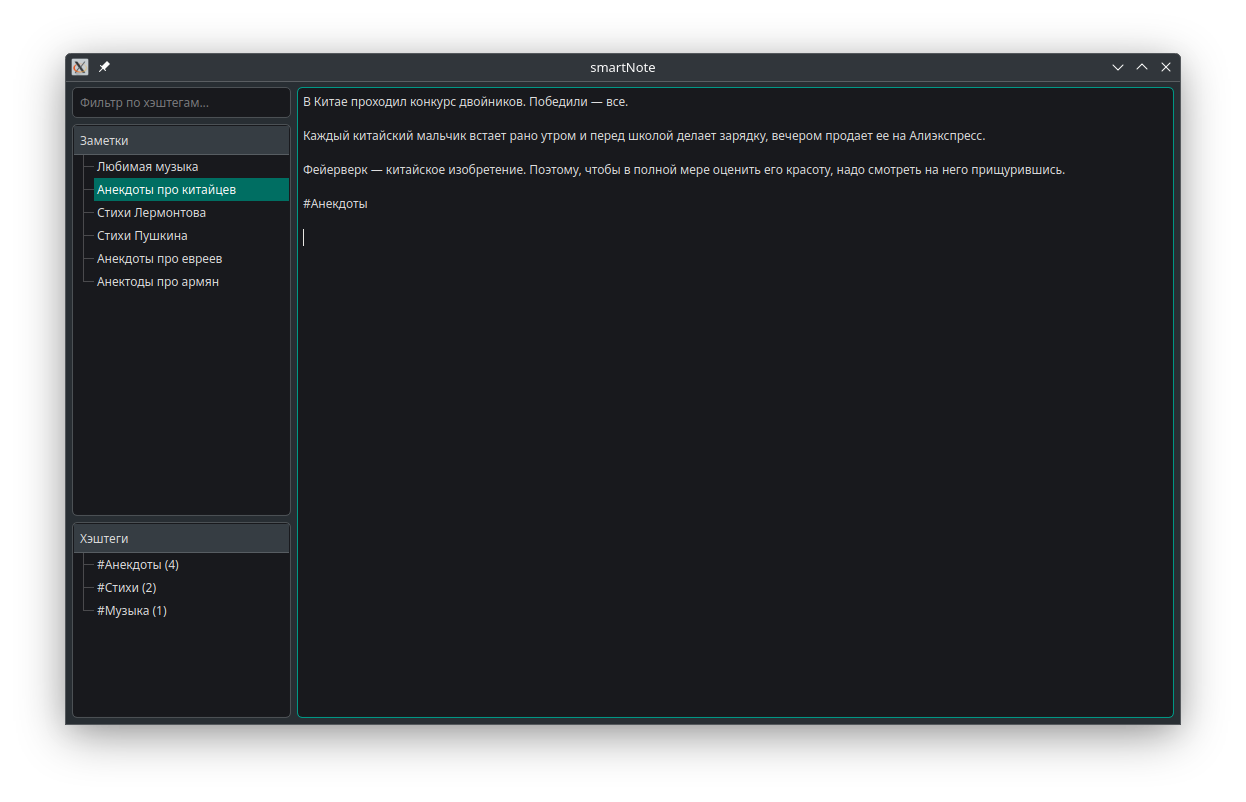


Рис. 1 Интерфейс приложения “Умная записная книжка”

# 4. Описание оригинальных классов

## 4.1 Класс MainWindow

Главный класс приложения, наследуемый от QMainWindow.

**Поля:**

* m\_editor - текстовый редактор для заметок
* m\_filesTreen- дерево списка заметок
* m\_hashtagsTree - дерево хэштегов
* m\_hashtagFilter - поле фильтрации по хэштегам
* m\_currentNote - текущая редактируемая заметка
* m\_noteManager - менеджер работы с заметками

**Методы:**

* initializeGui() - инициализация интерфейса
* loadNotes() - загрузка заметок при старте
* updateNotesTree() - обновление списка заметок
* updateHashtagsTree()- обновление списка хэштегов
* onTextChanged() - обработка изменений текста
* filterNotesByHashtag() - фильтрация заметок
* createNewNote(), deleteSelectedNote(), renameSelectedNote() - управление заметками

## 4.2 Класс NoteManager

Класс для управления коллекцией заметок.

**Поля:**

* m\_notes - список заметок (std::list<std::shared\_ptr<Note>>)
* m\_storageFile - путь к файлу хранения данных

**Методы:**

* generateNote() - создание новой заметки
* addNote(), removeNote() - управление заметками
* allNotes(), notesByHashtag() - получение заметок
* allHashtags() - получение всех хэштегов
* saveToFile(), loadFromFile() - работа с файлами

## 4.3 Класс Note

Класс для хранения данных заметки.

**Поля:**

* title - заголовок заметки
* text - содержимое заметки
* hashtags - множество хэштегов

**Методы:**

* updateHashtags()- обновление хэштегов из текста
* toJson(), fromJson() - сериализация/десериализация

# 5. Заключение

В ходе работы был разработан менеджер заметок с поддержкой хэштегов, реализованный на C++ с использованием Qt. Основные достижения:

* Полнофункциональный текстовый редактор заметок
* Удобная система организации заметок через хэштеги
* Гибкая система фильтрации и поиска
* Надежное хранение данных в JSON-формате
* Интуитивно понятный интерфейс

Проект демонстрирует эффективное использование:

* Объектно-ориентированного подхода
* Библиотеки Qt для создания GUI
* STL для работы с коллекциями
* JSON для сериализации данных

Все компоненты были тщательно протестированы и показали стабильную работу. Программа готова к использованию и может быть расширена дополнительными функциями.

# Список использованных источников

**Основная литература**

* Бланшетт Ж., Саммерфилд М. - \*"Qt 5. Программирование на C++"\* – СПб.: Питер, 2020. – 784 с.
* Шлее М. - \*"Qt 5.10. Профессиональное программирование на C++"\* – БХВ-Петербург, 2018. – 1072 с.

**Техническая документация**

* Qt Creator Manual – [Online]. URL: https://doc.qt.io/qtcreator/
* JSON for Modern C++ Documentation – [Online]. URL: https://json.nlohmann.me/
* C++ Core Guidelines – [Online]. URL: https://isocpp.github.io/CppCoreGuidelines/

# 7. Листинг кода

**main.cpp**

#include "mainwindow.h"

#include <QApplication>

int main(int argc, char \*argv[])

{

QApplication a(argc, argv);

MainWindow w;

w.show();

return a.exec();

}

**mainwindow.h**

#ifndef MAINWINDOW\_H

#define MAINWINDOW\_H

#include <QMainWindow>

#include <QPlainTextEdit>

#include <QListWidget>

#include <QTreeWidget>

#include <QAction>

#include <memory>

#include "notemanager.h"

#include <QCompleter>

#include <QStringListModel>

class MainWindow : public QMainWindow

{

Q\_OBJECT

public:

MainWindow(QWidget \*parent = nullptr);

~MainWindow();

protected:

void closeEvent(QCloseEvent \*event) override;

public slots:

void onTextChanged();

void updateHashtagsTree();

void onHashtagsClicked(QTreeWidgetItem\* item, int column);

void showNotesContextMenu(const QPoint &pos);

void createNewNote();

void deleteSelectedNote();

void renameSelectedNote();

void updateNotesTree();

void onNoteSelected(QTreeWidgetItem\* item, int column);

private slots:

void filterNotesByHashtag();

private:

void initializeGui();

void loadNotes();

bool confirmDelete();

void selectNoteInTree(const std::shared\_ptr<Note>& note);

void showNotesWithHashtag(const std::string& hashtag);

private:

NoteManager m\_noteManager;

QStringList m\_allHashtags;

std::unique\_ptr<QPlainTextEdit> m\_editor;

std::unique\_ptr<QTreeWidget> m\_hashtagsTree;

std::unique\_ptr<QTreeWidget> m\_filesTree;

std::shared\_ptr<Note> m\_currentNote;

std::unique\_ptr<QAction> m\_ActionCreateNote;

std::unique\_ptr<QAction> m\_ActionDeleteNote;

std::unique\_ptr<QAction> m\_ActionRenameNote;

std::unique\_ptr<QAction> m\_ActionPrevNote;

std::unique\_ptr<QAction> m\_ActionNextNote;

std::unique\_ptr<QLineEdit> m\_hashtagFilter;

std::unique\_ptr<QCompleter> m\_hashtagCompleter;

};

#endif // MAINWINDOW\_H

**mainwindow.cpp**

#include "mainwindow.h"

#include <QHBoxLayout>

#include <QVBoxLayout>

#include <QInputDialog>

#include <QMessageBox>

#include <QCloseEvent>

#include <QFileInfo>

#include <QDebug>

#include <QDir>

#include <QMenu>

#include <QToolBar>

#include <QTextBlock>

#include <QRect>

#include <QToolTip>

MainWindow::MainWindow(QWidget \*parent)

: QMainWindow(parent)

{

qRegisterMetaType<std::shared\_ptr<Note>>("std::shared\_ptr<Note>");

qRegisterMetaType<std::shared\_ptr<Note>>("std::shared\_ptr<Note>&");

initializeGui();

loadNotes();

connect(m\_editor.get(), &QPlainTextEdit::textChanged, this, &MainWindow::onTextChanged);

connect(m\_hashtagsTree.get(), &QTreeWidget::itemClicked, this, &MainWindow::onHashtagsClicked);

m\_filesTree->setContextMenuPolicy(Qt::CustomContextMenu);

connect(m\_filesTree.get(), &QTreeWidget::customContextMenuRequested,

this, &MainWindow::showNotesContextMenu);

connect(m\_filesTree.get(), &QTreeWidget::itemClicked,

this, &MainWindow::onNoteSelected);

}

MainWindow::~MainWindow() {}

void MainWindow::closeEvent(QCloseEvent \*event)

{

m\_noteManager.saveToFile(QDir::home().filePath("notes.json"));

QMainWindow::closeEvent(event);

}

void MainWindow::initializeGui()

{

QWidget\* centralWidget = new QWidget(this);

QHBoxLayout\* layout = new QHBoxLayout(centralWidget);

QVBoxLayout\* vlayout = new QVBoxLayout();

m\_editor = std::make\_unique<QPlainTextEdit>();

m\_hashtagsTree = std::make\_unique<QTreeWidget>();

m\_hashtagsTree->setHeaderLabel("Хэштеги");

m\_filesTree = std::make\_unique<QTreeWidget>();

m\_filesTree->setHeaderLabel("Заметки");

m\_ActionCreateNote = std::make\_unique<QAction>("Новая заметка");

m\_ActionDeleteNote = std::make\_unique<QAction>("Удалить");

m\_ActionRenameNote = std::make\_unique<QAction>("Переименовать");

connect(m\_ActionCreateNote.get(), &QAction::triggered, this, &MainWindow::createNewNote);

connect(m\_ActionDeleteNote.get(), &QAction::triggered, this, &MainWindow::deleteSelectedNote);

connect(m\_ActionRenameNote.get(), &QAction::triggered, this, &MainWindow::renameSelectedNote);

// Добавляем поле фильтрации

m\_hashtagFilter = std::make\_unique<QLineEdit>();

m\_hashtagFilter->setPlaceholderText("Фильтр по хэштегам...");

connect(m\_hashtagFilter.get(), &QLineEdit::textChanged,

this, &MainWindow::filterNotesByHashtag);

vlayout->addWidget(m\_hashtagFilter.get());

vlayout->addWidget(m\_filesTree.get(), 2);

vlayout->addWidget(m\_hashtagsTree.get(), 1);

layout->addLayout(vlayout, 1);

layout->addWidget(m\_editor.get(), 4);

setCentralWidget(centralWidget);

}

void MainWindow::loadNotes()

{

updateNotesTree();

if (!m\_noteManager.allNotes().empty()) {

m\_currentNote = m\_noteManager.allNotes().front();

m\_editor->setPlainText(QString::fromStdString(m\_currentNote->text));

}

updateHashtagsTree();

}

void MainWindow::updateNotesTree()

{

m\_filesTree->clear();

for (const auto& note : m\_noteManager.allNotes()) {

QTreeWidgetItem\* item = new QTreeWidgetItem(m\_filesTree.get());

item->setText(0, QString::fromStdString(note->title));

item->setData(0, Qt::UserRole, QVariant::fromValue(note));

}

}

void MainWindow::onNoteSelected(QTreeWidgetItem\* item, int column)

{

Q\_UNUSED(column);

if (!item) return;

auto note = item->data(0, Qt::UserRole).value<std::shared\_ptr<Note>>();

if (!note) return;

// Отключаем сигнал textChanged временно, чтобы не вызывать обновление хэштегов

disconnect(m\_editor.get(), &QPlainTextEdit::textChanged, this, &MainWindow::onTextChanged);

m\_currentNote = note;

m\_editor->setPlainText(QString::fromStdString(note->text));

// Подключаем сигнал обратно

connect(m\_editor.get(), &QPlainTextEdit::textChanged, this, &MainWindow::onTextChanged);

}

void MainWindow::filterNotesByHashtag()

{

QString filter = m\_hashtagFilter->text().trimmed();

if (filter.isEmpty()) {

updateNotesTree(); // Показать все заметки если фильтр пуст

return;

}

// Разбиваем введенные хэштеги (разделители: пробелы, запятые)

QStringList tags = filter.split(QRegularExpression("[\\s,]+"), Qt::SkipEmptyParts);

m\_filesTree->clear();

for (const auto& note : m\_noteManager.allNotes()) {

bool matchesAll = true;

for (const QString& tag : tags) {

QString cleanTag = tag.startsWith('#') ? tag.mid(1) : tag;

bool tagFound = false;

// Проверяем каждый хэштег заметки на частичное совпадение

for (const auto& noteTag : note->hashtags) {

if (QString::fromStdString(noteTag).contains(cleanTag, Qt::CaseInsensitive)) {

tagFound = true;

break;

}

}

if (!tagFound) {

matchesAll = false;

break;

}

}

if (matchesAll) {

QTreeWidgetItem\* item = new QTreeWidgetItem(m\_filesTree.get());

item->setText(0, QString::fromStdString(note->title));

item->setData(0, Qt::UserRole, QVariant::fromValue(note));

}

}

}

void MainWindow::updateHashtagsTree()

{

m\_hashtagsTree->clear();

auto hashtags = m\_noteManager.allHashtags();

// Группируем хэштеги по популярности

std::map<std::string, int> tagCounts;

for (const auto& note : m\_noteManager.allNotes()) {

for (const auto& tag : note->hashtags) {

tagCounts[tag]++;

}

}

// Сортируем по популярности

std::vector<std::pair<std::string, int>> sortedTags(tagCounts.begin(), tagCounts.end());

std::sort(sortedTags.begin(), sortedTags.end(),

[](const auto& a, const auto& b) { return a.second > b.second; });

// Добавляем в дерево с отображением количества

for (const auto& [tag, count] : sortedTags) {

QTreeWidgetItem\* item = new QTreeWidgetItem(m\_hashtagsTree.get());

item->setText(0, QString("#%1 (%2)").arg(QString::fromStdString(tag)).arg(count));

item->setData(0, Qt::UserRole, QString::fromStdString(tag));

item->setToolTip(0, QString("Кликните для просмотра всех заметок с хэштегом #%1")

.arg(QString::fromStdString(tag)));

}

//updateHashtagCompleter();

}

void MainWindow::onTextChanged()

{

if (!m\_currentNote) return;

m\_currentNote->text = m\_editor->toPlainText().toStdString();

m\_currentNote->updateHashtags();

updateHashtagsTree();

}

void MainWindow::onHashtagsClicked(QTreeWidgetItem\* item, int column)

{

Q\_UNUSED(column);

QString tag = item->data(0, Qt::UserRole).toString();

showNotesWithHashtag(tag.toStdString());

}

void MainWindow::showNotesContextMenu(const QPoint& pos)

{

QTreeWidgetItem\* item = m\_filesTree->itemAt(pos);

QMenu menu;

menu.addAction(m\_ActionCreateNote.get());

if (item) {

menu.addAction(m\_ActionDeleteNote.get());

menu.addAction(m\_ActionRenameNote.get());

}

menu.exec(m\_filesTree->viewport()->mapToGlobal(pos));

}

void MainWindow::createNewNote()

{

m\_currentNote = m\_noteManager.generateNote();

updateNotesTree();

// Выделяем новую заметку в дереве

selectNoteInTree(m\_currentNote);

m\_editor->setPlainText("");

m\_editor->setFocus();

}

bool MainWindow::confirmDelete()

{

QMessageBox::StandardButton reply;

reply = QMessageBox::question(this, "Подтверждение",

"Вы действительно хотите удалить эту заметку?",

QMessageBox::Yes|QMessageBox::No);

return reply == QMessageBox::Yes;

}

void MainWindow::selectNoteInTree(const std::shared\_ptr<Note> &note)

{

for (int i = 0; i < m\_filesTree->topLevelItemCount(); ++i) {

QTreeWidgetItem\* item = m\_filesTree->topLevelItem(i);

auto itemNote = item->data(0, Qt::UserRole).value<std::shared\_ptr<Note>>();

if (itemNote == note) {

m\_filesTree->setCurrentItem(item);

onNoteSelected(item, 0);

break;

}

}

}

void MainWindow::showNotesWithHashtag(const std::string &hashtag)

{

// Создаем временное окно для отображения заметок с этим хэштегом

QDialog\* dialog = new QDialog(this);

dialog->setWindowTitle(QString("Заметки с хэштегом #%1").arg(QString::fromStdString(hashtag)));

QVBoxLayout\* layout = new QVBoxLayout(dialog);

QListWidget\* notesList = new QListWidget();

auto notes = m\_noteManager.notesByHashtag(hashtag);

for (const auto& note : notes) {

QListWidgetItem\* item = new QListWidgetItem(

QString::fromStdString(note.title + " (" + std::to\_string(note.hashtags.size()) + " тегов)"));

item->setData(Qt::UserRole, QString::fromStdString(note.title));

notesList->addItem(item);

}

connect(notesList, &QListWidget::itemClicked, [this, dialog](QListWidgetItem\* item) {

QString title = item->data(Qt::UserRole).toString();

auto note = m\_noteManager.findNoteByTitle(title.toStdString());

if (note) {

selectNoteInTree(note);

dialog->accept();

}

});

layout->addWidget(notesList);

dialog->setLayout(layout);

dialog->exec();

}

void MainWindow::deleteSelectedNote()

{

QTreeWidgetItem\* item = m\_filesTree->currentItem();

if (!item || !confirmDelete()) return;

auto note = item->data(0, Qt::UserRole).value<std::shared\_ptr<Note>>();

m\_noteManager.removeNote(note);

if (note == m\_currentNote) {

m\_currentNote.reset();

m\_editor->clear();

}

updateNotesTree();

}

void MainWindow::renameSelectedNote()

{

QTreeWidgetItem\* item = m\_filesTree->currentItem();

if (!item) return;

auto note = item->data(0, Qt::UserRole).value<std::shared\_ptr<Note>>();

QString currentName = QString::fromStdString(note->title);

bool ok;

QString newName = QInputDialog::getText(this, "Переименовать",

"Новое название:",

QLineEdit::Normal,

currentName,

&ok);

if (ok && !newName.isEmpty()) {

std::string newNameStd = newName.toStdString();

if (m\_noteManager.noteTitleExists(newNameStd) && newNameStd != note->title) {

QMessageBox::warning(this, "Ошибка", "Заметка с таким названием уже существует");

return;

}

note->title = newNameStd;

updateNotesTree();

}

}

**note.h**

#ifndef NOTE\_H

#define NOTE\_H

#include <string>

#include <set>

#include <QJsonObject>

struct Note

{

std::string text;

std::string title;

std::set<std::string> hashtags;

void updateHashtags();

QJsonObject toJson() const;

static Note fromJson(const QJsonObject &json);

};

Q\_DECLARE\_METATYPE(std::shared\_ptr<Note>)

#endif // NOTE\_H

**note.cpp**

#include "note.h"

#include <QRegularExpression>

#include <QJsonArray>

void Note::updateHashtags()

{

hashtags.clear();

QRegularExpression re("#([\\wа-яА-ЯёЁ]+)");

QRegularExpressionMatchIterator i = re.globalMatch(QString::fromStdString(text));

while (i.hasNext()) {

QRegularExpressionMatch match = i.next();

hashtags.insert(match.captured(1).toStdString());

}

}

QJsonObject Note::toJson() const

{

QJsonObject json;

json["title"] = QString::fromStdString(title);

json["text"] = QString::fromStdString(text);

QJsonArray tagsArray;

for (const auto &tag : hashtags) {

tagsArray.append(QString::fromStdString(tag));

}

json["hashtags"] = tagsArray;

return json;

}

Note Note::fromJson(const QJsonObject &json)

{

Note note;

note.title = json["title"].toString().toStdString();

note.text = json["text"].toString().toStdString();

QJsonArray tagsArray = json["hashtags"].toArray();

for (const auto &tag : tagsArray) {

note.hashtags.insert(tag.toString().toStdString());

}

return note;

}

**notemanager.h**

#ifndef NOTEMANAGER\_H

#define NOTEMANAGER\_H

#include "note.h"

#include <list>

#include <memory>

#include <QString>

class NoteManager

{

public:

NoteManager();

~NoteManager();

std::shared\_ptr<Note> generateNote();

void addNote(const Note &note);

void removeNote(const std::shared\_ptr<Note> &note);

const std::list<std::shared\_ptr<Note>> &allNotes() const;

std::list<Note> notesByHashtag(const std::string &hashtag) const;

std::set<std::string> allHashtags() const;

std::shared\_ptr<Note> findNoteByTitle(const std::string &title) const;

bool noteTitleExists(const std::string &title) const;

void saveToFile(const QString &filename) const;

void loadFromFile(const QString &filename);

private:

std::list<std::shared\_ptr<Note>> m\_notes;

QString m\_storageFile;

};

#endif // NOTEMANAGER\_H

**notemanager.cpp**

#include "notemanager.h"

#include <algorithm>

#include <QFile>

#include <QJsonDocument>

#include <QJsonArray>

#include <QDir>

#include <QMessageBox>

NoteManager::NoteManager()

{

m\_storageFile = QDir::home().filePath("notes.json");

loadFromFile(m\_storageFile);

}

NoteManager::~NoteManager()

{

saveToFile(m\_storageFile);

}

std::shared\_ptr<Note> NoteManager::generateNote()

{

// Генерируем уникальное имя по умолчанию

int i = 1;

while (noteTitleExists("Новая заметка " + std::to\_string(i))) {

i++;

}

auto note = std::make\_shared<Note>();

note->title = "Новая заметка " + std::to\_string(i);

m\_notes.push\_front(note);

return m\_notes.front();

}

void NoteManager::addNote(const Note &note)

{

m\_notes.push\_front(std::make\_shared<Note>(note));

}

void NoteManager::removeNote(const std::shared\_ptr<Note>& note)

{

m\_notes.remove(note);

}

const std::list<std::shared\_ptr<Note>> &NoteManager::allNotes() const

{

return m\_notes;

}

std::list<Note> NoteManager::notesByHashtag(const std::string &hashtag) const

{

std::list<Note> notesByHashtag;

for (const auto& note : m\_notes) {

if (note->hashtags.find(hashtag) != note->hashtags.end()) {

notesByHashtag.push\_back(\*note);

}

}

return notesByHashtag;

}

std::set<std::string> NoteManager::allHashtags() const

{

std::set<std::string> hashtags;

for (const auto& note : m\_notes) {

hashtags.insert(note->hashtags.begin(), note->hashtags.end());

}

return hashtags;

}

std::shared\_ptr<Note> NoteManager::findNoteByTitle(const std::string& title) const

{

auto it = std::find\_if(m\_notes.begin(), m\_notes.end(),

[&title](const std::shared\_ptr<Note>& note) {

return note->title == title;

});

return it != m\_notes.end() ? \*it : nullptr;

}

bool NoteManager::noteTitleExists(const std::string& title) const

{

return findNoteByTitle(title) != nullptr;

}

void NoteManager::saveToFile(const QString& filename) const

{

QJsonArray notesArray;

for (const auto& note : m\_notes) {

notesArray.append(note->toJson());

}

QJsonDocument doc(notesArray);

QFile file(filename);

if (file.open(QIODevice::WriteOnly)) {

file.write(doc.toJson());

}

}

void NoteManager::loadFromFile(const QString& filename)

{

QFile file(filename);

if (!file.exists()) return;

if (file.open(QIODevice::ReadOnly)) {

QJsonDocument doc = QJsonDocument::fromJson(file.readAll());

QJsonArray notesArray = doc.array();

for (const auto& noteJson : notesArray) {

Note note = Note::fromJson(noteJson.toObject());

m\_notes.push\_back(std::make\_shared<Note>(note));

}

}

}