Министерство цифрового развития, связи и массовых коммуникаций РФ
Федеральное государственное бюджетное образовательное учреждение высшего образования
«Сибирский государственный университет телекоммуникаций и информатики»
(СибГУТИ)

КУРСОВАЯ РАБОТА

Вариант 7

«Редактор заметок с уведомлениями»

Выполнил студент группы ИП-116 Губин А.С.

Проверил доцент кафедры ПМиК к.т.н Мерзлякова Е.Ю.

Новосибирск 2023

Содержание

Введение

В современном быстро меняющемся мире, где каждый день приносит новые задачи и вызовы, эффективное управление своим временем и заданиями становится ключевым аспектом успешной повседневной жизни. Редакторы заметок, дополненные функцией уведомлений, играют важную роль в обеспечении организованности и оперативности в выполнении задач.

В данной работе мы исследуем современные стандарты и требования к редакторам заметок с уведомлениями, проанализируем их функциональные возможности и влияние на повседневную продуктивность. Освещение этих аспектов поможет глубже понять роль таких инструментов в современном информационном пространстве и их важность для эффективного управления личным и профессиональным временем.

Моё личное стремление планировать и контролировать всю важную информацию вдохновило меня на создание этого приложения.

1. Анализ задач и пользователей

Задачи: Разработать приложение для удобной, а главное быстрой записи какой-либо важной информации, а также предоставить возможность установить напоминание.

Пользователи и их требования: Пользователь должен быть знаком с основами использования компьютера для работы с приложением.

Для создания интерфейса программы "Заметки" был взят за основу опыт использования подобных приложений, так же было проведено тестирование в котором участвовало несколько человек, которые сказали что было понятно интуитивно, а что нет. Это позволило определить наиболее удобный интерфейс для обычного пользователя ПК.

2. Выбор репрезентативных задач

После проведения тестирования, я выяснил, какие главные задачи должны решаться при использовании системы:

1. Добавление новых заметок:
   - создание нового файла.
2. Редактирование текста по нажатию по заметке или в меню при нажатии ПКМ:
   - открытие текстового редактора разными способами.
3. Удаление заметки:
   - Удалить выбранную заметку в меню при нажатии ПКМ.
4. Добавление напоминания на заметку:
   - Установить напоминание на заметку выбранную заметку в меню при нажатии ПКМ.

Эти задачи обеспечивают функционал для приложения заметок, позволяя пользователям вести записи и устанавливать напоминания при необходимости.

3.Заимствование

После определения репрезентативных задач, следует найти существующий интерфейс с помощью которых пользователи могут выполнить требуемую работу. На телефонах компании Xiaomi с завода установлено приложение Заметки, от куда и будет позаимствован основой функционал и дизайн приложения. Например, при заходе в приложение будет отображаться примерно следующее (рис.1)

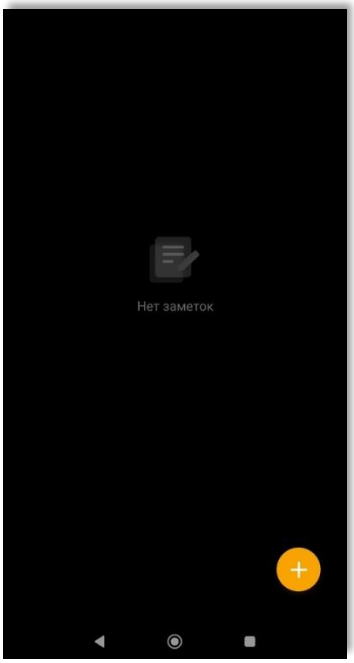

Рисунок 1. Стартовое окно приложения «Заметки».

Добавление событий будет выглядеть примерно так(рис.2):

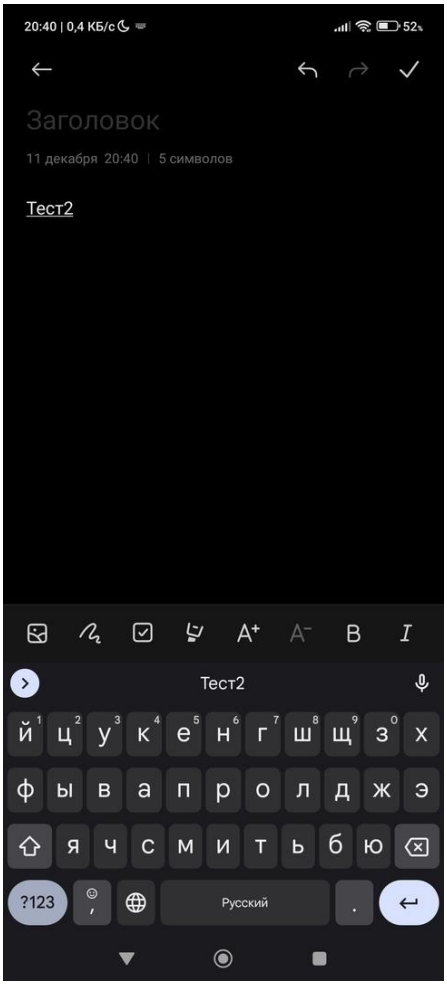

Рисунок 2. второе окно редактирование заметки «Редактор».

4.Прототип интерфейса

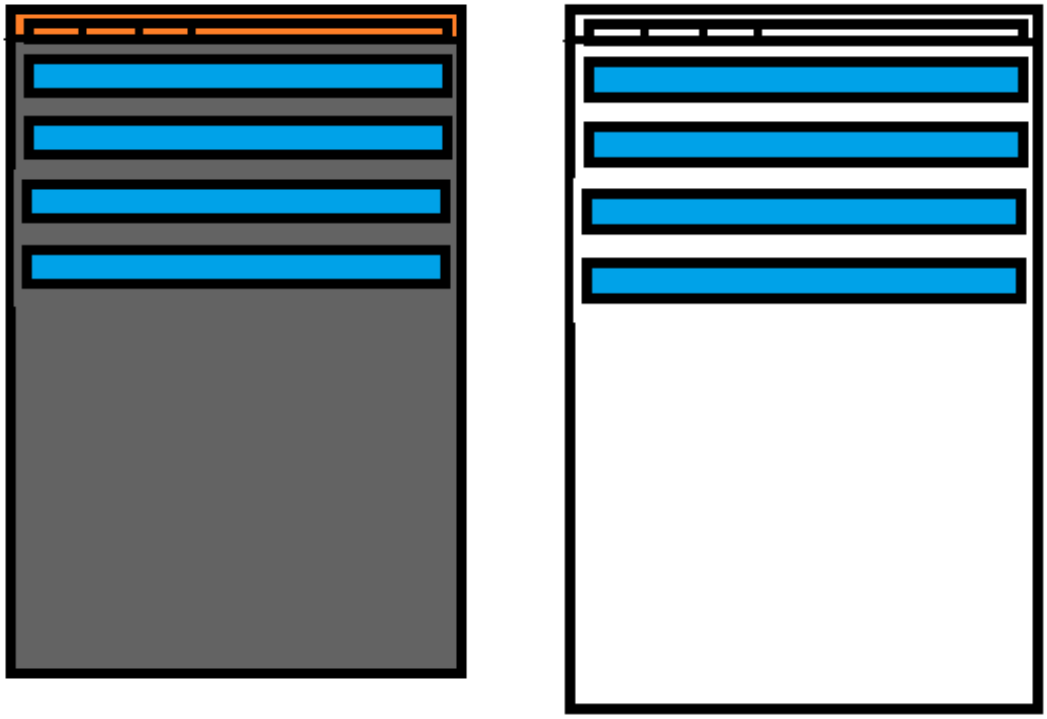

Рисунок 3.шаблон главного окна светлая и темная тема.

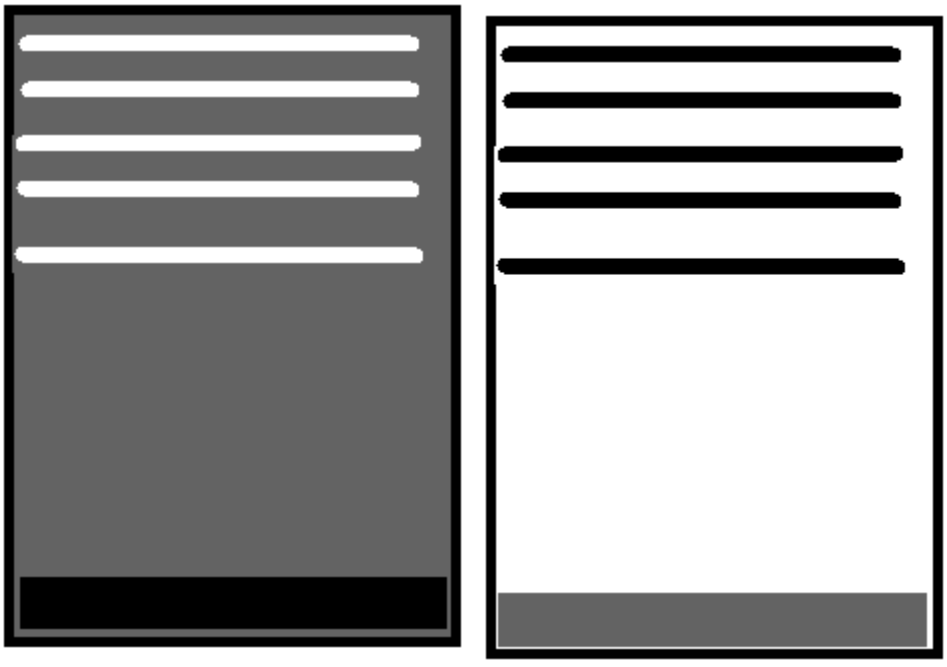

Рисунок 4.шаблон окна редактирования светлая и темная тема.

На данной вкладке будет производиться изменение текста заметки. При нажатии на кнопку «File», а далее «Добавить запись» создастся новая заметка. При нажатии на ЛКМ открывается окно создания новой записи. (рис.3)

5. Реализация

Функционал:

- Импорт и экспорт базы данных.
- Просмотр, создание, редактирование и удаление заметок.
- Сохранение текста заметок в файл.
- Функция установки напоминания с уведомлением на заметку.
- Изменение цветовой палитры приложения со светлой на темную и наоборот

База данных реализована на языке SQLite, взаимодействующим с функционалом библиотеки QSql.

Скриншоты работы программы

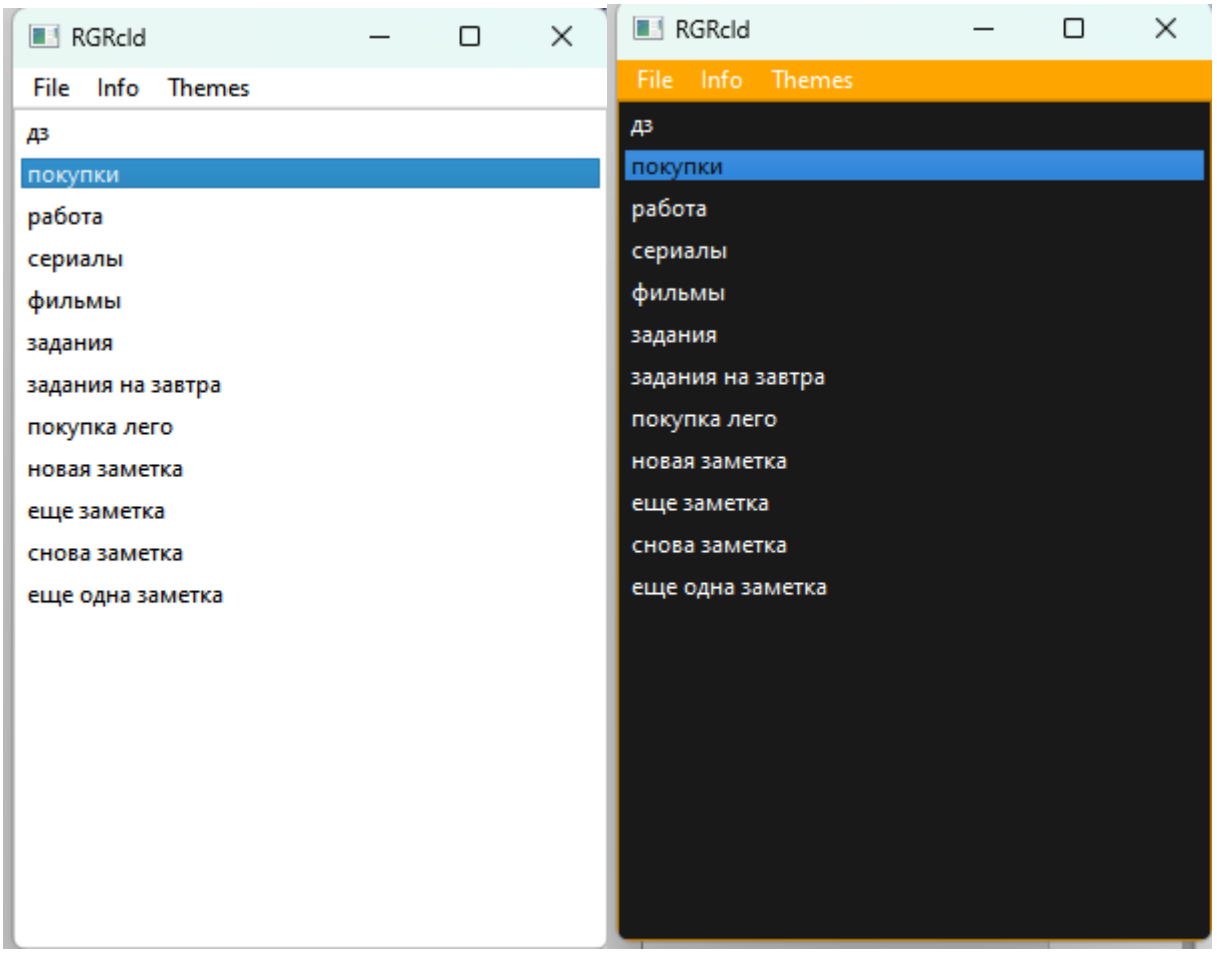

Рисунок 5. Начальное окно с заметками.

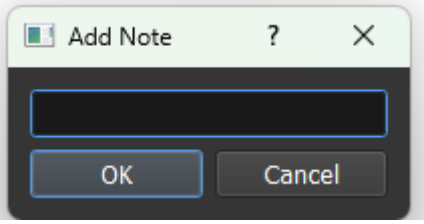

Рисунок 6. окно с добавлением новой заметки

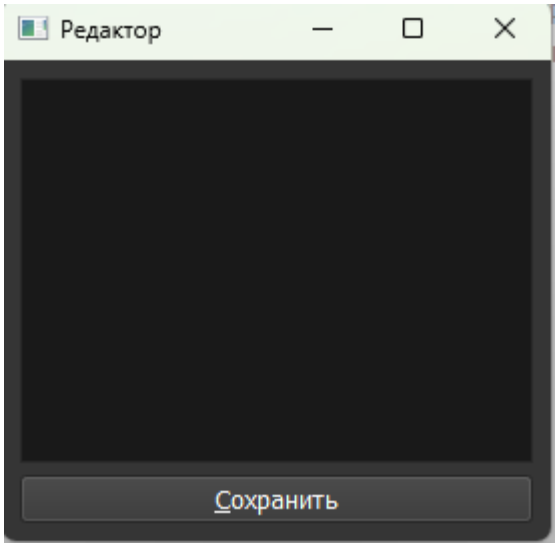

Рисунок 8.Окно для редактирования текста заметки.

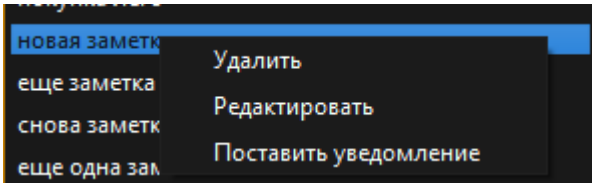

Рисунок 9. Панель открывающаяся при нажатии ПКМ.

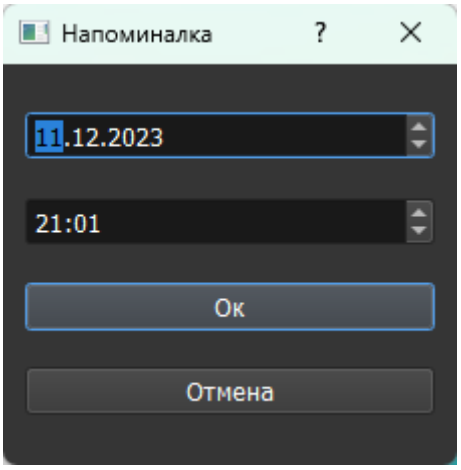

Рисунок 10. Меню для создания Напоминания с уведомлением для конкретной заметки.

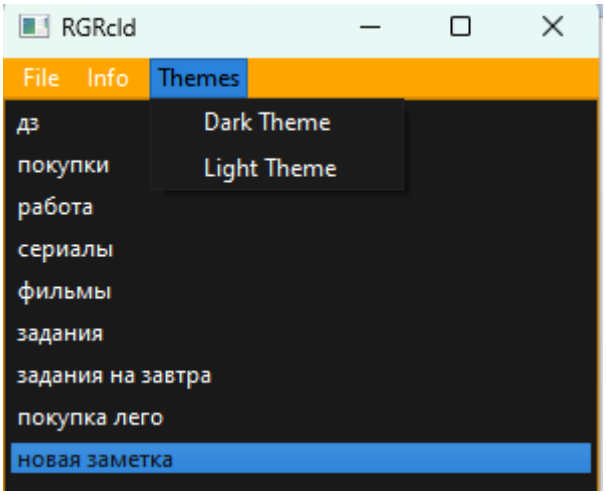

Рисунок 11. Смена палитры цветов со светлой на темну и наоборот.

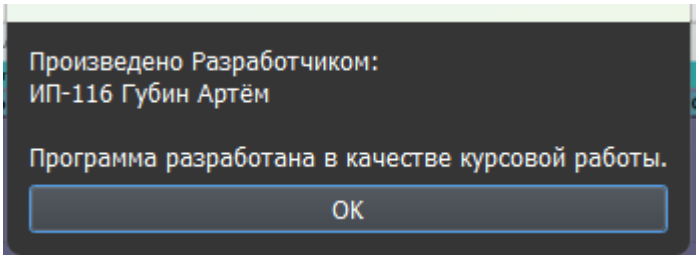

Рисунок 7. Окно “О программе

CWT-анализ

Задача 1: Удаление Заметки

Список действий:

1. Запустить приложение.
2. Выбрать заметку на которую мы хотим поставить уведомление
3. Нажать на вкладку “File”
4. Перейти в подпункт меню “работа с заметкой”
5. Нажать на кнопку удалить

Анализ действий задачи 1:

С первым действием у пользователя не должно возникнуть затруднений, замечаний нет.
Со вторым действием у пользователя не должно возникнуть затруднений, замечаний нет.
С третьим действием у пользователя не должно возникнуть затруднений, замечаний нет.

В четвертом пункте у пользователя может создаться затруднение так как обычно в приложениях такого рода кнопка удаления должна быть где-то на видном месте или в интуитивно понятном для пользователя

Решение:

Кнопка была вынесена в отдельное меню которое открывается при нажатии правой кнопкой мыши по выбранной заметке

Задача 2: Установка Напоминания с уведомлением

Список действий:

1. Запустить приложение.
2. Выбрать заметку на которую мы хотим поставить уведомление
3. Нажать на вкладку “File”
4. Перейти в подпункт меню “работа с заметкой”
5. Выбрать пункт “Поставить уведомление”
6. Выбрать время и дату
7. Нажать “Ок”

Анализ действий задачи 2:

С первым действием у пользователя не должно возникнуть затруднений, замечаний нет.
Со вторым действием у пользователя не должно возникнуть затруднений, замечаний нет.
С третьим действием у пользователя не должно возникнуть затруднений, замечаний нет.
В четвером пункте у пользователя могут возникнуть проблемы с отсутствием подсказок, так как не совсем ясно как создавать уведомления и где находится эта кнопка

Решение:

Для этого, как и в 1 задании кнопка создания уведомлений была перенесена в меню при нажатии правой кнопкой мыши по заметке.

GOMS-анализ

Описание анализа

Практически все интерфейсные взаимодействия в можно описать следующими операциями:

K – нажатие клавиши;

B – клик кнопкой мыши;

P – наведение указателя мыши;

R – ожидание ответной реакции компьютера;

H – перенос руки с клавиатуры на мышь или наоборот

D – проведение с помощью мыши прямой линии (например, выделение или прокрутка текста);

M – мыслительная подготовка (к осуществлению одной из перечисленных операций).

Разные пользователи выполняют указанные операции за разное время. Однако, GOMS исследует работу опытного пользователя. Многочисленные исследования выявили средние значения времени операций, выполняемых опытными пользователями.

K 0.2 с

B 0.2 с

P 1.1 с

H 0.4 с

M 1.0 с

Цель №1: Создать новую заметку и внести в нее изменения.

Для выполнения цели сформулируем подцели:

1. Создать новую заметку
2. Открыть заметку
3. Записать в нее текст

Теперь опишем методы для каждой подцели и распишем каждый метод с точностью до операции:

1. Создать новую заметку.
   1.1. Нажимаем на кнопку “File”,
   1.2. Нажимаем “Добавить запись”,
   1.3. придумываем название для новой заметки,
   1.4. вводим его
   1.5. нажимаем “ок”

MB B H MB H B

1 + 0.2 + 0.2 + 0.4 + 1 + 0.2 + 0.4 + 0.2 = 3.6 сек.

2. Открываем заметку:
   2.1. Выбираем нужную заметку
   2.2. Нажимаем левой кнопкой мыши

M K

1+ 0.2 = 1.2 сек

3. Записать в нее текст:
   3.1. Думаем что писать
   3.2. Пишем

3.3. Сохраняем

М Н К Н В

1 + 0.4 + 0.2 + 0.4 + 0.2 = 2.2 сек

Итог: **6.0 сек.**

Цель №2: Установить на заметку напоминание с уведомлением.

Для выполнения цели сформулируем подцели:

1. Выбрать заметку нажав правой кнопкой мыши
2. Выбрать пункт напоминание
3. Изменить время

Теперь опишем методы для каждой подцели и распишем каждый метод с точностью до операции:

1. Выбрать заметку нажав правой кнопкой мыши.
   1.1. Нажать на заметку правой кнопкой мыши
2. Нажать “Поставить уведомление”
3. Ввести время
   3.1. Нажать “Ок”

МВ В НК НВ

1 + 0.2 + 0.2 + 0.4 + 0.2 + 0.4 + 0.2 = 2.6 сек.

Итог: **2.6 сек.**

СПИСОК ИСПОЛЬЗУЕМЫХ ИСТОЧНИКОВ

- Qt Documentation – официальная документация по разработке в среде Qt;

  URL: https://doc.qt.io/

Приложение:

Основной код

```
void MainWindow::closeEvent(QCloseEvent *event) {
    QByteArray geometry = saveGeometry();
    QSettings settings("YourOrganizationName", "YourAppName");
    settings.setValue("MainWindow/Geometry", geometry);

    // Сохранение текущей темы
    QString currentTheme = settings.value("theme", "light").toString();
       if (currentTheme == "dark") {
           applyDarkTheme();
       } else {
           applyLightTheme();
       }

       QMainWindow::closeEvent(event);
}

void MainWindow::readSettings() {
    QSettings settings("YourOrganizationName", "YourAppName");

    // Восстановление геометрии
    QByteArray geometry = settings.value("MainWindow/Geometry").toByteArray();
    if (!geometry.isEmpty()) {
        restoreGeometry(geometry);
    }

    // Восстановление темы
    QString savedTheme = settings.value("theme", "light").toString();
    if (savedTheme == "dark") {
        applyDarkTheme();
    } else {
        applyLightTheme();
    }
}
```

```
MainWindow::MainWindow(QWidget *parent) : QMainWindow(parent) {
popUp = new PopUp();

database = QSqlDatabase::addDatabase("QSQLITE");
database.setDatabaseName("C:/Users/Kiddacloud/Documents/RGRcld/forbd/my.db");
if (!database.open()) {
qDebug() << "[FROM db] - Error opening database:" << database.lastError().text();
} else {
qDebug() << "[FROM db] - Database opened successfully!";
}
QSqlQuery query;
query.exec("CREATE TABLE IF NOT EXISTS notes (id INTEGER PRIMARY KEY
AUTOINCREMENT, title TEXT, filename TEXT)");

contextMenu = new QMenu(this);
notesList = new QListWidget(this);
setCentralWidget(notesList);

createMenu();
loadNotesFromManager();
notesList->setContextMenuPolicy(Qt::CustomContextMenu);

notesList->setSpacing(3);
```

```
}



void MainWindow::onNoteSelected(QListWidgetItem *item) {
    if (!(QGuiApplication::mouseButtons() && Qt::RightButton)) {
        QString filename = item->data(Qt::UserRole).toString();
        NoteEditor *editor = new NoteEditor(filename, this);
        editor->show();
        //contextMenu->exec(QCursor::pos());
    }
        contextMenu->setProperty("selectedNoteTitle", item->text());
        contextMenu->setProperty("selectedNoteFilename", item-
>data(Qt::UserRole).toString());
}

void MainWindow::onDeleteNote() {
QListWidgetItem *currentItem = notesList->currentItem();
if (currentItem) {
QString filename = currentItem->data(Qt::UserRole).toString();
noteManager.deleteNoteFromDatabase(filename);
delete currentItem;
}
}

void MainWindow::onOpenNote() {
QListWidgetItem *currentItem = notesList->currentItem();
if (currentItem) {
QString filename = currentItem->data(Qt::UserRole).toString();
NoteEditor *editor = new NoteEditor(filename, this);
editor->show();
}
}

void MainWindow::onSetReminder() {
QString selectedNoteTitle = contextMenu-
>property("selectedNoteTitle").toString();
QString selectedNoteContent = readNoteContentFromFile(contextMenu-
>property("selectedNoteFilename").toString());
qDebug() << selectedNoteTitle << selectedNoteContent;
// Проверка на пустое имя файла и контент
if (!selectedNoteTitle.isEmpty()) {
ReminderDialogg dialog(this);
if (dialog.exec() == QDialog::Accepted) {
if (selectedNoteContent.isEmpty()){
QDateTime reminderDateTime = dialog.getReminderDateTime();
scheduleReminder(selectedNoteTitle, "\nбез текста.", reminderDateTime);
}else{
QDateTime reminderDateTime = dialog.getReminderDateTime();
scheduleReminder(selectedNoteTitle, selectedNoteContent, reminderDateTime);
}
}
} else {
qDebug() << "[FROM FILE] - Empty filename or content specified.";
}
}

void MainWindow::onAddNote() {
    AddNoteDialog dialog(this);
    dialog.show();
    if (dialog.exec() == QDialog::Accepted) {
        QString title = dialog.getNoteTitle();
        QString filename = "D:/QT/dlyamyrgr/" + title + ".txt"; // Путь к файлу
заметки
```

```
        // Создаем файл и записываем в него данные
        QFile file(filename);
            if (file.open(QIODevice::WriteOnly | QIODevice::Text)) {
            file.close();
            } else {
                qDebug() << "[FROM FILE] - Error opening file for writing:" <<
file.errorString();
                return;
            }
        noteManager.addNoteToDatabase(title, filename);

        QListWidgetItem *newItem = new QListWidgetItem(title);
        newItem->setData(Qt::UserRole, filename);
        notesList->addItem(newItem);

        loadNotesFromManager(); // Обновляем список заметок в главном окне
    }
}



void MainWindow::scheduleReminder(const QString &title, const QString &content,
const QDateTime &reminderDateTime) {
    qDebug() << "Schedule Reminder for" << title << "at" << reminderDateTime;

    // Создайте QTimer
    QTimer *timer = new QTimer(this);

    // Подключите слот для обработки таймера
    connect(timer, &QTimer::timeout, this, [=]() {
        // Ваш код для отложенного вызова функции
        popUp->setPopupText("Напоминалка:\n\n" + title + "\n" + content);
        popUp->show();

        // Остановите таймер после выполнения функции (если нужно)
        timer->stop();

        // Освободите ресурсы таймера (если нужно)
        timer->deleteLater();
    });

    // Рассчитайте время задержки до reminderDateTime
    QDateTime currentDateTime = QDateTime::currentDateTime();
    int delay = currentDateTime.msecsTo(reminderDateTime);

    // Установите интервал таймера (в миллисекундах)
    timer->start(delay);
}


void MainWindow::AboutMe(){
    // Реализация метода aboutme
    aboutme aboutMeDialog;
    aboutMeDialog.exec();
}


QString MainWindow::readNoteContentFromFile(const QString &filename) {
QString content;
if (!filename.isEmpty()) {
QFile file(filename);
if (file.open(QIODevice::ReadOnly | QIODevice::Text)) {
```

```
QTextStream in(&file);
content = in.readAll();
file.close();
} else {
qDebug() << "Error opening file for reading:" << file.errorString();
}
} else {
qDebug() << "Empty filename specified.";
}
return content;
}

void MainWindow::createMenu() {
    QMenu *fileMenu = menuBar()->addMenu(tr("File"));
    QMenu *Aabout = menuBar()->addMenu(tr("Info"));
    QMenu *Themes = menuBar()->addMenu(tr("Themes"));

// Проверка открытия базы данных
    if (!QSqlDatabase::database().isOpen()) {
        qDebug() << "Database is not open!";
        return;
    }

// Используем NoteManager для получения списка заметок
    for (QListWidgetItem *item : noteManager.getNoteItems()) {
        item->whatsThis();
        notesList->addItem(item);
    }



//****************************************
    QAction *addNoteAction = fileMenu->addAction(tr("Добавить запись"));
    connect(addNoteAction, &QAction::triggered, this, &MainWindow::onAddNote);

    QAction *about = Aabout->addAction(tr("О программе"));
    connect(about, &QAction::triggered, this, &MainWindow::AboutMe);

//****************************************




    deleteNoteAction = new QAction(tr("Удалить"), this);
    connect(deleteNoteAction, &QAction::triggered, this,
&MainWindow::onDeleteNote);

    setReminderAction = new QAction(tr("Поставить уведомление"), this);
    connect(setReminderAction, &QAction::triggered, this,
&MainWindow::onSetReminder);

    openNoteAction = new QAction(tr("Редактировать"), this);
    connect(openNoteAction, &QAction::triggered, this, &MainWindow::onOpenNote);

    connect(notesList, &QListWidget::itemClicked, this,
&MainWindow::onNoteSelected);

    // Добавляем действия в меню для смены темы
        QAction *darkThemeAction = Themes->addAction(tr("Dark Theme"));
        QAction *lightThemeAction = Themes->addAction(tr("Light Theme"));

        // Подключаем слоты для действий
```

```
        connect(darkThemeAction, &QAction::triggered, this,
&MainWindow::applyDarkTheme);
        connect(lightThemeAction, &QAction::triggered, this,
&MainWindow::applyLightTheme);


// Добавляем действия в контекстное меню
    notesList->addAction(deleteNoteAction);
    notesList->addAction(openNoteAction);
    notesList->addAction(setReminderAction);


    // Создаем контекстное меню
    QMenu *contextMenu = new QMenu(this);
    contextMenu->addAction(deleteNoteAction);
    contextMenu->addAction(openNoteAction);
    contextMenu->addAction(setReminderAction);

// Подключаем контекстное меню к списку заметок
    notesList->setContextMenuPolicy(Qt::CustomContextMenu);
        connect(notesList, &QListWidget::customContextMenuRequested, [=](const
QPoint &pos){
            contextMenu->setProperty("selectedNoteTitle", notesList->itemAt(pos)-
>text());
            contextMenu->setProperty("selectedNoteFilename", notesList-
>itemAt(pos)->data(Qt::UserRole).toString());
            contextMenu->exec(notesList->mapToGlobal(pos));
        });
}

void MainWindow::applyDarkTheme() {
    QPalette palette = this->palette();
    palette.setColor(QPalette::Window, QColor(255, 165, 0)); // Orange background
color
    palette.setColor(QPalette::WindowText, Qt::black); // White text color

    // Применяем установленные цвета
    this->setPalette(palette);
    QPalette darkPalette;
        darkPalette.setColor(QPalette::Window, QColor(53, 53, 53));
        darkPalette.setColor(QPalette::WindowText, Qt::white);
        darkPalette.setColor(QPalette::Base, QColor(25, 25, 25));
        darkPalette.setColor(QPalette::AlternateBase, QColor(53, 53, 53));
        darkPalette.setColor(QPalette::ToolTipBase, Qt::white);
        darkPalette.setColor(QPalette::ToolTipText, Qt::white);
        darkPalette.setColor(QPalette::Text, Qt::white);
        darkPalette.setColor(QPalette::Button, QColor(53, 53, 53));
        darkPalette.setColor(QPalette::ButtonText, Qt::white);
        darkPalette.setColor(QPalette::BrightText, Qt::red);
        darkPalette.setColor(QPalette::Link, QColor(42, 130, 218));
        darkPalette.setColor(QPalette::Highlight, QColor(42, 130, 218));
        darkPalette.setColor(QPalette::HighlightedText, Qt::black);

    qApp->setPalette(darkPalette);
    QSettings settings("YourOrganizationName", "YourAppName");
        settings.setValue("theme", "dark");
}
void MainWindow::applyLightTheme() {
    QPalette palette = this->palette();
    palette.setColor(QPalette::Window, QColor(255, 255, 255)); // Orange
background color
    palette.setColor(QPalette::WindowText, Qt::black); // White text color

    // Применяем установленные цвет
```

```
    this->setPalette(palette);
    // Сбросите стили и палитру на стандартные
    qApp->setStyle(QStyleFactory::create("Fusion"));
    qApp->setPalette(QApplication::style()->standardPalette());
    qApp->setStyleSheet("");
    QSettings settings("YourOrganizationName", "YourAppName");
        settings.setValue("theme", "light");
}
void MainWindow::loadNotesFromManager() {
    for (QListWidgetItem *item : noteManager.getNoteItems()) {
        notesList->addItem(item);
    }
}
```

Уведомления

```
PopUp::PopUp(QWidget *parent) : QWidget(parent){
    setWindowFlags(Qt::FramelessWindowHint |        // Отключаем оформление окна
                   Qt::Tool |                       // Отменяем показ в качестве
отдельного окна
                   Qt::WindowStaysOnTopHint);       // Устанавливаем поверх всех
окон
    setAttribute(Qt::WA_TranslucentBackground);     // Указываем, что фон будет
прозрачным
    setAttribute(Qt::WA_ShowWithoutActivating);     // При показе, виджет не
получается фокуса автоматически

    animation.setTargetObject(this);                // Устанавливаем целевой
объект анимации
    animation.setPropertyName("popupOpacity");      // Устанавливаем анимируемое
свойство
    connect(&animation, &QAbstractAnimation::finished, this, &PopUp::hide); /*
Подключаем сигнал окончания
                                                                             *
анимации к слоты скрытия
                                                                             * */

    // Настройка текста уведомления
    label.setAlignment(Qt::AlignHCenter | Qt::AlignVCenter); // Устанавливаем по
центру
    // И настраиваем стили
    label.setStyleSheet("QLabel { color : white; "
                        "margin-top: 6px;"
                        "margin-bottom: 6px;"
                        "margin-left: 10px;"
                        "margin-right: 10px; }");

    // Производим установку текста в размещение, ...
    layout.addWidget(&label, 0, 0);
    setLayout(&layout); // которое помещаем в виджет

    // По сигналу таймера будет произведено скрытие уведомления, если оно видимо
    timer = new QTimer();
    connect(timer, &QTimer::timeout, this, &PopUp::hideAnimation);
}
void PopUp::paintEvent(QPaintEvent *event){
    Q_UNUSED(event)

    /* А теперь настраиваем фон уведомления,
     * который является прямоугольником с чёрным фоном
     * */
    QPainter painter(this);
    painter.setRenderHint(QPainter::Antialiasing); // Включаем сглаживание
```

```
    // Подготавливаем фон. rect() возвращает внутреннюю геометрию виджета
уведомления, по содержимому
    QRect roundedRect;
    roundedRect.setX(rect().x() + 5);
    roundedRect.setY(rect().y() + 5);
    roundedRect.setWidth(rect().width() - 10);
    roundedRect.setHeight(rect().height() - 10);

    // Кисть настраиваем на чёрный цвет в режиме полупрозрачности 180 из 255
    painter.setBrush(QBrush(QColor(0,0,0,180)));
    painter.setPen(Qt::NoPen); // Край уведомления не будет выделен

    // Отрисовываем фон с закруглением краёв в 10px
    painter.drawRoundedRect(roundedRect, 10, 10);
}
void PopUp::setPopupText(const QString &text)
{
    label.setText(text);    // Устанавливаем текст в Label
    adjustSize();           // С пересчётом размеров уведомления
}
void PopUp::show(){
    setWindowOpacity(0.0);  // Устанавливаем прозрачность в ноль

    animation.setDuration(300);     // Настраиваем длительность анимации
    animation.setStartValue(0.0);   // Стартовое значение будет 0 (полностью
прозрачный виджет)
    animation.setEndValue(1.0);     // Конечное - полностью непрозрачный виджет

    setGeometry(QApplication::desktop()->availableGeometry().width() - 36 -
width() + QApplication::desktop() -> availableGeometry().x(),
                QApplication::desktop()->availableGeometry().height() - 36 -
height() + QApplication::desktop() -> availableGeometry().y(),
                width(),
                height());
    QWidget::show();                // Отображаем виджет, который полностью
прозрачен

    animation.start();              // И запускаем анимацию
    timer->start(6000);             // А также стартуем таймер, который запустит
скрытие уведомления через 3 секунды
}

void PopUp::hideAnimation(){
    timer->stop();                  // Останавливаем таймер
    animation.setDuration(1000);    // Настраиваем длительность анимации
    animation.setStartValue(1.0);   // Стартовое значение будет 1 (полностью
непрозрачный виджет)
    animation.setEndValue(0.0);     // Конечное - полностью прозрачный виджет
    animation.start();              // И запускаем анимацию
}

void PopUp::hide(){
    // Если виджет прозрачный, то скрываем его
    if(getPopupOpacity() == 0.0){
        QWidget::hide();
    }
}

void PopUp::setPopupOpacity(float opacity){
    popupOpacity = opacity;

    setWindowOpacity(opacity);
}
```

```
float PopUp::getPopupOpacity() const{
    return popupOpacity;
}
```

Редактирование заметок

```
#include "noteeditor.h"
#include <QDebug>

NoteEditor::NoteEditor(const QString &filename, QWidget *parent):
QMainWindow(parent), currentFilename(filename)
{
    setWindowTitle(tr("Редактор"));

    noteTextEdit = new QTextEdit(this);
    saveButton = new QPushButton(tr("&Сохранить"), this);

    // Загружаем содержимое заметки из файла
    QFile file(currentFilename);
    if (file.open(QIODevice::ReadOnly | QIODevice::Text)) {
        QTextStream in(&file);
        // Установка кодировки для QTextStream (в данном случае, UTF-8)
        in.setCodec("CRLF");
        in.setGenerateByteOrderMark(true);
        QString content = in.readAll();

        noteTextEdit->setPlainText(content);

        file.close();
    }

    // Связываем кнопку с обработчиком события
    connect(saveButton, &QPushButton::clicked, this, &NoteEditor::onSave);

    // Размещаем элементы интерфейса в вертикальном лейауте
    QVBoxLayout *mainLayout = new QVBoxLayout(this);
    mainLayout->addWidget(noteTextEdit);
    mainLayout->addWidget(saveButton);

    // Устанавливаем основной лейаут для окна
    QWidget *centralWidget = new QWidget(this);
    centralWidget->setLayout(mainLayout);
    setCentralWidget(centralWidget);
}
void NoteEditor::onSave() {
    qDebug() << "Save button clicked!";

    QFile file(currentFilename);
    if (file.open(QIODevice::WriteOnly | QIODevice::Text)) {
        QTextStream out(&file);
        //out.setCodec("UTF-8");

        out << noteTextEdit->toPlainText();
        file.close();

        qDebug() << "saved!";
        close();
    } else {
        qDebug() << "Error saving note!";}
```