

Министерство образования Республики Беларусь  
Учреждение образования  
«БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ  
ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ»

Факультет компьютерных систем и сетей

Кафедра электронных вычислительных машин

Дисциплина: Программирование на языках высокого уровня

*К ЗАЩИТЕ ДОПУСТИТЬ*

\_\_\_\_\_ *А.В. Марзалюк*

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА  
к курсовому проекту  
на тему  
КУЛИНАРНЫЙ СПРАВОЧНИК  
БГУИР КП 1–40 02 01 401 ПЗ

Студент:

Алхава Р.А.

Руководитель:

Марзалюк А.В.

Минск 2023

Учреждение образования  
«Белорусский государственный университет информатики  
и радиоэлектроники»

Факультет компьютерных систем и сетей

УТВЕРЖДАЮ  
Заведующий кафедрой

\_\_\_\_\_  
(подпись)

\_\_\_\_\_  
2023г.

ЗАДАНИЕ  
по курсовому проектированию

Студенту \_\_\_\_\_ *Алхава Руслану Абдулькаримовичу*

1. Тема проекта \_\_\_\_\_ *Приложение «Кулинарный справочник»*
2. Срок сдачи студентом законченного проекта \_\_\_\_\_ *11 декабря 2023 г.*
3. Исходные данные к проекту \_\_\_\_\_ *Язык программирования – C++, библиотека Qt, среда разработки QT Creator, база данных SQLite.*
4. Содержание расчетно-пояснительной записки (перечень вопросов, которые подлежат разработке)
  1. Лист задания.
  2. Введение.
  3. Постановка задачи.
  4. Обзор литературы.
    - 4.1. Обзор методов и алгоритмов решения поставленной задачи.
5. Функциональное проектирование.
  - 5.1. Структура входных и выходных данных.
  - 5.2. Разработка диаграммы классов.
  - 5.3. Описание классов.
6. Разработка программных модулей.
  - 6.1. Разработка схем алгоритмов (два важных метода).
  - 6.2. Разработка алгоритмов (описание алгоритмов по шагам, для двух методов).
7. Результаты работы.
8. Заключение

9. Литература

10. Приложения

5. Перечень графического материала (с точным обозначением обязательных чертежей и графиков)

1. Диаграмма классов.

2. Схема алгоритма метода `getRecipeRowById()`.

3. Схема алгоритма метода `on_buttonEdit_clicked()`.

6. Консультант по проекту (с обозначением разделов проекта) А. В. Марзалюк

7. Дата выдачи задания 15.09.2023г

8. Календарный график работы над проектом на весь период проектирования (с обозначением сроков выполнения и трудоемкости отдельных этапов):

1. Выбор задания. Разработка содержания пояснительной записки. Перечень графического материала – к 01.10.2023 5 %;

разделы 2, 3 – к 01.10.2023 10 %;

разделы 4, 5 – к 01.11.2023 25 %;

разделы 6 – к 01.12.2023 35 %;

разделы 7,8,9 – к 05.12.2023 10 %;

разделы 10 – к 11.12.2023 5 %;

оформление пояснительной записки и графического материала – к 11.12.2023 10 %;

Защита курсового проекта с 21.12 по 28.12.23г.

РУКОВОДИТЕЛЬ

Марзалюк А. В.

(подпись)

Задание принял к исполнению

Алхава Р.А.

(дата и подпись студента)

## СОДЕРЖАНИЕ

ВВЕДЕНИЕ .....	5
1 ПОСТАНОВКА ЗАДАЧИ.....	6
2 ОБЗОР ЛИТЕРАТУРЫ .....	8
2.1 Обзор методов и алгоритмов решения поставленной задачи .....	8
3 ФУНКЦИОНАЛЬНОЕ ПРОЕКТИРОВАНИЕ.....	10
3.1 Структура входных и выходных данных .....	10
3.2 Разработка диаграммы классов .....	11
3.3 Описание классов .....	11
3.3.1 Класс для создания рецептов .....	11
3.3.2 Класс для управления рецептами .....	13
3.3.3 Класс для реализации работы с базой данных .....	13
3.3.4 Класс для работы с данными в ComboBox .....	14
3.3.5 Класс для загрузки шаблона окон.....	14
3.3.6 Класс окна для отображения полного рецепта.....	15
3.3.7 Класс окна для добавления рецепта .....	16
3.3.8 Класс окна для редактирования рецепта.....	16
3.3.9 Класс главного окна .....	16
4 РАЗРАБОТКА ПРОГРАММНЫХ МОДУЛЕЙ .....	19
4.1 Разработка схем алгоритмов.....	19
4.2 Разработка алгоритмов.....	19
4.2.1 Алгоритм по шагам для добавления рецепта в базу данных .....	19
4.2.2 Алгоритм по шагам для обновления рецепта в базе данных .....	19
5 РЕЗУЛЬТАТЫ РАБОТЫ .....	21
ЗАКЛЮЧЕНИЕ .....	28
СПИСОК ЛИТЕРАТУРЫ .....	30
ПРИЛОЖЕНИЕ А.....	31
ПРИЛОЖЕНИЕ Б .....	32
ПРИЛОЖЕНИЕ В.....	33
ПРИЛОЖЕНИЕ Г .....	34
ПРИЛОЖЕНИЕ Д.....	35

## ВВЕДЕНИЕ

Современное программное обеспечение все более требует эффективной организации данных и управления ими. Для достижения этой цели часто используются средства, обеспечивающие хранение и обработку информации. В рамках данной курсовой работы исследуется использование языка программирования C++ совместно с библиотекой Qt и системой управления базами данных SQLite для создания приложения с графическим интерфейсом.

Qt предоставляет мощные инструменты для разработки кроссплатформенных приложений с уникальным дизайном и высокой степенью гибкости. Qt Creator, в свою очередь, представляет собой интегрированную среду разработки, упрощающую процесс создания программного обеспечения с использованием Qt.

В центре внимания данной работы находится также SQLite – встраиваемая система управления базами данных, известная своей легковесностью, надежностью и простотой в использовании. Интеграция SQLite в приложение, разрабатываемое на C++ с использованием Qt, предоставляет уникальные возможности для эффективной работы с данными.

Цель данной курсовой работы – проанализировать процесс разработки приложения на C++ с использованием библиотеки Qt и интеграции с базой данных SQLite. В рамках исследования рассматриваются особенности проектирования графического интерфейса, взаимодействия с базой данных, а также оптимизация процессов обработки и отображения информации в разрабатываемом приложении.

## 1 ПОСТАНОВКА ЗАДАЧИ

Целью данного проекта является разработка приложения, предоставляющего пользователю функциональные возможности по созданию, редактированию, удалению рецептов. Приложение представляет собой удобный кулинарный справочник с интуитивно понятным графическим интерфейсом.

Для разработки данного приложения мы будем использовать библиотеку Qt.

Библиотека Qt – это кроссплатформенный набор инструментов и библиотек для разработки программного обеспечения на C++. Она включает в себя широкий набор классов для создания графических интерфейсов, работы с файлами, сетями, базами данных и многих других задач.

Иерархия классов в Qt:

Qt предоставляет обширную иерархию классов. Основные модули включают классы для виджетов (`QWidget`, `QLabel`, `QPushButton`), графики (`QPainter`, `QPixmap`), работу с файлами (`QFile`, `QDir`), сетевые операции (`QTcpSocket`, `QUdpSocket`), а также структуры данных и контейнеры (`QList`, `QMap`, `QVector`), и многое другое.

Механизм сигналов и слотов:

Одна из ключевых особенностей Qt – это механизм сигналов и слотов. Он позволяет объектам взаимодействовать асинхронно. Сигналы генерируются объектами при определенных событиях, а слоты – это методы, которые реагируют на эти сигналы. Между собой объекты могут связываться, и при возникновении сигнала у одного объекта будут вызываться соответствующие слоты у другого.

Для «Кулинарного справочника» этот механизм особенно полезен, поскольку позволяет реагировать на различные события в пользовательском интерфейсе (например, нажатие кнопки, изменение значения поля ввода) и обрабатывать их асинхронно, упрощая связь между различными компонентами вашего приложения.

Вы можете создавать свои собственные сигналы и слоты в классах, что позволяет гибко организовывать взаимодействие между различными частями вашего приложения.

Основные функции приложения включают:

1. Создание, удаление и редактирование рецептов: пользователь имеет возможность создавать новые рецепты, удалять и редактировать уже существующие. Создание и редактирование рецептов происходит в специальном диалоговом окне, где можно заполнять нужные поля для рецепта. Также доступно добавление изображения к каждому рецепту.

2. Динамическое сохранение результатов: все созданные рецепты записываются в базу данных SQLite. Каждый рецепт в базе данных представлен несколькими полями (name(TEXT), description(TEXT), ingredients(TEXT), instruction(TEXT), prepTime(INTEGER), servings(INTEGER), photo(BLOB), category(TEXT), kitchen(TEXT)). Эти данные динамически обновляются в базе данных, обеспечивая актуальность информации.

3. Управление приложением: Управление осуществляется при помощи интерфейса приложения.

Особенности приложения:

Приложение разработано для операционной системы MacOS. Для запуска на других устройствах необходимо пересобрать проект с учетом специфики и требований конкретной операционной системы.

Цель разработки приложения:

Целью данного проекта является предоставление пользователю интуитивно понятного и эффективного инструмента для создания и управления рецептами с возможностью добавления изображений. Используя функциональность библиотеки Qt, систему управления базами данных SQLite и языка программирования C++, мы стремимся создать удобное приложение, которое обеспечит гибкость в работе с рецептами и обеспечит их хранение и доступность для пользователя.

## **2 ОБЗОР ЛИТЕРАТУРЫ**

В данном разделе мы проведем обзор существующих методов и алгоритмов, которые могут быть применены в кулинарном справочнике. Этот обзор поможет определить подходы, которые можно использовать при разработке информационной системы для улучшения процессов управления недвижимостью.

Кулинарный справочник – это сборник информации о кулинарных терминах, приемах приготовления блюд, ингредиентах, рецептах и других аспектах, связанных с готовкой пищи. Это ресурс, который предоставляет пользователю подробные сведения о различных аспектах кулинарии с целью помочь ему в приготовлении вкусных и разнообразных блюд.

Примеры различных кулинарных справочников:

1. Рецепты. Предоставление разнообразных рецептов с пошаговыми инструкциями и списками ингредиентов. Поиск рецептов по типу блюда, кухне, времени приготовления и др.

2. Ингредиенты и их свойства. Информация о различных продуктах: их характеристики, пищевая ценность и совместимость с другими ингредиентами. Советы по замене ингредиентов в рецептах.

3. Техники приготовления. Объяснение различных методов приготовления: варка, жарка, запекание, тушение и т.д. Видеоматериалы или анимации, демонстрирующие техники приготовления.

### **2.1 Обзор методов и алгоритмов решения поставленной задачи**

1. Использование графического интерфейса Qt:

Это кроссплатформенная библиотека и фреймворк для разработки приложений на языке программирования C++. Он позволяет создавать графические интерфейсы, обеспечивает множество инструментов для работы с базами данных, многопоточностью и файловой системой. Qt имеет механизм сигналов и слотов для обработки событий, обеспечивает кроссплатформенность, а также является open source с возможностью коммерческого использования. Этот фреймворк широко применяется для создания приложений различного типа, включая настольные, мобильные и встроенные системы.

2. Использование базы данных SQLite:

Это встраиваемая реляционная база данных, которая предоставляет легковесное и простое в использовании хранилище данных. Она не требует отдельного сервера и хранит всю информацию в одном файле. SQLite поддерживает стандартный SQL, обладает высокой производительностью и небольшим размером библиотеки. Она широко используется в приложениях с



невысокими требованиями к масштабируемости и предоставляет надежный механизм для хранения структурированных данных.

### 3. Использование языка программирования C++:

Это высокоуровневый язык программирования, который объединяет возможности языка C с объектно-ориентированным программированием. Он предоставляет широкий спектр функциональных возможностей, включая многопоточность, шаблоны, обработку исключений. C++ поддерживает непосредственное управление памятью, что дает программисту контроль над ресурсами. Язык применяется в различных областях, от системного программирования до разработки приложений с графическим интерфейсом. C++ используется для создания эффективных и производительных программ, поддерживая принципы общности и переносимости кода.

В заключение, обзор литературы показывает, что существуют разнообразные методы и подходы для организации кулинарного справочника. Интерактивность, безопасность данных и удобство использования играют важную роль в разработке подобных приложений. Применение языка C++ и фреймворка Qt и предоставляет мощные инструменты для создания высококачественных графических приложений.

### 3 ФУНКЦИОНАЛЬНОЕ ПРОЕКТИРОВАНИЕ

#### 3.1 Структура входных и выходных данных

Файл RecipeNames.db хранит всю информацию о рецептах.

Таблица 3.1.1 – структура SQLite таблицы "RecipesInfo.db"

Имя	Тип данных	Данные
recipeId	INTEGER	Уникальный номер
name	TEXT	Название рецепта
description	TEXT	Краткое описание
ingredients	TEXT	Ингредиенты
instruction	TEXT	Инструкция
prepTime	INTEGER	Время приготовления
servings	INTEGER	Количество порций
photo	BLOB	Фотография блюда
category	TEXT	Категория блюда
kitchen	TEXT	Кухня (происхождение)

Файл comboBoxCategoryInfo.txt хранит всю информацию о существующих категориях (категории могут пополняться).

Таблица 3.1.2 – пример данных в файле comboBoxCategoryInfo.txt

Категория
Все категории
Закуски
Салаты
Супы
Холодные блюда
Горячие блюда
Десерты
Выпечка
Напитки

Файл `comboBoxKitchenInfo.txt` хранит всю информацию о существующих кухнях (кухни могут пополняться).

Таблица 3.1.3 – пример данных в файле `comboBoxKitchenInfo.txt`

Кухня
Все кухни
Российская кухня
Итальянская кухня
Индийская кухня
Грузинская кухня
Японская кухня
Французская кухня
Американская кухня

## 3.2 Разработка диаграммы классов

Диаграмма классов данной курсовой работы приведена в приложении Б.

## 3.3 Описание классов

### 3.3.1 Класс для создания рецептов

```
class Recipe
public:
    Recipe(); – конструктор без аргументов
    Recipe(int recipeId, const QString &name, const QString
&description, const QString &ingredients, const QString
&instruction, int prepTime, int servings, const QByteArray &photo,
const QString &category, const QString &kitchen); – конструктор с
аргументами
    ~Recipe(); – деструктор
    int getRecipeId() const; – возвращает уникальный номер рецепта
    QString getName() const; – возвращает название рецепта
    QString getDescription() const; – возвращает краткое описание
рецепта
    QString getIngredients() const; – возвращает ингредиенты для
рецепта
```

QString getInstruction() const; – возвращает инструкцию по  
 приготовлению  
 int getPrepTime() const; – возвращает время приготовления  
 int getServings() const; – возвращает количество порций  
 QByteArray getPhoto() const; – возвращает фотографию блюда  
 QString getCategory() const; – возвращает категорию рецепта  
 QString getKitchen() const; – возвращает кухню (происхождение)  
 блюда  
 void setRecipeId(int recipeId); – устанавливает значение для  
 уникального номера рецепта  
 void setName(const QString &name); – устанавливает значение для  
 названия рецепта  
 void setDescription(const QString &description); –  
 устанавливает значение для краткого описания рецепта  
 void setIngredients(const QString &ingredients); –  
 устанавливает значение для ингредиентов  
 void setInstruction(const QString &instruction); –  
 устанавливает значение для инструкции по приготовлению  
 void setPrepTime(int prepTime); – устанавливает значение для  
 времени приготовления  
 void setServings(int servings); – устанавливает значение для  
 количества порций  
 void setPhoto(const QByteArray &photo); – устанавливает значение  
 для фотографии блюда  
 void setCategory(const QString &category); – устанавливает  
 значение для категории рецепта  
 void setKitchen(const QString &kitchen); – устанавливает  
 значение для кухни (происхождения) рецепта  
 private:  
 int recipeId; – уникальный номер рецепта  
 QString name; – название рецепта  
 QString description; – краткое описание рецепта  
 QString ingredients; – ингредиенты  
 QString instruction; – инструкция по приготовлению  
 int prepTime; – время приготовления блюда  
 int servings; – количество порций  
 QByteArray photo; – фотография блюда  
 QString category; – категория рецепта  
 QString kitchen; – кухня (происхождение) рецепта

### 3.3.2 Класс для управления рецептами

```
class RecipeManager
public:
    RecipeManager(); – конструктор без аргументов
    RecipeManager(QSqlTableModel *model); – конструктор с
аргументом
    ~RecipeManager(); – деструктор
    bool deleteRecipeById(int recipeId); – удаляет рецепт из базы
данных по уникальному номеру рецепта
    int getRecipeRowById(int recipeId); – возвращает номер строки
рецепта в таблице по уникальному номеру
    Recipe getRecipeById(int recipeId); – возвращает рецепт из базы
данных по уникальному номеру
    int getRecipeIdFromIndex(const QModelIndex &index,
QSqlQueryModel *model); – возвращает уникальный номер рецепта по
индексу в таблице
    void addRecipeToDatabase(const Recipe& recipe, const QString&
photoPath); – добавляет рецепт в базу данных
    bool updateRecipe(int recipeId, const Recipe& recipe); –
обновляет данные рецепта в базе данных
private:
    QSqlTableModel *model; – модель таблицы базы данных
```

### 3.3.3 Класс для реализации работы с базой данных

```
class DatabaseManager
public:
    DatabaseManager(const QString &databasePath); – конструктор с
аргументом
    ~DatabaseManager(); – деструктор
    bool openDatabase(); – добавляет базу данных к проекту и открывает
её
    void closeDatabase(); – закрывает базу данных
    QSqlTableModel* createRecipeModel(); – создаёт модель таблицы
базы данных для списка рецептов
    QSqlTableModel* createRecipeModelForSearch(); – создаёт модель
таблицы базы данных для списка рецептов после фильтрации
    QSqlQuery executeQuery(const QString &queryString, const
QMap<QString, QVariant> &bindValues); – создает SQL-запрос в базу
данных
private:
```

QSqlDatabase db; – база данных

QString databasePath; – путь к базе данных

### 3.3.4 Класс для работы с данными в ComboBox

```
class ComboBoxLoader
public:
    ComboBoxLoader() = default; – конструктор по-умолчанию
    void loadComboBoxItems(QComboBox* comboBox, const QString&
filePath); – загружает данные из файла в ComboBox
    void saveComboBoxItems(QComboBox* comboBox, const QString&
filePath); – сохраняет данные из ComboBox в файл
```

### 3.3.5 Класс для загрузки шаблона окон

```
class RecipeWindowBase : public QDialog
Q_OBJECT – макрос Qt для слотов и сигналов
public:
    explicit RecipeWindowBase(QWidget *parent = nullptr); –
конструктор
    ~RecipeWindowBase(); – деструктор класса
private slots:
    void browsePhoto(); – открывает диалоговое окно с выбором
фотографии
    virtual void saveRecipe() = 0; – сохраняет рецепт в базу данных
private:
    const QString categoryPath = "/Users/willygodx/Qt/qt
projets/3sem-CulinaryBook-
C++/DataBase/comboBoxCategoryInfo.txt"; – путь к файлу, в котором
хранятся все категории
    const QString kitchenPath = "/Users/willygodx/Qt/qt
projets/3sem-CulinaryBook-C++/DataBase/comboBoxKitchenInfo.txt";
– путь к файлу, в котором хранятся все кухни
protected:
    void setupUI(); – установка основного шаблона окна
    void loadComboBoxItems(QComboBox *comboBox, const QString
&filePath); – загружает данные(категории/кухни) из файлов в ComboBox
    QLabel *titleLabel; – заголовок
    QLabel *categoryLabel; – надпись для обозначения категории
    QComboBox *categoryComboBox; – виджет ComboBox для выбора
категории
    QLabel *kitchenLabel; – надпись для обозначения кухни
    QComboBox *kitchenComboBox; – виджет ComboBox для выбора кухни
```

```

    QLabel *nameLabel; – надпись для обозначения названия рецепта
    QLineEdit *nameLineEdit; – виджет QLineEdit для ввода названия
рецепта
    QLabel *descriptionLabel; – надпись для обозначения краткого
описания рецепта
    QTextEdit *descriptionTextEdit; – виджет QLineEdit для ввода
краткого описания рецепта
    QLabel *ingredientsLabel; - надпись для обозначения ингредиентов
    QTextEdit *ingredientsTextEdit; – виджет QLineEdit для ввода
ингредиентов
    QLabel *instructionLabel; - надпись для обозначения инструкции по
приготовлению
    QTextEdit *instructionTextEdit; – виджет QLineEdit для ввода
инструкции по приготовлению
    QLabel *prepTimeLabel; – надпись для обозначения времени
приготовления
    QSpinBox *prepTimeSpinBox; – виджет SpinBox для выбора времени
приготовления
    QLabel *servingsLabel; – надпись для обозначения количества
порций
    QSpinBox *servingsSpinBox; – виджет SpinBox для выбора количества
порций
    QLabel *photoLabel; – надпись для обозначения фотографии блюда
    QLabel *selectedPhotoLabel; – надпись для обозначения выбранной
фотографии блюда
    QPushButton *browsePhotoButton; – кнопка для выбора фотографии
    QPushButton *saveButton; – кнопка для сохранения рецепта
    QPushButton *exitButton; – кнопка для выхода из диалогового окна
    QString selectedPhotoPath; – путь к выбранной фотографии

```

### 3.3.6 Класс окна для отображения полного рецепта

```

namespace Ui
class RecipeDetailsWindow; – пространство имен для хранения
класса Ui, сгенерированного Qt Designer
class RecipeDetailsWindow : public QDialog
Q_OBJECT – макрос Qt для слотов и сигналов
public:
    explicit RecipeDetailsWindow(QWidget *parent = nullptr); –
конструктор

```

```
RecipeDetailsWindow(const Recipe &recipe, QWidget *parent); –
конструктор с аргументами
~RecipeDetailsWindow(); – деструктор
private:
    QLabel *recipeNameLabel; – надпись с названием рецепта
    QLabel *descriptionLabel; – надпись с кратким описанием рецепта
    QLabel *ingredientsLabel; – надпись с ингредиентами для рецепта
    QLabel *instructionLabel; – надпись с инструкцией по
приготовлению
    QLabel *prepTimeLabel; – надпись с временем приготовления
    QLabel *servingsLabel; – надпись с количеством порций
    QLabel *photoLabel; – надпись с фотографией
    QLabel *kitchenLabel; – надпись с кухней
    QPushButton *exitButton; – кнопка для выхода из диалогового окна
```

### 3.3.7 Класс окна для добавления рецепта

```
class AddRecipeWindow : public RecipeWindowBase
Q_OBJECT – макрос Qt для слотов и сигналов
public:
    explicit AddRecipeWindow(QWidget *parent = nullptr); –
конструктор класса, который хранит всю логику
private slots:
    virtual void saveRecipe() override; – переопределенный метод
для сохранения рецепта в базу данных
```

### 3.3.8 Класс окна для редактирования рецепта

```
class EditRecipeWindow : public RecipeWindowBase
Q_OBJECT – макрос Qt для слотов и сигналов
public:
    explicit EditRecipeWindow(int recipeId, QWidget* parent =
nullptr); – конструктор класса, который хранит всю логику
private slots:
    virtual void saveRecipe() override; – переопределенный метод
для сохранения рецепта в базу данных
    void loadRecipeData(); – заполняет все поля данными рецепта
private:
    int recipeId; – уникальный номер рецепта
```

### 3.3.9 Класс главного окна



```

namespace Ui
class MainWindow; – пространство имен для хранения класса Ui,
сгенерированного Qt Designer
class MainWindow : public QMainWindow
Q_OBJECT – макрос Qt для слотов и сигналов
public:
MainWindow(QWidget *parent = nullptr); – конструктор класса
~MainWindow(); – деструктор
private slots:
void on_action_triggered(); – сигнал выхода из программы
void on_buttonAdd_clicked(); – кнопка для вызова диалогового окна
с добавлением рецепта
void on_buttonDelete_clicked(); – кнопка для удаления рецепта из
модели таблицы базы данных
void on_buttonFind_clicked(); – кнопка для поиска рецепта и
вывода найденных рецептов во вторую модель таблицы базы данных
void on_tableView_doubleClicked(const QModelIndex &index); –
вызов сигнала открытия данных о рецепте по двойному нажатию на рецепт в
модели таблицы базы данных всех рецептов
void on_tableView1_doubleClicked(const QModelIndex &index); –
вызов сигнала открытия данных о рецепте по двойному нажатию на рецепт в
модели таблицы базы данных найденных рецептов
void on_buttonEdit_clicked(); – кнопка для вызова диалогового
окна с редактированием рецепта
void on_tableView_clicked(); – вызов сигнала выбора рецепта по
нажатию в модели таблицы базы данных всех рецептов
void on_tableView1_clicked(); – вызов сигнала выбора рецепта по
нажатию в модели таблицы базы данных найденных рецептов
void on_comboBox_currentIndexChanged(); – вызов обновления
моделей таблиц базы данных после изменения категории
void on_comboBox_2_currentIndexChanged(); – вызов обновления
моделей таблиц базы данных после изменения кухни
void updateRecipeView(const QString& category, const QString&
kitchen); – обновление моделей таблиц базы данных по категории и кухне
void on_actionAddCategory_triggered(); – вызов сигнала
добавления новой категории
void on_actionDeleteCategory_triggered(); – вызов сигнала
удаления существующей категории
void on_actionAddKitchen_triggered(); – вызов сигнала добавления
новой кухни

```

```

void on_actionDeleteKitchen_triggered(); – вызов сигнала
удаления существующей кухни
private:
    QSqlTableModel *model; – модель таблицы базы данных всех рецептов
    QSqlTableModel *modelForSearch; – модель таблицы базы данных
найденных рецептов
    int currentRow; – номер текущей строки в таблице рецептов
    DatabaseManager *dbManager; – переменная для управления
запросами из базы данных
    ComboBoxLoader *comboBoxLoader; – переменная для
загрузки/сохранения данных виджета ComboBox
    const QString categoryPath = "/Users/willygodx/Qt/qt
projets/3sem-CulinaryBook-
C++/DataBase/comboBoxCategoryInfo.txt"; – путь к файлу, в котором
хранятся все категории
    const QString kitchenPath = "/Users/willygodx/Qt/qt
projets/3sem-CulinaryBook-C++/DataBase/comboBoxKitchenInfo.txt";
– путь к файлу, в котором хранятся все кухни

```

## **4 РАЗРАБОТКА ПРОГРАММНЫХ МОДУЛЕЙ**

### **4.1 Разработка схем алгоритмов**

Схемы алгоритмов приведены в приложениях В и Г.

### **4.2 Разработка алгоритмов**

#### **4.2.1 Алгоритм по шагам для добавления рецепта в базу данных**

Этот алгоритм представляет собой процесс добавления рецепта в базу данных.

Входные данные – объект класса `'Recipe'`, путь к фотографии.

Выходные данные – нет.

1. Получаем доступ к существующей базе данных.
2. Создаем объект `'QSqlQuery'`, затем подготавливаем SQL-запрос для вставки данных в таблицу базы данных.
3. С помощью метода `'bindValue()'` связываем значения полей рецепта с соответствующими параметрами запроса из п.2.
4. Открываем файл с фотографией в режиме чтения `'QIODevice::ReadOnly'`. Если файл успешно открыт, переходим к п.5, иначе выводится ошибка.
5. Читаем все байтовые данные из файла в объект `'QByteArray'`.
6. С помощью метода `'bindValue()'` связываем данные фотографии из п.5 с параметром SQL-запроса.
7. Закрываем файл с фотографией.
8. Исполняем SQL-запрос с использованием `'exec()'`. Если запрос не выполнен успешно, выводится сообщение об ошибке в консоль отладки и откатывается транзакция базы данных с помощью `'rollback()'`. Если запрос выполнен успешно, переходим к п.9.
9. Фиксируем изменения в базе данных с помощью `'commit()'`.

#### **4.2.2 Алгоритм по шагам для обновления рецепта в базе данных**

Метод, реализующий данный алгоритм позволяет редактировать информацию о существующем рецепте в базе данных.

Входные данные – уникальный номер рецепта, объект класса `'Recipe'`.

Выходные данные – значение `'true'` или `'false'`.

1. Получаем доступ к существующей базе данных.
2. Создаем объект `'QSqlQuery'`, затем подготавливаем SQL-запрос для обновления данных в таблице по заданному уникальному номеру.

3. С помощью метода `bindValue()` связываем значения полей рецепта с соответствующими параметрами из п.2. Также связываем значения уникального номера с параметром из запроса, чтобы обновить конкретную запись в таблице.

4. Исполняем SQL-запрос с использованием `exec()`. Если запрос не выполнен успешно, выводится сообщение об ошибке в консоль отладки и откатывается транзакция базы данных с помощью `rollback()`. Возвращается значение `false`. Если запрос выполнен успешно, переходим к п.5.

5. Фиксируем изменения в базе данных с помощью `commit()`.

6. Возвращается значение `true`.

## 5 РЕЗУЛЬТАТЫ РАБОТЫ

На рисунке 5.1 изображено главное окно приложения. Оно же и представляет собой интерфейсную часть приложения, отображающую полный список рецептов, а также результаты поиска списков.

Каждый рецепт представлен в виде строки в модели таблицы базы данных SQLite. Данная модель интегрируется в приложение с помощью определенных методов и настраивается таким образом, каким хочет её видеть разработчик.

Нажатие на кнопку "Добавить блюдо": открывается новое диалоговое окно с незаполненными полями, которые характерны рецепту. Данные поля можно заполнить и успешно сохранить рецепт в базу данных.

Нажатие на кнопку "Удалить блюдо": для удаления рецепта необходимо его выделить в модели таблицы. После выделения рецепта и нажатия на кнопку рецепт удаляется из базы данных и модель таблицы в приложении обновляется.

Нажатие на кнопку "Редактировать блюдо": для редактирования рецепта необходимо его выделить в модели таблицы. После выделения рецепта и нажатия на кнопку открывается новое диалоговое окно с заполненными данными этого же рецепта полями. Данные можно отредактировать и успешно сохранить отредактированный рецепт в базу данных.

Двойное нажатие на рецепт в таблице: открывается новое диалоговое окно, в котором мы можем увидеть все данные о рецепте.

Нажатие на виджет с надписью "Все категории": открывается список доступных категорий рецепта. При выборе определенной категории модель таблицы рецептов фильтруется под выбранную категорию.

Нажатие на виджет с надписью "Все кухни": открывается список доступных кухонь. При выборе определенной кухни модель таблицы рецептов фильтруется под выбранную кухню.

Ввод данных в поисковую строку и последующее нажатие на кнопку "Найти": производится фильтрация модели таблицы рецептов под введенную строку, и отфильтрованная таблица выводится в другую модель, над которой есть надпись "Результаты поиска".

Нажатие на надпись "Название рецепта" или "Краткое описание": производится сортировка рецептов по алфавиту и по названию рецепта, либо по краткому описанию соответственно. При первом нажатии производится сортировка по возрастанию, при втором нажатии производится сортировка по убыванию.

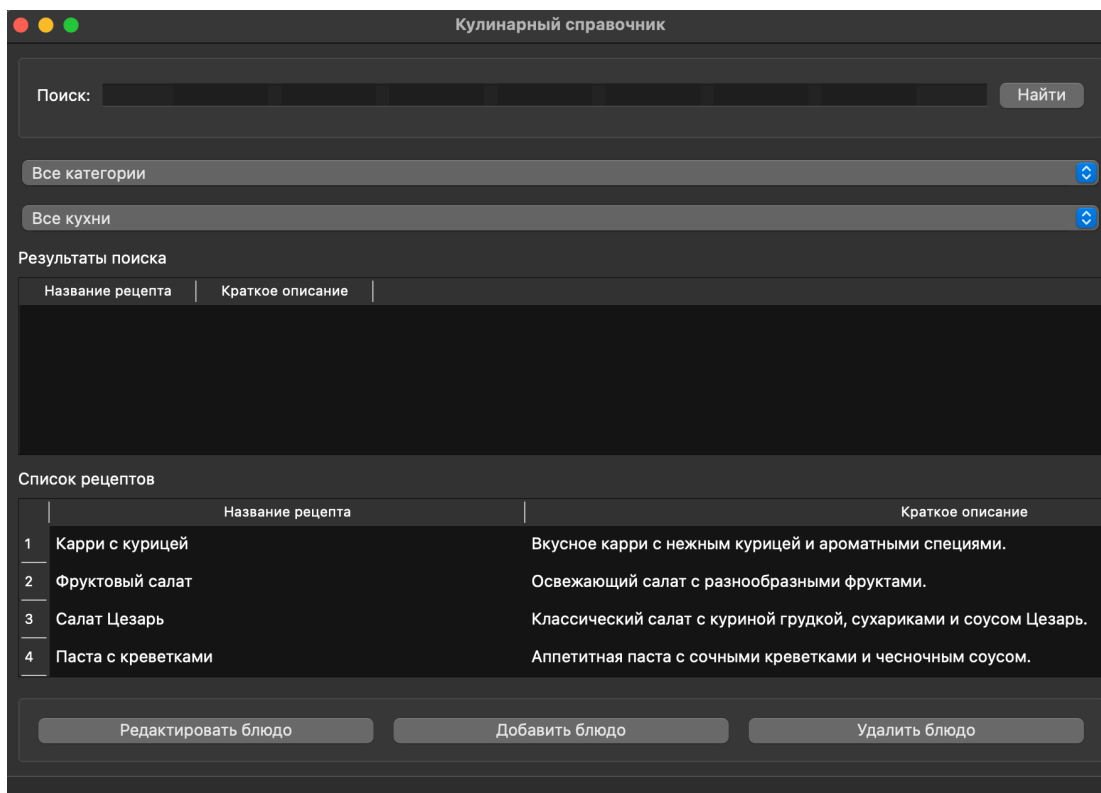


Рисунок 5.1 – главное окно приложения

На рисунке 5.2 изображено окно, представляющее собой интерфейсную часть приложения, в которой можно заполнить данные для нового рецепта и успешно сохранить в базу данных.

При нажатии на виджеты с надписями "Все категории" и "Все кухни" откроется список со всеми существующими на данный момент категориями и кухнями соответственно.

Поля "Название", "Краткое описание", "Ингредиенты", "Инструкция" можно заполнять необходимыми данными для пользователя, соответствующими названиям полей.

В виджетах "Время приготовления" и "Количество порций" можно указывать числа, соответствующие необходимым полям. Время приготовления указывается только в минутах.

При нажатии на кнопку "Выбрать фото" откроется системный каталог для выбора необходимого файла. Файлы принимаются только в .png/.jpg форматах.

При нажатии на кнопку "Сохранить" рецепт сохраняется в базу данных.

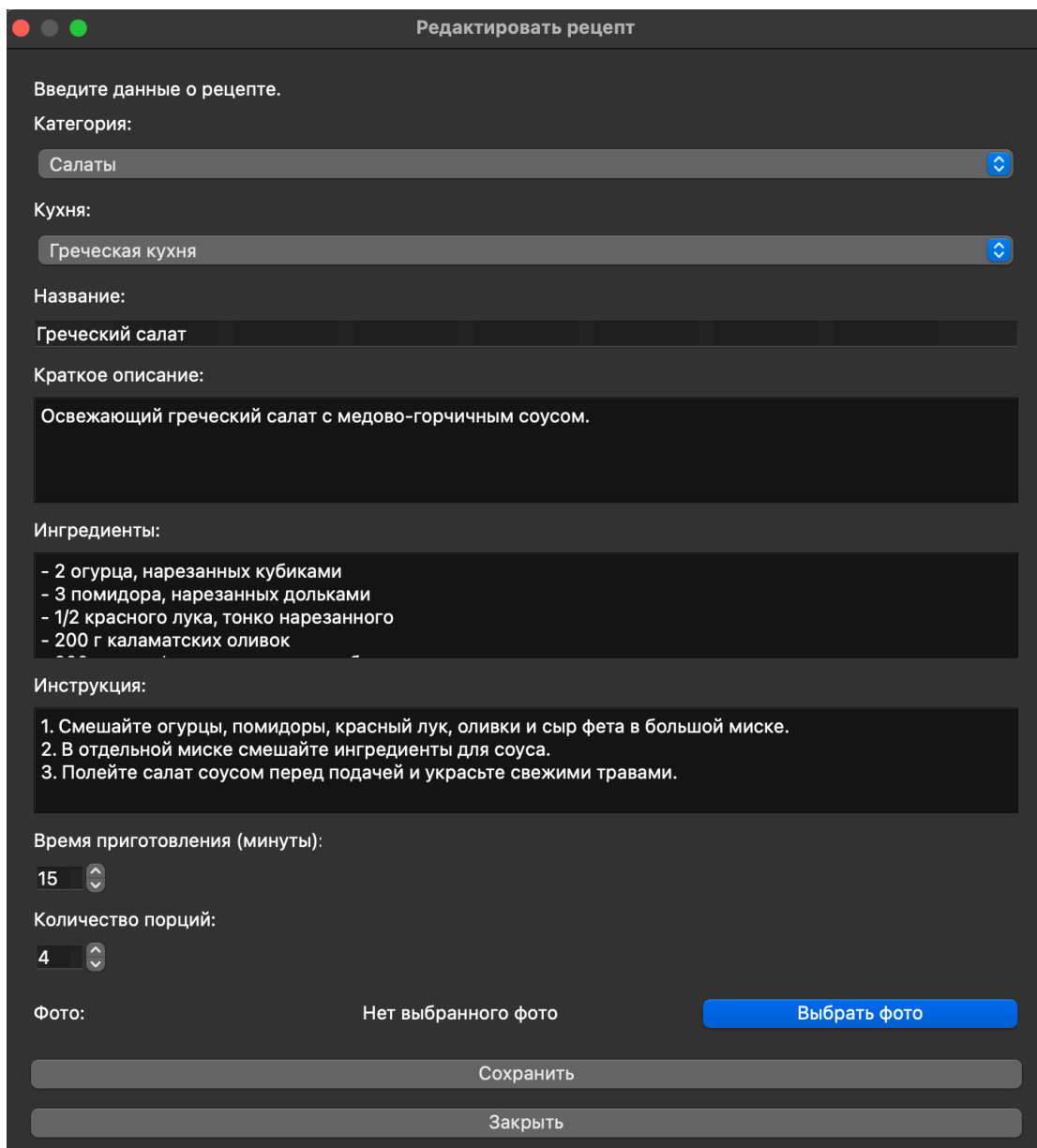
При нажатии на кнопку "Закрыть" рецепт не сохраняется и диалоговое окно закрывается.

Пользователь не сможет добавить новый рецепт, не заполнив поля "Название", "Время приготовления", "Количество порций".

Рисунок 5.2 – окно добавления рецепта

На рисунке 5.3 изображено окно, представляющее собой интерфейсную часть приложения, в которой можно отредактировать ранее заполненные данные существующего рецепта и успешно сохранить в базу данных.

Интерфейс окна редактирования рецепта идентичен интерфейсу окна добавления рецепта. Однако все поля уже заранее заполнены данными того рецепта, который пользователь собирается отредактировать.



Редактировать рецепт

Введите данные о рецепте.

Категория:

Салаты

Кухня:

Греческая кухня

Название:

Греческий салат

Краткое описание:

Освежающий греческий салат с медово-горчичным соусом.

Ингредиенты:

- 2 огурца, нарезанных кубиками
- 3 помидора, нарезанных дольками
- 1/2 красного лука, тонко нарезанного
- 200 г каламатских оливок

Инструкция:

1. Смешайте огурцы, помидоры, красный лук, оливки и сыр фета в большой миске.
2. В отдельной миске смешайте ингредиенты для соуса.
3. Полейте салат соусом перед подачей и украсьте свежими травами.

Время приготовления (минуты):

15

Количество порций:

4

Фото: Нет выбранного фото [Выбрать фото](#)

[Сохранить](#)

[Заккрыть](#)

Рисунок 5.3 – окно редактирования рецепта

На рисунке 5.4 изображено окно отображения всех данных о рецепте. В этом окне отображены все поля, которые можно было заполнить при добавлении рецепта в базу данных.

При нажатии на кнопку "Заккрыть" окно с отображением всех данных о рецепте закрывается.





Рисунок 5.4 – окно отображения рецепта

На рисунке 5.5 изображены дополнительные кнопки для событий. На операционной системе MacOS данные кнопки расположены в самом верху системного интерфейса.

При наведении курсора на кнопку "Приложение" выпадает меню, предлагающее только одну опцию "Выйти из приложения" (рисунок 5.6). При нажатии на эту опцию приложение закрывается.

При наведении курсора на кнопку "Категории" выпадает меню, предлагающее две опции "Добавить категорию" и "Удалить категорию" (рисунок 5.7). При нажатии на опцию "Добавить категорию" открывается диалоговое окно, в котором можно добавить новую категорию. При нажатии на опцию "Удалить категорию" удаляется категория, которая была выбрана в главном окне приложения.

При наведении курсора на кнопку "Кухни" выпадает меню, предлагающее две опции "Добавить кухню" и "Удалить кухню" (рисунок 5.8).

При нажатии на опцию "Добавить кухню" открывается диалоговое окно, в котором можно добавлять новую кухню. При нажатии на опцию "Удалить кухню" удаляется кухня, которая была выбрана в главном окне приложения.

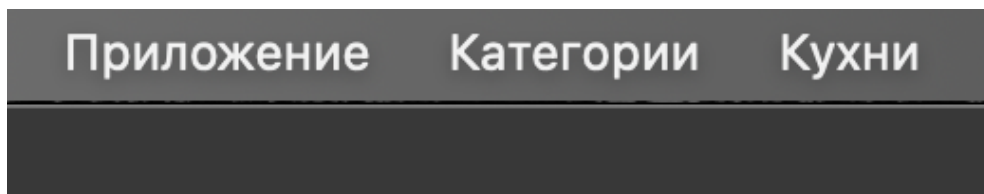


Рисунок 5.5 – дополнительные кнопки

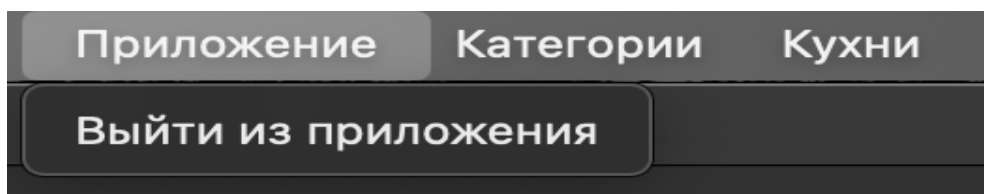


Рисунок 5.6 – опция кнопки "Приложение"

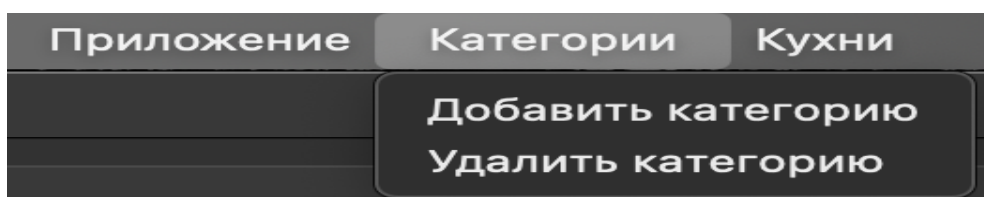


Рисунок 5.7 – опции кнопки "Категории"

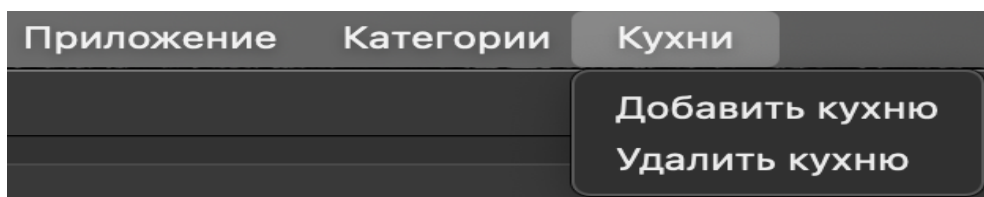


Рисунок 5.8 – опции кнопки "Кухни"

На рисунке 5.9 представлено диалоговое окно, в котором пользователь может добавить новую категорию.

При нажатии на кнопку "ОК" новая категория будет добавлена в файл `comboBoxCategory.txt` и появится в виджете `ComboBox`.

При нажатии на кнопку "Cancel" диалоговое окно закроется.

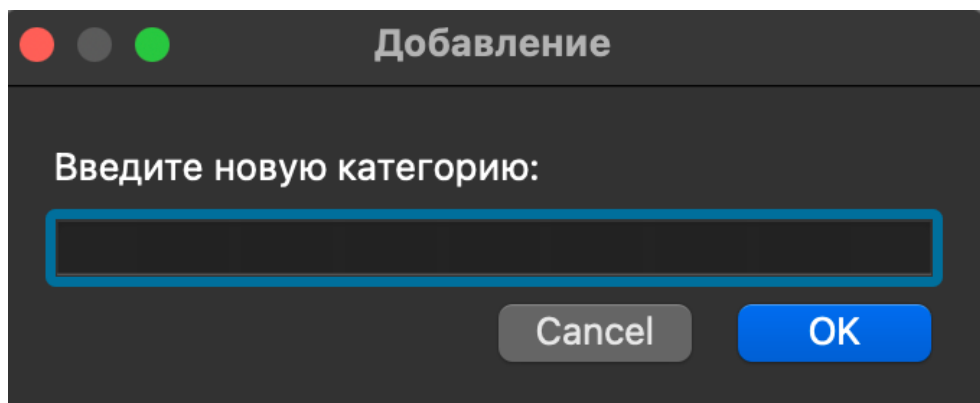


Рисунок 5.9 – диалоговое окно добавления категории

На рисунке 5.10 представлено диалоговое окно, в котором пользователь может добавить новую кухню.

При нажатии на кнопку "OK" новая кухня будет добавлена в файл `comboBoxKitchen.txt` и появится в виджете `ComboBox`.

При нажатии на кнопку "Cancel" диалоговое окно закроется.

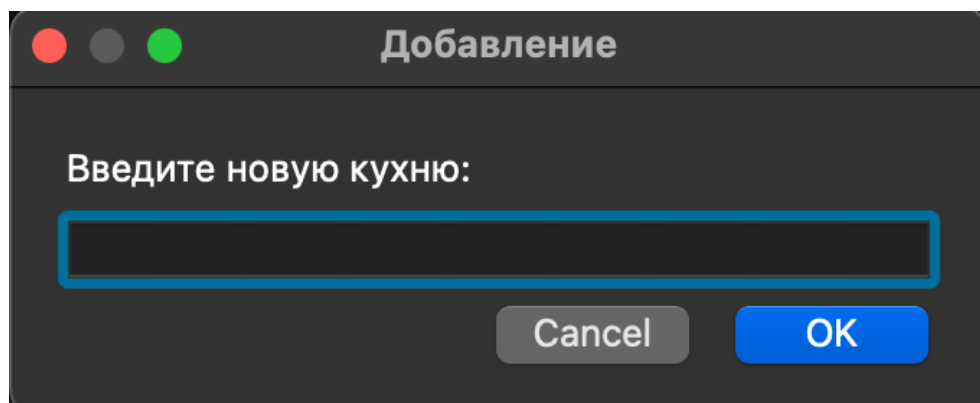


Рисунок 5.10 – диалоговое окно добавления кухни

## ЗАКЛЮЧЕНИЕ

"Кулинарный справочник" представляет собой полноценное приложение для управления кулинарными рецептами, предоставляя пользователям удобный и функциональный инструмент для хранения и управления своими кулинарными заметками.

В ходе разработки "Кулинарного справочника" были использованы передовые технологии, возможности библиотеки Qt и система управления базами данных SQLite, что позволило создать удобное приложение для персональной работы с рецептами.

Используемые технологии и инструменты:

В процессе разработки были использованы классы и функционал библиотеки Qt, такие как `'QWidget'`, `'QString'`, `'QFile'`, `'QLineEdit'`, `'QMessageBox'`, `'QPixmap'`, `'QHBoxLayout'`, `'QDialog'`, `'QLayout'`, `'QPushButton'` и многие другие. Эти инструменты обеспечили создание многооконного интерфейса, обработку событий по клавишам.

Также в процессе разработки была использована система управления базами данных SQLite. В работе были использованы такие SQL-запросы, как `"SELECT * FROM"`, `"INSERT INTO"`, `"UPDATE"`.

Работа с механизмом слотов и сигналов:

Одной из ключевых частей разработки было использование механизма слотов и сигналов библиотеки Qt. Этот мощный инструмент позволил эффективно организовать взаимодействие между элементами интерфейса и логикой приложения, обеспечивая отзывчивость и плавную работу кулинарного справочника.

Польза от приложения "Кулинарный справочник":

Кулинарное приложение может быть полезным инструментом для широкого круга пользователей. Оно предоставляет подробные рецепты с пошаговыми инструкциями, позволяя пользователям научиться готовить разнообразные блюда. Также имеется возможность добавлять свои собственные рецепты под своими личными категориями.

Интерфейс и хранение тестов:

Разработанный интерфейс представляет собой многооконную систему с интуитивными элементами управления. Рецепты хранятся в базах данных, что позволяет не нагружать систему и обеспечивает быстрый доступ ко всем данным.

Применение принципов ООП в разработке приложения:

Использование принципов ООП, таких как наследование, позволило создать собственные классы, унаследованные от базовых классов библиотеки Qt. Это позволило эффективно расширять функционал и адаптировать приложение для конкретных задач. Виртуальные методы, в свою очередь, обеспечили гибкость в работе с абстрактными классами Qt, позволяя создавать адаптивные решения.

Возможности пересборки под разные платформы: приложение успешно собрано для операционной системы MacOS, но благодаря правильной настройке qMake файла, можно осуществить пересборку под мобильные устройства и Windows. Это демонстрирует гибкость и переносимость разработанного приложения на различные платформы. Значение знаний ООП для будущих задач: знания объектно-ориентированного программирования (ООП) являются ключевыми для дальнейших профессиональных задач. Они позволяют разрабатывать более структурированный и расширяемый код, а также более легко адаптировать и модифицировать приложения под различные потребности и платформы.

Полученный опыт работы с ООП и библиотекой Qt будет ценным активом в дальнейшей разработке программного обеспечения. "Кулинарный справочник" является удобным приложением для создания и хранения разного рода рецептов. Используемые технологии Qt, механизм слотов и сигналов, а также созданный интерфейс делают это приложение максимально удобным и интуитивно понятным для любого пользователя.

## СПИСОК ЛИТЕРАТУРЫ

- [1] "Объектно-ориентированное программирование на C++" Бьярн Страуструп
- [2] "Язык программирования C++" Герберт Шилдт
- [3] "C++ Primer" Липман, Лажойе, Му, Хопкинс
- [4] "Алгоритмы. Построение и анализ" Кормен, Лейзерсон, Ривест, Штайн
- [5] "Введение в алгоритмы" Кормен, Лейзерсон, Ривест, Штайн
- [6] "Алгоритмы на C++" Роберт Седжвик, Кевин Уэйн
- [7] C++: эффективное программирование. 55 способов улучшения структуры программ и стиля кода" Scott Meyers
- [8] "Алгоритмы. Построение и анализ" Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, Clifford Stein
- [9] "Структуры данных и алгоритмы в C++" Robert Lafore
- [10] "C++ for Game Programmers" Noel Llopis
- [11] "SQLite Database System: Design and Implementation" D. Richard Hipp
- [12] "SQL Performance Explained" Markus Winand

**ПРИЛОЖЕНИЕ А**  
**(Обязательное)**  
Листинг кода

**ПРИЛОЖЕНИЕ Б**  
(Обязательное)  
Диаграмма классов



## **ПРИЛОЖЕНИЕ В**

**(Обязательное)**

Блок-схема алгоритма для метода `getRecipeRowById()`

## **ПРИЛОЖЕНИЕ Г**

**(Обязательное)**

Блок-схема алгоритма для метода on\_buttonEdit\_clicked()

**ПРИЛОЖЕНИЕ Д**  
**(Обязательное)**  
Ведомость документов