МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ

РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ

ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

«ВОРОНЕЖСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»

(ФГБОУ ВО «ВГТУ»)

Факультет информационных технологий и компьютерной безопасности

(факультет)

Кафедра Систем управления и информационных технологий в строительстве

КУРСОВАЯ РАБОТА

по дисциплине Программная инженерия

тема Разработка программного продукта для записи кулинарных рецептов

**Расчетно-пояснительная записка**

Разработал студент С.И. Комов

Подпись, дата Инициалы, фамилия

Руководитель О.В. Минакова

Подпись, дата Инициалы, фамилия

Члены комиссии

Подпись, дата Инициалы, фамилия

Подпись, дата Инициалы, фамилия

Нормоконтролер О.В. Минакова

Подпись, дата Инициалы, фамилия

Защищена \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Оценка \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

дата

2024

СОДЕРЖАНИЕ

[ВВЕДЕНИЕ 3](#_Toc167629464)

[1. Функциональное назначение программного средства 4](#_Toc167629465)

[1.1 Классы решаемых задач: 4](#_Toc167629466)

[1.1.1 Задачи поиска: 4](#_Toc167629467)

[1.1.2 Задачи анализа: 4](#_Toc167629468)

[1.1.3 Задачи оптимизации трудозатрат при решении задач поиска и анализа: 4](#_Toc167629469)

[1.1.4 Анализ прототипов 5](#_Toc167629470)

[1.2 Функциональное назначение: 6](#_Toc167629471)

[1.3 Нефункциональные требования, ограничения и интерфейсы 6](#_Toc167629472)

[2 Проектная часть 8](#_Toc167629473)

[2.1 Архитектура приложения 8](#_Toc167629474)

[2.2. Проектирование пользовательского интерфейса 10](#_Toc167629475)

[2.3. Модель данных ПС 16](#_Toc167629476)

[2.4. Моделирование работы приложения 17](#_Toc167629477)

[3. Конструирование программного продукта 19](#_Toc167629478)

[3.1 Диаграмма классов и их описание 19](#_Toc167629479)

[3.2 Тестирование ПС 26](#_Toc167629480)

[3.3 Создание документации 29](#_Toc167629481)

[ЗАКЛЮЧЕНИЕ 30](#_Toc167629482)

[СПИСОК ИСПОЛЬЗУЕМЫХ ИСТОЧНИКОВ 31](#_Toc167629483)

# ВВЕДЕНИЕ

В условиях стремительного развития информационных технологий потребность в удобных и функциональных инструментах для управления любыми бытовыми вещами становится все более актуальной. Современные тенденции в области кулинарии и питания требуют создания удобных и функциональных инструментов для управления рецептами и их подготовки. Рост интереса к домашнему приготовлению пищи, а также потребность в структурировании рецептов для более удобного использования, делает разработку специализированного программного обеспечения актуальной задачей. Программы для управления рецептами могут значительно облегчить процесс поиска, сортировки и приготовления блюд, предоставляя пользователям возможность хранить свои любимые рецепты в одном месте, легко находить и сортировать

Целью курсового проекта является разработка программы для управления кулинарными рецептами, которая позволит пользователям добавлять, редактировать и удалять рецепты, а также сортировать их по различным критериям и искать по ключевым словам.

Для достижения поставленной цели необходимо решить следующие задачи:

* Определить сценарии использования программы, включая добавление новых рецептов, редактирование существующих, удаление рецептов, сортировка и поиск по различным параметрам.
* Спроектировать пользовательский интерфейс, обеспечивающий удобное и интуитивно понятное взаимодействие с программой.
* Реализовать и протестировать прототип программы, включающий основные функциональные возможности, такие как добавление и редактирование рецептов, их сортировка и поиск, а также возможность помечать избранные рецепты.

# Функциональное назначение программного средства

Программное средство предназначено для управления кулинарными рецептами и обладает следующими функциональными возможностями:

## Классы решаемых задач:

### 1.1.1 Задачи поиска:

* Поиск рецептов по ключевым словам: Пользователь может искать рецепты по названию блюда, ингредиентам или другим ключевым словам, что позволяет быстро находить нужные рецепты.
* Фильтрация по критериям: Возможность фильтрации рецептов по различным критериям, таким как время приготовления, популярность, наличие ингредиентов и др.
* Сортировка результатов поиска: Рецепты могут быть отсортированы по релевантности, алфавиту, дате добавления или другим параметрам, что упрощает процесс поиска нужного рецепта.

### 1.1.2 Задачи анализа:

* Валидация данных рецепта: Проверка введенных пользователем данных для обеспечения их корректности, например, проверка формата времени приготовления или правильности списка ингредиентов.
* Анализ содержания рецептов: Автоматический анализ рецептов для выявления дубликатов или противоречий в описании, что позволяет поддерживать базу данных рецептов в актуальном и чистом состоянии.

### 1.1.3 Задачи оптимизации трудозатрат при решении задач поиска и анализа:

* Автоматизация рутинных процессов: Внедрение автоматических алгоритмов для рутинных задач, таких как сортировка и фильтрация рецептов, что значительно снижает временные затраты пользователя.
* Интуитивно понятный интерфейс: Разработка интерфейса, который позволяет пользователю легко и быстро выполнять поиск и анализ рецептов без необходимости изучать сложные инструкции.
* Интеграция с внешними источниками данных: Возможность интеграции с внешними базами данных для автоматического обновления и пополнения базы рецептов, что сокращает время на ручной ввод данных.

Так же существует концепция программы, представленная в приложении 1.

### 1.1.4 Анализ прототипов

Cookbook: Cookbook - это онлайн-инструмент, который предоставляет базовый функционал для хранения и управления рецептами. Пользователи могут добавлять, редактировать и организовывать свои рецепты, создавая категории и метки для удобства поиска. Однако он ограничен в функционале и не предлагает возможности обмена рецептами или автоматического импорта из Интернета.

My Kitchen Assistant: My Kitchen Assistant - более сложный программный продукт, который предоставляет расширенный функционал для хранения и управления рецептами. Он позволяет пользователям не только хранить и редактировать свои рецепты, но и делиться ими с другими пользователями, следить за кулинарными трендами и автоматически импортировать рецепты из различных источников в Интернете. Однако он может показаться сложным для новичков из-за большого количества функций и информации, доступной в приложении.

Таблица 1 – Анализ прототипов

|  |  |  |
| --- | --- | --- |
| Проблема | Cookbook | My Kitchen Assistant |
| Работа в offline режиме | - | - |
| Возможность сохранения данных на локальный компьютер | - | + |
| Наличие web версии | + | + |
| Интеграция с социальными сетями | + | - |
| Персонализированные рекомендации рецептов | - | + |

## Функциональное назначение:

Пользователь имеет возможность добавлять новые рецепты, редактировать существующие, удалять неактуальные записи, искать рецепты по различным критериям, таким как название, ингредиенты или ключевые слова в описании. Он также может просматривать детали каждого рецепта, включая ингредиенты и процесс приготовления, а также отмечать понравившиеся блюда как "избранные".

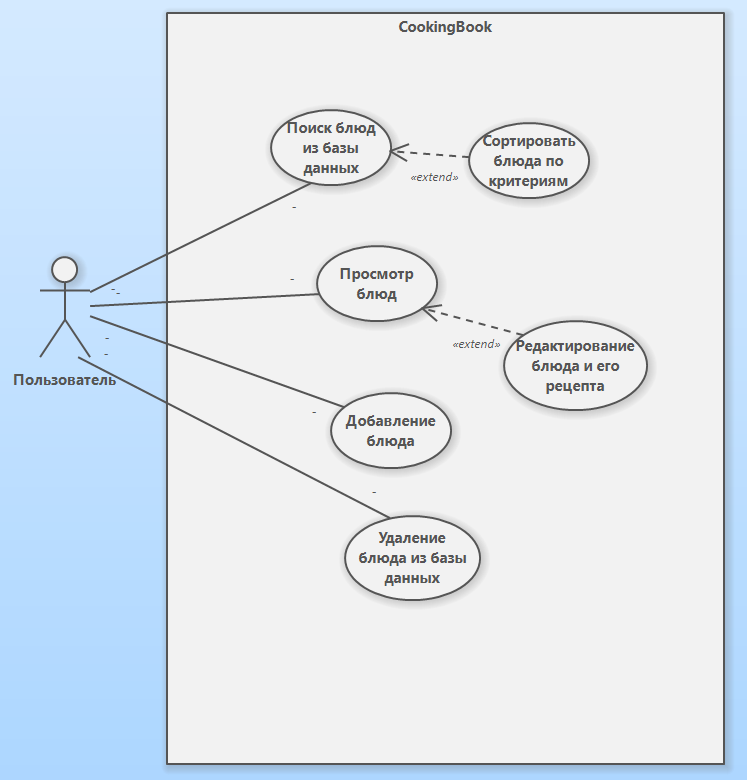


Рисунок 1 – Диаграмма вариантов использования

## Нефункциональные требования, ограничения и интерфейсы

**Производительность:**

* Система должна быть отзывчивой и быстрой, обеспечивая оперативный доступ к данным и мгновенный поиск рецептов.
* Время загрузки интерфейса и выполнения запросов должно быть минимальным, даже при работе с большим объемом данных.

**Безопасность:**

* Все данные пользователей должны храниться в безопасной базе данных.

**Надежность:**

* Система должна быть стабильной и надежной, предотвращая потерю данных или сбои в работе.

**Удобство использования:**

* Интерфейс приложения должен быть интуитивно понятным и простым в использовании, даже для неопытных пользователей.
* Все функции должны быть легко доступны через понятный и удобный интерфейс.

**Масштабируемость:**

* Система должна быть легко масштабируемой, позволяя добавлять новые функции и расширять возможности без серьезных изменений в архитектуре.
* База данных должна быть спроектирована таким образом, чтобы поддерживать рост количества пользователей и объема данных.

**Интерфейсы:**

* Для работы с внешними системами должны быть реализованы соответствующие интерфейсы и протоколы обмена данными.

Больше о функциональных и нефункциональных данных описаны в техничном задании в приложении.

# 2 Проектная часть

## 2.1 Архитектура приложения

Для реализации программного средства была выделена многослойная архитектура, которая представлена на рисунке 2.

MVC (Model-View-Controller) - это популярный шаблон проектирования, используемый для разработки программного обеспечения. Он разделяет приложение на три основных компонента:

* Модель (Model): Представляет данные и бизнес-логику приложения. Модель содержит методы для доступа, изменения и обработки данных. Она независима от пользовательского интерфейса и не знает, как эти данные будут отображаться.
* Представление (View): Отвечает за отображение данных пользователю. Представление получает данные из модели и отображает их в удобном для восприятия виде пользователю. Оно не содержит бизнес-логики и не обрабатывает данные, только отображает их.
* Контроллер (Controller): Связывает модель и представление. Он обрабатывает пользовательский ввод, взаимодействует с моделью для получения и обновления данных, а затем обновляет представление в соответствии с этими изменениями. Контроллер обеспечивает связь между моделью и представлением, но не содержит бизнес-логики.

Плюсы шаблона MVC:

* Разделение ответственности: MVC позволяет разделить приложение на три отдельных компонента. Каждый из них отвечает только за свою область функциональности. Это упрощает разработку, тестирование и поддержку приложения.
* Улучшенная модульность: Благодаря разделению на компоненты, каждый из них можно разрабатывать и изменять независимо от других.
* Гибкость и масштабируемость: MVC обеспечивает гибкую архитектуру, которая легко масштабируется и адаптируется к изменениям требований. Изменения в одной части приложения редко затрагивают другие части, что упрощает сопровождение и расширение приложения.
* Поддержка параллельной разработки: Разделение на компоненты позволяет нескольким разработчикам работать над различными частями приложения одновременно без значительных конфликтов.

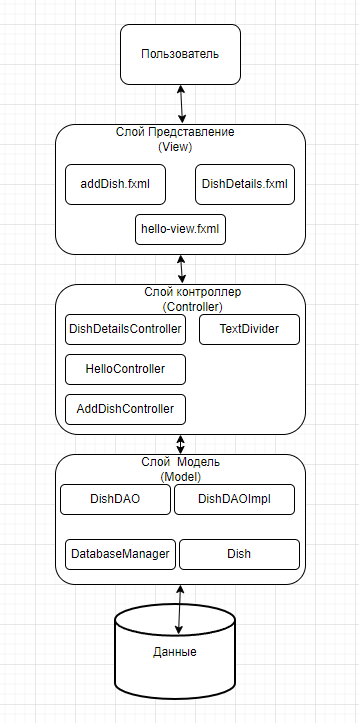


Рисунок 2 – Многоуровневая архитектура

Модель (Model): В данном случае, модель представлена классами DatabaseManager и Dish, интерфейс DishDAO и класс DishDAOimpl. DatabaseManager обеспечивает доступ к базе данных, Dish представляет модель данных блюда, DishDAO определяет стандартные операции, которые могут выполняться с объектами типа Dish в базе данных, а класс DishDAOimpl является конкретной реализацией интерфейса DishDAO.

Представление (View): Представление отображается пользователю и включает в себя окна и элементы управления GUI, описанные в FXML файлах addDish.fxml, dishDetails.fxml, hello-view.fxml.

Контроллер (Controller): Контроллеры обрабатывают пользовательский ввод и обновляют модель и представление в соответствии с этим вводом. AddDishController обрабатывает добавление нового блюда, HelloController управляет главным представлением и его действиями, DishDetailsController управляет данными конкретного блюда.

## 2.2. Проектирование пользовательского интерфейса

В соотвествии с ТЗ интерфейс пользователя должен быть реализован в виде 3 окон с разным функционалом.

**Окно добавления нового блюда:**

* Это окно предназначено для добавления новых записей о блюдах в приложение.
* Пользователь может вводить название блюда, указывать время приготовления, перечислять ингредиенты и описывать процесс приготовления.
* Также имеется возможность отметить блюдо как избранное.
* После заполнения всех полей пользователь может сохранить данные, чтобы они были доступны в приложении.

**Окно просмотра деталей блюда:**

* Это окно предназначено для просмотра подробной информации о выбранном блюде.
* Пользователь может просмотреть название, время приготовления, список ингредиентов и процесс приготовления блюда.
* Также пользователь может отметить блюдо как избранное.
* Есть возможность внести изменения и сохранить их.

**Окно списка блюд:**

* Это главное окно приложения, в котором отображается список всех блюд.
* Пользователь может просматривать список блюд, их названия, время приготовления и список ингредиентов.
* Также пользователь может фильтровать и сортировать список, а также удалять выбранные блюда.
* Есть возможность добавить новое блюдо, открыв окно добавления нового блюда.

Каждое окно обеспечивает определенный функционал для удобного взаимодействия пользователя с данными о блюдах в приложении, позволяя ему добавлять, просматривать, редактировать и удалять информацию о блюдах в соответствии с требованиями ТЗ.

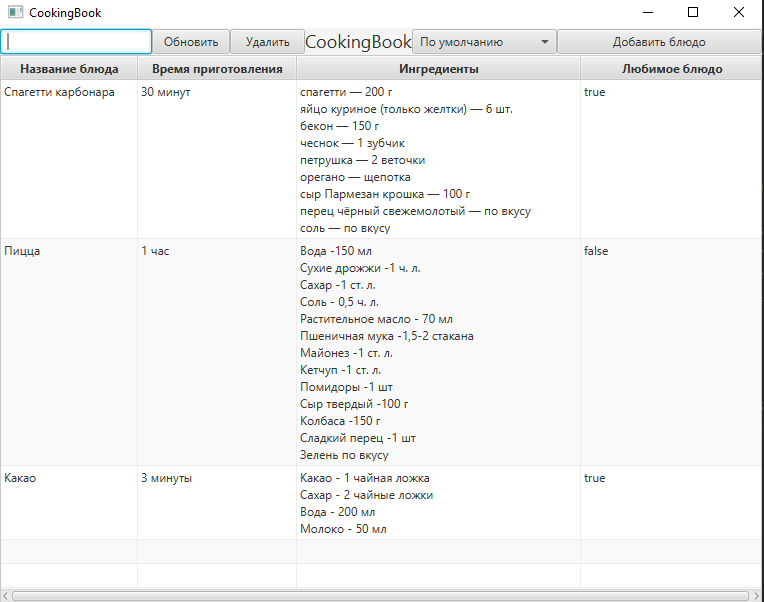


Рисунок 3 - Окно списка блюд

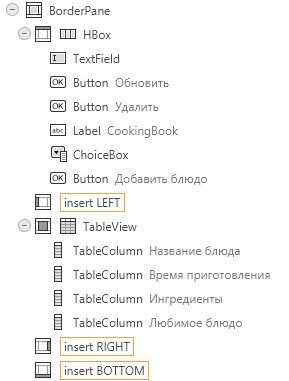


Рисунок 4 – Иерархия компонентов окна списка блюд

Таблица 2 – Элементы управления окна списка блюд

|  |  |
| --- | --- |
| Элементы управления | Назначение |
| TextField (fx  ="searchField"): | Это текстовое поле, предназначенное для ввода текста для поиска блюда. |
| Button (onAction="#refreshTable"): | Это кнопка "Обновить", которая запускает обновление списка блюд. |
| Button (onAction="#deleteSelectedDish"): | Это кнопка "Удалить", которая позволяет пользователю удалить выбранное блюдо из списка. |
| Label | Это метка "CookingBook", которая отображает название приложения в центре верхней части окна. |
| ChoiceBox (fx  ="sortChoiceBox"): | Это выпадающий список, который предоставляет пользователю возможность выбрать параметр сортировки списка блюд. |
| Button (onAction="#openAddDishWindow"): | Это кнопка "Добавить блюдо", которая открывает окно добавления нового блюда. |
| TableView (fx  ="dishesTable"): | Это таблица, в которой отображается список блюд. Каждая строка таблицы соответствует одному блюду, а столбцы содержат информацию о названии блюда, времени приготовления, ингредиентах и отметке "Любимое блюдо". |

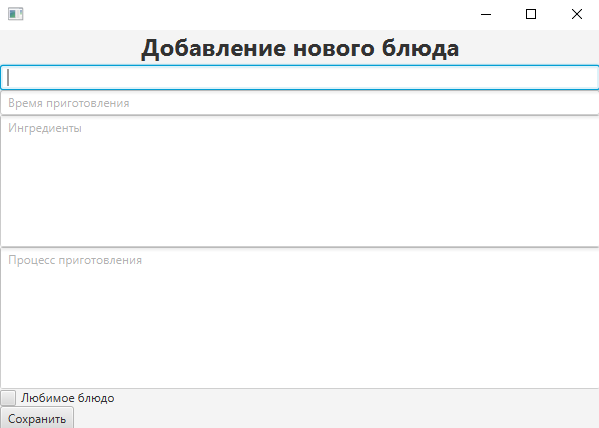


Рисунок 5 - Окно добавления новых блюд

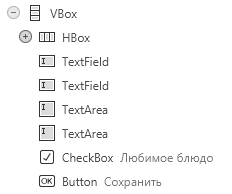


Рисунок 6 - Иерархия компонентов окна добавления новых блюд

Таблица 3 – Элементы управления окна добавления новых блюд

|  |  |
| --- | --- |
| Элементы управления | Назначение |
| Label | Это метка с заголовком "Добавление нового блюда", которая располагается в верхней части окна. |
| TextField, TextArea | Предназначены для заполнения информации о блюдах |
| CheckBox | Это флажок, который пользователь может установить, чтобы отметить новое блюдо как "Любимое блюдо". |
| Button (onAction="#saveDish"): | Это кнопка "Сохранить", которая запускает процесс сохранения нового блюда. |

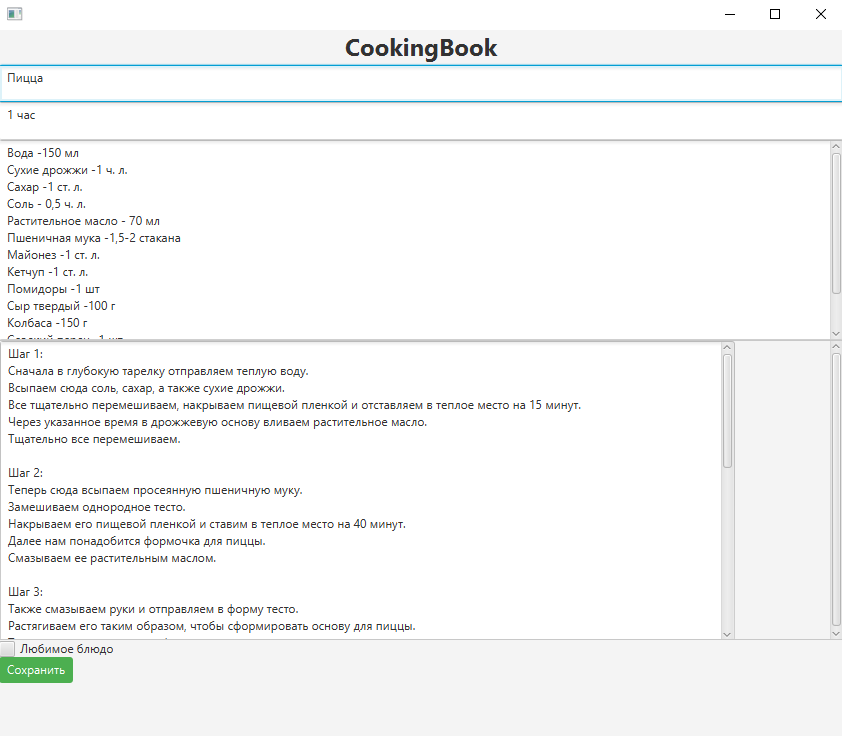


Рисунок 7 - Окно просмотра деталей блюда

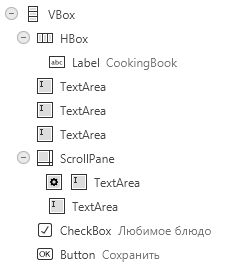


Рисунок 8 - Иерархия компонентов окна просмотра деталей блюда

Таблица 4 – Элементы управления окна просмотра деталей блюда

|  |  |
| --- | --- |
| Элементы управления | Назначение |
| Label | Метка с заголовком "CookingBook", расположенная в верхней части окна. |
| TextArea | Многострочное текстовое поле для отображения и редактирования данных о блюде. |
| ScrollPane | Предназначен для отображения и редактирования процесса приготовления блюда. |
| CheckBox | Флажок для отметки блюда как "Любимое блюдо". |
| Button | Кнопка "Сохранить", которая запускает процесс сохранения изменений в блюде. |

## 2.3. Модель данных ПС

Для данного программного средства была создана база данных “CookingBook” с одной таблицей “dishes”. Для создания базы данных была выбрана СУБД SQLite так как отличается простотой и легковесностью. Она не требует отдельного серверного процесса, а вся база данных хранится в одном файле на диске. Это делает SQLite идеальным выбором для программного средства.

Таблица 5 – таблица “dishes”

|  |  |  |
| --- | --- | --- |
| **Атрибуты** | **Тип данных** | **Ограничения** |
| ID | INTEGER | PRIMARY KEY AUTOINCREMENT |
| Name | TEXT | NOT NULL |
| Cooking\_Time | TEXT |  |
| Ingredients | TEXT |  |
| Cooking\_Process | TEXT |  |
| Favorite | INTEGER | DEFAULT 0 |

* ID - отвечает за уникальный идентификатор для каждой записи в таблице. Имеет ограничения: PRIMARY KEY AUTOINCREMENT. Тип данных: INTEGER, используется для обеспечения уникальности каждой строки в таблице и автоматически увеличивается при добавлении новой записи.
* Name - отвечает за название блюда. Имеет ограничения: NOT NULL. Тип данных: TEXT, используется для идентификации блюда и является обязательным полем, которое не может быть пустым.
* Cooking\_Time - отвечает за указание времени приготовления блюда. Ограничений нет. Тип данных: TEXT, используется для хранения времени приготовления, например, "30 минут" или "1 час".
* Ingredients - отвечает за список ингредиентов, необходимых для приготовления блюда. Ограничений нет. Тип данных: TEXT, используется для хранения текста с перечнем ингредиентов и их количеством.
* Cooking\_Process - отвечает за описание процесса приготовления блюда. Ограничений нет. Тип данных: TEXT, используется для хранения пошагового описания действий, необходимых для приготовления блюда.
* Favorite - отвечает за флаг, указывающий, является ли блюдо избранным. Имеет ограничения: DEFAULT 0. Тип данных: INTEGER, используется для указания, является ли блюдо избранным (0 - не избранное, 1 - избранное), значение по умолчанию - 0.

## 2.4. Моделирование работы приложения

На рисунке 10 представлены диаграммы последовательности, которые отображают полное взаимодействие пользователя с системой.

Пользователь начинает взаимодействие с системой, вводя запрос в поисковое поле. Этот запрос передается контроллеру HelloController, который обращается к DishDAO для поиска соответствующих блюд в базе данных. Результаты поиска отображаются в таблице.

При необходимости обновить список блюд, пользователь нажимает кнопку "Обновить", что приводит к запросу всех блюд у DishDAO, и их отображению в таблице.

Удаление блюда осуществляется путем выбора строки в таблице и нажатия кнопки "Удалить". Контроллер HelloController удаляет выбранное блюдо через DishDAO, что приводит к обновлению таблицы.

Добавление нового блюда начинается с нажатия кнопки "Добавить блюдо", что открывает окно добавления. После заполнения формы и сохранения данных, контроллер AddDishController добавляет новое блюдо через DishDAO.

При двойном клике на строку с блюдом, открывается окно с деталями блюда, где отображается вся информация о выбранном блюде. Редактирование данных возможно прямо в этом окне, и после сохранения изменений, они обновляются в базе данных.

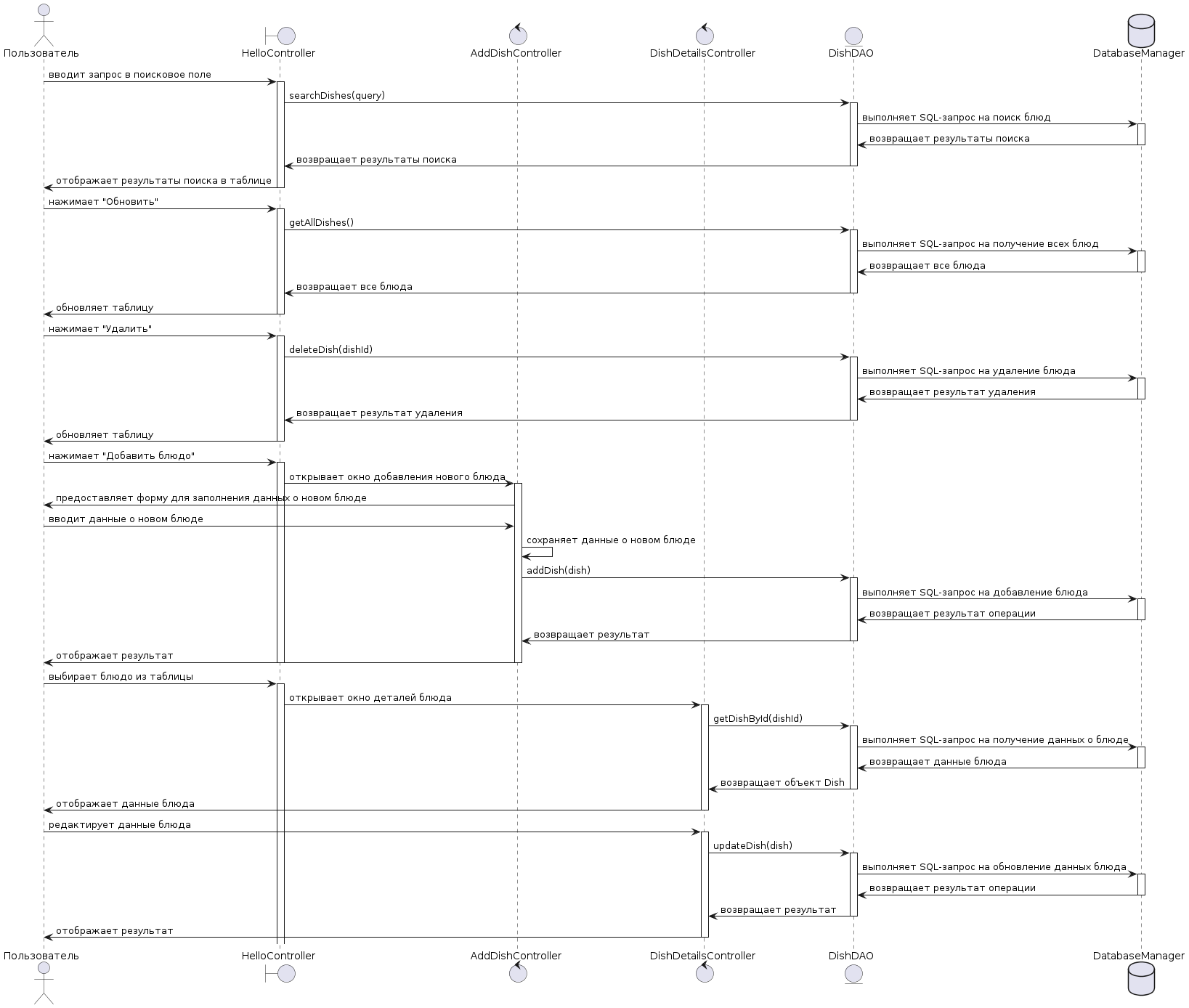


Рисунок 9 - Диаграмма последовательности взаимодействия пользователя с программным средством

# 3. Конструирование программного продукта

В качестве среды разработки программного продукта была выбрана интегрированная среда разработки программного обеспечения IntelliJ IDEA 2019. Так же был применён инструментарий GUI для Java – JavaFX.

## 3.1 Диаграмма классов и их описание

В ходе разработки логики программного продукта «CookingBook» были созданы следующие классы:

* Dish
* DishDAO
* DishDAOImpl
* TextDivider
* DatabaseManager
* HelloController
* AddDishController
* DishDetailsController
* HelloApplication

Структура проекта представлена на рисунке 6.

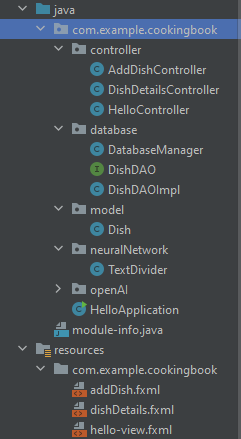


Рисунок 10 - Структура проекта

На рисунках 11-14 представлены диаграммы классов программы по пакетам

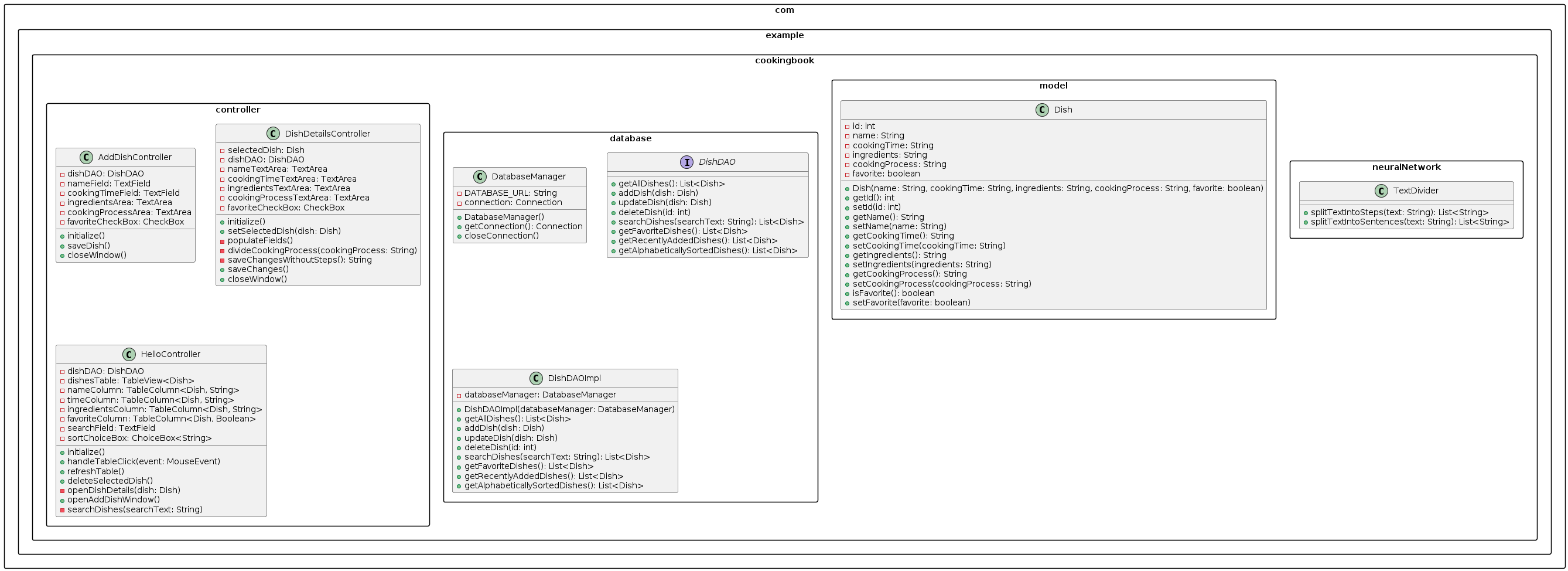


Рисунок 11 – Диаграмма классов пакета cookingbook

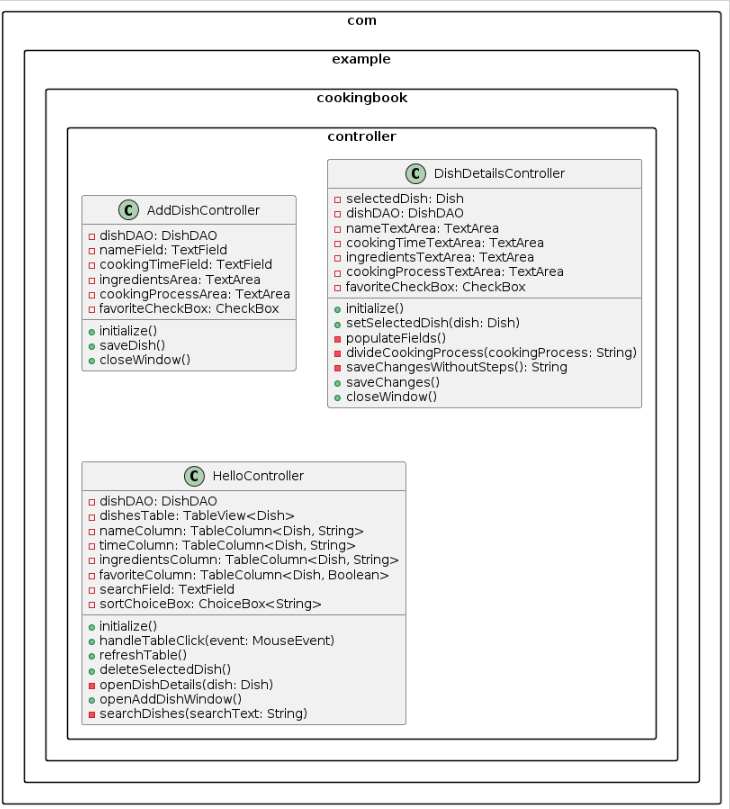


Рисунок 12 – Диаграмма классов пакета controller

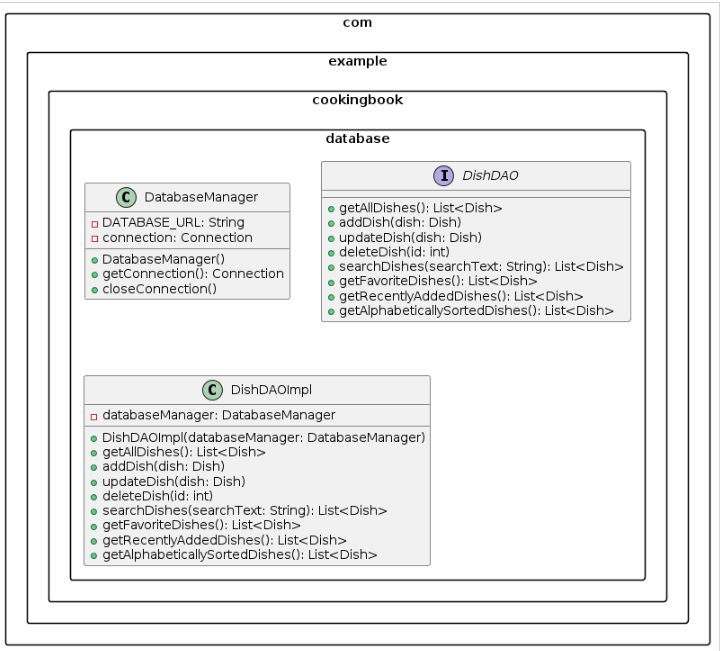


Рисунок 13 – Диаграмма классов пакета database



Рисунок 14 – Диаграмма классов пакета model и neturalNetwork

Пакет com.example.cookingbook.controller содержит:

Класс AddDishController:

* initialize(): Инициализирует контроллер, создавая объект доступа к базе данных.
* saveDish(): Сохраняет новое блюдо, считывая данные из полей формы и передавая их в базу данных.
* closeWindow(): Закрывает окно формы добавления блюда.

Класс DishDetailsController:

* initialize(): Инициализирует контроллер, создавая объект доступа к базе данных.
* setSelectedDish(dish: Dish): Устанавливает выбранное блюдо и заполняет соответствующие поля формы данными о нем.
* populateFields(): Заполняет поля формы данными о выбранном блюде.
* divideCookingProcess(cookingProcess: String): Разделяет текст процесса приготовления на отдельные шаги.
* saveChangesWithoutSteps(): String: Сохраняет изменения в блюде без учета шагов приготовления.
* saveChanges(): Сохраняет изменения в блюде.
* closeWindow(): Закрывает окно формы деталей блюда.

Класс HelloController:

* initialize(): Инициализирует контроллер, загружая данные о блюдах и настраивая обработчики событий.
* handleTableClick(event: MouseEvent): Обрабатывает клик по строке таблицы с блюдами.
* refreshTable(): Обновляет список блюд в таблице.
* deleteSelectedDish(): Удаляет выбранное блюдо из базы данных и из списка отображаемых блюд.
* openDishDetails(dish: Dish): Открывает окно с подробностями о выбранном блюде.
* openAddDishWindow(): Открывает окно для добавления нового блюда.
* searchDishes(searchText: String): Выполняет поиск блюд по тексту и отображает результаты в таблице.

Пакет com.example.cookingbook. database содержит:

Класс DatabaseManager:

* DatabaseManager(): Конструктор класса, инициализирует подключение к базе данных.
* getConnection(): Connection: Возвращает объект подключения к базе данных.
* closeConnection(): Закрывает соединение с базой данных.ъ

Интерфейс DishDAO:

* getAllDishes(): List<Dish>: Получает список всех блюд из базы данных.
* addDish(dish: Dish): Добавляет новое блюдо в базу данных.
* updateDish(dish: Dish): Обновляет информацию о блюде в базе данных.
* deleteDish(id: int): Удаляет блюдо из базы данных по его идентификатору.
* searchDishes(searchText: String): List<Dish>: Выполняет поиск блюд по заданному тексту.
* getFavoriteDishes(): List<Dish>: Получает список избранных блюд из базы данных.
* getRecentlyAddedDishes(): List<Dish>: Получает список последних добавленных блюд из базы данных.
* getAlphabeticallySortedDishes(): List<Dish>: Получает список блюд, отсортированных по алфавиту.

Класс DishDAOImpl:

* DishDAOImpl(databaseManager: DatabaseManager): Конструктор класса, устанавливает экземпляр DatabaseManager.
* getAllDishes(): List<Dish>: Получает список всех блюд из базы данных.
* addDish(dish: Dish): Добавляет новое блюдо в базу данных.
* updateDish(dish: Dish): Обновляет информацию о блюде в базе данных.
* deleteDish(id: int): Удаляет блюдо из базы данных по его идентификатору.
* searchDishes(searchText: String): List<Dish>: Выполняет поиск блюд по заданному тексту.
* getFavoriteDishes(): List<Dish>: Получает список избранных блюд из базы данных.
* getRecentlyAddedDishes(): List<Dish>: Получает список последних добавленных блюд из базы данных.
* getAlphabeticallySortedDishes(): List<Dish>: Получает список блюд, отсортированных по алфавиту.

Пакет com.example.cookingbook.model

Класс Dish:

* Dish(name: String, cookingTime: String, ingredients: String, cookingProcess: String, favorite: boolean): Конструктор класса, инициализирует объект блюда с заданными параметрами.
* getId(): int: Возвращает идентификатор блюда.
* setId(id: int): Устанавливает идентификатор блюда.
* getName(): String: Возвращает название блюда.
* setName(name: String): Устанавливает название блюда.
* getCookingTime(): String: Возвращает время приготовления блюда.
* setCookingTime(cookingTime: String): Устанавливает время приготовления блюда.
* getIngredients(): String: Возвращает список ингредиентов блюда.
* setIngredients(ingredients: String): Устанавливает список ингредиентов блюда.
* getCookingProcess(): String: Возвращает процесс приготовления блюда.
* setCookingProcess(cookingProcess: String): Устанавливает процесс приготовления блюда.
* isFavorite(): boolean: Возвращает значение, является ли блюдо избранным.
* setFavorite(favorite: boolean): Устанавливает значение, является ли блюдо избранным.

Пакет com.example.cookingbook.neuralNetwork:

Класс TextDivider:

* splitTextIntoSteps(text: String): List<String>: Разделяет текст на шаги приготовления и возвращает список шагов.
* splitTextIntoSentences(text: String): List<String>: Разделяет текст на предложения и возвращает список предложений.

## 3.2 Тестирование ПС

Таблица 6 – Test-Case для проверки сценария добавления нового блюда в базу данных

|  |  |  |  |
| --- | --- | --- | --- |
| **Название:** | Тест добавления нового блюда в базу данных | | |
| **Функция:** | Добавления нового блюда | | |
| **Действие** | | **Ожидаемый результат** | Результат теста:   * пройден * провален * заблокирован |
| **Предусловие:** | |  | |
| Запустить программное средство | | Программное средство успешно запущено, открыто главное окно | пройден |
| Нажать на кнопку «Добавить блюдо» | | Открывается окно добавления нового блюда | пройден |
| **Шаги теста:** | |  | |
| Вписать полную информацию о блюде в соответствующие поля. | | Набранный текст отображается в полях для ввода | пройден |
| Нажать кнопку «сохранить» | | Окно добавления блюда закрывается | пройден |
| **Постусловие:** | |  | |
| Нажать на кнопку «обновить» в главном окне приложения | | В списке блюд, появилось добавленное блюдо и вся информация о нём | пройден |

Таблица 7 – Test-Case для проверки сценария проверка удаления блюда из базы данных

|  |  |  |  |
| --- | --- | --- | --- |
| **Название:** | Тест удаления блюда из базы данных | | |
| **Функция:** | Удаления блюда | | |
| **Действие** | | **Ожидаемый результат** | **Результат теста:**   * пройден * провален * заблокирован |
| **Предусловие:** | |  | |
| Запустить программное средство | | Программное средство успешно запущено, открыто главное окно | пройден |
| **Шаги теста:** | |  | |
| Выбрать, одним кликом, определённую запись из списка блюд | | Запись подсвечивается синим | пройден |
| Нажать на кнопку «удалить» | | Выбранная запись исчезает из списка всех блюд | пройден |
| **Постусловие:** | |  | |
| Нажать на кнопку «обновить» в главном окне приложения | | В списке блюд, не появилась запись, которая была удалена | пройден |

Таблица 8 – Test-Case обновления информации о блюде в базе данных

|  |  |  |  |
| --- | --- | --- | --- |
| **Название:** | Тест обновления информации о блюде в базе данных | | |
| **Функция:** | Обновления информации о блюде | | |
| **Действие** | | **Ожидаемый результат** | **Результат теста:**   * пройден * провален * заблокирован |
| **Предусловие:** | |  | |
| Запустить программное средство | | Программное средство успешно запущено, открыто главное окно | пройден |
| Выбрать, двойным кликом, определённую запись из списка блюд | | Открывается новое окно с полной заполненной информацией о выбранном блюде | пройден |
| **Шаги теста:** | |  | |
| Изменить информацию, переписав текст или удалив строки, кликнуть по CheckBox. | | Изменённый текст отображается в полях | пройден |
| Нажать на кнопку «Сохранить» | | Окно с полной заполненной информацией о выбранном блюде закрывается | пройден |
| **Постусловие:** | |  | |
| Нажать на кнопку «обновить» в главном окне приложения | | В списке блюд изменилась информация , которая была изменена | пройден |

## 3.3 Создание документации

Код программного проекта загружен на <ссылка на репозиторий> с описанием по ГОСТ 19.402-78.

Разработанное руководство пользователя в приложении 4.

# ЗАКЛЮЧЕНИЕ

В ходе выполнения курсового проекта был разработан программный продукт CookingBook. В ходе выполнения проекта были реализованы следующие этапы:

Формулирование требований к ПО:

* Составлен список функций, необходимых для корректного функционирования калькулятора антропометрических показателей.
* Определены ограничения и требования к интерфейсу пользователя для удобства взаимодействия.

Проектирование программного средства:

* Выбрана подходящая архитектура приложения.
* Разработана структура базы данных.
* Спроектирован пользовательский интерфейс.

Реализация функций программного средства:

* Добавление нового рецепта.
* Редактирование существующего рецепта.
* Удаление рецепта.
* Поиск рецептов по названию или ингредиентам.
* Просмотр избранных рецептов.
* Просмотр недавно добавленных рецептов.
* Сортировка рецептов по алфавиту.

# СПИСОК ИСПОЛЬЗУЕМЫХ ИСТОЧНИКОВ

1. "Java Полное руководство" Г. Шилдт (Эксмо)
2. "Java. Эффективное программирование" Дж. Блох (Питер)
3. "Java. Паттерны проектирования" Т. Гамма, Р. Хелм, Р. Джонсон (Питер)
4. Jet Brains. IntelliJ IDEA - Режим доступа: https://[www.jetbrains.com/ru-ru/idea/.](http://www.jetbrains.com/ru-ru/idea/)
5. "Java. Методы программирования" А. Немцев и А. Федеров (Вильямс)
6. "Объектно-ориентированное проектирование на Java" Б. Селектор и Д. Прыцков (ДМК Пресс)
7. Диаграммы вариантов использования - Режим доступа: https://habr.com/ru/articles/566218/
8. Использование Дигорам классов - Режим доступа: https://flexberry.github.io/ru/gpg\_class-diagram.html
9. Тестирование ПС - Режим доступа: https://www.atlassian.com/ru/continuous-delivery/software-testing/types-of-software-testing
10. Диорамы последовательности - Режим доступа: https://flexberry.github.io/ru/fd\_sequence-diagram.html
11. Клещев А.С., Грибова В.В. Методы и средства разработки пользовательского интерфейса: современное состояние. - Режим доступа: https://guimachine.ru/?p=937.
12. Влацкая, И.В. Проектирование и реализация прикладного программного обеспечения: учебное пособие/ И.В. Влацкая, Н.А. Заельская, Н.С. Надточий; Оренбурский гос. ун-т. – Оренбург: ОГУ, 2015. – 118 с.

**Приложение 1. Концепция программы**

**Концепция программы**

**1. Введение**

В условиях стремительного развития информационных технологий потребность в удобных и функциональных инструментах для управления любыми бытовыми вещами становится все более актуальной. Современные тенденции в области кулинарии и питания требуют создания удобных и функциональных инструментов для управления рецептами и их подготовки. Рост интереса к домашнему приготовлению пищи, а также потребность в структурировании рецептов для более удобного использования, делает разработку специализированного программного обеспечения актуальной задачей.

**2. Определение программного продукта**

**2.1 Описание проблемы**

Проблема: Затрачивается большое количество времени на поиск и организацию рецептов в различных кулинарных книгах и веб-сайтах. Это может вызвать задержки в процессе приготовления блюд и увеличивает временные затраты на поиск нужных рецептов.

**Воздействует на:** Пользователей, которые хотят быстро и легко находить рецепты и следовать инструкциям для приготовления блюд.

Результатом чего является: Задержки в поиске и организации рецептов могут затруднить процесс приготовления пищи, что может снизить удовлетворенность пользователей и их желание готовить дома.

**Выигрыш от ПП:** Повышение удовлетворенности пользователей за счет быстрого и эффективного поиска и организации рецептов, что способствует более удобному и продуктивному приготовлению пищи.

**2.2 Цель программного продукта**

Для: Пользователей, интересующихся кулинарией, а также профессиональных поваров, желающих быстро находить и управлять рецептами.

Который: Имеет быстрый и удобный способ поиска, хранения и сортировки рецептов, а также возможность отметки избранных рецептов и добавления новых.

В отличие от: Кулинарных книг или других онлайн-ресурсов, которые могут быть неудобны в использовании или не предоставляют возможность эффективно управлять рецептами.

Наш продукт: Обеспечивает мгновенный доступ к рецептам и их организация через простой и интуитивно понятный интерфейс, что позволяет пользователям быстро и удобно находить и следовать рецептам, добавлять свои собственные рецепты и отмечать избранные блюда, способствуя более продуктивному и приятному процессу приготовления пищи.

**3. Контекст использования**

**3.1 Описание заинтересованных лиц**

|  |  |  |
| --- | --- | --- |
| Имя | Описание | Цели, интересы, предпочтения |
| Пользователь | Лицо, использующее приложение CookingBook | Быстро и удобно находить рецепты, следовать инструкциям для приготовления блюд, хранить и управлять своими рецептами, отмечать любимые рецепты и искать новые идеи для готовки. |

**3.2 Окружение продукта**

Количество людей, участвующих в выполнении задачи: В процессе разработки приложения CookingBook участвует один разработчик.

Продолжительность цикла задачи: Продолжительность цикла задачи составляет 18 недель.

Уникальные ограничения окружающей среды: Нет уникальных ограничений окружающей среды.

Используемые системные платформы: Проект разрабатывается с использованием IntelliJ IDEA, и что обеспечивает гибкость в разработке и возможность использования на различных операционных системах, таких как Windows, macOS и Linux.

**4. Общее описание**

**4.1 Назначение программного средства**

**Какие системы смогут применять программное средство для решения своих задач:**

**Различные кулинарные приложения: Программное средство может быть интегрировано в кулинарные приложения для управления рецептами, поиска новых рецептов и создания персонализированных кулинарных книг.**

От каких систем/пользователей поступают входные данные программного средства:

* Пользователи: Вводят данные о рецептах, такие как название блюда, время приготовления, ингредиенты, процесс приготовления и отмечают любимые рецепты.

Какие сторонние системы/сервисы участвуют в процессе обработки данных: Библиотека sqlite-jdbc используется для подключения и взаимодействия с базой данных SQLite. Она обеспечивает все необходимые драйверы и методы для работы с SQLite. Библиотека javafx-fxml используется для загрузки и работы с FXML файлами, которые описывают структуру пользовательского интерфейса JavaFX приложения.

**4.2 Ограничения, допущения и зависимости**

**Ограничения:**

Программное обеспечение должно обеспечивать быстрый доступ к рецептам и выполнение операций с базой данных, с временем отклика не более 5 секунд для основных операций (поиск рецептов, добавление новых рецептов, обновление и удаление рецептов).

**Допущения:**

Корректность вводимых данных: Программа предполагает, что пользователи вводят корректные и точные данные о рецептах.

**Зависимости:**

Библиотека sqlite-jdbc используется для подключения и взаимодействия с базой данных SQLite. Она обеспечивает все необходимые драйверы и методы для работы с SQLite. Библиотека javafx-fxml используется для загрузки и работы с FXML файлами, которые описывают структуру пользовательского интерфейса JavaFX приложения.

**4.3 Характеристики потребностей**

|  |  |  |  |
| --- | --- | --- | --- |
| **Необходимость** | **Приоритет** | **Характеристика** | **Планируемая реализация** |
| Быстрый доступ к рецептам | Высокий | Программа должна обеспечивать быстрый доступ к рецептам, минимизируя время ожидания пользователей. | Оптимизация запросов к базе данных и использование кэширования для ускорения получения данных. |
| Интуитивно понятный интерфейс | Высокий | Пользователи должны легко находить, добавлять, редактировать и удалять рецепты через простой и удобный интерфейс. | Разработка удобного и интуитивного пользовательского интерфейса с использованием компонентов JavaFX. |
| Поиск и фильтрация рецептов | Высокий | Пользователи должны иметь возможность легко искать и фильтровать рецепты по различным критериям, таким как название, время приготовления, ингредиенты и т.д. | Реализация функций поиска и фильтрации с использованием SQL-запросов и соответствующих интерфейсов. |

**4.4 Альтернативы**

Cookbook: Cookbook - это онлайн-инструмент, который предоставляет базовый функционал для хранения и управления рецептами. Пользователи могут добавлять, редактировать и организовывать свои рецепты, создавая категории и метки для удобства поиска. Однако он ограничен в функционале и не предлагает возможности обмена рецептами или автоматического импорта из Интернета.

My Kitchen Assistant: My Kitchen Assistant - более сложный программный продукт, который предоставляет расширенный функционал для хранения и управления рецептами. Он позволяет пользователям не только хранить и редактировать свои рецепты, но и делиться ими с другими пользователями, следить за кулинарными трендами и автоматически импортировать рецепты из различных источников в Интернете. Однако он может показаться сложным для новичков из-за большого количества функций и информации, доступной в приложении.

**5. Другие требования к продукту**

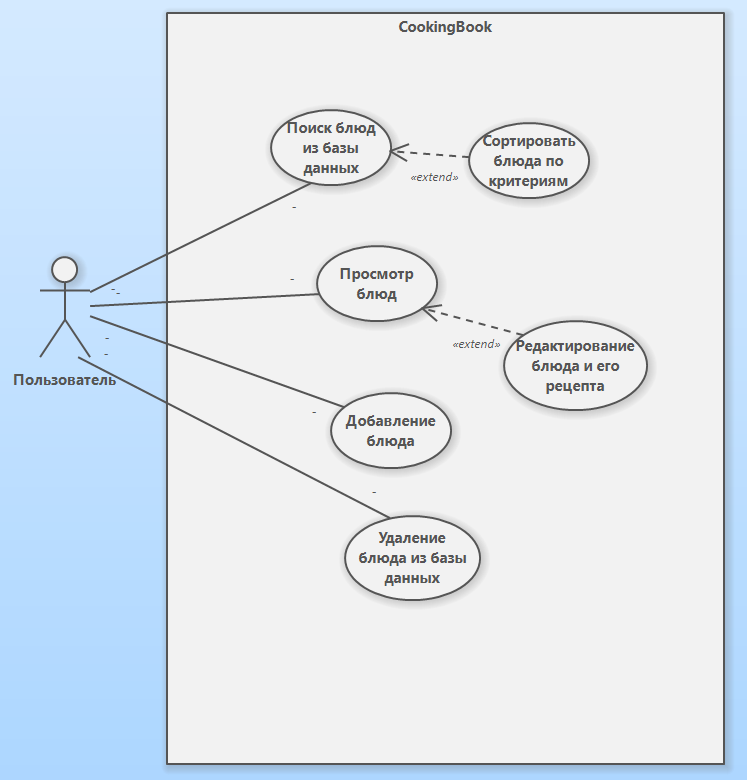
Требования к надежности:

Восстановление после сбоя не более чем за 5 минут

В случае сбоев или отказов программного обеспечения, важно свести к минимуму время, необходимое для восстановления доступа к данным и возобновления работы функциональности.

Устойчивость к сбоям: Программа должна автоматически восстанавливаться после сбоев.

**Приложение 2. Диаграмма вариантов использования**



**Приложение 3. Техническое задание**

**1 Введение**

Настоящее техническое задание распространяется на разработку программного продукта «CookingBook» для применения в области управления и хранения рецептов блюд. Программа предназначена для упрощения процесса организации рецептов, предоставляя пользователям возможность добавлять новые рецепты, редактировать и удалять существующие.

**2 Основания для разработки**

Программный продукт разрабатывается в соответствии с учебным планом кафедры «Систем управления и информационных технологий в строительстве» ВГТУ по дисциплине «Программная инженерия».

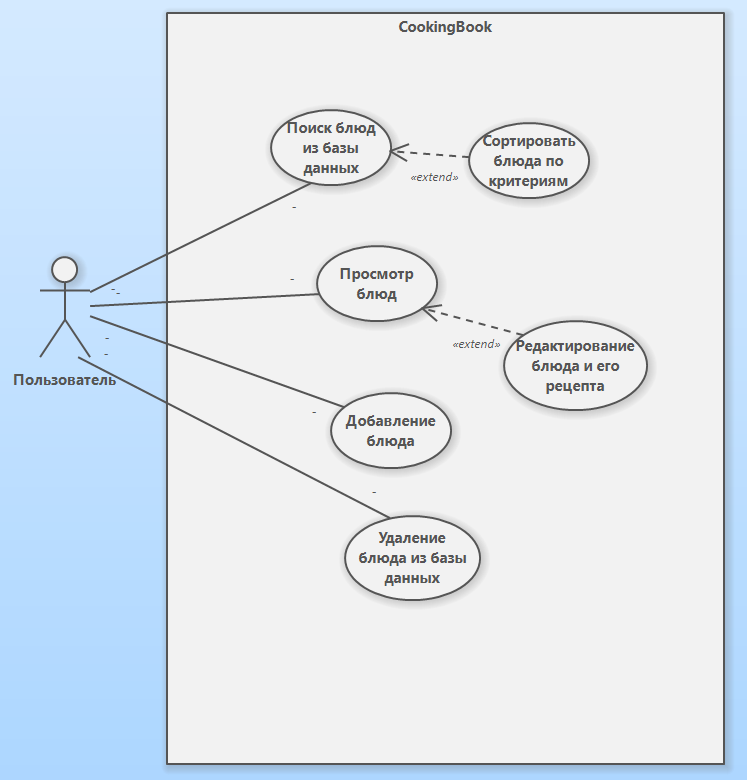
**3 Назначение разработки**

Программный продукт «CookingBook» разработан для упрощения организации и управления кулинарными рецептами и блюдами. Основная цель данного продукта - облегчить процесс создания, редактирования и хранения рецептов, а также обеспечить пользователей инструментами для удобного поиска и сортировки блюд. Программа предназначена для повышения эффективности пользователей в приготовлении разнообразных блюд, делая процесс кулинарного творчества более организованным.

4 Требования к функциональным характеристикам программного продукта

4.1 Функциональные требования

|  |  |
| --- | --- |
| Идентификатор функционального требования | Описание |
| FUN-01 | Пользователь может добавить новое блюдо, введя его название, время приготовления, ингредиенты, процесс приготовления и отметив, является ли это блюдо его любимым. |
| FUN-02 | Редактирование существующего блюда |
| FUN-03 | Пользователь может удалить выбранное блюдо из базы данных. |
| FUN-04 | Поиск блюд и базы данных |
| FUN-05 | Пользователь может сортировать блюда по различным критериям, таким как понравившиеся, последние добавленные и по алфавиту. |
| FUN-06 | Разбиение рецепта на конкретные шаги. |



4.2 Требования к надежности

|  |  |
| --- | --- |
| Идентификатор требования надежности | Описание |
| REL-01 | Программное обеспечение использует локальную базу данных для хранения данных. В случае возникновения ошибок или сбоев система обрабатывает их корректно, обеспечивая целостность данных. |
| REL-02 | В случае сбоев или отказов программного обеспечения, важно свести к минимуму время, необходимое для восстановления доступа к данным и возобновления работы функциональности. |
| REL-03 | Корректное сохранение данных при сбоях |

4.3 Условия эксплуатации

|  |  |
| --- | --- |
| Идентификатор требования производительности | Описание |
| PER-01 | Обработка не менее 10 запросов в секунду при максимальной нагрузке |
| PER-02 | Время отклика системы не превышает 5 секунд при обычной нагрузке |
| PER-03 | Восстановление после сбоя не более чем за 5 минут |

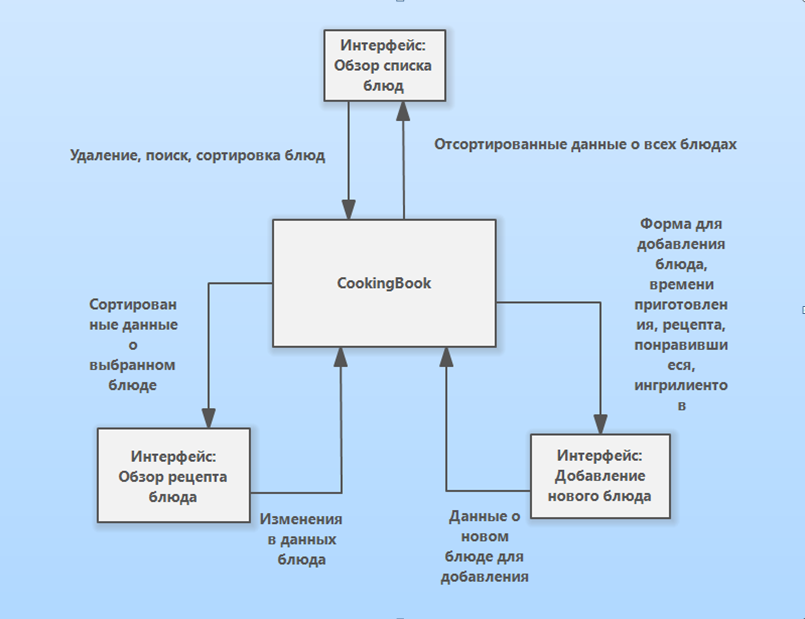
|  |  |
| --- | --- |
| Идентификатор требования поддерживаемости | Описание |
| SUP-01 | Система должна поддерживать возможность проведения различных тестов для обеспечения её корректной работы. |
| SUP-02 | Система должна быть спроектирована таким образом, чтобы её функциональность можно было легко расширять при необходимости. |
| SUP-03 | Система должна быть легко восстанавливаемой после сбоев и ошибок, обеспечивая минимальные простои. |
| SUP-04 | ПО должно быть удобным для технического обслуживания и обновления. |
| SUP-05 | ПО должно корректно работать в различных средах и на различных устройствах, обеспечивая одинаковый уровень функциональности. |

**4.4 Требования к составу и параметрам технических средств**

Минимальная конфигурация технических средств:

* Тип процессора Core i3 - 330M
* Объем ОЗУ 3072 Мб.
* Жесткий диск HDD 50 Гб
* Любая видеокарта (например, GeForce 310M)

**4.5 Требования к информационной и программной совместимости**



|  |  |
| --- | --- |
| Идентификатор требования к интерфейсам | Описание |
| INT-01 | Интерфейс: “Обзор списка блюд”:  INT-1.1: Возможность удаления данных блюда.  INT-1.2: Возможность сортировки вывода данных о блюдах.  INT-1.2: Возможность поиска блюда по названию. |
| INT-02 | Интерфейс: “Добавление нового блюда”:  INT-2.1: Предоставления формы для заполнения новых данных и кнопка для сохранения. |
| INT-03 | Интерфейс: “Обзор рецепта блюда”:  INT-3.1: Вывод всех данных об определённом блюде.  INT-3.2: Возможность изменить данные о блюде.  INT-3.3: Вывод данных рецепта по определённым шагам. |

**4.6 Требования к маркировке и упаковке.**

Не предъявляются.

**4.7 Требования к транспортированию и хранению.**

Не предъявляются.

4.8 Специальные требования.

|  |  |
| --- | --- |
| Идентификатор требования удобства | Описание |
| USA-01 | Возможность менять размер окна |
| USA-02 | Возможность менять размер таблиц |
| USA-03 | Перемещение окна в любую часть экрана |

Графический пользовательский интерфейс будет иметь форму десктопного приложения, где все функциональные возможности визуально и логически сгруппированы в тематические блоки, соответствующие типам объектов, с которыми они связаны, например, управление списком рецептов и добавление новых блюд.

Особое внимание уделено эргономике приложения. Основная концепция заключается в обеспечении удобства и простоты использования, с акцентом на минимизацию усилий пользователя для выполнения основных задач. Все элементы интерфейса организованы так, чтобы обеспечить интуитивно понятное взаимодействие и легкость навигации.

**5 Требования к программной документации**

Разрабатываемое программное обеспечение должно включать:

* Техническое задание;
* Пояснительную записку;
* Руководство пользователя;

**6 Технико-экономические показатели**

Трудоемкость разработки – 108 чел/ч

**7 Стадии и этапы разработки**

|  |  |  |  |
| --- | --- | --- | --- |
| 1 | Выбор темы и согласование ТЗ | 1-2 неделя | Документ Концепция проекта |
| 2 | Анализ предметной области и точная постановка задачи в форме внешнего описания | 3-5 неделя | Документ ТЗ |
| 3 | Проектирование ПС | 6-9 неделя | Архитектурная спецификация |
| 4 | Кодирование и отладка | 10-13 неделя | Код программы с комментариями |
| 5 | Подготовка программной документации | 14-17 неделя | Руководство пользователя |
| 6 | Защита проекта | 18 неделя |  |

**8 Порядок контроля и приемки**

Испытание представленной модели и контроль качества ее работы провести на базе компьютерного класса. Во время испытаний проверить работу системы по тестовым сценариям (приложение 3):

* добавления нового блюда
* обновления информации о блюде
* проверка удаления блюда

**Приложение 4. Руководство пользователя**

В данном разделе приводится описание всех операций, существующих в программе.

1. При запуске программы открывается главное окно с некоторыми функциями.

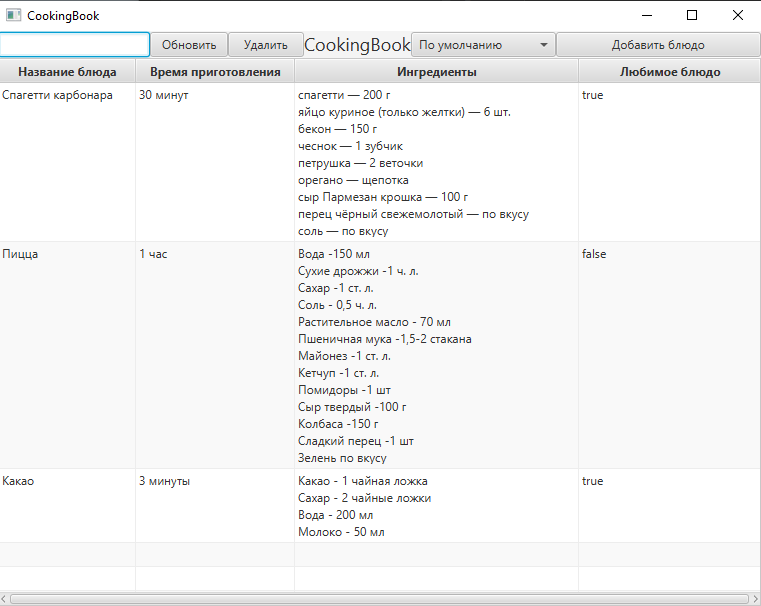


Рисунок 1 – Главное окно программы

1. При нажатии на кнопку “Добавить блюдо” открывается окно добавления блюда, где можно заполнить новый рецепт и нажав на кнопку “сохранить” заполненная информация сохраняется в базе данных, окно автоматически закрывается. Нажав на кнопку “обновить”, запись добавляется в общий список блюд в главном окне.

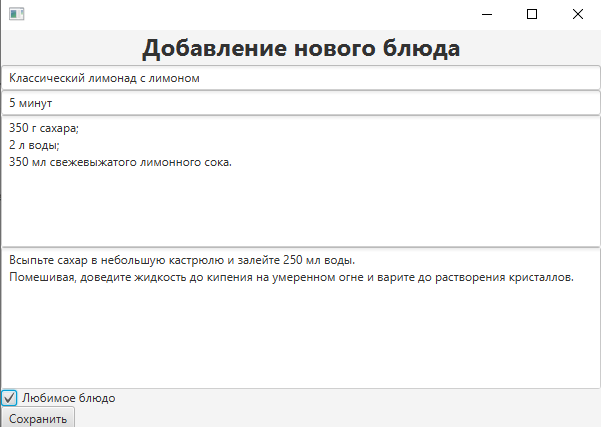


Рисунок 2 – Окно добавления нового блюда

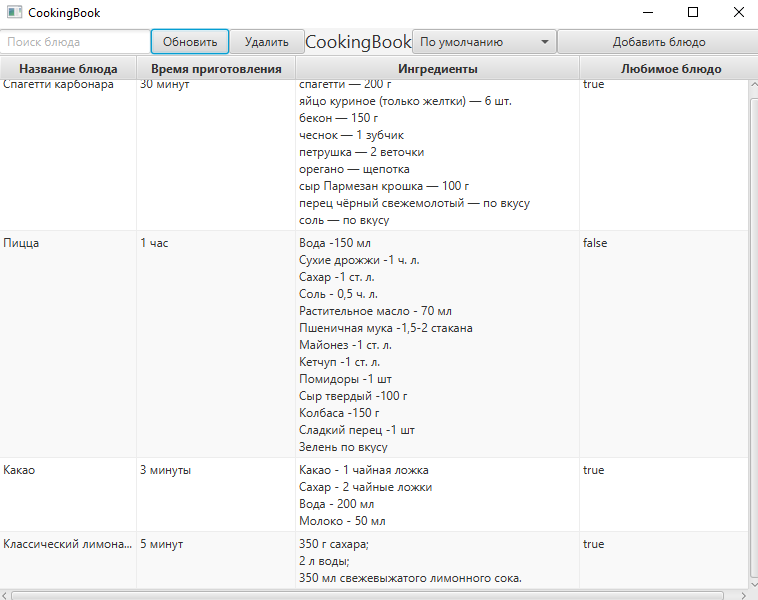


Рисунок 3 – Новое блюдо в списке блюд

1. Нажав на сортировку вывода базы данных, можно отсортировать вывод блюд по нужным критериям.

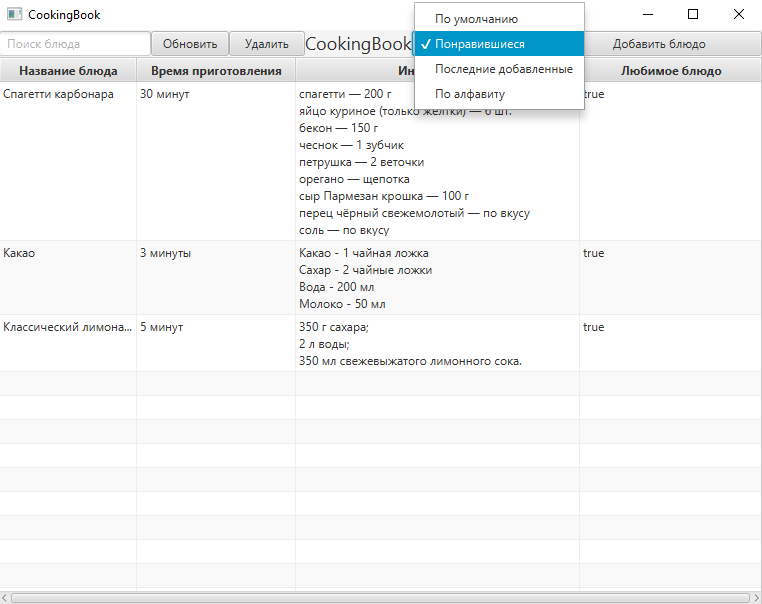


Рисунок 3 – Сортировка вывода списка блюд по критерию “Понравившиеся”

1. Кликнув два раза на запись, открывается окно с выбранным блюдом и всей информацией о нём, информацию можно отредактировать, изменив текст и нажав кнопку “Сохранить”, изменения сохраняться в базу данных.

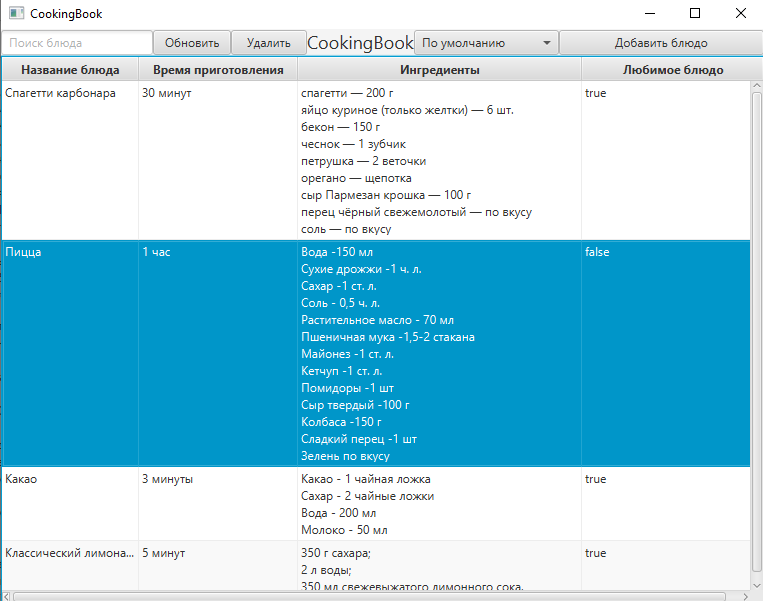


Рисунок 4 – Выбор блюда, у которого нужно посмотреть рецепт

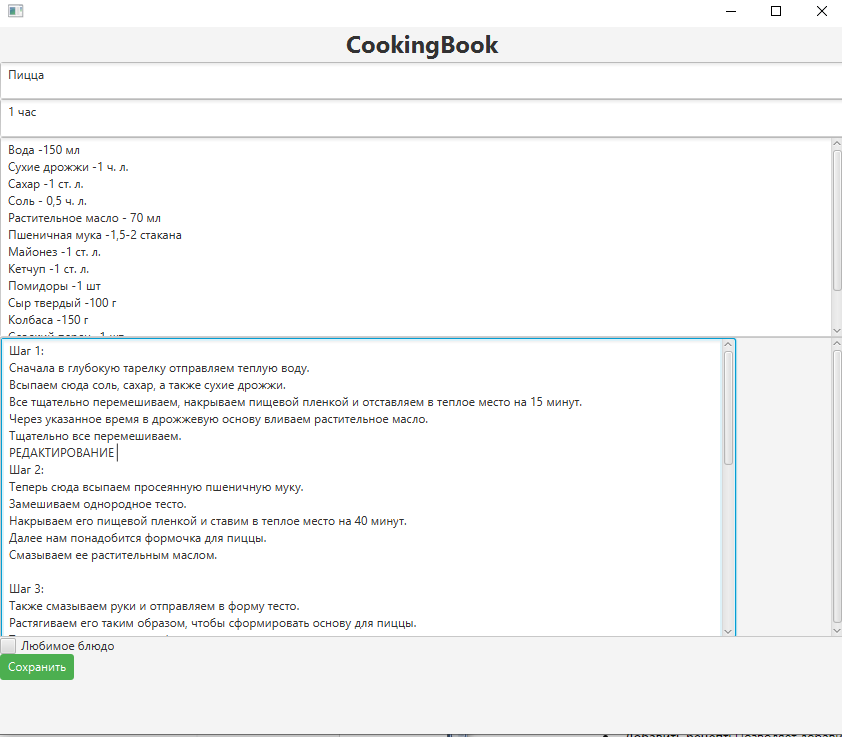


Рисунок 5 – Окно с полной информацией о блюде с делением рецепта по шагам

1. Нажав на окно слева вверху, можно ввести текст и найти блюдо по названию.

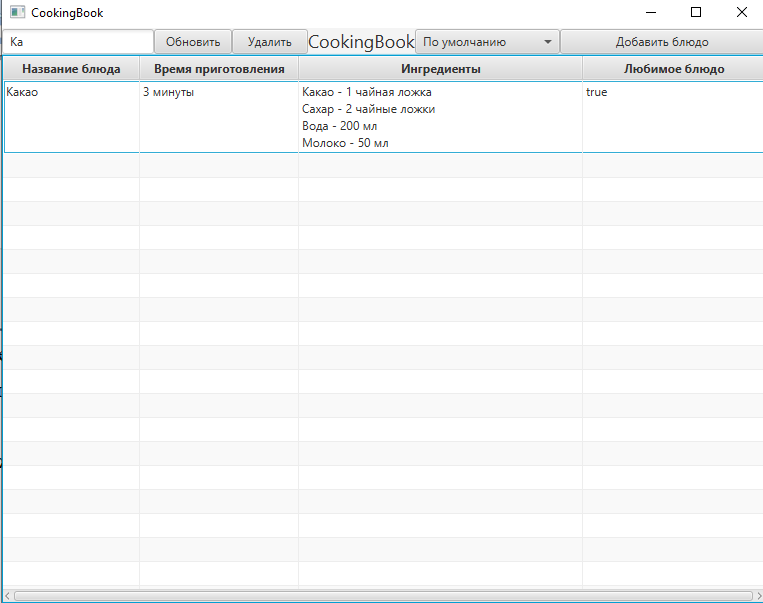


Рисунок 5 – Поиск блюда

1. Нажав на кнопку “Удалить”, можно удалить блюдо.

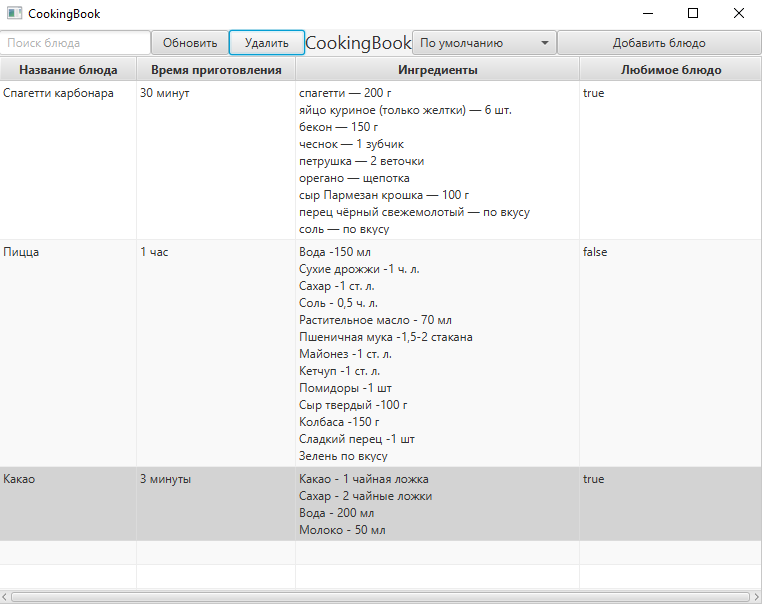


Рисунок 6 – Удаление блюда

**Приложение 5. Описание кода**

package com.example.cookingbook.controller;

import com.example.cookingbook.database.DatabaseManager;

import com.example.cookingbook.database.DishDAO;

import com.example.cookingbook.database.DishDAOImpl;

import com.example.cookingbook.model.Dish;

import javafx.fxml.FXML;

import javafx.scene.control.CheckBox;

import javafx.scene.control.TextArea;

import javafx.scene.control.TextField;

/\*\*

\* Контроллер для добавления нового блюда.

\* Управляет полями ввода и кнопкой сохранения блюда.

\*

\* @autor Комов Степан

\* @version 1.0

\* @since 2024-05-20

\*/

public class AddDishController {

private DishDAO dishDAO;

@FXML

private TextField nameField;

@FXML

private TextField cookingTimeField;

@FXML

private TextArea ingredientsArea;

@FXML

private TextArea cookingProcessArea;

@FXML

private CheckBox favoriteCheckBox;

/\*\*

\* Метод инициализации, вызываемый после загрузки FXML.

\* Инициализирует DAO для работы с базой данных.

\*/

@FXML

private void initialize() {

this.dishDAO = new DishDAOImpl(new DatabaseManager());

}

/\*\*

\* Сохраняет введенные данные о новом блюде.

\*/

@FXML

private void saveDish() {

String name = nameField.getText();

String cookingTime = cookingTimeField.getText();

String ingredients = ingredientsArea.getText();

String cookingProcess = cookingProcessArea.getText();

boolean favorite = favoriteCheckBox.isSelected();

Dish newDish = new Dish(name, cookingTime, ingredients, cookingProcess, favorite);

dishDAO.addDish(newDish);

closeWindow();

}

/\*\*

\* Закрывает окно добавления блюда.

\*/

@FXML

private void closeWindow() {

nameField.getScene().getWindow().hide();

}

}

package com.example.cookingbook.controller;

import com.example.cookingbook.database.DatabaseManager;

import com.example.cookingbook.database.DishDAO;

import com.example.cookingbook.database.DishDAOImpl;

import com.example.cookingbook.model.Dish;

import com.example.cookingbook.neuralNetwork.TextDivider;

import javafx.fxml.FXML;

import javafx.scene.control.CheckBox;

import javafx.scene.control.TextArea;

import javafx.scene.layout.HBox;

import javafx.scene.layout.VBox;

import java.util.List;

/\*\*

\* Контроллер для работы с окном деталей блюда.

\* Управляет полями ввода и кнопками сохранения изменений.

\*

\* @autor Комов Степан

\* @version 1.0

\* @since 2024-05-20

\*/

public class DishDetailsController {

@FXML

private TextArea nameTextArea;

@FXML

private TextArea cookingTimeTextArea;

@FXML

private TextArea ingredientsTextArea;

@FXML

private TextArea cookingProcessTextArea;

@FXML

private CheckBox favoriteCheckBox;

private Dish selectedDish;

private DishDAO dishDAO;

/\*\*

\* Метод инициализации, вызываемый после загрузки FXML.

\* Инициализирует DAO для работы с базой данных.

\*/

@FXML

private void initialize() {

this.dishDAO = new DishDAOImpl(new DatabaseManager());

}

/\*\*

\* Устанавливает выбранное блюдо и заполняет поля его деталями.

\*

\* @param dish Выбранное блюдо.

\*/

public void setSelectedDish(Dish dish) {

this.selectedDish = dish;

populateFields();

}

/\*\*

\* Заполняет поля деталями выбранного блюда.

\*/

private void populateFields() {

if (selectedDish != null) {

nameTextArea.setText(selectedDish.getName());

cookingTimeTextArea.setText(selectedDish.getCookingTime());

ingredientsTextArea.setText(selectedDish.getIngredients());

divideCookingProcess(selectedDish.getCookingProcess());

favoriteCheckBox.setSelected(selectedDish.isFavorite());

}

}

/\*\*

\* Разделяет процесс приготовления на отдельные шаги и отображает в текстовом поле.

\*

\* @param cookingProcess Полный текст процесса приготовления.

\*/

private void divideCookingProcess(String cookingProcess) {

List<String> steps = TextDivider.splitTextIntoSteps(cookingProcess);

StringBuilder dividedText = new StringBuilder();

for (int i = 0; i < steps.size(); i++) {

String[] sentences = steps.get(i).split("\\.\\s\*");

dividedText.append("Шаг ").append(i + 1).append(":\n");

for (String sentence : sentences) {

dividedText.append(sentence).append(".\n");

}

dividedText.append("\n");

}

cookingProcessTextArea.setText(dividedText.toString());

}

/\*\*

\* Сохраняет внесенные изменения в блюде и обновляет его в базе данных.

\*/

@FXML

private void saveChanges() {

if (selectedDish != null) {

selectedDish.setName(nameTextArea.getText());

selectedDish.setCookingTime(cookingTimeTextArea.getText());

selectedDish.setIngredients(ingredientsTextArea.getText());

selectedDish.setCookingProcess(saveChangesWithoutSteps());

selectedDish.setFavorite(favoriteCheckBox.isSelected());

dishDAO.updateDish(selectedDish);

closeWindow();

}

}

/\*\*

\* Закрывает окно деталей блюда.

\*/

@FXML

private void closeWindow() {

nameTextArea.getScene().getWindow().hide();

}

/\*\*

\* Удаляет префиксы "Шаг N:" из текста процесса приготовления перед сохранением.

\*

\* @return Текст процесса приготовления без префиксов "Шаг N:".

\*/

private String saveChangesWithoutSteps() {

String fullText = cookingProcessTextArea.getText();

return fullText.replaceAll("Шаг \\d+:\\n", "");

}

}

import com.example.cookingbook.database.DatabaseManager;

import com.example.cookingbook.database.DishDAO;

import com.example.cookingbook.database.DishDAOImpl;

import com.example.cookingbook.model.Dish;

import javafx.fxml.FXML;

import javafx.fxml.FXMLLoader;

import javafx.scene.Parent;

import javafx.scene.Scene;

import javafx.scene.control.\*;

import javafx.scene.control.cell.PropertyValueFactory;

import javafx.scene.input.MouseButton;

import javafx.scene.input.MouseEvent;

import javafx.stage.Stage;

import java.io.IOException;

import java.util.List;

/\*\*

\* Контроллер для работы с главной сценой приложения.

\* Управляет таблицей блюд, полем поиска и выпадающим списком сортировки.

\*

\* @autor Комов Степан

\* @version 1.0

\* @since 2024-05-20

\*/

public class HelloController {

@FXML

private TableView<Dish> dishesTable;

@FXML

private TableColumn<Dish, String> nameColumn, timeColumn, ingredientsColumn;

@FXML

private TableColumn<Dish, Boolean> favoriteColumn;

@FXML

private TextField searchField;

@FXML

private ChoiceBox<String> sortChoiceBox;

private final DishDAO dishDAO;

/\*\*

\* Конструктор класса HelloController.

\* Инициализирует DAO для работы с базой данных.

\*/

public HelloController() {

this.dishDAO = new DishDAOImpl(new DatabaseManager());

}

/\*\*

\* Метод инициализации, вызываемый после загрузки FXML.

\* Настраивает колонки таблицы, заполняет данные и добавляет обработчики событий.

\*/

@FXML

public void initialize() {

nameColumn.setCellValueFactory(new PropertyValueFactory<>("name"));

timeColumn.setCellValueFactory(new PropertyValueFactory<>("cookingTime"));

ingredientsColumn.setCellValueFactory(new PropertyValueFactory<>("ingredients"));

favoriteColumn.setCellValueFactory(new PropertyValueFactory<>("favorite"));

List<Dish> allDishes = dishDAO.getAllDishes();

dishesTable.getItems().addAll(allDishes);

searchField.textProperty().addListener((observable, oldValue, newValue) -> {

searchDishes(newValue.trim());

});

sortChoiceBox.getItems().addAll("По умолчанию", "Понравившиеся", "Последние добавленные", "По алфавиту");

sortChoiceBox.setValue("По умолчанию");

sortChoiceBox.setOnAction(event -> {

String selectedSortType = sortChoiceBox.getValue();

List<Dish> sortedDishes = null;

switch (selectedSortType) {

case "По умолчанию":

sortedDishes = dishDAO.getAllDishes();

break;

case "Понравившиеся":

sortedDishes = dishDAO.getFavoriteDishes();

break;

case "Последние добавленные":

sortedDishes = dishDAO.getRecentlyAddedDishes();

break;

case "По алфавиту":

sortedDishes = dishDAO.getAlphabeticallySortedDishes();

break;

}

if (sortedDishes != null) {

dishesTable.getItems().clear();

dishesTable.getItems().addAll(sortedDishes);

}

});

}

/\*\*

\* Обрабатывает двойной клик по строке таблицы.

\* Открывает окно с деталями выбранного блюда.

\*

\* @param event Событие клика мыши.

\*/

@FXML

private void handleTableClick(MouseEvent event) {

if (event.getButton().equals(MouseButton.PRIMARY) && event.getClickCount() == 2) {

Dish selectedDish = dishesTable.getSelectionModel().getSelectedItem();

openDishDetails(selectedDish);

}

}

/\*\*

\* Обновляет таблицу блюд.

\* Загружает все блюда из базы данных и отображает их в таблице.

\*/

@FXML

private void refreshTable() {

dishesTable.getItems().clear();

List<Dish> allDishes = dishDAO.getAllDishes();

dishesTable.getItems().addAll(allDishes);

}

/\*\*

\* Удаляет выбранное блюдо из таблицы и базы данных.

\*/

@FXML

private void deleteSelectedDish() {

Dish selectedDish = dishesTable.getSelectionModel().getSelectedItem();

if (selectedDish != null) {

dishDAO.deleteDish(selectedDish.getId());

dishesTable.getItems().remove(selectedDish);

}

}

/\*\*

\* Открывает окно с деталями выбранного блюда.

\*

\* @param dish Блюдо, детали которого нужно отобразить.

\*/

private void openDishDetails(Dish dish) {

try {

FXMLLoader loader = new FXMLLoader(getClass().getResource("/com/example/cookingbook/DishDetails.fxml"));

Parent root = loader.load();

DishDetailsController controller = loader.getController();

controller.setSelectedDish(dish);

Stage stage = new Stage();

stage.setScene(new Scene(root));

stage.show();

} catch (IOException e) {

e.printStackTrace();

}

}

/\*\*

\* Открывает окно для добавления нового блюда.

\*/

@FXML

private void openAddDishWindow() {

try {

FXMLLoader loader = new FXMLLoader(getClass().getResource("/com/example/cookingbook/addDish.fxml"));

Parent root = loader.load();

AddDishController controller = loader.getController();

Stage stage = new Stage();

stage.setScene(new Scene(root));

stage.show();

} catch (IOException e) {

e.printStackTrace();

}

}

/\*\*

\* Выполняет поиск блюд по введенному тексту.

\*

\* @param searchText Текст для поиска.

\*/

private void searchDishes(String searchText) {

dishesTable.getItems().clear();

if (searchText.isEmpty()) {

List<Dish> allDishes = dishDAO.getAllDishes();

dishesTable.getItems().addAll(allDishes);

return;

}

List<Dish> foundDishes = dishDAO.searchDishes(searchText);

dishesTable.getItems().addAll(foundDishes);

}

}

import java.sql.Connection;

import java.sql.DriverManager;

import java.sql.SQLException;

/\*\*

\* Менеджер базы данных для управления подключением к SQLite базе данных.

\* Предоставляет методы для установления и закрытия соединения с базой данных.

\*

\* @autor Комов Степан

\* @version 1.0

\* @since 2024-05-20

\*/

public class DatabaseManager {

private static final String DATABASE\_URL = "jdbc:sqlite:C:/Program Files/SQLiteStudio/storage/cookingbook";

private Connection connection;

/\*\*

\* Конструктор класса. Устанавливает соединение с базой данных.

\*/

public DatabaseManager() {

try {

connection = DriverManager.getConnection(DATABASE\_URL);

System.out.println("Connected to the database");

} catch (SQLException e) {

System.err.println("Error connecting to the database: " + e.getMessage());

}

}

/\*\*

\* Получает текущее соединение с базой данных.

\*

\* @return Соединение с базой данных.

\*/

public Connection getConnection() {

return connection;

}

/\*\*

\* Закрывает соединение с базой данных.

\*/

public void closeConnection() {

try {

if (connection != null) {

connection.close();

System.out.println("Connection closed");

}

} catch (SQLException e) {

System.err.println("Error closing connection: " + e.getMessage());

}

}

}

import com.example.cookingbook.model.Dish;

import java.util.List;

/\*\*

\* Интерфейс, определяющий методы для работы с базой данных блюд.

\*

\* @autor Комов Степан

\* @version 1.0

\* @since 2024-05-20

\*/

public interface DishDAO {

/\*\*

\* Получает список всех блюд из базы данных.

\*

\* @return Список всех блюд.

\*/

List<Dish> getAllDishes();

/\*\*

\* Добавляет новое блюдо в базу данных.

\*

\* @param dish Новое блюдо.

\*/

void addDish(Dish dish);

/\*\*

\* Обновляет информацию о блюде в базе данных.

\*

\* @param dish Блюдо для обновления.

\*/

void updateDish(Dish dish);

/\*\*

\* Удаляет блюдо из базы данных по его идентификатору.

\*

\* @param id Идентификатор удаляемого блюда.

\*/

void deleteDish(int id);

/\*\*

\* Выполняет поиск блюд по заданному текстовому запросу.

\*

\* @param searchText Текст запроса для поиска.

\* @return Список найденных блюд.

\*/

List<Dish> searchDishes(String searchText);

/\*\*

\* Получает список избранных блюд из базы данных.

\*

\* @return Список избранных блюд.

\*/

List<Dish> getFavoriteDishes();

/\*\*

\* Получает список недавно добавленных блюд из базы данных.

\*

\* @return Список недавно добавленных блюд.

\*/

List<Dish> getRecentlyAddedDishes();

/\*\*

\* Получает список блюд, отсортированных по алфавиту.

\*

\* @return Список блюд, отсортированных по алфавиту.

\*/

List<Dish> getAlphabeticallySortedDishes();

}

import com.example.cookingbook.model.Dish;

import java.util.ArrayList;

import java.util.List;

import java.sql.PreparedStatement;

import java.sql.ResultSet;

import java.sql.SQLException;

import java.sql.Statement;

/\*\*

\* Реализация интерфейса DishDAO для работы с базой данных блюд.

\* @autor Комов Степан

\* @version 1.0

\* @since 2024-05-20

\*/

public class DishDAOImpl implements DishDAO{

private final DatabaseManager databaseManager;

/\*\*

\* Конструктор для инициализации объекта DishDAOImpl.

\*

\* @param databaseManager Менеджер базы данных.

\*/

public DishDAOImpl(DatabaseManager databaseManager) {

this.databaseManager = databaseManager;

}

/\*\*

\* Получает список всех блюд из базы данных.

\*

\* @return Список всех блюд.

\*/

@Override

public List<Dish> getAllDishes() {

List<Dish> dishes = new ArrayList<>();

try {

String query = "SELECT id, name, cooking\_time, ingredients, cooking\_process, favorite FROM dishes";

PreparedStatement preparedStatement = databaseManager.getConnection().prepareStatement(query);

ResultSet resultSet = preparedStatement.executeQuery();

while (resultSet.next()) {

int id = resultSet.getInt("id");

String name = resultSet.getString("name");

String cookingTime = resultSet.getString("cooking\_time");

String ingredients = resultSet.getString("ingredients");

String cookingProcess = resultSet.getString("cooking\_process");

boolean favorite = resultSet.getBoolean("favorite");

Dish dish = new Dish(name, cookingTime, ingredients, cookingProcess, favorite);

dish.setId(id);

dishes.add(dish);

}

} catch (SQLException e) {

e.printStackTrace();

}

return dishes;

}

/\*\*

\* Добавляет новое блюдо в базу данных.

\*

\* @param dish Новое блюдо для добавления.

\*/

@Override

public void addDish(Dish dish) {

String query = "INSERT INTO dishes (name, cooking\_time, ingredients, cooking\_process, favorite) VALUES (?, ?, ?, ?, ?)";

try (PreparedStatement preparedStatement = databaseManager.getConnection().prepareStatement(query, Statement.RETURN\_GENERATED\_KEYS)) {

preparedStatement.setString(1, dish.getName());

preparedStatement.setString(2, dish.getCookingTime());

preparedStatement.setString(3, dish.getIngredients());

preparedStatement.setString(4, dish.getCookingProcess());

preparedStatement.setBoolean(5, dish.isFavorite());

preparedStatement.executeUpdate();

ResultSet rs = preparedStatement.getGeneratedKeys();

if (rs.next()) {

int generatedId = rs.getInt(1);

dish.setId(generatedId);

}

} catch (SQLException e) {

e.printStackTrace();

}

}

/\*\*

\* Обновляет информацию о блюде в базе данных.

\*

\* @param dish Объект блюда с обновленными данными.

\*/

@Override

public void updateDish(Dish dish) {

String query = "UPDATE dishes SET name = ?, cooking\_time = ?, ingredients = ?, cooking\_process = ?, favorite = ? WHERE id = ?";

try (PreparedStatement preparedStatement = databaseManager.getConnection().prepareStatement(query)) {

preparedStatement.setString(1, dish.getName());

preparedStatement.setString(2, dish.getCookingTime());

preparedStatement.setString(3, dish.getIngredients());

preparedStatement.setString(4, dish.getCookingProcess());

preparedStatement.setBoolean(5, dish.isFavorite());

preparedStatement.setInt(6, dish.getId());

preparedStatement.executeUpdate();

} catch (SQLException e) {

e.printStackTrace();

}

}

/\*\*

\* Удаляет блюдо из базы данных по его ID.

\*

\* @param id ID блюда для удаления.

\*/

@Override

public void deleteDish(int id) {

String query = "DELETE FROM dishes WHERE id = ?";

try (PreparedStatement preparedStatement = databaseManager.getConnection().prepareStatement(query)) {

preparedStatement.setInt(1, id);

preparedStatement.executeUpdate();

} catch (SQLException e) {

e.printStackTrace();

}

}

/\*\*

\* Ищет блюда в базе данных по заданному тексту.

\*

\* @param searchText Текст для поиска в названиях блюд.

\* @return Список найденных блюд.

\*/

@Override

public List<Dish> searchDishes(String searchText) {

List<Dish> foundDishes = new ArrayList<>();

try {

String query = "SELECT id, name, cooking\_time, ingredients, cooking\_process, favorite " +

"FROM dishes " +

"WHERE name LIKE ?";

PreparedStatement preparedStatement = databaseManager.getConnection().prepareStatement(query);

preparedStatement.setString(1, "%" + searchText + "%"); // Используем параметр для поиска совпадений внутри строки

ResultSet resultSet = preparedStatement.executeQuery();

while (resultSet.next()) {

int id = resultSet.getInt("id");

String name = resultSet.getString("name");

String cookingTime = resultSet.getString("cooking\_time");

String ingredients = resultSet.getString("ingredients");

String cookingProcess = resultSet.getString("cooking\_process");

boolean favorite = resultSet.getBoolean("favorite");

Dish dish = new Dish(name, cookingTime, ingredients, cookingProcess, favorite);

dish.setId(id);

foundDishes.add(dish);

}

} catch (SQLException e) {

e.printStackTrace();

}

return foundDishes;

}

/\*\*

\* Получает список избранных блюд из базы данных.

\*

\* @return Список избранных блюд.

\*/

@Override

public List<Dish> getFavoriteDishes() {

List<Dish> favoriteDishes = new ArrayList<>();

try {

String query = "SELECT id, name, cooking\_time, ingredients, cooking\_process, favorite FROM dishes WHERE favorite = ?";

PreparedStatement preparedStatement = databaseManager.getConnection().prepareStatement(query);

preparedStatement.setBoolean(1, true);

ResultSet resultSet = preparedStatement.executeQuery();

while (resultSet.next()) {

int id = resultSet.getInt("id");

String name = resultSet.getString("name");

String cookingTime = resultSet.getString("cooking\_time");

String ingredients = resultSet.getString("ingredients");

String cookingProcess = resultSet.getString("cooking\_process");

boolean favorite = resultSet.getBoolean("favorite");

Dish dish = new Dish(name, cookingTime, ingredients, cookingProcess, favorite);

dish.setId(id);

favoriteDishes.add(dish);

}

} catch (SQLException e) {

e.printStackTrace();

}

return favoriteDishes;

}

/\*\*

\* Получает список недавно добавленных блюд из базы данных.

\*

\* @return Список недавно добавленных блюд.

\*/

@Override

public List<Dish> getRecentlyAddedDishes() {

List<Dish> recentlyAddedDishes = new ArrayList<>();

try {

String query = "SELECT id, name, cooking\_time, ingredients, cooking\_process, favorite FROM dishes ORDER BY id DESC LIMIT 10";

PreparedStatement preparedStatement = databaseManager.getConnection().prepareStatement(query);

ResultSet resultSet = preparedStatement.executeQuery();

while (resultSet.next()) {

int id = resultSet.getInt("id");

String name = resultSet.getString("name");

String cookingTime = resultSet.getString("cooking\_time");

String ingredients = resultSet.getString("ingredients");

String cookingProcess = resultSet.getString("cooking\_process");

boolean favorite = resultSet.getBoolean("favorite");

Dish dish = new Dish(name, cookingTime, ingredients, cookingProcess, favorite);

dish.setId(id);

recentlyAddedDishes.add(dish);

}

} catch (SQLException e) {

e.printStackTrace();

}

return recentlyAddedDishes;

}

/\*\*

\* Получает список блюд из базы данных, отсортированных по алфавиту.

\*

\* @return Список блюд, отсортированных по алфавиту.

\*/

@Override

public List<Dish> getAlphabeticallySortedDishes() {

List<Dish> alphabeticallySortedDishes = new ArrayList<>();

try {

String query = "SELECT id, name, cooking\_time, ingredients, cooking\_process, favorite FROM dishes ORDER BY name";

PreparedStatement preparedStatement = databaseManager.getConnection().prepareStatement(query);

ResultSet resultSet = preparedStatement.executeQuery();

while (resultSet.next()) {

int id = resultSet.getInt("id");

String name = resultSet.getString("name");

String cookingTime = resultSet.getString("cooking\_time");

String ingredients = resultSet.getString("ingredients");

String cookingProcess = resultSet.getString("cooking\_process");

boolean favorite = resultSet.getBoolean("favorite");

Dish dish = new Dish(name, cookingTime, ingredients, cookingProcess, favorite);

dish.setId(id);

alphabeticallySortedDishes.add(dish);

}

} catch (SQLException e) {

e.printStackTrace();

}

return alphabeticallySortedDishes;

}

}

package com.example.cookingbook.model;

/\*\*

\* Класс, представляющий объект блюда.

\* @autor Комов Степан

\* @version 1.0

\* @since 2024-05-20

\*/

public class Dish {

private int id; // Идентификатор блюда

private String name; // Название блюда

private String cookingTime; // Время приготовления

private String ingredients; // Ингредиенты

private String cookingProcess; // Процесс приготовления

private boolean favorite; // Флаг, указывающий на избранность блюда

/\*\*

\* Конструктор класса Dish.

\*

\* @param name Название блюда.

\* @param cookingTime Время приготовления блюда.

\* @param ingredients Ингредиенты для приготовления блюда.

\* @param cookingProcess Процесс приготовления блюда.

\* @param favorite Флаг, указывающий, является ли блюдо избранным.

\*/

public Dish(String name, String cookingTime, String ingredients, String cookingProcess, boolean favorite) {

this.name = name;

this.cookingTime = cookingTime;

this.ingredients = ingredients;

this.cookingProcess = cookingProcess;

this.favorite = favorite;

}

public int getId() {

return id;

}

public void setId(int id) {

this.id = id;

}

public String getName() {

return name;

}

public void setName(String name) {

this.name = name;

}

public String getCookingTime() {

return cookingTime;

}

public void setCookingTime(String cookingTime) {

this.cookingTime = cookingTime;

}

public String getIngredients() {

return ingredients;

}

public void setIngredients(String ingredients) {

this.ingredients = ingredients;

}

public String getCookingProcess() {

return cookingProcess;

}

public void setCookingProcess(String cookingProcess) {

this.cookingProcess = cookingProcess;

}

public boolean isFavorite() {

return favorite;

}

public void setFavorite(boolean favorite) {

this.favorite = favorite;

}

}  
package com.example.cookingbook.neuralNetwork;

import java.util.ArrayList;

import java.util.List;

import java.util.regex.Matcher;

import java.util.regex.Pattern;

/\*\*

\* Класс для разделения текста на предложения и шаги.

\*

\* @autor Комов Степан

\* @version 1.0

\* @since 2024-05-20

\*/

public class TextDivider {

/\*\*

\* Разделяет текст на шаги с указанным количеством предложений в каждом шаге.

\*

\* @param text Текст для разделения.

\* @return Список шагов.

\*/

public static List<String> splitTextIntoSteps(String text) {

List<String> steps = new ArrayList<>();

List<String> sentences = splitTextIntoSentences(text);

for (int i = 0; i < sentences.size(); i += 5) {

int end = Math.min(i + 5, sentences.size());

List<String> stepSentences = sentences.subList(i, end);

steps.add(String.join(" ", stepSentences));

}

return steps;

}

/\*\*

\* Разделяет текст на предложения.

\*

\* @param text Текст для разделения.

\* @return Список предложений.

\*/

public static List<String> splitTextIntoSentences(String text) {

List<String> sentences = new ArrayList<>();

// Регулярное выражение для разделения текста на предложения

Pattern pattern = Pattern.compile("[^.!?\\s][^.!?]\*(?:\\.(?!\\s|$)|[.!?](?=\\s|$))");

Matcher matcher = pattern.matcher(text);

// Добавляем найденные предложения в список

while (matcher.find()) {

sentences.add(matcher.group());

}

return sentences;

}

}

package com.example.cookingbook;

/\*\*

\*

\*/

import javafx.application.Application;

import javafx.fxml.FXMLLoader;

import javafx.scene.Scene;

import javafx.stage.Stage;

import java.io.IOException;

public class HelloApplication extends Application {

@Override

public void start(Stage stage) throws IOException {

FXMLLoader fxmlLoader = new FXMLLoader(HelloApplication.class.getResource("hello-view.fxml"));

Scene scene = new Scene(fxmlLoader.load(), 704, 575);

stage.setTitle("CookingBook");

stage.setScene(scene);

stage.show();

}

public static void main(String[] args) {

launch();

}

}