МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ
УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«МОСКОВСКИЙ ПОЛИТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»
(МОСКОВСКИЙ ПОЛИТЕХ)

Факультет информационных технологий
Кафедра «Инфокогнитивные технологии»

КУРСОВОЙ ПРОЕКТ

на тему: *«Разработка клиент-серверного desktop-приложения для поддержки бизнес-процессов ресторана»*

Направление подготовки 09.03.03 «Прикладная информатика»
Профиль «Корпоративные информационные системы»

Выполнила:

студентка группы 221-362

Буракова Полина Юрьевна

01.07.2023

Москва 2023

Введение

Главным запросом владельца небольшого ресторана является необходимость в создании новой информационной системы, которая бы обеспечила эффективное хранение и учет данных.

Одним из ключевых аспектов является учет ингредиентов. Для эффективного управления рестораном необходимо иметь полную информацию о наличии ингредиентов на складе. Разработка декстоп-приложения позволит владельцу ресторана легко отслеживать количество ингредиентов, их единицы измерения и названия, исключая возможность дублирования. Это поможет избежать ошибок при составлении меню и планировании блюд.

Следующим важным аспектом в предметной области является составление блюд. Ресторан предлагает различные блюда, каждое из которых состоит из нескольких ингредиентов в определенном количестве. Информационная система позволит оптимально управлять ингредиентами, учитывая их использование в разных блюдах. Владелец ресторана сможет легко добавлять или удалять ингредиенты, а также контролировать наличие необходимых ингредиентов на складе.

В данной предметной области становится очевидной необходимость создания информационной системы, которая бы упростила и улучшила процессы управления данными в ресторане. Такая система позволит владельцу ресторана эффективно контролировать запасы ингредиентов, управлять блюдами.

Постановка задачи

Целью данного курсового проекта является разработка простого оконного клиент-серверного приложения, позволяющего пользователю посредством графического интерфейса и согласно предоставляемым ему правам мандатного доступа обрабатывать данные, хранящиеся на сервере ресторана.

Основными этапами работы являются:

-Анализ основных функциональных требований к приложению.

-Составление инфологической (описание сущностей и связей между ними) и реляционной модели (реализация модели, используя таблицы, столбцы и отношения между ними; преобразование сущностей в таблицы).

-Проектирование базы данных: разработка структуры базы данных, которая будет хранить данные на сервере. Определение таблиц и связей между ними.

-Создание удаленного репозитория посредством системы контроля версий Git.

-Осуществление подключения к базе данных, доступ к которому предоставляется через fit.mospolytech.ru.

-Реализация пользовательского интерфейса: разработка дизайна и интерфейса приложения.

-Разработка функциональности: создание основной функциональности приложения.

-Тестирование приложения для выявления и исправления возможных ошибок.

Проектирование и разработка

После осведомления о необходимых требованиях исследования происходит переход ко второму этапу разработки, а именно, к составлению информационной и реляционной моделей с использованием ER-диаграммы (рис 1, 2).

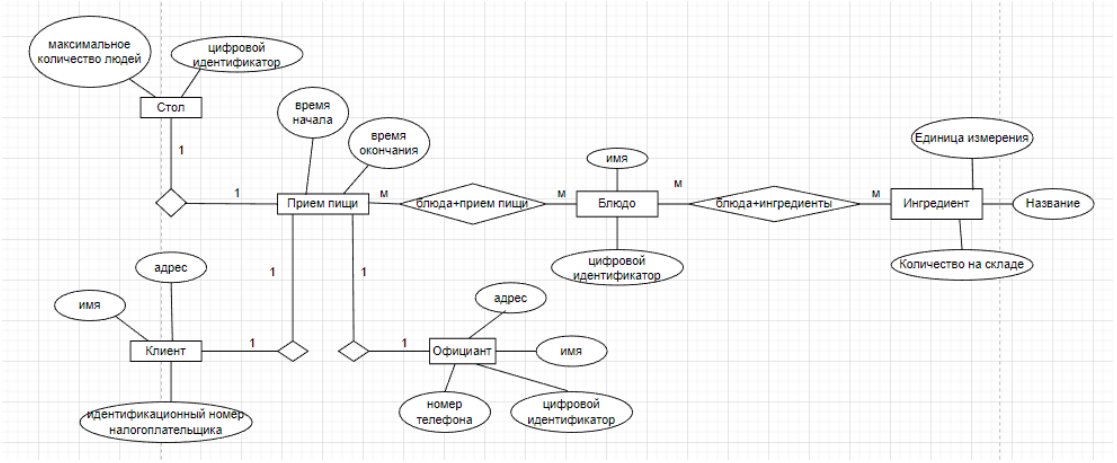


Рисунок 1 — Инфологическая модель

В инфологической модели представлены следующие сущности: ингредиент (атрибуты: название, единица измерения и количество на складе), блюдо (атрибуты: имя и цифровой идентификатор), стол (атрибуты: цифровой идентификатор и максимальное количество людей), прием пищи (атрибуты: время начала и окончания), официант (атрибуты: цифровой идентификатор, имя, адрес и номер телефона), клиент (атрибуты: идентификационный номер налогоплательщика, имя и адрес).

В инфологической модели реализованы следующие связи:

- Блюдо-Ингредиент: связь M:M - одно блюдо может состоять из нескольких ингредиентов, а каждый ингредиент может быть использован в разных блюдах.
- Прием пищи-Блюдо: связь M:M - при каждом приеме пищи за определенным столом употребляется несколько блюд, и одно и то же блюдо может быть употреблено более одного раза за один и тот же прием пищи.
- Прием пищи-Стол: связь 1:1 - прием пищи происходит за определенным столиком.
- Прием пищи-Официант: связь 1:1 - каждый прием пищи имеет ответственного официанта.
- Прием пищи-Клиент: связь 1:1 - каждый прием пищи может быть потреблен одним клиентом.

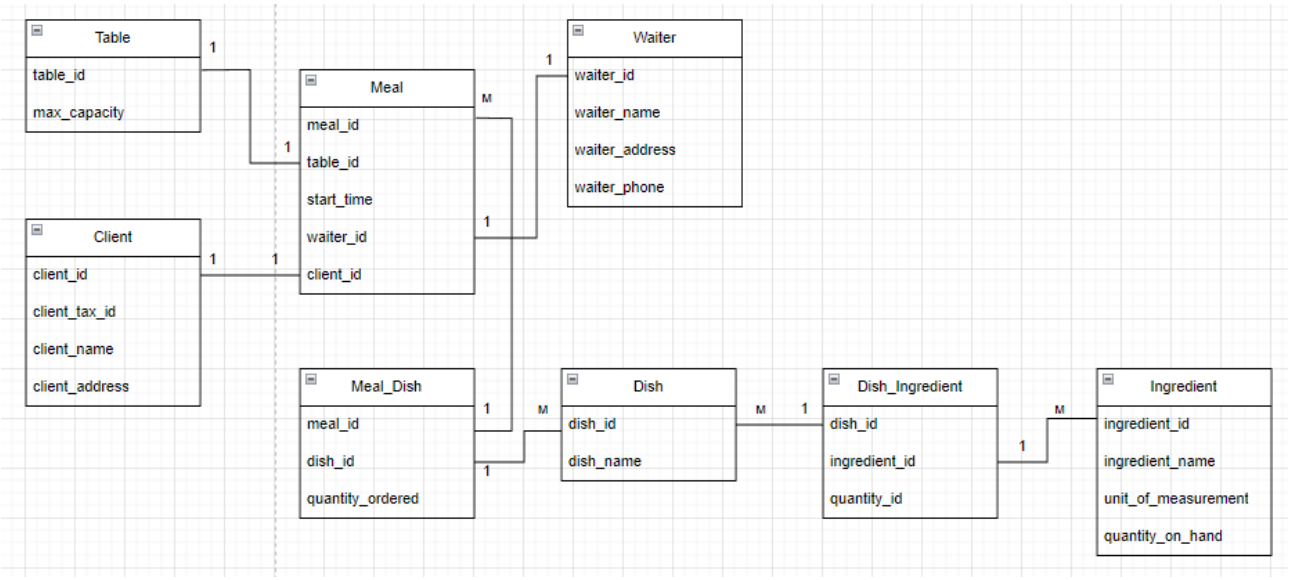


Рисунок 2 — Реляционная модель

Связи в реляционной модели базы данных ресторана:

-Таблица "Dish_Ingredient" связана с таблицами "Dish" и "Ingredient" через внешние ключи dish_id и ingredient_id. Эта связь показывает, какие ингредиенты требуются для приготовления определенного блюда, и указывает количество необходимых ингредиентов.

-Таблица "Meal" связана с таблицей "Table" через внешний ключ table_id. Эта связь показывает, какие столы зарезервированы для определенного приема пищи.

-Таблица "Meal" также связана с таблицей "Waiters" через внешний ключ waiter_id. Эта связь показывает, какой официант обслуживает определенный прием пищи.

-Таблица "Meal_Dish" связана с таблицами "Meal" и "Dish" через внешние ключи meal_id и dish_id. Эта связь показывает, какие блюда были заказаны для определенного приема пищи и указывает количество заказанных блюд.

-Таблица "Client" связана с таблицей "Meal" через внешний ключ client_id, если требуется отслеживать, какие клиенты посещают ресторан и какие приемы пищи они заказывают.

Разработка базы данных является важным этапом в создании программного приложения. Она представляет собой процесс проектирования и создания структуры, схемы и отношений между данными, которые будут храниться и обрабатываться в приложении (рис 3).

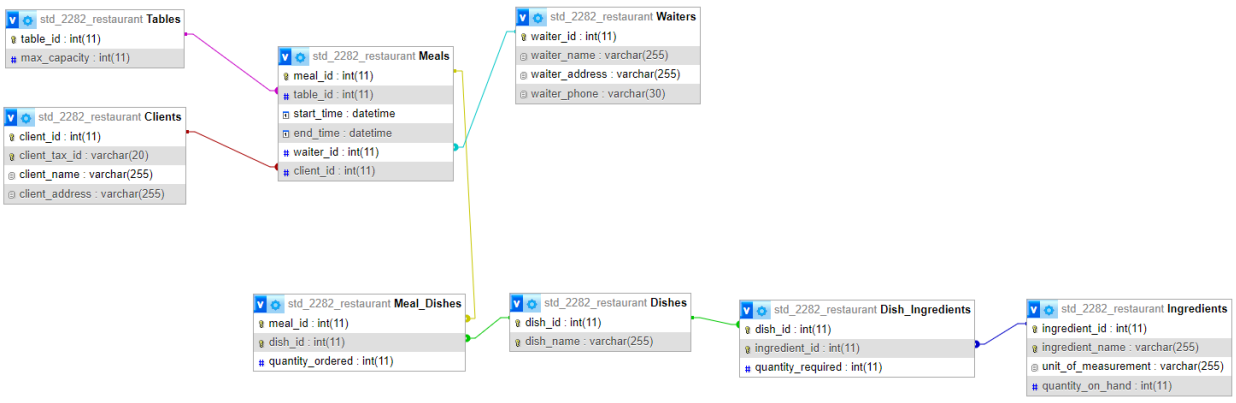


Рисунок 3 — Физическая модель, опирающаяся на СУБД MySQL

Далее производилось установление соединения с базой данных путем вызова метода .getInstance в классе sql.java в соответствии с шаблоном проектирования Singleton (рис. 4).

```
public class sql {
    3 usages
    private static sql instance;
    3 usages
    private static Connection connection;
    Polina
    protected sql() throws SQLException, ClassNotFoundException {
        Class.forName( className: "com.mysql.cj.jdbc.Driver");
        connection = DriverManager.getConnection( url: "jdbc:mysql://std-mysql.ist.mospolytech.ru:3306/std_2282_restaurant",  user: "std_2282_restaurant",  password: "polina2810");
    }
    11 usages   Polina
    public static sql getInstance() throws SQLException, ClassNotFoundException {
        if (instance == null) { //если объект еще не создан
            instance = new sql(); //создать новый объект
        }
        return instance; //вернуть ранее созданный объект
    }
    Polina
    public static Connection getConnection() { return connection; }
```

Рисунок 4 — Осуществление подключения к базе данных

Для реализации пользовательского интерфейса приложения с использованием Scene Builder были выполнены следующие шаги.

-Разработка дизайна: В начале процесса был разработан общий дизайн приложения, определены цветовая гамма, шрифты и расположение элементов интерфейса. Это позволило создать единый стиль и обеспечить легкость использования для конечных пользователей.

-Создание макетов: Затем были созданы макеты страниц, которые будут отображаться в приложении. Макеты включали в себя различные элементы интерфейса, такие как кнопки, текстовые поля, таблицы и другие элементы, необходимые для отображения информации и взаимодействия с пользователем.

-Использование Scene Builder: Для реализации дизайна и интерфейса приложения был использован Scene Builder. Scene Builder- это инструмент, который позволяет создавать пользовательские темы и стили для приложений. Он предоставляет возможность настроить внешний вид элементов интерфейса, таких как цвета, шрифты, размеры и другие атрибуты.

-Настройка стилей: С помощью Scene Builder были настроены стили элементов интерфейса в соответствии с разработанным дизайном. Были определены цвета для фона, текста, кнопок и других элементов. Также были настроены размеры и расположение элементов для достижения оптимального пользовательского опыта.

-Разработка функциональности: После настройки стилей была разработана функциональность приложения. Были созданы обработчики событий для кнопок и других элементов интерфейса, чтобы пользователи могли взаимодействовать с приложением. Например, при нажатии на кнопку "Блюда" или "Ингредиенты" вызывалась соответствующая функция для отображения списка блюд или ингредиентов.

В проекте присутствуют следующие fxml-файлы, для каждого из которых был создан класс-контроллер:

-pol.fxml и Pol.java: Файл pol.fxml содержит макет для экрана входа в систему, где пользователь должен ввести свои учетные данные. Класс Pol.java является контроллером для этого fxml-файла и содержит логику, связанную с аутентификацией пользователя и обработкой событий, связанных с экраном входа в систему.

-pol2.fxml и Pol2.java: Файл pol.fxml содержит макет для экрана регистрации в систему, где пользователь должен ввести свои учетные данные. Класс Pol2.java является контроллером для этого fxml-файла и содержит логику, связанную с регистрацией пользователя и обработкой событий, связанных с экраном регистрации в систему.

-pol6.fxml и Pol6.java: Файл Pol6.fxml содержит макет для экрана профиля пользователя, где пользователь может изменить свои персональные данные. Класс Pol6.java является контроллером для этого fxml-файла и содержит логику, связанную с обновлением профиля пользователя.

-pol1.fxml и Pol1.java: Файл pol.fxml содержит макет для панели инструментов, которая отображает основную информацию и функциональность приложения для администраторов. Класс Pol1.java является контроллером для этого fxml-файла и содержит логику, связанную с обновлением и отображением данных на панели инструментов.

-pol7.fxml и Pol7.java: Файл pol.fxml содержит макет для панели инструментов, которая отображает основную информацию и функциональность приложения для пользователей. Класс Pol1.java является контроллером для этого fxml-файла и содержит логику, связанную с обновлением и отображением данных на панели инструментов.

-pol3.fxml и Pol3.java: Файл pol.fxml содержит макет для экрана «Ингредиенты», который отображает данные из базы данных ресторана для администраторов. Класс Pol3.java является контроллером для этого fxml-файла и содержит логику, связанную с обработкой событий при нажатии кнопок, с помощью которых можно добавить, изменить или удалить ингредиенты из таблицы.

-pol4.fxml и Pol4.java: Файл pol.fxml содержит макет для экрана «Блюда», который отображает данные из базы данных ресторана для администраторов. Класс Pol4.java является контроллером для этого fxml-файла и содержит логику, связанную с обработкой событий при нажатии кнопок, с помощью которых можно добавить, изменить или удалить блюда из таблицы.

-pol5.fxml и Pol5.java: Файл pol.fxml содержит макет для экрана «Информация», который отображает данные из базы данных ресторана для администраторов и пользователей. Класс Pol5.java является контроллером для этого fxml-файла и содержит логику, связанную с представлением общей информации из сервера.

Каждый из этих классов-контроллеров отвечает за управление соответствующим fxml-файлом и обеспечивает взаимодействие между пользовательским интерфейсом и бизнес-логикой приложения.

Метод, представленный на рисунке 5, выполняет необходимую логику для обработки события нажатия кнопки в контроллере Pol4. В данном контексте, кнопка "add" представляет собой элемент интерфейса, при нажатии на который происходит добавление определенного блюда.

```
@FXML
void addMeals(ActionEvent event) {
    try {
        int id= Integer.parseInt(fieldId.getText());
        String name = fdname.getText();
        sql bd = new sql();
        bd.addMeals(id, name);
        // Обновляем данные в таблице
        list = sql.getMealsFromDatabase();
        table.setItems(list);
    } catch (SQLException e) {
        throw new RuntimeException(e);
    } catch (ClassNotFoundException e) {
        throw new RuntimeException(e);
    }
}
```

Рисунок 5 — Обработка события

В данном проекте, результаты запроса к базе данных обрабатываются с использованием специально разработанных классов.

Один из таких классов - это класс Ingredient, который играет ключевую роль в обработке информации об ингредиентах. Этот класс позволяет удобно хранить и работать с данными о каждом ингредиенте, предоставляя необходимые методы и свойства для их обработки.

Еще одним важным классом является класс Meal, который отвечает за хранение информации о блюде. Класс Meal обеспечивает удобный доступ и

манипуляцию с данными о блюде, а также содержит методы для выполнения определенных операций с блюдом.

Для управления пользователями в проекте используется класс Register. Этот класс представляет собой объект, который создается при регистрации пользователя и используется для аутентификации при входе в систему. Класс Register содержит информацию о пользователе, такую как имя, пароль и доступ. Он также предоставляет методы для проверки правильности введенных данных и выполнения других операций, связанных с управлением пользователями.

Наконец, класс Info используется для хранения общей информации, полученной из сервера ресторана. Класс Info предоставляет удобный доступ к этим данным и содержит методы для их отображения.

Использование таких специально разработанных классов позволяет более эффективно управлять и обрабатывать данные в проекте. Они обеспечивают удобный интерфейс для работы с информацией об ингредиентах, блюдах, пользователях и других аспектах приложения. Такой подход способствует повышению читаемости и понимаемости кода.

На рисунке 6 представлен пример использования данных классов. Создается экземпляр класса register, который представляет собой объект, содержащий информацию о пользователе, такую как имя, пароль и режим доступа. Затем вызывается метод regist() объекта bd, который выполняет операцию регистрации пользователя в базе данных, передавая ему созданный объект register.

```
1 usage   Polina
private void register1() throws SQLException, ClassNotFoundException {
    try {
        sql bd = sql.getInstance();
        String access = "";
        String username = fieldlogin.getText();
        String password = fieldpassword.getText();
        if (buttonadmin.isSelected()) {
            access = "Администратор";
        } else {
            access = "Пользователь";
        }
        register user = new register(username, password, access);
        bd.regist(user);
    } catch (SQLException e) {
        throw new RuntimeException(e);
    } catch (ClassNotFoundException e) {
        throw new RuntimeException(e);
    }
}
```

Рисунок 6 — Реализация обработки результата запроса в специально разработанный класс

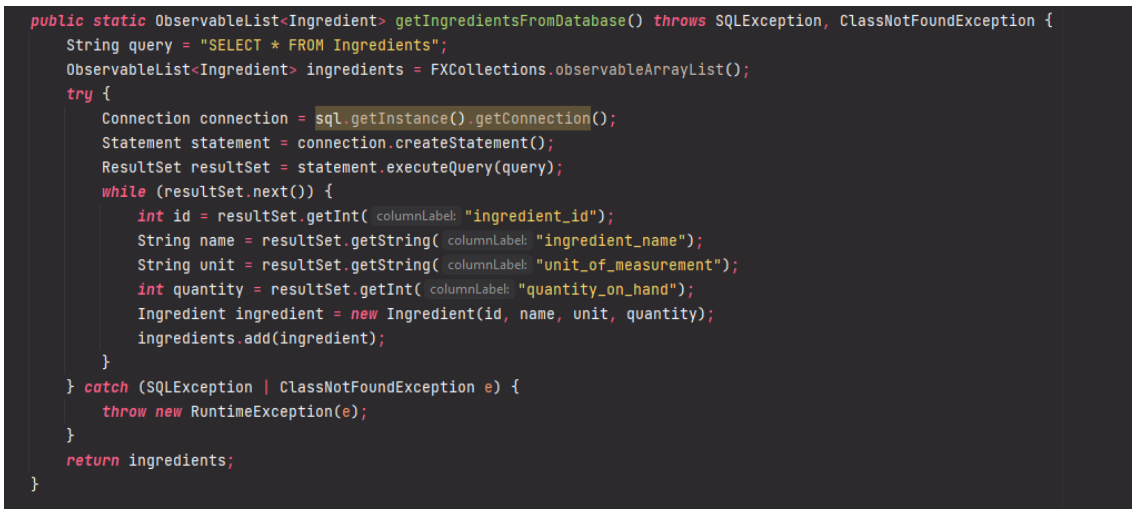
Остальные классы, используемые в проекте:

-Класс PolModel представляет собой модель в архитектуре MVC (Model-View-Controller). Класс PolModel отвечает за авторизацию пользователя в системе, проверяя его данные в базе данных и осуществляя соответствующие действия в зависимости от результата проверки.

- Класс Pol6Model представляет собой модель в архитектуре MVC (Model-View-Controller). Класс Pol6Model отвечает за обновление данных пользователя в базе данных, вызывая соответствующий метод объекта для работы с базой данных.

- Класс sql является синглтоном, то есть он позволяет создать только один экземпляр объекта и предоставляет глобальную точку доступа к этому объекту, отвечает за подключение базы данных к проекту. С помощью этого класса реализуются все sql-запроса.

На рисунке 7 представлен запрос, реализующий получение всех данных с таблицы Ингредиенты, в последующем данный метод используется в другом классе для записи данных из базы в таблицу, реализованную в окне.



```
public static ObservableList<Ingredient> getIngredientsFromDatabase() throws SQLException, ClassNotFoundException {
    String query = "SELECT * FROM Ingredients";
    ObservableList<Ingredient> ingredients = FXCollections.observableArrayList();
    try {
        Connection connection = sql.getInstance().getConnection();
        Statement statement = connection.createStatement();
        ResultSet resultSet = statement.executeQuery(query);
        while (resultSet.next()) {
            int id = resultSet.getInt( columnLabel: "ingredient_id");
            String name = resultSet.getString( columnLabel: "ingredient_name");
            String unit = resultSet.getString( columnLabel: "unit_of_measurement");
            int quantity = resultSet.getInt( columnLabel: "quantity_on_hand");
            Ingredient ingredient = new Ingredient(id, name, unit, quantity);
            ingredients.add(ingredient);
        }
    } catch (SQLException | ClassNotFoundException e) {
        throw new RuntimeException(e);
    }
    return ingredients;
}
```

Рисунок 7 — Пример запроса из класса sql

- Класс dobavlenie содержит статический метод hashPassword, который используется для хэширования пароля пользователя. Метод hashPassword принимает строку password в качестве параметра и возвращает хэшированную версию этого пароля.

Внутри метода происходит следующее: сначала проверяется, что входной пароль не является null. Если пароль равен null, то возвращается null.Затем создается экземпляр класса MessageDigest с использованием алгоритма хэширования SHA-256, вычисляется хэш пароля, вызывая метод digest у объекта digest и передавая ему байтовое представление пароля в кодировке UTF-8. Создается объект StringBuilder для построения строки, представляющей хэш в шестнадцатеричном формате.

Для каждого байта в хэше выполняются следующие действия:

- Преобразование байта в шестнадцатеричное представление с помощью метода toHexString.

- Если шестнадцатеричное представление имеет длину 1, то добавляется символ '0' перед шестнадцатеричным представлением.

- Шестнадцатеричное представление добавляется к объекту hexString.

В конце возвращается строковое представление объекта hexString, которое представляет собой хэш пароля (рис. 8).

```
2 usages  Polina
public class dobavlenie {
    2 usages  Polina
    protected static String hashPassword(String password) {
        if (password == null) {
            return null;
        }
        try {
            MessageDigest digest = MessageDigest.getInstance( algorithm: "SHA-256");
            byte[] hash = digest.digest(password.getBytes(StandardCharsets.UTF_8));
            StringBuilder hexString = new StringBuilder();
            for (byte b : hash) {
                String hex = Integer.toHexString( i: 0xff & b);
                if (hex.length() == 1) hexString.append('0');
                hexString.append(hex);
            }
            return hexString.toString();
        } catch (NoSuchAlgorithmException e) {
            e.printStackTrace();
            return null;
        }
    }
```

Рисунок 8 — Хэширование пароля

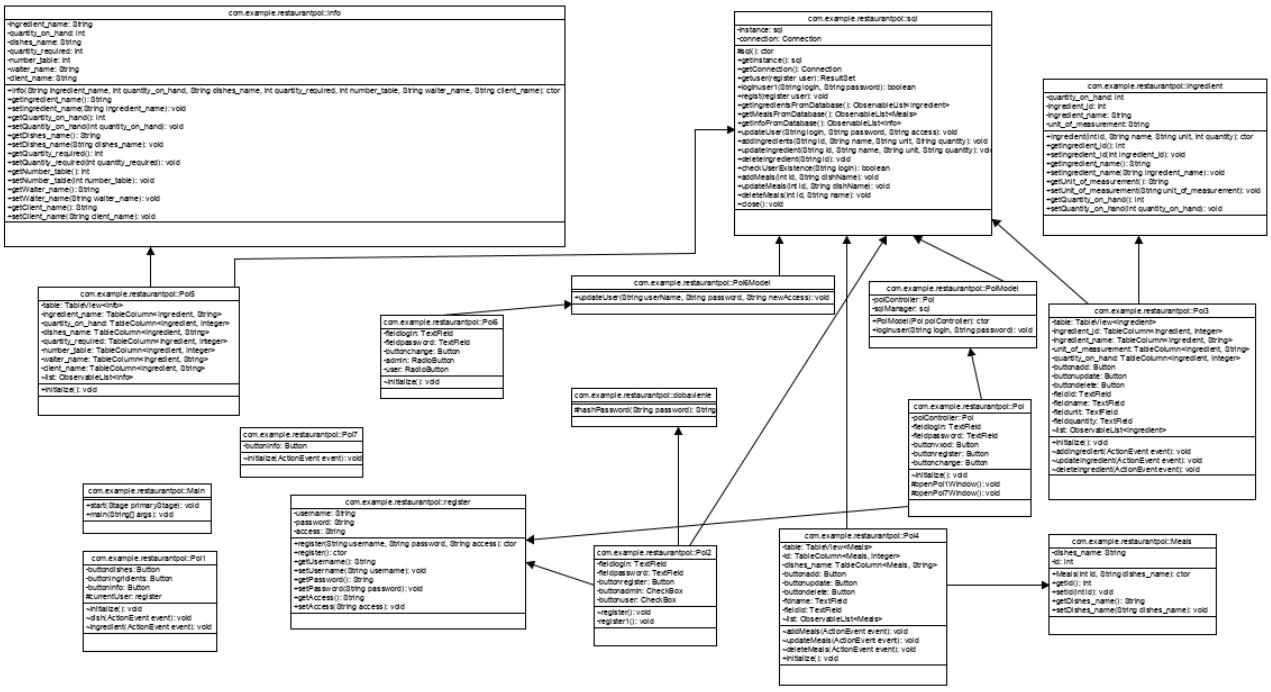


Рисунок 9 — Диаграмма классов

Основные сценарии работы

При инициализации приложения, пользователь встречается с главным окном, которое предоставляет различные функциональности для управления аккаунтом. Основной целью главного окна является предоставление пользователю возможности авторизоваться, зарегистрироваться или изменить свои данные (рис. 10).

Пользователь может использовать поле для ввода имени пользователя и поле для ввода пароля, чтобы войти в свою учетную запись. После ввода соответствующих данных, пользователь должен нажать на кнопку "Войти", чтобы выполнить процесс аутентификации. В случае успешной аутентификации, пользователь будет перенаправлен на другое окно приложения, где ему будут доступны дополнительные функциональности.

Если пользователь не имеет учетной записи, он может нажать на кнопку "Зарегистрироваться", чтобы создать новую учетную запись. После нажатия на эту кнопку, пользователю будет предоставлена форма регистрации, где он должен будет указать свой логин, пароль и уровень доступа. После заполнения всех полей, пользователь должен нажать на кнопку "Зарегистрироваться", чтобы завершить процесс регистрации.

Если пользователь уже имеет учетную запись и хочет изменить свои данные, он может нажать на кнопку "Изменить данные". После нажатия на эту кнопку, пользователю будет предоставлена форма, где он может внести необходимые изменения в свою учетную запись.

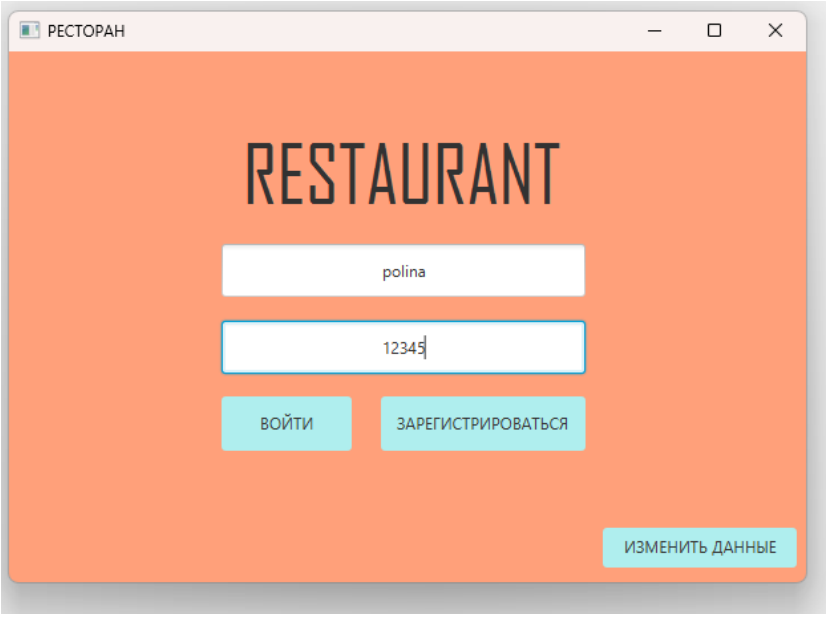


Рисунок 10 — Окно входа

После успешной аутентификации, пользователю будет предоставлен доступ к серверу ресторана, который предоставляет общую информацию о базе данных. Этот сервер предоставляет пользователю возможность просмотра различных данных, связанных с рестораном, таких как список ингредиентов и другие сведения о блюдах.

Однако, если пользователь обладает привилегиями администратора, то ему будет доступна дополнительная функциональность на сервере ресторана. В частности, пользователь с правами администратора сможет нажать на кнопку "Блюда" или "Ингредиенты", что позволит ему добавлять, изменять или удалять данные, связанные с блюдами и ингредиентами в базе данных ресторана.

Нажатие на кнопку "Блюда" откроет пользователю окно, где он сможет просмотреть список доступных блюд. Пользователь с правами администратора также сможет добавлять новые блюда, изменять информацию о существующих блюдах или удалять их из базы данных.

Аналогично, нажатие на кнопку "Ингредиенты" откроет пользователю окно, где он сможет просмотреть список доступных ингредиентов. Пользователь с правами администратора сможет добавлять новые ингредиенты, изменять информацию о существующих ингредиентах или удалять их из базы данных (рис. 11, 12).

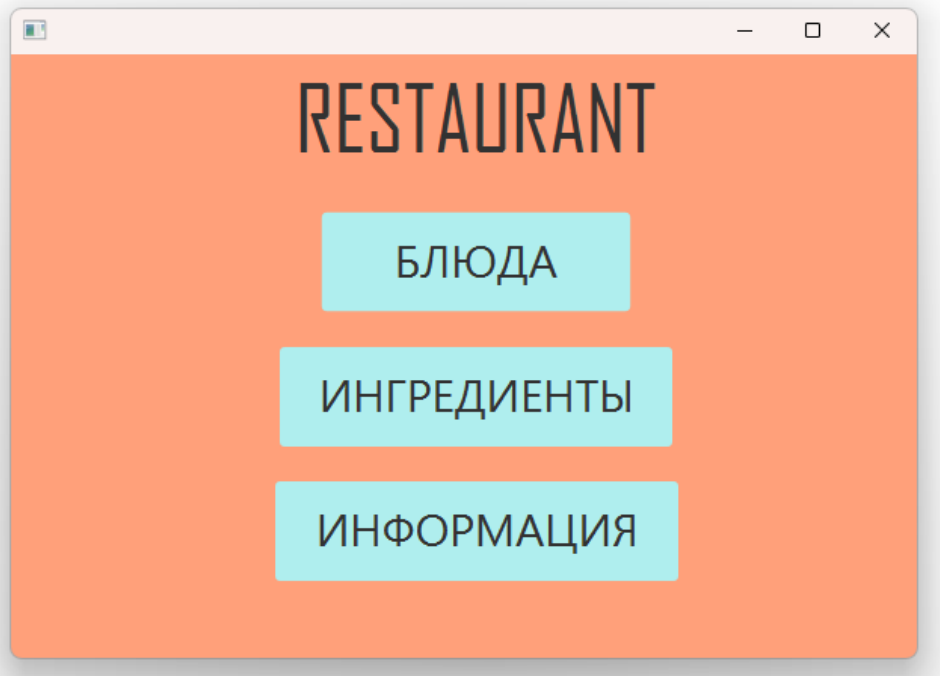


Рисунок 11 — Окно «меню» для администратора

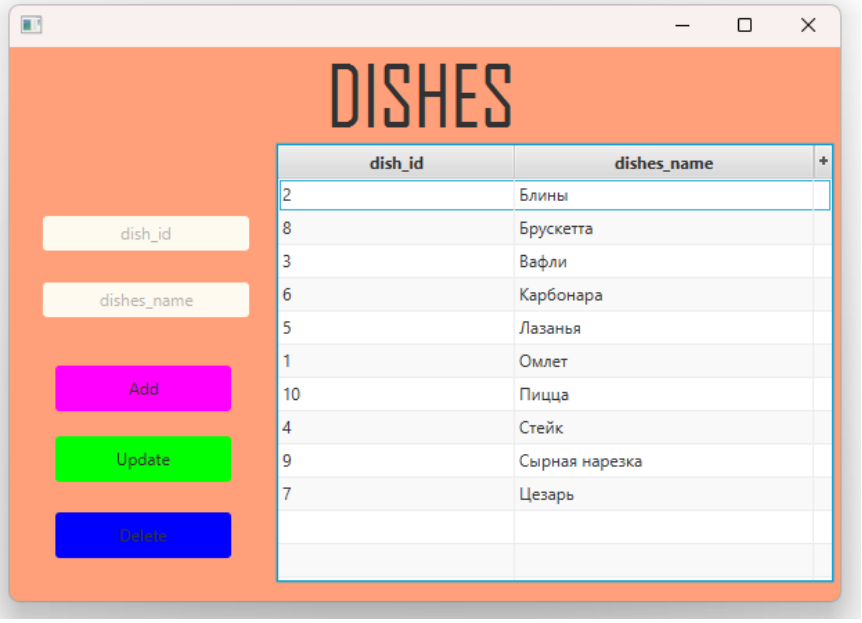


Рисунок 12 — Окно просмотра списка блюд

Заключение

В рамках проведенного исследования было разработано приложение для ресторана, которое предоставляет возможность управления ингридиентами и блюдами. Приложение обладает несколькими значительными достоинствами.

Во-первых, интерфейс приложения является приятным для пользователей. Он разработан с применением современных дизайнерских принципов, что делает его использование интуитивно понятным и эстетически привлекательным.

Во-вторых, расположение кнопок в приложении является удобным и логичным. Они распределены таким образом, чтобы пользователю было удобно выполнять основные операции без необходимости совершать излишние клики или перемещения по интерфейсу.

В-третьих, код приложения является простым и понятным. Он написан с учетом принципов читаемости и поддерживаемости, что облегчает его дальнейшую разработку.

Однако в ходе исследования были также выявлены некоторые недостатки приложения.

Во-первых, возникают небольшие ошибки при обработке событий. Вместо вывода сообщения об ошибочном введении данных, приложение иногда выдает исключение. Это является нежелательным моментом, который может создать путаницу у пользователей.

Во-вторых, отсутствует проверка соответствия паролей из базы данных с введенным паролем пользователем. Это означает, что приложение не гарантирует безопасность пользовательских данных, что является серьезным недостатком.

Тем не менее, данные недостатки относительно просты в исправлении. В дальнейшем планируется проведение работ над их устранением. Исправление ошибок позволит улучшить работу и безопасность приложения.

При разработке приложения осуществлялось ведение удаленного репозитория посредством системы контроля версий Git: https://github.com/sweetmintt/polinka.

Список литературы и интернет-ресурсы:

1. Статья: " Учебник по JavaFX: FXML и SceneBuilder" Habr, 12 ноября 2019 г., https://habr.com/ru/articles/474982/.

2. Руководство: "JavaFX Tutorial - Часть 1: Введение в JavaFX." Code.Makery, https://code.makery.ch/ru/library/javafx-tutorial/part1/.

3. Статья: " Использование Scene Builder для разработки приложений на JavaFX" Open Source For You, 12 сентября 2017 г., https://www.opensourceforu.com/2017/09/develop-javafx-apps/.

4. Статья: " Работаем с Git: первые шаги в GitHub" Habr, 22 ноября 2022 г., https://habr.com/ru/companies/yandex_praktikum/articles/700708/.