МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ НИЖЕГОРОДСКОЙ ОБЛАСТИ
Государственное бюджетное профессиональное образовательное учреждение
НИЖЕГОРОДСКИЙ РАДИОТЕХНИЧЕСКИЙ КОЛЛЕДЖ

Специальность 09.02.07 Информационные системы и программирование

Квалификация Программист

КУРСОВАЯ РАБОТА

МДК 11.01 Технология разработки и защиты баз данных

Тема: «Проектирование и разработка базы данных для маникюрного салона»

Выполнил студент группы

3ИСиП-21-1

Булычева Арина Александровна

Проверил преподаватель
Гутянская Е.М.

Проект защищен с оценкой
_____
Дата защиты_____
Подпись_____

Нижний Новгород, 2023

ЗАДАНИЕ

на курсовую работу
Специальность 09.02.07 Информационные системы и программирование
МДК 11.01 Технология разработки и защиты баз данных

Обучающемуся Булычевой Арине Александровне

Группа 3ИСиП-21-1

Тема курсовой работы Проектирование и разработка базы данных для маникюрного салона

Требования к разрабатываемой системе

1. Многопользовательский доступ
2. Разграничение функционала по ролям
3. Клиент и сотрудник салона могут осуществлять следующие действия:
    a. получать оперативную информацию о наличии свободного времени у мастера;
    b. подбирать маникюр по ряду критериев;
    c. записываться на услугу на любое свободное время;
    d. находить необходимую информацию о средствах используемы при оказании услуги;
    e. клиент может при необходимости указать наличие у него аллергии.
4. Сотрудник салона может осуществлять следующие действия:
    a. вести справочники (добавление, удаление, редактирование);
    b. оформлять заказ на услуги (в одном заказе может быть несколько различных услуг);
    c. стоимость заказа рассчитывается динамически.

Дата выдачи задания «_____» ноября 2023 г.
Срок сдачи работы «_____» ноября 2023 г.

Перечень вопросов, подлежащих разработке:

- Введение (название выбранной темы, обзор раскрываемых вопросов, актуальность).
- Анализ предметной области.

- Проектирование базы данных информационной системы.
- Разработка информационной системы.
- Руководство пользователя.
- Заключение.
- Список использованной литературы.

Задание выдал преподаватель _____

Задание принял обучающийся_____

Оглавление

Введение

Проектирование и разработка базы данных для маникюрного салона представляет собой важную задачу в контексте современной индустрии красоты и ухода за собой. С увеличением числа клиентов и услуг, предоставляемых маникюрными салонами, эффективное управление информацией о клиентах, услугах, запасах и расписании становится ключевым фактором успеха. В связи с этим, создание и оптимизация базы данных для маникюрного салона имеет большое значение для повышения эффективности бизнеса, улучшения обслуживания клиентов и увеличения конкурентоспособности.

Развитие информационных технологий и программных решений в сфере красоты и ухода за собой также подчеркивает актуальность данной темы. Внедрение современных баз данных позволяет автоматизировать процессы учета клиентов, управления записями, анализа предпочтений клиентов и оптимизации инвентаря. Таким образом, разработка базы данных для маникюрного салона является важным шагом в направлении повышения эффективности и конкурентоспособности бизнеса в сфере красоты и ухода за собой.

В рамках курсовой работы необходимо разработать базу данных для маникюрного салона в которой будет храниться информация об услугах, записях, пользователях системы и др. После чего я будет создавать консольное приложение для осуществления необходимых действий в системе.

1. Теоретические основы разрабатываемой темы.

1.1. Анализ предметной области

В курсовом проекте должны выполняться следующие ключевые аспекты:

1. Многопользовательский доступ: Система должна поддерживать одновременный доступ нескольких пользователей. Это может включать в себя как клиентов, так и сотрудников салона. Каждый из этих пользователей должен иметь возможность выполнять определенные действия в системе в зависимости от их роли.

2. Разграничение функционала по ролям: Система должна поддерживать роль-ориентированный контроль доступа (RBAC). Это означает, что разные пользователи (в зависимости от их ролей) будут иметь доступ к различным функциям системы. Например, сотрудники салона могут иметь доступ к функциям управления заказами, в то время как клиенты могут иметь доступ к функциям бронирования услуг.

3. Клиент и сотрудник салона могут осуществлять следующие действия: Система должна предоставлять возможность клиентам и сотрудникам салона выполнять ряд действий. Это может включать в себя получение оперативной информации о наличии свободного времени у мастера, подбор маникюра по ряду критериев, запись на услугу на любое свободное время, поиск информации о средствах, используемых при оказании услуги, и указание наличия у клиента аллергии.

4. Сотрудник салона может осуществлять следующие действия: Система должна предоставлять возможность сотрудникам салона вести справочники (добавление, удаление, редактирование), оформлять заказ на услуги (в одном заказе может быть несколько различных услуг), и рассчитывать стоимость заказа динамически.

2. Проектирование базы данных информационной системы

2.1. Информационно-логическая модель базы данных

В СУБД MySQL Workbench я создала базу данных «Salon» (рис. 1) со следующими таблицами: Orders, Resources, Services, Users, Masters, Schedule.

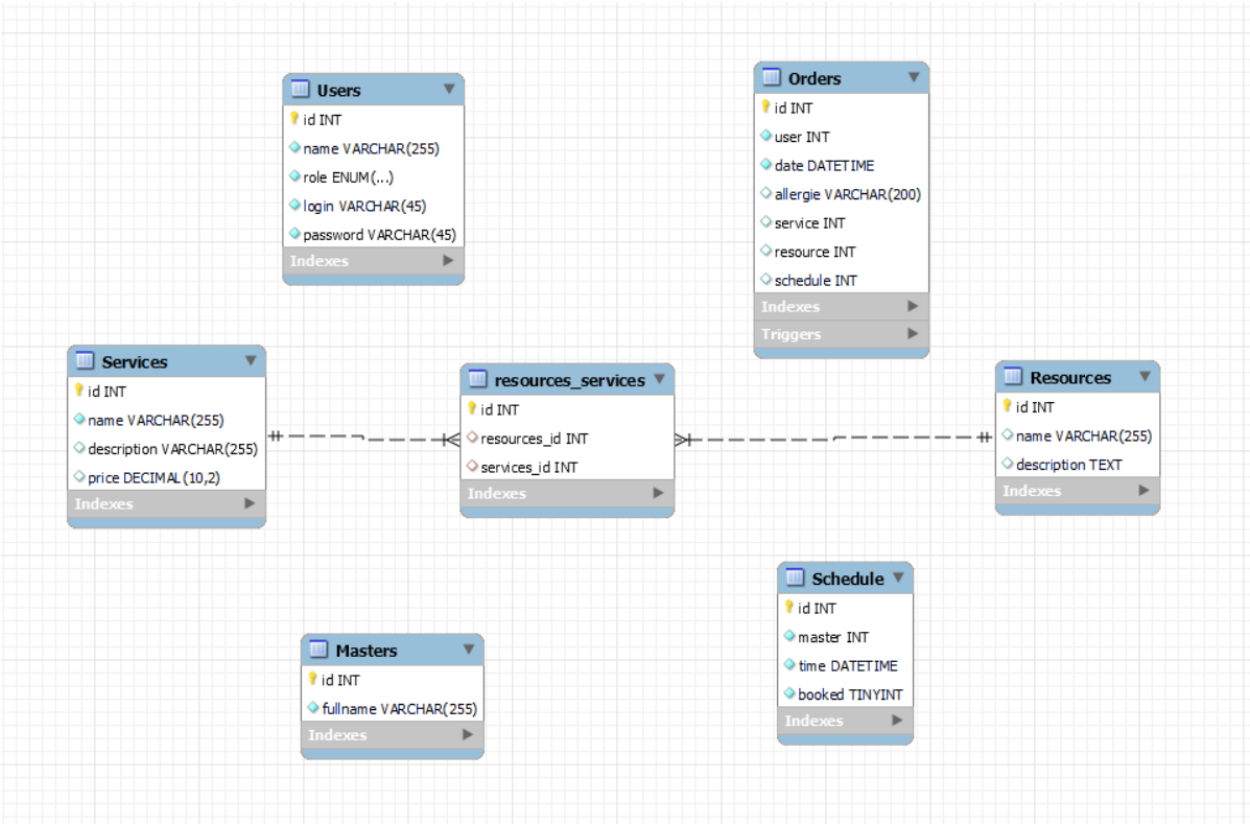


Рис. 1 «Схема базы данных «Salon»»

2.2. Словарь данных

База данных «Salon»
Таблицы:
1. Orders (это таблица заказов)
Id (INT) – id заказа
User (INT)– id пользователь
Date (DATETIME) – дата создания записи
Allergie (VARCHAR (200)) – тут пользователь указывает есть ли у него аллергия

Service (INT) – id услуги
Resourse (INT) – id используемые инструменты из таблицы resourse
Schedule (INT) – id свободной записи

2. Resources (таблица используемых инструментов при оказании услуг)
Id (INT) – id инструмента
Name (VARCHAR(255)) – наименование предмета
Description (TEXT) – описание инструмента

3. Services (таблица с услугами)
Id (INT) – id услуги
Name (VARCHAR(255)) – наименование услуги
Description (TEXT) – описание услуги
Price (DECIMAL(10,2)) – цена услуги

4. Users (информация о пользователях)
Id (INT) – id пользователя
Name (VARCHAR(255)) – имя пользователя
Role (ENUM('Клиент', 'Сотрудник')) – роль пользователя (кем является пользователь: клиент/сотрудник)
Login (VARCHAR(45)) – логин пользователя
Password (VARCHAR(45)) – пароль пользователя

5. Masters (список мастеров)
Id (INT) – id мастера
Fullname (VARCHAR(255)) – имя и фамилия мастера

6. Schedule (таблица хранит информацию о свободном времени)
Id (INT) – id
Master (INT) – id свободного мастера
time (DATETIME) – дата и время на которую ещё не оформлена запись
booked (TINYINT) – показатель занято время или нет (0 – нет, 1 - да)

7. resources_services (таблица показывает какие ресурсы используются для выполнения конкретной услуги)
Id(INT) – объединяющий id для ресурсов и услуг
Resources_id(INT) -id ресурса
Services_id(INT) – id услуги

2.3. Ограничения ссылочной целостности

В моей базе данных существует несколько таблиц, и между ними установлены связи. Эти связи обеспечивают ссылочную целостность в базе данных и позволяют связывать данные между разными таблицами. Давайте рассмотрим каждую из таблиц и их связи:

Таблица resources_services связана с resources связью «много-один» и таблица resources_services связана с services связью «много-один», таким образом получается, есть гарантия, что ресурс, связанный с конкретной услугой, существует в таблице Resources и что услуга, связанная с конкретным ресурсом, существует в таблице Services. Таким образом в таблицу в поле с id, например, 1, добавляется одно поле из таблицы resources и одно поле из services.

Эти связи обеспечивают целостность данных и предотвращают появление некорректных данных.

2.4. Обоснование выбора СУБД

Выбор MySQL Workbench в качестве СУБД обоснован следующими причинами:

1. Простота использования.

Интуитивно понятный интерфейс, который упрощает работу с базами данных. Он предлагает функции для проектирования и моделирования баз данных, а также для выполнения SQL-запросов и управления базами данных.

2. Поддержка SQL.

MySQL Workbench позволяет создавать, управлять и конфигурировать соединения и параметры соединения с серверами баз данных MySQL. Он также позволяет выполнять SQL-запросы на этих соединениях с помощью встроенного редактора.

3. Моделирование и дизайн.

MySQL Workbench позволяет создавать и манипулировать моделями баз данных, включая обратную инженерию живой базы данных в модель и создание и редактирование таблиц и вставку данных.

4. Администрирование пользователей.

MySQL Workbench упрощает управление пользователями. Вы можете легко создавать, редактировать и удалять пользователей, а также управлять их привилегиями.

5. Поддержка различных типов данных.

MySQL поддерживает множество типов данных, включая числовые, строковые, даты и времени, и даже специализированные типы данных, такие как JSON и GIS. Это делает его гибким инструментом для различных типов проектов.

6. Поддержка стандартов.

MySQL поддерживает большинство стандартов SQL, что облегчает интеграцию с другими системами и инструментами.

7. Безопасность.

MySQL предлагает множество функций безопасности, включая шифрование данных, поддержку аутентификации и авторизации, а также механизмы защиты от атак.

2.5. Триггеры и хранимые процедуры.

Триггеры и хранимые процедуры – это именованные блоки кода SQL, которые заранее откомпилированы и хранятся на сервере для того, чтобы быстро производить обработку запросов, валидацию данных и др.

Хранимая процедура CreateNewOrder

Эта процедура предназначена для создания нового заказа. Она принимает различные параметры, связанные с заказом, такие как идентификатор пользователя, дата заказа, аллергии, критерии, детали услуги, ресурса, расписания и полное имя мастера. Процедура выполняет следующие шаги:

Объявление переменных. Объявляет несколько переменных для хранения идентификаторов мастера, расписания, ресурса, услуги и самого заказа.

Начало транзакции. Инициирует транзакцию для обеспечения выполнения всех запросов как единого блока работы. Это помогает поддерживать согласованность данных.

Извлечение данных. Извлекает идентификаторы мастера, расписания, ресурса и услуги на основе предоставленной информации.

Обновление расписания. Обновляет расписание, назначая выбранного мастера и отмечая его как забронированное.

Вставка данных в таблицу заказов. Вставляет детали заказа в таблицу Orders, используя полученные идентификаторы для услуги, ресурса и расписания.

Получение идентификатора заказа. Извлекает идентификатор нового заказа с помощью LAST_INSERT_ID ().

Завершение транзакции. Фиксирует транзакцию в случае успешного выполнения всех шагов или выполняет откат в случае ошибки.

```
DELIMITER $$

CREATE PROCEDURE CreateNewOrder(

  order_user INT,
  order_date DATETIME,
  order_allergie VARCHAR(255),
  order_criteria TEXT,
  service_name VARCHAR(255),
  service_description TEXT,
  service_price DECIMAL(10,2),
  resource_name VARCHAR(255),
  resourse_description TEXT,
  schedule_time DATETIME,
  schedule_booked TINYINT,
  master_fullname VARCHAR(255)

)
BEGIN

-- объявление новой переменных с идентификатором
    DECLARE order_id BOOLEAN DEFAULT false;
  DECLARE master_id INT DEFAULT null;
    DECLARE schedule_id INT DEFAULT null;
    DECLARE resource_id INT DEFAULT null;
    DECLARE service_id INT DEFAULT null;


-- начало транзакции что бы все запросы выполнить за одно обращение
    START TRANSACTION;

    SELECT id INTO master_id FROM Masters WHERE Masters.fullname = master_fullname;
    SELECT id INTO schedule_id FROM Schedule WHERE Schedule.time = schedule_time;
    UPDATE Schedule SET Schedule.master = master_id WHERE Schedule.id = schedule_id;
    SELECT id INTO resource_id FROM Resources WHERE Resources.name = resource_name;
    SELECT id INTO service_id FROM Services WHERE Services.name = service_name;

-- вставка данных в основную таблицу с заказами
    INSERT INTO Orders (user, date, allergie, criteria, service, resource, schedule)
    VALUES (order_user, order_date, order_allergie, order_criteria, service_id, resource_id, schedule_id);

-- получение идентификатора нового заказа
    SET order_id = LAST_INSERT_ID();

-- завершение транзакции успешно или ROLLBACK в случае ошибки
    COMMIT;

END$$

DELIMITER ;
```

Рис. 1 «Хранимая процедура CreateNewOrder»

Триггер after_order_created

Этот триггер автоматически выполняется после вставки новой строки в таблицу Orders. Он обновляет статус booked соответствующего расписания на true на основе идентификатора расписания, связанного с только что созданным заказом.

```
1   DELIMITER $$
2
3   CREATE TRIGGER after_order_created AFTER INSERT ON `Orders` FOR EACH ROW
4
5   BEGIN
6     SET @scheduleId := NEW.schedule;
7     UPDATE `Schedule` SET `booked` = true WHERE `Schedule`.`id` = @scheduleId;
8   END$$
9
10  DELIMITER ;
```

Рис. 1 «Триггер after_order_created»

Хранимая процедура UpdateExistOrder

Эта процедура предназначена для обновления существующего заказа. Она принимает параметры, включая идентификатор заказа, обновленные детали заказа, детали услуги, ресурса, расписания и полное имя мастера. Процедура выполняет следующие шаги:

Объявление переменных. Объявляет переменные для хранения идентификаторов мастера, расписания и услуги.

Начало транзакции. Инициирует транзакцию для обеспечения согласованности данных.

Извлечение данных. Извлекает идентификаторы мастера, расписания и услуги на основе предоставленной информации.

Обновление расписания. Обновляет расписание, назначая нового мастера.

Обновление заказа. Обновляет данные заказа в таблице Orders с новыми деталями заказа, новой услугой и новым расписанием.

Завершение транзакции. Фиксирует транзакцию в случае успешного выполнения всех шагов или выполняет откат в случае ошибки.

```
DELIMITER $$

CREATE PROCEDURE UpdateExistOrder(

  order_id INT,
  order_date DATETIME,
  order_allergie VARCHAR(255),
  service_name VARCHAR(255),
  service_description TEXT,
  service_price DECIMAL(10,2),
  resource_name VARCHAR(255),
  resourse_description TEXT,
  schedule_time DATETIME,
  schedule_booked TINYINT,
  master_fullname VARCHAR(255)

)
BEGIN
  -- объявление новой переменной с идентификатором
    DECLARE master_id INT DEFAULT null;
    DECLARE schedule_id INT DEFAULT null;
    DECLARE service_id INT DEFAULT null;


  -- начало транзакции что бы все запросы выполнить за одно обращение
    START TRANSACTION;

    SELECT id INTO master_id FROM `Masters` WHERE `Masters`.`fullname` = master_fullname;
    SELECT id INTO schedule_id FROM `Schedule` WHERE `Schedule`.`time` = schedule_time;
    UPDATE `Schedule` SET `Schedule`.`master` = master_id WHERE `Schedule`.`id` = schedule_id;
    SELECT id INTO service_id FROM `Services` WHERE `Services`.`name` = service_name;

    UPDATE `Orders` SET `Orders`.`date` = order_date, `Orders`.`allergie` = order_allergie, `Orders`.`service` = service_id, `Orders`.`schedule` = schedule_id WHERE `Orders`.`

  -- завершение транзакции успешно или ROLLBACK в случае ошибки
    COMMIT;
END$$

DELIMITER ;
```

Рис. 1 «Хранимая процедура UpdateExistOrder»

3. Разработка информационной системы

В данном разделе подробно описывается процесс разработки информационной системы, написанной на языке программирования Java с использованием технологии JavaFX. Реализация кода сосредоточена на создании графического пользовательского интерфейса (GUI) для обеспечения удобного и эффективного взаимодействия пользователя с системой.

3.1. Основные классы программы

DatabaseController

Класс контроллер базы данных, который управляет подключением к базе данных MySQL, выполняет SQL-запросы и обрабатывает результаты.

В каждом из этих методов выполняются различные операции с базой данных, такие как установка соединения, разрыв соединения, авторизация пользователя, регистрация пользователя, получение списка заказов, создание нового заказа, обновление существующего заказа и удаление заказа. Все эти операции выполняются с использованием SQL-запросов и объектов Connection, PreparedStatement и ResultSet.

```
package com.example.salonchik;

import java.sql.*;
import java.util.ArrayList;
import java.util.List;
import java.util.Properties;

public class DatabaseController {

    // Константы для настройки подключения к базе данных
    private static final String DATABASE_DRIVER = "com.mysql.cj.jdbc.Driver";
    private static final String DATABASE_URL =
"jdbc:mysql://localhost:3306/Salon";
    private static final String DATABASE_USERNAME = "root";
    private static final String DATABASE_PASSWORD = "";
    private static final String DATABASE_MAX_POOL = "250";


    // Объект соединения с базой данных
    private Connection connection;
    public static int idK;


    // Объект для хранения свойств подключения к базе данных
    private Properties properties;


    // Метод для инициализации свойств подключения к базе данных
    private Properties properties() {
        if (this.properties == null) {
            this.properties = new Properties();
```

```
            this.properties.setProperty("user", DATABASE_USERNAME);
            this.properties.setProperty("password", DATABASE_PASSWORD);
            this.properties.setProperty("MaxPooledStatements",
DATABASE_MAX_POOL);
        }
        return this.properties;
    }


    // Метод для установки соединения с базой данных
    public Connection connect() {
        if (this.connection == null) {
            try {
                // Загрузка драйвера и установка соединения
                Class.forName(DATABASE_DRIVER);
                this.connection = DriverManager.getConnection(DATABASE_URL,
properties());
            } catch (Exception error) {
                error.printStackTrace();
            }
        }
        return this.connection;
    }


    // Метод для разрыва соединения с базой данных
    public void disconnect() {
        if (this.connection != null) {
            try {
                // Закрытие соединения
                this.connection.close();
                this.connection = null;
            } catch (Exception error) {
                error.printStackTrace();
            }
        }
    }


    // Метод для авторизации пользователя
    public DatabaseUser authorization(String login, String password) {
        // Проверка наличия соединения с базой данных
        if (this.connection == null) {
            return null;
        }
        try {
            // Подготовка и выполнение SQL-запроса для авторизации
            String query = "SELECT * FROM `Users` WHERE `Users`.`login` = ?
AND `Users`.`password` = ?;";
            PreparedStatement statement =
this.connection.prepareStatement(query);
            // Установка параметров запроса
            statement.setString(1, login);
            statement.setString(2, password);
            // Выполнение SQL запроса и получение результата
            ResultSet result = statement.executeQuery();

            // Обработка результата запроса
            if (result.next()) {
                DatabaseUser user = new DatabaseUser();
                // Заполнение объекта пользователя данными из результата
запроса
                user.idK       = result.getInt("id");
```

```
                user.name     = result.getString("name");
                user.role     = result.getString("role");
                user.login    = result.getString("login");
                user.password = result.getString("password");
                // Закрытие PreparedStatement
                statement.close();
                // Возвращение объекта пользователя
                return user;
            }
            ResultSet resultSet = statement.executeQuery();
            if (resultSet.next()) {
                idK = resultSet.getInt("id"); // Получить id
            }
            statement.close();
            throw new NullPointerException("User with given login is not
found in database!");
        } catch (Exception error) {
            // Вывод информации об ошибке
            error.printStackTrace();
            // Возвращение null в случае ошибки
            return null;
        }
    }


    // Метод для регистрации пользователя
    public boolean registration (String name, String login, String password)
throws SQLException {
        // Проверка наличия соединения с базой данных
        if (this.connection == null) {
            return false;
        }
        try {
            // Создание SQL запроса для вставки нового пользователя
            String query = "INSERT INTO `Users` (`name`, `login`, `password`)
VALUES (?, ?, ?);";
            // Создание PreparedStatement для выполнения SQL запроса
            PreparedStatement statement =
this.connection.prepareStatement(query);
            // Установка параметров запроса
            statement.setString(1, name);
            statement.setString(2, login);
            statement.setString(3, password);
            // Выполнение SQL запроса и получение результата
            boolean result = statement.execute();
            // Закрытие PreparedStatement
            statement.close();
            // Возвращение результата
            return result;
        } catch (Exception error) {
            // Вывод информации об ошибке
            error.printStackTrace();
            // Выбрасывание исключения SQLException
            throw new SQLException(error.getMessage());
        }
    }


    // Метод для получения заказов клиента
    public List<DatabaseOrder> clientOrders(int id) {
        // Проверка наличия соединения с базой данных
        if (this.connection == null) {
            return new ArrayList<DatabaseOrder>();
```

```
        }
        try {
            // Подготовка и выполнение SQL-запроса для получения заказов
клиента
            String query = "SELECT * FROM `Orders` LEFT JOIN `Services` ON
`Orders`.`service` = `Services`.`id` LEFT JOIN `Schedule` ON
`Orders`.`schedule` = `Schedule`.`id` LEFT JOIN `Masters` ON
`Schedule`.`master` = `Masters`.`id` WHERE `Orders`.`user` = ?;";
            PreparedStatement statement =
this.connection.prepareStatement(query);
            // Установка параметра запроса
            statement.setInt(1, id);
            // Выполнение SQL запроса и получение результата
            ResultSet result = statement.executeQuery();
            // Создание списка для хранения заказов
            List<DatabaseOrder> orders = new ArrayList<DatabaseOrder>();
            // Обработка результата запроса
            while (result.next()) {
                orders.add(fillDatabaseOrder(result));
            }
            // Закрытие PreparedStatement
            statement.close();
            // Возвращение списка заказов
            return orders;
        } catch (Exception error) {
            // Вывод информации об ошибке
            error.printStackTrace();
            // Возвращение пустого списка в случае ошибки
            return new ArrayList<DatabaseOrder>();
        }
    }


    // Метод для получения заказов сотрудников
    public List<DatabaseOrder> employeeOrders() {
        // Проверка наличия соединения с базой данных
        if (this.connection == null) {
            return new ArrayList<DatabaseOrder>();
        }
        try {
            // Подготовка и выполнение SQL-запроса для получения заказов
сотрудников
            String query = "SELECT * FROM `Orders` LEFT JOIN `Services` ON
`Orders`.`service` = `Services`.`id` LEFT JOIN `Schedule` ON
`Orders`.`schedule` = `Schedule`.`id` LEFT JOIN `Masters` ON
`Schedule`.`master` = `Masters`.`id`;";
            PreparedStatement statement =
this.connection.prepareStatement(query);
            // Выполнение SQL запроса и получение результата
            ResultSet result = statement.executeQuery();
            // Создание списка для хранения заказов
            List<DatabaseOrder> orders = new ArrayList<DatabaseOrder>();
            // Обработка результата запроса
            while (result.next()) {
                orders.add(fillDatabaseOrder(result));
            }
            // Закрытие PreparedStatement
            statement.close();
            // Возвращение списка заказов
            return orders;
        } catch (Exception error) {
            // Вывод информации об ошибке
            error.printStackTrace();
```

```
            // Возвращение пустого списка в случае ошибки
            return new ArrayList<DatabaseOrder>();
        }
    }


    // Метод для заполнения объекта DatabaseOrder из ResultSet
    private DatabaseOrder fillDatabaseOrder(ResultSet result) throws
SQLException {
        // Создание нового объекта DatabaseOrder
        DatabaseOrder order = new DatabaseOrder();
        // Заполнение объекта DatabaseOrder данными из ResultSet
        order.setId(result.getInt(1));
        order.setUser(result.getInt(2));
        order.setDate(result.getString(3));
        order.setAllergie(result.getString(4));
        order.setService(result.getString(9));
        order.setDescriptionM(result.getString(10));
        order.setPrice(result.getFloat(11));
        order.setTime(result.getString(14));
        order.setBooked(result.getBoolean(15)?"Занято":"Свободно");
        order.setMaster(result.getString(17));
        // Возвращение объекта DatabaseOrder
        return order;
    }


    // Метод для получения списка мастеров
    public List<String> masters() {
        // Попытка выполнения операции
        try {
            // Подготовка и выполнение SQL-запроса для получения списка
мастеров
            String query = "SELECT * FROM `Masters`;";
            PreparedStatement statement =
this.connection.prepareStatement(query);
            // Выполнение SQL запроса и получение результата
            ResultSet result = statement.executeQuery();
            // Создание списка для хранения мастеров
            List<String> masters = new ArrayList<>();
            // Обработка результата запроса
            while (result.next()) {
                masters.add(result.getString("fullname"));
            }
            // Закрытие PreparedStatement
            statement.close();
            // Возвращение списка мастеров
            return masters;
        } catch (Exception error) {
            // Вывод информации об ошибке
            error.printStackTrace();
            // Возвращение пустого списка в случае ошибки
            return new ArrayList<>();
        }
    }


    // Метод для получения списка услуг
    public List<String> services(String field) {
        try {
            // Подготовка и выполнение SQL-запроса для получения списка услуг
            String query = "SELECT * FROM `Services`;";
            PreparedStatement statement =
```

```
this.connection.prepareStatement(query);
            // Выполнение SQL запроса и получение результата
            ResultSet result = statement.executeQuery();
            // Создание списка для хранения услуг
            List<String> services = new ArrayList<>();
            // Обработка результата запроса
            while (result.next()) {
                services.add(result.getString(field));
            }
            // Закрытие PreparedStatement
            statement.close();
            // Возвращение списка услуг
            return services;
        } catch (Exception error) {
            // Вывод информации об ошибке
            error.printStackTrace();
            // Возвращение пустого списка в случае ошибки
            return new ArrayList<>();
        }
    }


    // Метод для получения списка ресурсов
    public List<String> resources(String field) {
        try {
            // Подготовка и выполнение SQL-запроса для получения списка
ресурсов
            String query = "SELECT * FROM `Resources`;";
            PreparedStatement statement =
this.connection.prepareStatement(query);
            // Выполнение SQL запроса и получение результата
            ResultSet result = statement.executeQuery();
            // Создание списка для хранения ресурсов
            List<String> resources = new ArrayList<>();
            // Обработка результата запроса
            while (result.next()) {
                resources.add(result.getString(field));
            }
            // Закрытие PreparedStatement
            statement.close();
            // Возвращение списка ресурсов
            return resources;
        } catch (Exception error) {
            // Вывод информации об ошибке
            error.printStackTrace();
            // Возвращение пустого списка в случае ошибки
            return new ArrayList<>();
        }
    }


    // Метод для получения расписания
    public List<String> schedule(String field) {
        try {
            // Подготовка и выполнение SQL-запроса для получения списка
расписания
            String query = "SELECT * FROM `Schedule`;";
            PreparedStatement statement =
this.connection.prepareStatement(query);
            // Выполнение SQL запроса и получение результата
            ResultSet result = statement.executeQuery();
            // Создание списка для хранения расписания
            List<String> schedule = new ArrayList<>();
```

```
            // Обработка результата запроса
            while (result.next()) {
                schedule.add(result.getString(field));
            }
            // Закрытие PreparedStatement
            statement.close();
            // Возвращение списка расписания
            return schedule;
        } catch (Exception error) {
            // Вывод информации об ошибке
            error.printStackTrace();
            // Возвращение пустого списка в случае ошибки
            return new ArrayList<>();
        }
    }


    // Метод для создания нового заказа
    public boolean create (DatabaseOrder order) {
        try {
            // Подготовка и выполнение SQL-запроса для создания нового заказа
            String query = "CALL CreateNewOrder(?, ?, ?, ?, ?, ?, ?, ?, ?, ?,
?, ?);";
            PreparedStatement statement =
this.connection.prepareStatement(query);
            // Установка значений параметров запроса
            statement.setInt(1, order.getUser());
            statement.setString(2, order.getDate());
            statement.setString(3, order.getAllergie());
            statement.setString(4, order.getCriteria());
            statement.setString(5, order.getService());
            statement.setString(6, order.getDescriptionM());
            statement.setFloat(7, order.getPrice());
            statement.setString(8, order.getResource());
            statement.setString(9, order.getDescriptionR());
            statement.setString(10, order.getTime());
            statement.setBoolean(11,
order.getBooked().equalsIgnoreCase("Занято")?true:false);
            statement.setString(12, order.getMaster());
            // Выполнение SQL запроса и получение результата
            boolean result = statement.execute();
            // Закрытие PreparedStatement
            statement.close();
            // Возвращение результата операции
            return result ;
        } catch (Exception error) {
            // Вывод информации об ошибке
            error.printStackTrace();
            // Возвращение false в случае ошибки
            return false;
        }
    }


    // Метод для обновления существующего заказа
    public boolean update (DatabaseOrder order) {
        try {
            // Подготовка и выполнение SQL-запроса для обновления
существующего заказа
            String query = "CALL UpdateExistOrder(?, ?, ?, ?, ?, ?, ?, ?, ?,
?, ?);";
            PreparedStatement statement =
this.connection.prepareStatement(query);
```

```
            // Установка значений параметров запроса
            statement.setInt(1, order.getId());
            statement.setString(2, order.getDate());
            statement.setString(3, order.getAllergie());
            statement.setString(4, order.getService());
            statement.setString(5, order.getDescriptionM());
            statement.setFloat(6, order.getPrice());
            statement.setString(7, order.getResource());
            statement.setString(8, order.getDescriptionR());
            statement.setString(9, order.getTime());
            statement.setBoolean(10,
order.getBooked().equalsIgnoreCase("Занято")?true:false);
            statement.setString(11, order.getMaster());
            // Выполнение SQL запроса и получение результата
            int result = statement.executeUpdate();
            // Закрытие PreparedStatement
            statement.close();
            // Возвращение результата операции
            return result > 0;
        } catch (Exception error) {
            // Вывод информации об ошибке
            error.printStackTrace();
            // Возвращение false в случае ошибки
            return false;
        }
    }


    // Метод для удаления заказа
    public boolean delete (int id) {
        try {
            // Подготовка и выполнение SQL-запроса для удаления заказа
            String query = "DELETE FROM `Orders` WHERE `Orders`.`id` = ?;";
            PreparedStatement statement =
this.connection.prepareStatement(query);
            // Установка значения параметра запроса
            statement.setInt(1, id);
            // Выполнение SQL запроса и получение результата
            boolean result = statement.execute();
            // Закрытие PreparedStatement
            statement.close();
            // Возвращение результата операции
            return result;
        } catch (Exception error) {
            // Вывод информации об ошибке
            error.printStackTrace();
            // Возвращение false в случае ошибки
            return false;

        }
    }
}
```

SalonchikController

С помощью этого интерфейса я объявляю новый тип данных. Я создала свой тип данных с тремя полями. В обычном классе типы с реализацией, а я сделала просто типы. Необходимо для того, чтобы классы умели работать и являлись таким типом данных.

```
package com.example.salonchik;

import javafx.collections.ObservableList;
import javafx.scene.control.TableView;

public interface SalonchikController {

    // Метод retrieveUserOrders получает список заказов пользователя из базы
данных
    public ObservableList retrieveUserOrders(int userId);

    // Метод getTableView возвращает таблицу заказов пользователя
    public TableView getTableView();
}
```

DatabaseOrder

Класс DatabaseOrder представляет собой модель данных для заказа в базе данных.

В этом классе определены поля, которые соответствуют атрибутам заказа, такие как id, user, date, allergie, criteria, service, descriptionM, price, resource, descriptionR, time, booked, master. Эти поля представлены в виде объектов SimpleIntegerProperty, SimpleStringProperty и SimpleFloatProperty из JavaFX, что позволяет привязать их к пользовательскому интерфейсу и следить за изменениями этих свойств.

Конструктор DatabaseOrder() инициализирует все эти свойства.

Также в классе определены геттеры и сеттеры для каждого свойства. Геттеры используются для получения значения свойства, а сеттеры - для установки значения свойства.

Метод toString() возвращает строковое представление объекта DatabaseOrder, что может быть полезно для отладки или для вывода информации о заказе.

В общем, этот класс служит для представления и обработки данных заказа в приложении.

```
package com.example.salonchik;

import javafx.beans.property.SimpleBooleanProperty;
import javafx.beans.property.SimpleStringProperty;
import javafx.beans.property.SimpleIntegerProperty;
import javafx.beans.property.SimpleFloatProperty;

public class DatabaseOrder {


    // Объекты свойств, которые представляют атрибуты заказа
    public SimpleIntegerProperty id;
    public SimpleIntegerProperty user;
    public SimpleStringProperty date;
    public SimpleStringProperty allergie;
    public SimpleStringProperty criteria;
    public SimpleStringProperty service;
    public SimpleStringProperty descriptionM;
    public SimpleFloatProperty price;
    public SimpleStringProperty resource;
    public SimpleStringProperty descriptionR;
    public SimpleStringProperty time;
    public SimpleStringProperty booked;
    public SimpleStringProperty master;


    // Конструктор, который инициализирует все свойства
    public DatabaseOrder() {
       this.id = new SimpleIntegerProperty();
       this.user = new SimpleIntegerProperty();
       this.date = new SimpleStringProperty();
       this.allergie = new SimpleStringProperty();
       this.criteria = new SimpleStringProperty();
       this.service = new SimpleStringProperty();
       this.descriptionM = new SimpleStringProperty();
       this.price = new SimpleFloatProperty();
       this.resource = new SimpleStringProperty();
       this.descriptionR = new SimpleStringProperty();
       this.time = new SimpleStringProperty();
       this.booked = new SimpleStringProperty();
       this.master = new SimpleStringProperty();
    }


    // Методы get и set для каждого свойства
    // Эти методы используются для получения и установки значений свойств
    // Методы property возвращают объект свойства, который может быть
использован для привязки к пользовательскому интерфейсу
    public int getId() {
       return id.get();
    }

    public SimpleIntegerProperty idProperty() {
       return id;
    }

    public void setId(int id) {
       this.id.set(id);
    }

    public int getUser() {
       return user.get();
```

```
}

public SimpleIntegerProperty userProperty() {
   return user;
}

public void setUser(int user) {
   this.user.set(user);
}

public String getDate() {
   return date.get();
}

public SimpleStringProperty dateProperty() {
   return date;
}

public void setDate(String date) {
   this.date.set(date);
}

public String getAllergie() {
   return allergie.get();
}

public SimpleStringProperty allergieProperty() {
   return allergie;
}

public void setAllergie(String allergie) {
   this.allergie.set(allergie);
}

public String getCriteria() {
   return criteria.get();
}

public SimpleStringProperty criteriaProperty() {
   return criteria;
}

public void setCriteria(String criteria) {
   this.criteria.set(criteria);
}

public String getService() {
   return service.get();
}

public SimpleStringProperty serviceProperty() {
   return service;
}

public void setService(String service) {
   this.service.set(service);
}

public String getDescriptionM() {
   return descriptionM.get();
}

public SimpleStringProperty descriptionMProperty() {
```

```
    return descriptionM;
}

public void setDescriptionM(String descriptionM) {
    this.descriptionM.set(descriptionM);
}

public float getPrice() {
    return price.get();
}

public SimpleFloatProperty priceProperty() {
    return price;
}

public void setPrice(float price) {
    this.price.set(price);
}

public String getResource() {
    return resource.get();
}

public SimpleStringProperty resourceProperty() {
    return resource;
}

public void setResource(String resource) {
    this.resource.set(resource);
}

public String getDescriptionR() {
    return descriptionR.get();
}

public SimpleStringProperty descriptionRProperty() {
    return descriptionR;
}

public void setDescriptionR(String descriptionR) {
    this.descriptionR.set(descriptionR);
}

public String getTime() {
    return time.get();
}

public SimpleStringProperty timeProperty() {
    return time;
}

public void setTime(String time) {
    this.time.set(time);
}

public String getBooked() {
    return booked.get();
}

public SimpleStringProperty bookedProperty() {
    return booked;
}
```

```
    public void setBooked(String booked) {
        this.booked.set(booked);
    }

    public String getMaster() {
        return master.get();
    }

    public SimpleStringProperty masterProperty() {
        return master;
    }

    public void setMaster(String master) {
        this.master.set(master);
    }


    // Метод toString возвращает строковое представление объекта
DatabaseOrder
    // Это может быть полезно для отладки или для вывода информации о заказе
    @Override
    public String toString() {
        return "DatabaseOrder{" +
                "id=" + id +
                ", user=" + user +
                ", date=" + date +
                ", allergie=" + allergie +
                ", criteria=" + criteria +
                ", service=" + service +
                ", descriptionM=" + descriptionM +
                ", price=" + price +
                ", resource=" + resource +
                ", descriptionR=" + descriptionR +
                ", time=" + time +
                ", booked=" + booked +
                ", master=" + master +
                '}';
    }
}
```

DatabaseUser

Класс DatabaseUser представляет собой модель данных для пользователя в базе данных.

Все эти атрибуты являются публичными и могут быть доступны для чтения и записи из любого места в коде.

Этот класс используется для представления и работы с информацией о пользователях в базе данных. Он может быть использован для создания новых пользователей, обновления информации о существующих пользователях, а также для получения информации о пользователях для различных целей, таких как аутентификация и авторизация пользователей.

```
package com.example.salonchik;

public class DatabaseUser {

    // Атрибут idK представляет уникальный идентификатор пользователя в базе
данных
    public int idK = 0;

    // Атрибут name представляет имя пользователя
    public String name = null;

    // Атрибут role представляет роль пользователя (например, "admin",
"user")
    public String role = null;

    // Атрибут login представляет логин пользователя для входа в систему
    public String login = null;

    // Атрибут password представляет пароль пользователя для входа в систему
    public String password = null;

    @Override
    public String toString() {
       return "DatabaseUser{" +
             "idK=" + idK +
             ", name='" + name + '\'' +
             ", role='" + role + '\'' +
             ", login='" + login + '\'' +
             ", password='" + password + '\'' +
             '}';
    }
}
```

SalonchikApp

Этот класс используется для управления информацией о текущем пользователе в системе. Он позволяет устанавливать текущего пользователя и получать его идентификатор. Это может быть полезно для различных целей, таких как аутентификация и авторизация пользователей, а также для отслеживания действий текущего пользователя в системе.

```
package com.example.salonchik;

public class SalonchikApp {

    // Атрибут loggedInUser представляет пользователя, который в настоящее
время вошел в систему
    private static DatabaseUser loggedInUser;

    // Метод client устанавливает пользователя как текущего пользователя,
который вошел в систему
    public static void client(DatabaseUser user) {
        loggedInUser = user;
    }
```

```
    // Метод employee устанавливает пользователя как текущего пользователя,
который вошел в систему
    public static void employee(DatabaseUser user) {
        loggedInUser = user;
    }

    // Метод getLoggedInUserId возвращает идентификатор текущего
пользователя, который вошел в систему
    // Если пользователь не вошел в систему, метод возвращает -1
    public static int getLoggedInUserId() {
        return loggedInUser != null ? loggedInUser.idK : -1;
    }
}
```

SalonchikApplication

Класс SalonchikApplication представляет собой основу для JavaFX-приложения "Salonchik". Этот класс расширяет Application, который является основным классом для всех JavaFX-приложений.

Отвечает за запуск приложения и открытие различных окон в зависимости от действий пользователя. Он использует JavaFX для создания графического пользовательского интерфейса и FXML для определения структуры интерфейса.

```
package com.example.salonchik;

import javafx.application.Application;
import javafx.fxml.FXMLLoader;
import javafx.scene.Scene;
import javafx.scene.image.Image;
import javafx.stage.Modality;
import javafx.stage.Stage;
import java.io.IOException;

public class SalonchikApplication extends Application {


    // Метод main запускает приложение
    public static void main(String[] args) {
        launch();
    }


    // Метод vybor открывает окно выбора пользователя
    public static void vybor() throws IOException {
        Stage stage = new Stage();
        stage.getIcons().add(new Image("/photo/icon.png"));
        FXMLLoader loader = new
FXMLLoader(SalonchikApplication.class.getResource("salonchik-vybor.fxml"));
        Scene scene = new Scene(loader.load(), 400, 400);
        stage.setTitle("Пользователь");
        stage.setScene(scene);
        stage.show();
    }
```

```
    // Метод vybr открывает окно создания нового пользователя
    public static void vybr(SalonchikVybor.User user) throws IOException {
        Stage stage = new Stage();
        stage.getIcons().add(new Image("/photo/icon.png"));
        FXMLLoader loader = new
FXMLLoader(SalonchikApplication.class.getResource("salonchik-
sozdanie.fxml"));
        Scene scene = new Scene(loader.load(), 600, 400);
        stage.setTitle("Пользователь");
        stage.setScene(scene);
        stage.show();

        SalonchikSozdanie controller = loader.getController();
        controller.fillSozdanieData(stage, user.getId());
    }


    // Метод start запускает приложение и открывает окно авторизации
    @Override
    public void start(Stage stage) throws IOException {
        stage.getIcons().add(new Image("/photo/icon.png"));
        FXMLLoader loader = new
FXMLLoader(SalonchikApplication.class.getResource("salonchik-
authorization.fxml"));
        Scene scene = new Scene(loader.load(), 320, 480);
        stage.setTitle("Авторизация");
        stage.setScene(scene);
        stage.show();
    }

    // Метод client открывает окно клиента
    public static void client(DatabaseUser user) throws IOException {
        FXMLLoader loader = new
FXMLLoader(SalonchikApplication.class.getResource("salonchik-client.fxml"));
        Scene scene = new Scene(loader.load(), 640, 480);
        Stage stage = new Stage();
        stage.getIcons().add(new Image("/photo/icon.png"));
        stage.initModality(Modality.WINDOW_MODAL);
        stage.setTitle("Клиент");
        stage.setScene(scene);
        stage.show();

       SalonchikClient controller = loader.getController();
       controller.fillUserData(user, stage);
    }


    // Метод employee открывает окно сотрудника
    public static void employee(DatabaseUser user) throws IOException {

        FXMLLoader loader = new
FXMLLoader(SalonchikApplication.class.getResource("salonchik-
employee.fxml"));
        Scene scene = new Scene(loader.load(), 640, 480);
        Stage stage = new Stage();
        stage.getIcons().add(new Image("/photo/icon.png"));
        stage.initModality(Modality.WINDOW_MODAL);
        stage.setTitle("Сотрудник");
        stage.setScene(scene);
        stage.show();

       SalonchikEmployee controller = loader.getController();
```

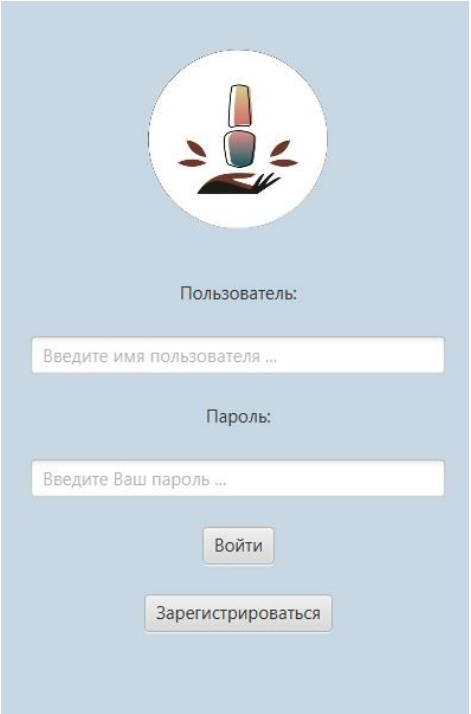
```
        controller.fillUserData(user, stage);
    }


    // Метод order открывает окно заказа
    public static void order(DatabaseOrder order, String role,
SalonchikController view) throws IOException {

        FXMLLoader loader = new
FXMLLoader(SalonchikApplication.class.getResource("salonchik-order.fxml"));
        Scene scene = new Scene(loader.load(), 640, 600);
        Stage stage = new Stage();
        stage.getIcons().add(new Image("/photo/icon.png"));
        stage.initModality(Modality.WINDOW_MODAL);
        stage.setTitle("Заказ");
        stage.setScene(scene);
        stage.show();

        SalonchikOrder controller = loader.getController();
        controller.fillOrderData(order, stage, role, view);
    }


    // Метод sozdanie открывает окно создания нового заказа
    public static void sozdanie(int userId) throws IOException {

        FXMLLoader loader = new
FXMLLoader(SalonchikApplication.class.getResource("salonchik-
sozdanie.fxml"));
        Scene scene = new Scene(loader.load(), 600, 400);
        Stage stage = new Stage();
        stage.getIcons().add(new Image("/photo/icon.png"));
        stage.initModality(Modality.WINDOW_MODAL);
        stage.setTitle("Новый заказ");
        stage.setScene(scene);
        stage.show();

        SalonchikSozdanie controller = loader.getController();
        controller.fillSozdanieData(stage, userId);
    }


    // Метод registration открывает окно регистрации
    public static void registration() throws IOException {

        FXMLLoader loader = new
FXMLLoader(SalonchikApplication.class.getResource("salonchik-
registration.fxml"));
        Scene scene = new Scene(loader.load(), 320, 520);
        Stage stage = new Stage();
        stage.getIcons().add(new Image("/photo/icon.png"));
        stage.initModality(Modality.WINDOW_MODAL);
        stage.setTitle("Регистрация");
        stage.setScene(scene);
        stage.show();

    }
}
```

3.2. Классы интерфейсов

3.2.1. Окно авторизации пользователя

salonchik-autorization

Окно для авторизации пользователя.



SalonchikAutorization

Этот класс отвечает за обработку процесса авторизации и регистрации пользователя в приложении "Salonchik". Он использует JavaFX для создания графического пользовательского интерфейса и взаимодействует с базой данных для проверки логина и пароля пользователя.

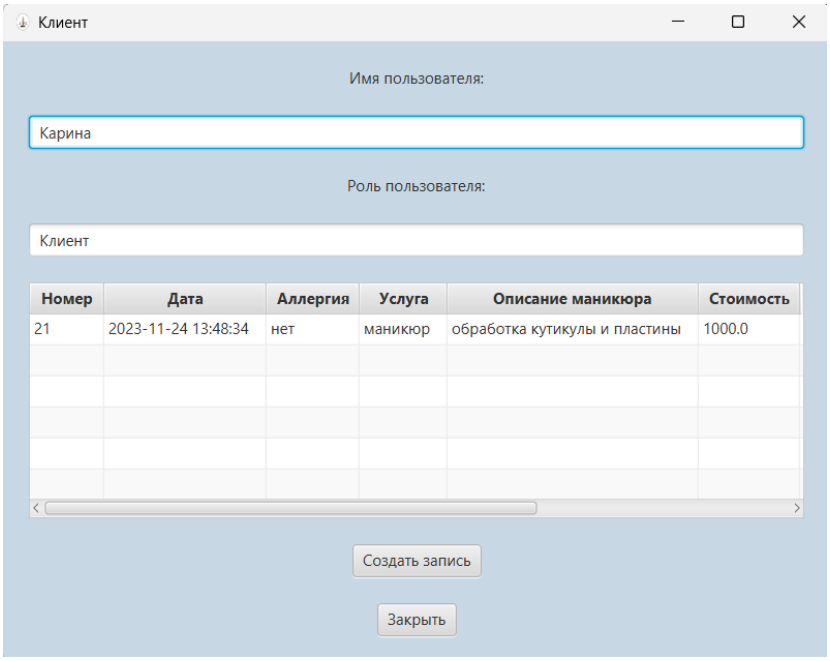
```
package com.example.salonchik;

import javafx.event.ActionEvent;
import javafx.fxml.FXML;
import javafx.scene.control.Label;
import javafx.scene.control.PasswordField;
import javafx.scene.control.TextField;

public class SalonchikAuthorization {


    // Атрибут database представляет контроллер базы данных
    private  DatabaseController database = new DatabaseController();


    // Атрибуты userLogin, userPassword и userResult связаны с элементами
пользовательского интерфейса
```

```
    @FXML
    private TextField userLogin;
    @FXML
    private PasswordField userPassword;
    @FXML
    private Label userResult;


    // Метод onAuthorizationButtonClick обрабатывает событие нажатия кнопки
авторизации
    @FXML
    protected void onAuthorizationButtonClick() {
        // Очищаем текстовое поле результата авторизации
        this.userResult.setText("");
        // Попытка подключения к базе данных
        try {
            this.database.connect();
        } catch (Exception error) {
            // Если подключение не удалось, выводим сообщение об ошибке и
прерываем выполнение метода
            this.userResult.setText("Невозможно подключиться к базе
данных!");
            return;
        }
        // Попытка авторизации пользователя
        try {
            // Получаем объект пользователя из базы данных
            DatabaseUser user =
this.database.authorization(userLogin.getText(), userPassword.getText());
            // Если пользователь найден и его роль "Клиент", открываем окно
клиента
            // Если пользователь найден и его роль не "Клиент", открываем
окно сотрудника
            if (user != null) {
                if (user.role.equals("Клиент")) {
                SalonchikApplication.client(user);
             } else {
                SalonchikApplication.employee(user);
             }
             return;
            }
            // Если пользователь не найден, выводим сообщение об ошибке
            throw new NullPointerException();
        } catch (Exception error) {
            this.userResult.setText("Пользователь не найден!");
        } finally {
            // В любом случае, после выполнения попытки авторизации,
отключаемся от базы данных
          this.database.disconnect();
       }
    }


    // Метод onRegistrationButtonClick обрабатывает событие нажатия кнопки
регистрации
    public void onRegistrationButtonClick(ActionEvent actionEvent) {
        try {
            SalonchikApplication.registration();
        } catch (Exception error) {
            error.printStackTrace();
        }
    }
}
```

3.2.2. Окно отображения записей клиента

salonchik-client

Окно для вывода информации о том, кто авторизирован и его заказы (если они есть, то в окне вместо «У вас нет записей!» будут отображать строки с записями).



SalonchikClient

Класс SalonchikClient реализует интерфейс SalonchikController и представляет собой контроллер для клиентской части приложения "Salonchik".

Этот класс отвечает за управление пользовательским интерфейсом и взаимодействие с базой данных для клиентской части приложения "Salonchik".

```
package com.example.salonchik;

import javafx.collections.FXCollections;
import javafx.collections.ObservableList;
import javafx.event.ActionEvent;
import javafx.fxml.FXML;
import javafx.scene.control.Label;
import javafx.scene.control.TableColumn;
import javafx.scene.control.TableView;
import javafx.scene.control.TextField;
import javafx.scene.control.cell.PropertyValueFactory;
import javafx.stage.Stage;

import java.util.ArrayList;
import java.util.List;

public class SalonchikClient implements SalonchikController {
```

```
    // Атрибут database представляет контроллер базы данных
    private  DatabaseController database = new DatabaseController();

    // Атрибут stage представляет текущее окно приложения
    private  Stage stage;

    // Атрибут userId хранит идентификатор текущего пользователя
    private int userId;


    // Атрибуты userName, userRole и userOrders связаны с элементами
пользовательского интерфейса
    @FXML
    private TextField userName;
    @FXML
    private TextField userRole;
    @FXML
    private TableView userOrders;


    // Метод onExitButtonClick обрабатывает событие нажатия кнопки выхода
    @FXML
    protected void onExitButtonClick() {
        stage.close();
    }


    // Метод onTableViewSelect обрабатывает событие выбора элемента в таблице
    @FXML
    protected void onTableViewSelect() {
        try {
            // Получаем выбранный элемент из таблицы и приводим его к типу
DatabaseOrder
            DatabaseOrder order = (DatabaseOrder)
userOrders.getSelectionModel().getSelectedItem();
            // Открываем окно заказа с выбранным элементом
            SalonchikApplication.order(order, "Клиент", this);
        } catch (Exception error) {
            // Если произошла ошибка, выводим ее в консоль
            error.printStackTrace();
        }

    }


    // Метод getTableView возвращает таблицу заказов пользователя
    @Override
    public TableView getTableView(){
        return this.userOrders;
    }


    // Метод fillUserData заполняет данные пользователя в пользовательском
интерфейсе
    public void fillUserData(DatabaseUser user, Stage stage) {
        // Устанавливаем ссылку на Stage
        this.stage = stage;
        // Устанавливаем идентификатор пользователя
       this.userId = user.idK;
        // Заполняем поля имени пользователя и его роли
        userName.setText(user.name);
       userRole.setText(user.role);
        // Получаем заказы пользователя и заполняем таблицу заказов
```

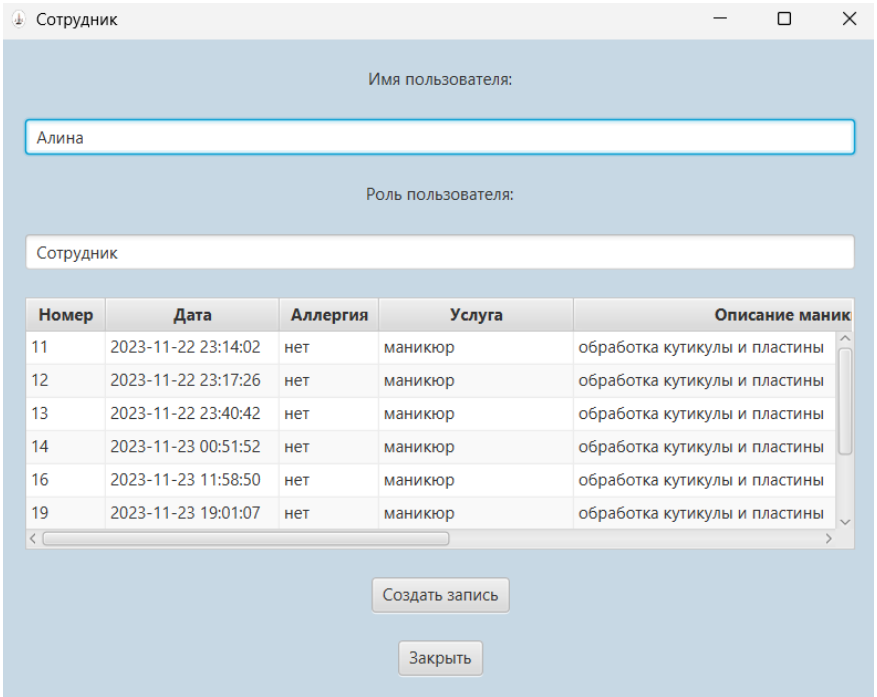
```
        userOrders.setItems(retrieveUserOrders(user.idK));
         // Создание и настройка колонок таблицы
         TableColumn idColumn = new TableColumn("Номер");
         idColumn.setCellValueFactory(new PropertyValueFactory("id"));
         TableColumn dateColumn = new TableColumn("Дата");
         dateColumn.setCellValueFactory(new PropertyValueFactory("date"));
         TableColumn allergieColumn = new TableColumn("Аллергия");
         allergieColumn.setCellValueFactory(new
PropertyValueFactory("allergie"));
         TableColumn serviceColumn = new TableColumn("Услуга");
         serviceColumn.setCellValueFactory(new
PropertyValueFactory("service"));
         TableColumn descriptionMColumn = new TableColumn("Описание
маникюра");
         descriptionMColumn.setCellValueFactory(new
PropertyValueFactory("descriptionM"));
         TableColumn priceColumn = new TableColumn("Стоимость");
         priceColumn.setCellValueFactory(new PropertyValueFactory("price"));
         TableColumn timeColumn = new TableColumn("Время записи");
         timeColumn.setCellValueFactory(new PropertyValueFactory("time"));
         TableColumn bookedColumn = new TableColumn("Статус");
         bookedColumn.setCellValueFactory(new PropertyValueFactory("booked"));
         TableColumn masterColumn = new TableColumn("Мастер");
         masterColumn.setCellValueFactory(new PropertyValueFactory("master"));
         // Устанавливаем все колонки таблицы
         userOrders.getColumns().setAll(idColumn, dateColumn, allergieColumn,
serviceColumn, descriptionMColumn, priceColumn, timeColumn, bookedColumn,
masterColumn);
         // Обновляем таблицу
         userOrders.refresh();
    }



    // Метод retrieveUserOrders получает список заказов пользователя из базы
данных
    @Override
    public ObservableList retrieveUserOrders(int userId) {
         // Попытка подключения к базе данных
        try {
             this.database.connect();
         } catch (Exception error) {
             // Если подключение не удалось, выводим ошибку в консоль и
возвращаем null
             error.printStackTrace();
             return null;
         }
        try {
             // Попытка получения заказов пользователя из базы данных
             if (userId > 0) {
                 // Получаем список заказов пользователя и преобразуем его в
ObservableList
              List<DatabaseOrder> orders = this.database.clientOrders(userId);
              return FXCollections.observableList(orders);
             }
             // Если идентификатор пользователя меньше или равен 0,
выбрасываем исключение
             throw new NullPointerException("Невозможно получить список
заказов для данного пользователя!");
         } catch (Exception error) {
             // Если произошла ошибка, выводим ее в консоль и возвращаем null
             error.printStackTrace();
           return null;
```

```
        } finally {
            // В любом случае, после выполнения попытки получения заказов,
отключаемся от базы данных
            this.database.disconnect();
        }

    }


    // Метод onCreateButtonClick обрабатывает событие нажатия кнопки создания
нового заказа
    public void onCreateButtonClick(ActionEvent actionEvent) {
        // Попытка открытия окна создания нового заказа для текущего
пользователя
        try {
            SalonchikApplication.sozdanie(this.userId);
        } catch (Exception error) {
            // Если произошла ошибка, выводим ее в консоль
            error.printStackTrace();
        }
    }
}
```

3.2.3. Окно всех заказов

salonchik-employee

Окно для вывода информации о том, кто авторизирован и записи каждого сотрудника (если они есть, то в окне вместо «Нет ни одной записи!» будут отображать строки с записями).



SalonchikEmployee

Этот класс отвечает за управление данными сотрудника, включая заполнение информации о сотруднике, получение списка его заказов из базы данных и обработку событий пользовательского интерфейса, таких как выбор элемента в таблице или нажатие кнопки создания нового заказа.

```
package com.example.salonchik;

import javafx.collections.FXCollections;
import javafx.collections.ObservableList;
import javafx.event.ActionEvent;
import javafx.fxml.FXML;
import javafx.scene.control.Label;
import javafx.scene.control.TableColumn;
import javafx.scene.control.TableView;
import javafx.scene.control.TextField;
import javafx.scene.control.cell.PropertyValueFactory;
import javafx.stage.Stage;

import java.util.List;

public class SalonchikEmployee implements SalonchikController {


    // Атрибут database представляет контроллер базы данных
    private  DatabaseController database = new DatabaseController();

    // Атрибут stage представляет текущее окно приложения
    private  Stage stage;

    // Атрибут userId хранит идентификатор текущего пользователя
    private int userId;


    // Атрибуты userName, userRole и userOrders связаны с элементами
пользовательского интерфейса
    @FXML
    private TextField userName;
    @FXML
    private TextField userRole;
    @FXML
    private TableView userOrders;


    // Метод onExitButtonClick обрабатывает событие нажатия кнопки выхода
    @FXML
    protected void onExitButtonClick() {
        stage.close();
    }


    // Метод onTableViewSelect обрабатывает событие выбора элемента в таблице
    @FXML
    protected void onTableViewSelect() {
        // Попытка получения выбранного элемента из таблицы и его приведения
к типу DatabaseOrder
        try {
            DatabaseOrder order = (DatabaseOrder)
userOrders.getSelectionModel().getSelectedItem();
```

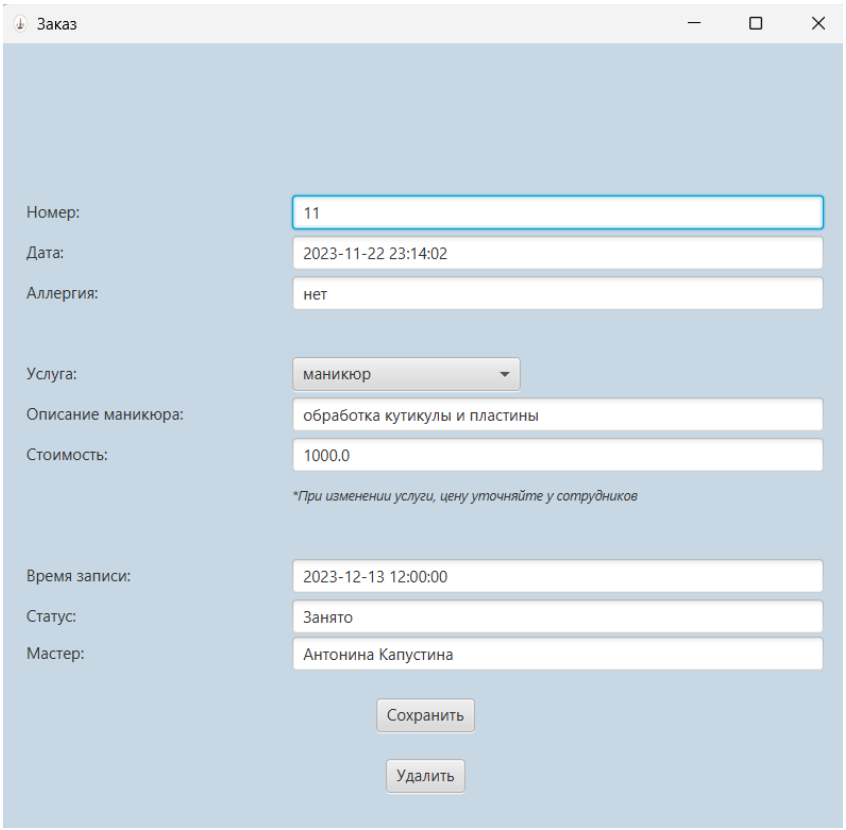
```
            // Открываем окно заказа с выбранным элементом
            SalonchikApplication.order(order, "Сотрудник", this);
        } catch (Exception error) {
            // Если произошла ошибка, выводим ее в консоль
            error.printStackTrace();
        }

    }


    // Метод getTableView возвращает таблицу заказов сотрудника
    @Override
    public TableView getTableView(){
        return this.userOrders;
    }


    // Метод fillUserData заполняет данные сотрудника в пользовательском
интерфейсе
    public void fillUserData(DatabaseUser user, Stage stage) {
        // Устанавливаем ссылку на Stage
        this.stage = stage;
        // Устанавливаем идентификатор пользователя
        this.userId = user.idK;
        // Заполняем поля имени пользователя и его роли
        userName.setText(user.name);
        userRole.setText(user.role);
        // Получаем заказы пользователя и заполняем таблицу заказов
        userOrders.setItems(retrieveUserOrders(user.idK));
        // Создание и настройка колонок таблицы
        TableColumn idColumn = new TableColumn("Номер");
        idColumn.setCellValueFactory(new PropertyValueFactory("id"));
        TableColumn dateColumn = new TableColumn("Дата");
        dateColumn.setCellValueFactory(new PropertyValueFactory("date"));
        TableColumn allergieColumn = new TableColumn("Аллергия");
        allergieColumn.setCellValueFactory(new
PropertyValueFactory("allergie"));

        TableColumn serviceColumn = new TableColumn("Услуга");
        serviceColumn.setCellValueFactory(new
PropertyValueFactory("service"));
        TableColumn descriptionMColumn = new TableColumn("Описание
маникюра");
        descriptionMColumn.setCellValueFactory(new
PropertyValueFactory("descriptionM"));
        TableColumn priceColumn = new TableColumn("Стоимость");
        priceColumn.setCellValueFactory(new PropertyValueFactory("price"));

        TableColumn timeColumn = new TableColumn("Время записи");
        timeColumn.setCellValueFactory(new PropertyValueFactory("time"));
        TableColumn bookedColumn = new TableColumn("Статус");
        bookedColumn.setCellValueFactory(new PropertyValueFactory("booked"));
        TableColumn masterColumn = new TableColumn("Мастер");
        masterColumn.setCellValueFactory(new PropertyValueFactory("master"));
        // Устанавливаем все колонки таблицы
        userOrders.getColumns().setAll(idColumn, dateColumn, allergieColumn,
serviceColumn, descriptionMColumn, priceColumn, timeColumn, bookedColumn,
masterColumn);
        // Обновляем таблицу
        userOrders.refresh();
    }
```

```
    // Метод retrieveUserOrders получает список заказов сотрудника из базы
данных
    @Override
    public ObservableList retrieveUserOrders(int userId) {
        // Попытка подключения к базе данных
        try {
            this.database.connect();
        } catch (Exception error) {
            // Если подключение не удалось, выводим ошибку в консоль и
возвращаем null
            error.printStackTrace();
            return null;
        }
        // Попытка получения заказов сотрудника из базы данных
        try {
                List<DatabaseOrder> orders = this.database.employeeOrders();
                // Получаем список заказов сотрудника и преобразуем его в
ObservableList
                return FXCollections.observableList(orders);
        } catch (Exception error) {
            // Если произошла ошибка, выводим ее в консоль и возвращаем null
            error.printStackTrace();
            return null;
        } finally {
            // В любом случае, после выполнения попытки получения заказов,
отключаемся от базы данных
            this.database.disconnect();
        }

    }


    // Метод onCreateButtonClick обрабатывает событие нажатия кнопки создания
нового заказа
    public void onCreateButtonClick(ActionEvent actionEvent) {
        // Попытка открыть окно выбора
        try {
            SalonchikApplication.vybor();
        } catch (Exception error) {
            // Если произошла ошибка, выводим ее в консоль
            error.printStackTrace();
        }
    }
}
```

3.2.4. Окно редактирования заказа

salonchik-order

Окно, в котором поля заполняются данными уже созданного заказа для их редактирования или удаления записи.



SalonchikOrder

Класс SalonchikOrder является центральной частью приложения, которая управляет функциями, связанными с заказами, взаимодействует с базой данных и обрабатывает события пользовательского интерфейса.

```
package com.example.salonchik;

import javafx.collections.FXCollections;
import javafx.collections.ObservableList;
import javafx.event.ActionEvent;
import javafx.fxml.FXML;
import javafx.scene.control.*;
import javafx.scene.control.cell.PropertyValueFactory;
import javafx.stage.Stage;

import java.util.List;

public class SalonchikOrder {

    // Атрибут database представляет контроллер базы данных
    private  DatabaseController database = new DatabaseController();
```

```
    // Атрибут stage представляет текущее окно приложения
    private  Stage stage;

    // Атрибут order представляет заказ
    private DatabaseOrder order;

    // Атрибут view представляет контроллер пользовательского интерфейса
    private SalonchikController view;


    // Атрибуты user, numb, date, allerg, service, desM, price, time, stat,
master связаны с элементами пользовательского интерфейса
    @FXML
    private TextField user;
    @FXML
    private TextField numb;
    @FXML
    private TextField date;
    @FXML
    private TextField allerg;
    @FXML
    private ComboBox service;
    @FXML
    private TextField desM;
    @FXML
    private TextField price;
    @FXML
    private TextField time;
    @FXML
    private TextField stat;
    @FXML
    private TextField master;


    // Метод onExitButtonClick обрабатывает событие нажатия кнопки выхода
    @FXML
    protected void onExitButtonClick() {
        stage.close();
    }


    // Метод fillOrderData заполняет данные заказа в пользовательском
интерфейсе
    public void fillOrderData(DatabaseOrder order, Stage stage, String role,
SalonchikController view) {
        // Устанавливаем ссылку на Stage и объект заказа
        this.stage = stage;
        this.order = order;
        this.view = view;
        // Заполняем поля данных заказа
        this.user.setText(String.valueOf(order.getUser()));
        this.date.setText(order.getDate());
        this.numb.setText(String.valueOf(order.getId()));
        this.allerg.setText(order.getAllergie());
        this.service.setItems(retrieveServices("name"));
        this.service.setValue(order.getService());
        this.desM.setText(order.getDescriptionM());
        this.price.setText(String.valueOf(order.getPrice()));
        this.time.setText(order.getTime());
        this.stat.setText(order.getBooked());
        this.master.setText(order.getMaster());
    }
```

```
    // Метод retrieveMasters получает список мастеров из базы данных
    private ObservableList retrieveMasters() {
       try {
            // Подключение к базе данных
            this.database.connect();
        } catch (Exception error) {
            // Если возникнет ошибка при подключении к базе данных, выводим
ошибку и возвращаем null
            error.printStackTrace();
            return null;
        }
       try {
            // Получение списка мастеров из базы данных
            List<String> masters = this.database.masters();
            // Преобразование списка в ObservableList
            return FXCollections.observableList(masters);
        } catch (Exception error) {
            // Если возникнет ошибка при получении списка мастеров, выводим
ошибку и возвращаем null
            error.printStackTrace();
          return null;
        } finally {
            // Независимо от того, возникла ли ошибка, отключаемся от базы
данных
          this.database.disconnect();
       }

    }


    // Метод retrieveServices получает список услуг из базы данных
    private ObservableList retrieveServices(String field) {
        try {
            // Подключение к базе данных
            this.database.connect();
        } catch (Exception error) {
            // Если возникнет ошибка при подключении к базе данных, выводим
ошибку и возвращаем null
            error.printStackTrace();
            return null;
        }
        try {
            // Получение списка услуг из базы данных
            List<String> services = this.database.services(field);
            // Преобразование списка в ObservableList
            return FXCollections.observableList(services);
        } catch (Exception error) {
            // Если возникнет ошибка при подключении к базе данных, выводим
ошибку и возвращаем null
            error.printStackTrace();
            return null;
        } finally {
            // Независимо от того, возникла ли ошибка, отключаемся от базы
данных
            this.database.disconnect();
        }

    }


    // Метод retrieveSchedule получает расписание из базы данных
```

```
    private ObservableList retrieveSchedule(String field) {
        try {
            // Подключение к базе данных
            this.database.connect();
        } catch (Exception error) {
            // Если возникнет ошибка при подключении к базе данных, выводим
ошибку и возвращаем null
            error.printStackTrace();
            return null;
        }
        try {
            // Получение расписания из базы данных
            List<String> schedule = this.database.schedule(field);
            // Преобразование расписания в ObservableList
            return FXCollections.observableList(schedule);
        } catch (Exception error) {
            // Если возникнет ошибка при получении расписания, выводим ошибку
и возвращаем null
            error.printStackTrace();
            return null;
        } finally {
            // Независимо от того, возникла ли ошибка, отключаемся от базы
данных
            this.database.disconnect();
        }

    }


    // Метод onCreateButtonClick обрабатывает событие нажатия кнопки создания
нового заказа
    public void onCreateButtonClick(ActionEvent actionEvent) {
        // Создание нового объекта DatabaseOrder
        DatabaseOrder formData = new DatabaseOrder();
        // Заполнение полей объекта данными из формы
        formData.setId(Integer.parseInt(numb.getText()));
        formData.setUser(Integer.parseInt(user.getText()));
        formData.setDate(date.getText());
        formData.setAllergie(allerg.getText());
        formData.setService(String.valueOf(service.getValue()));
        formData.setDescriptionM(String.valueOf(desM.getText()));
        formData.setPrice(Float.parseFloat(String.valueOf(price.getText())));
        formData.setTime(String.valueOf(time.getText()));
        formData.setBooked(String.valueOf(stat.getText()));
        formData.setMaster(String.valueOf(master.getText()));
        try {
            // Подключение к базе данных
            this.database.connect();
        } catch (Exception error) {
            // Если возникнет ошибка при подключении к базе данных, выводим
ошибку и возвращаем
            error.printStackTrace();
            return;
        }
        try {
            // Создание нового заказа в базе данных
            this.database.create(formData);
            // Обновление таблицы заказов в пользовательском интерфейсе

this.view.getTableView().setItems(this.view.retrieveUserOrders(formData.getUs
er()));
            this.view.getTableView().refresh();
            // Закрытие текущего окна
```

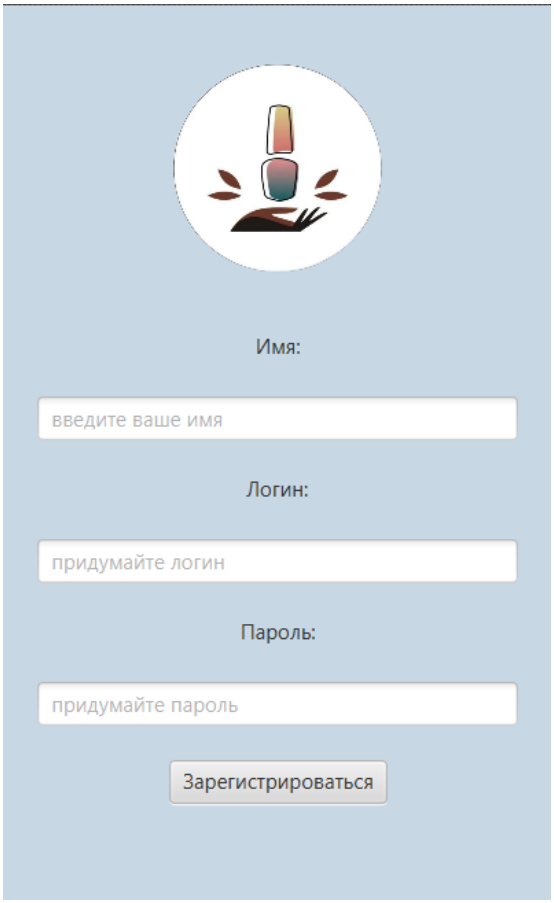
```
            this.stage.close();
        } catch (Exception error) {
            // Если возникнет ошибка при создании заказа, выводим ошибку и
закрываем окно
            error.printStackTrace();
            this.stage.close();
        } finally {
            // Независимо от того, возникла ли ошибка, отключаемся от базы
данных
            this.database.disconnect();
        }
    }


    // Метод onSaveButtonClick обрабатывает событие нажатия кнопки сохранения
заказа
    public void onSaveButtonClick(ActionEvent actionEvent) {
        // Создание нового объекта DatabaseOrder
        DatabaseOrder formData = new DatabaseOrder();
        // Заполнение полей объекта данными из формы
        formData.setId(Integer.parseInt(numb.getText()));
        formData.setUser(Integer.parseInt(user.getText()));
        formData.setDate(date.getText());
        formData.setAllergie(allerg.getText());
        formData.setService(String.valueOf(service.getValue()));
        formData.setDescriptionM(String.valueOf(desM.getText()));
        formData.setPrice(Float.parseFloat(String.valueOf(price.getText())));
        formData.setTime(String.valueOf(time.getText()));
        formData.setBooked(String.valueOf(stat.getText()));
        formData.setMaster(String.valueOf(master.getText()));
        try {
            // Подключение к базе данных
            this.database.connect();
        } catch (Exception error) {
            // Если возникнет ошибка при подключении к базе данных, выводим
ошибку и возвращаем
            error.printStackTrace();
            return;
        }
        try {
            // Обновление заказа в базе данных
            this.database.update(formData);
            // Обновление таблицы заказов в пользовательском интерфейсе

this.view.getTableView().setItems(this.view.retrieveUserOrders(formData.getUs
er()));
            this.view.getTableView().refresh();
            // Закрытие текущего окна
            this.stage.close();
        } catch (Exception error) {
            // Если возникнет ошибка при обновлении заказа, выводим ошибку и
закрываем окно
            error.printStackTrace();
            this.stage.close();
        } finally {
            // Независимо от того, возникла ли ошибка, отключаемся от базы
данных
            this.database.disconnect();
        }
    }


    // Метод onDeleteButtonClick обрабатывает событие нажатия кнопки удаления
```

```
заказа
    public void onDeleteButtonClick(ActionEvent actionEvent) {
        try {
            // Подключение к базе данных
            this.database.connect();
        } catch (Exception error) {
            // Если возникнет ошибка при подключении к базе данных, выводим
ошибку и возвращаем
            error.printStackTrace();
            return;
        }
        try {
            // Удаление заказа из базы данных
            this.database.delete(Integer.parseInt(numb.getText()));
            // Обновление таблицы заказов в пользовательском интерфейсе

this.view.getTableView().setItems(this.view.retrieveUserOrders(Integer.parseI
nt(user.getText())));
            this.view.getTableView().refresh();
            // Закрытие текущего окна
            this.stage.close();
        } catch (Exception error) {
            // Если возникнет ошибка при удалении заказа, выводим ошибку и
закрываем окно
            error.printStackTrace();
            this.stage.close();
        } finally {
            // Независимо от того, возникла ли ошибка, отключаемся от базы
данных
            this.database.disconnect();
        }
    }
}
```

3.2.5. Окно регистрации пользователя

salonchik-registration

Окно для регистрации нового пользователя.



SalonchikRegistration

Этот класс служит для обработки процесса регистрации пользователя, включая ввод данных пользователем, подключение к базе данных и запись данных пользователя в базу данных.

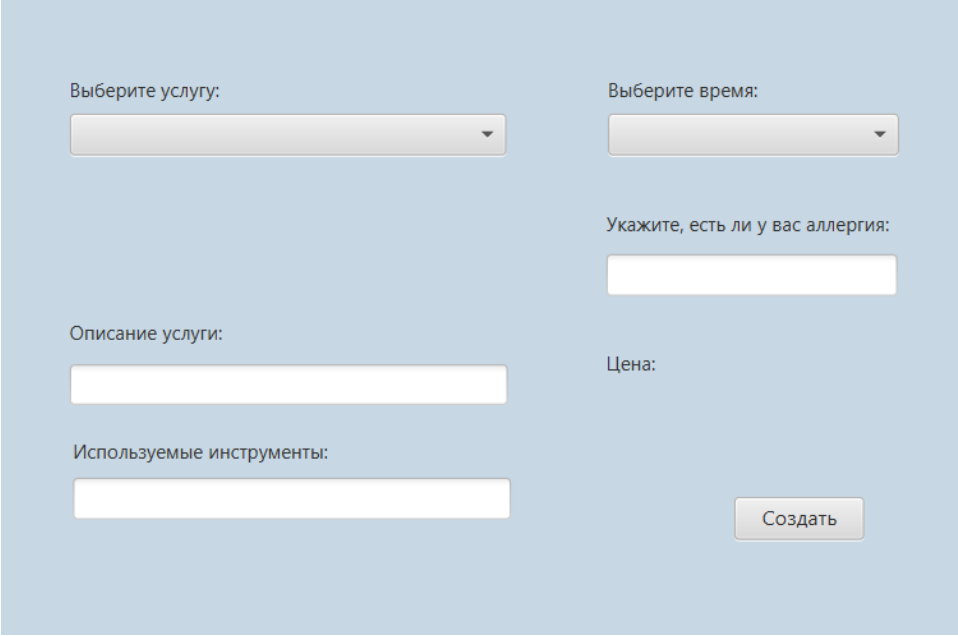
```
package com.example.salonchik;

import javafx.collections.FXCollections;
import javafx.collections.ObservableList;
import javafx.fxml.FXML;
import javafx.scene.control.Label;
import javafx.scene.control.TextField;
import javafx.stage.Stage;

import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.PreparedStatement;
import java.sql.SQLException;
import java.util.List;

public class SalonchikRegistration {
```

```
    // Атрибут database представляет контроллер базы данных
    private DatabaseController database = new DatabaseController();

    // Атрибут stage представляет текущее окно приложения
    private Stage stage;


    // Атрибуты userName, userLogin, userPassword, userResult связаны с
элементами пользовательского интерфейса
    @FXML
    private TextField userName;
    @FXML
    private TextField userLogin;
    @FXML
    private TextField userPassword;
    @FXML
    private Label userResult;


    // Метод onRegButtonClick обрабатывает событие нажатия кнопки регистрации
    @FXML
    private void onRegButtonClick() {
        // Очищаем текстовое поле userResult
        this.userResult.setText("");
        // Пытаемся подключиться к базе данных
        try {
            this.database.connect();
        } catch (Exception error) {
            // Если возникнет ошибка при подключении к базе данных, выводим
сообщение об ошибке
            this.userResult.setText("Невозможно подключиться к базе
данных!");
            return;
        }
        // Пытаемся зарегистрировать нового пользователя в базе данных
        try {
            boolean result = this.database.registration(userName.getText(),
userLogin.getText(), userPassword.getText());
            // Если регистрация прошла успешно, выводим сообщение об успехе
            this.userResult.setText("Регистрация прошла успешно!");
        } catch (Exception error) {
            // Если возникнет ошибка при регистрации, выводим сообщение об
ошибке
            this.userResult.setText("Не получилось зарегистрироваться!");
        } finally {
            // В конце отключаемся от базы данных
            this.database.disconnect();
        }


    }
}
```

3.2.6. Окно создания заказа

salonchik-sozdanie

Окно для создания записи.



SalonchikSozdanie

Этот класс отвечает за взаимодействие пользователя с формой создания заказа и обеспечивает взаимодействие с базой данных для получения и сохранения информации о заказах.

```
package com.example.salonchik;

import javafx.event.ActionEvent;
import javafx.fxml.FXML;
import javafx.scene.control.Button;
import javafx.scene.control.ComboBox;
import javafx.scene.control.Label;
import javafx.scene.control.TextField;
import javafx.event.ActionEvent;
import javafx.stage.Stage;

import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.SQLException;

public class SalonchikSozdanie {

    // Экземпляр Stage, который представляет окно приложения
    private Stage stage;
    // Комбобокс для выбора услуги
    @FXML
    private ComboBox<String> serviceComboBox;
    // Метка для отображения цены услуги
    @FXML
```

```
    private Label idPrice;
    // Текстовое поле для ввода ID пользователя
    @FXML
    private TextField user;
    // Комбобокс для выбора времени
    @FXML
    private ComboBox<String> scheduleComboBox;
    // Текстовое поле для ввода аллергий
    @FXML
    private TextField allergyTextField;
    // Текстовое поле для отображения описания услуги
    @FXML
    private TextField textService;
    // Текстовое поле для отображения ресурсов, связанных с услугой
    @FXML
    private TextField textResource;
    // Кнопка для создания заказа
    @FXML
    private Button idButton;


    // URL базы данных
    private static final String DB_URL = "jdbc:mysql://localhost:3306/Salon";
    // Имя пользователя для подключения к базе данных
    private static final String DB_USER = "root";
    // Пароль пользователя для подключения к базе данных
    private static final String DB_PASSWORD = "";


    // Метод инициализации, который загружает услуги и расписание в
комбобоксы
    @FXML
    public void initialize() {
        // Загрузка услуг в комбобокс
        loadServicesIntoComboBox();
        // Загрузка расписания в комбобокс
        loadScheduleIntoComboBox();
    }


    // Метод для заполнения данных формы создания заказа
    public void fillSozdanieData(Stage stage, int userId) {
        // Установка текущего окна приложения
        this.stage = stage;
        // Установка идентификатора пользователя в текстовое поле
        user.setText(String.valueOf(userId));
    }


    // Метод обработки нажатия кнопки создания заказа
    @FXML
    private void onCreateButtonClick(ActionEvent event) {
        try (Connection connection = DriverManager.getConnection(DB_URL,
DB_USER, DB_PASSWORD)) {
            // Получение нового идентификатора заказа
            String getIdQuery = "SELECT MAX(id) + 1 as newId FROM Orders";
            int newOrderId;
            try (PreparedStatement getIdStatement =
connection.prepareStatement(getIdQuery);
                 ResultSet getIdResult = getIdStatement.executeQuery()) {
                newOrderId = getIdResult.next() ? getIdResult.getInt("newId")
: 1;
            }
```

```
            // Получение идентификатора выбранного сервиса
            String selectedService =
serviceComboBox.getSelectionModel().getSelectedItem();
            int serviceId;
            String getServiceIdQuery = "SELECT id FROM Services WHERE name =
?";
            try (PreparedStatement getServiceIdStatement =
connection.prepareStatement(getServiceIdQuery)) {
                getServiceIdStatement.setString(1, selectedService);
                try (ResultSet getServiceIdResult =
getServiceIdStatement.executeQuery()) {
                    serviceId = getServiceIdResult.next() ?
getServiceIdResult.getInt("id") : -1;
                }
            }
            // Получение идентификатора выбранного времени
            String selectedTime =
scheduleComboBox.getSelectionModel().getSelectedItem();
            int scheduleId;
            String getScheduleIdQuery = "SELECT id FROM Schedule WHERE time =
?";
            try (PreparedStatement getScheduleIdStatement =
connection.prepareStatement(getScheduleIdQuery)) {
                getScheduleIdStatement.setString(1, selectedTime);
                try (ResultSet getScheduleIdResult =
getScheduleIdStatement.executeQuery()) {
                    scheduleId = getScheduleIdResult.next() ?
getScheduleIdResult.getInt("id") : -1;
                }
            }
            // Получение информации об аллергии
            String allergy = !allergyTextField.getText().isEmpty() ?
allergyTextField.getText() : "нет";
            // Получение идентификатора пользователя
            int userId = DatabaseController.idK;
            // Вставка нового заказа в базу данных
            String insertOrderQuery = "INSERT INTO Orders (id, user, service,
schedule, allergie) VALUES (?, ?, ?, ?, ?)";
            try (PreparedStatement insertOrderStatement =
connection.prepareStatement(insertOrderQuery)) {
                insertOrderStatement.setInt(1, newOrderId);
                insertOrderStatement.setInt(2,
Integer.parseInt(user.getText()));
                insertOrderStatement.setInt(3, serviceId);
                insertOrderStatement.setInt(4, scheduleId);
                insertOrderStatement.setString(5, allergy);
                insertOrderStatement.executeUpdate();
            }
            // Закрытие окна после успешного создания заказа
        this.stage.close();
        } catch (SQLException e) {
            // Вывод сообщения об ошибке в случае возникновения исключения
SQLException
            e.printStackTrace();
        }
    }


    // Метод загрузки услуг в комбобокс
    private void loadServicesIntoComboBox() {
        // Подключение к базе данных
        try (Connection connection = DriverManager.getConnection(DB_URL,
DB_USER, DB_PASSWORD)) {
```

```
            // SQL-запрос для получения списка услуг
            String query = "SELECT name FROM Services";
            try (PreparedStatement preparedStatement =
connection.prepareStatement(query);
                 ResultSet resultSet = preparedStatement.executeQuery()) {
                // Добавление каждой услуги в комбобокс
                while (resultSet.next()) {

serviceComboBox.getItems().add(resultSet.getString("name"));
                }
            }
        } catch (SQLException e) {
            // Вывод сообщения об ошибке в случае возникновения исключения
SQLException
            e.printStackTrace();
        }
    }


    // Метод загрузки расписания в комбобокс
    private void loadScheduleIntoComboBox() {
        try (Connection connection = DriverManager.getConnection(DB_URL,
DB_USER, DB_PASSWORD)) {
            // Создание SQL-запроса для выборки свободных времени
            String query = "SELECT time FROM Schedule WHERE booked = 0";
            try (PreparedStatement preparedStatement =
connection.prepareStatement(query);
                 ResultSet resultSet = preparedStatement.executeQuery()) {
                // Проверка на наличие свободных времени
                if (!resultSet.isBeforeFirst()) {
                    showErrorWindow("Нет свободных окон, попробуйте создать
заказ позже");
                } else {
                    // Если есть свободные времена, они добавляются в
комбобокс
                    while (resultSet.next()) {

scheduleComboBox.getItems().add(resultSet.getString("time"));
                    }
                }
            }
        } catch (SQLException e) {
            // Если произошла ошибка при работе с базой данных, она выводится
на экран
            e.printStackTrace();
        }
    }


    // Метод отображения окна с ошибкой
    private void showErrorWindow(String message) {
    }


    // Метод загрузки описания услуги в текстовое поле
    private void loadDescriptionIntoTextService(String selectedService) {
        try (Connection connection = DriverManager.getConnection(DB_URL,
DB_USER, DB_PASSWORD)) {
            // Создание SQL-запроса для выборки описания выбранной услуги
            String query = "SELECT description FROM Services WHERE name = ?";
            try (PreparedStatement preparedStatement =
connection.prepareStatement(query)) {
                // Установка параметра в SQL-запрос
```

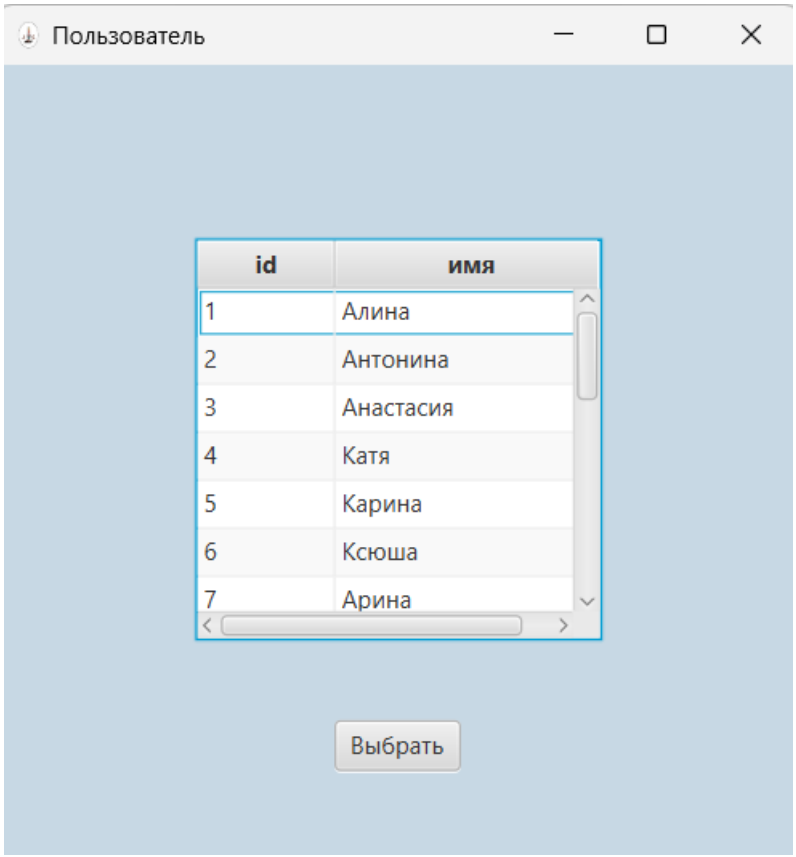
```
                preparedStatement.setString(1, selectedService);
                try (ResultSet resultSet = preparedStatement.executeQuery())
{
                    // Если описание найдено, оно устанавливается в текстовое
поле
                    if (resultSet.next()) {

textService.setText(resultSet.getString("description"));
                    }
                }
            }
        } catch (SQLException e) {
            // Если произошла ошибка при работе с базой данных, она выводится
на экран
            e.printStackTrace();
        }
    }


    // Метод загрузки ресурсов, связанных с услугой, в текстовое поле
    private void loadResourcesIntoTextResource(String selectedService) {
        try (Connection connection = DriverManager.getConnection(DB_URL,
DB_USER, DB_PASSWORD)) {
            // Создание SQL-запроса для выборки ресурсов, связанных с
выбранной услугой
            // Используем JOIN для объединения данных из двух таблиц
            String query = "SELECT r.name FROM Resources r " +
                    "JOIN resources_services rs ON r.id = rs.resources_id " +
                    "WHERE rs.services_id = (SELECT id FROM Services WHERE
name = ?)";
            try (PreparedStatement preparedStatement =
connection.prepareStatement(query)) {
                // Установка параметра в SQL-запрос
                preparedStatement.setString(1, selectedService);
                try (ResultSet resultSet = preparedStatement.executeQuery())
{
                    StringBuilder resources = new StringBuilder();
                    // Получение и объединение всех ресурсов, связанных с
выбранной услугой
                    while (resultSet.next()) {

resources.append(resultSet.getString("name")).append(", ");
                    }

                    // Убираем последнюю запятую и пробел, если они есть
                    if (resources.length() > 0) {
                        resources.setLength(resources.length() - 2);
                    }
                    // Установка полученных ресурсов в текстовое поле
                    textResource.setText(resources.toString());
                }
            }
        } catch (SQLException e) {
            // Если произошла ошибка при работе с базой данных, она выводится
на экран
            e.printStackTrace();
        }
    }


    // Метод обработки изменения выбора услуги в комбобоксе
    @FXML
    private void onServiceComboBoxSelectionChange(ActionEvent event) {
```

```
        // Получение выбранной услуги из комбобокса
        String selectedService =
serviceComboBox.getSelectionModel().getSelectedItem();
        // Если была выбрана услуга
        if (selectedService != null) {
            // Загрузка описания выбранной услуги в текстовое поле
            loadDescriptionIntoTextService(selectedService);
            // Загрузка ресурсов, связанных с выбранной услугой, в текстовое
поле
            loadResourcesIntoTextResource(selectedService);
            // Обновление цены выбранной услуги
            updateIdPrice(selectedService);
        }
    }


    // Метод обновления цены услуги
    private void updateIdPrice(String selectedService) {
        try (Connection connection = DriverManager.getConnection(DB_URL,
DB_USER, DB_PASSWORD)) {
            // Создание SQL-запроса для выборки цены выбранной услуги
            String query = "SELECT price FROM Services WHERE name = ?";
            try (PreparedStatement preparedStatement =
connection.prepareStatement(query)) {
                // Установка параметра в SQL-запрос
                preparedStatement.setString(1, selectedService);
                try (ResultSet resultSet = preparedStatement.executeQuery())
{
                    // Если цена найдена, она устанавливается в текстовое
поле
                    if (resultSet.next()) {

idPrice.setText(String.valueOf(resultSet.getDouble("price")));
                    }
                }
            }
        } catch (SQLException e) {
            // Если произошла ошибка при работе с базой данных, она выводится
на экран
            e.printStackTrace();
        }
    }
}
```

3.2.7. Окно выбора пользователя для дальнейшего создания записи

salonchik-vybor

Окно для выбора сотрудником клиента, на которого будет оформлена запись.



SalonchikVybor

Класс SalonchikVybor представляет собой контроллер для представления пользователей в таблице и обработки действий пользователя, таких как выбор пользователя.

```
package com.example.salonchik;
import javafx.beans.property.SimpleIntegerProperty;
import javafx.beans.property.SimpleStringProperty;
import javafx.collections.FXCollections;
import javafx.collections.ObservableList;
import javafx.event.ActionEvent;
import javafx.fxml.FXML;
import javafx.scene.control.Button;
import javafx.scene.control.TableColumn;
import javafx.scene.control.TableView;

import java.sql.*;

    public class SalonchikVybor {

        // Таблица для отображения списка пользователей
        @FXML
```

```
        private TableView<User> tableView;
        // Столбец таблицы для отображения ID пользователя
        @FXML
        private TableColumn<User, Integer> idColumn;
        // Столбец таблицы для отображения имени пользователя
        @FXML
        private TableColumn<User, String> nameColumn;
        // Кнопка для выбора пользователя
        @FXML
        private Button vybr;


        // Соединение с базой данных
        private Connection connection;


        // Метод инициализации, который устанавливает обработчики для
столбцов таблицы и загружает данные из базы данных
        public void initialize() {
            // Устанавливаем обработчики для столбцов таблицы
            idColumn.setCellValueFactory(cellData ->
cellData.getValue().idProperty().asObject());
            nameColumn.setCellValueFactory(cellData ->
cellData.getValue().nameProperty());
            // Соединение с базой данных
            String jdbcUrl = "jdbc:mysql://localhost:3306/Salon";
            String username = "root";
            String password = "";

            try {
                // Попытка установить соединение с базой данных
                connection = DriverManager.getConnection(jdbcUrl, username,
password);
            } catch (SQLException e) {
                // Если возникает ошибка при соединении с базой данных,
выводим ее в консоль
                e.printStackTrace();
            }
            // Загрузка данных из базы данных и установка их в таблицу
            tableView.setItems(getUsersFromDatabase());
            loadData();
        }


        // Метод для получения списка пользователей из базы данных
    private ObservableList<User> getUsersFromDatabase() {
        // Создаем пустой список пользователей
        ObservableList<User> users = FXCollections.observableArrayList();

        // Соединение с базой данных
        String jdbcUrl = "jdbc:mysql://localhost:3306/Salon";
        String username = "root";
        String password = "";

        try (Connection connection = DriverManager.getConnection(jdbcUrl,
username, password)) {
            // Попытка установить соединение с базой данных
            String query = "SELECT id, name FROM Users";
            // Подготовка и выполнение запроса
            try (PreparedStatement preparedStatement =
connection.prepareStatement(query);
                 ResultSet resultSet = preparedStatement.executeQuery()) {
                // Получение данных из результата запроса
```

```
                while (resultSet.next()) {
                    int userId = resultSet.getInt("id");
                    String name = resultSet.getString("name");
                    // Добавление пользователя в список
                    users.add(new User(userId, name));
                }
            }
        } catch (SQLException e) {
            // Если возникает ошибка при работе с базой данных, выводим ее в
консоль
            e.printStackTrace();
        }
        // Возвращаем список пользователей
        return users;
    }


        // Метод обработки нажатия кнопки выбора пользователя
    public void onVybrButtonClick(ActionEvent actionEvent) {
        try {
            User user = (User)
tableView.getSelectionModel().getSelectedItem();
            //Если пользователь был успешно выбран, вызывается метод vybr
            SalonchikApplication.vybr(user);
        } catch (Exception error) {
            // Если произошла ошибка, она выводится в консоль
            error.printStackTrace();
        }
    }


        // Внутренний класс для представления пользователя
    public static class User {
            // Свойства пользователя: id и name
        private final SimpleIntegerProperty id;
        private final SimpleStringProperty name;
            // Конструктор класса User
        public User(int id, String name) {
            // Инициализация свойств id и name
            this.id = new SimpleIntegerProperty(id);
            this.name = new SimpleStringProperty(name);
        }
        // Геттеры для свойств id и name
        public int getId() {
            return id.get();
        }

        public SimpleIntegerProperty idProperty() {
            return id;
        }

        public String getName() {
            return name.get();
        }

        public SimpleStringProperty nameProperty() {
            return name;
        }
    }


        // Метод для загрузки данных в таблицу
    private void loadData() {
```

```
        // Попытка создать объект Statement для выполнения SQL запроса
        try {
            Statement statement = connection.createStatement();
            // Выполнение SQL запроса для получения всех пользователей
            ResultSet resultSet = statement.executeQuery("SELECT * FROM
Users");
            // Получение данных из результата запроса и добавление их в
таблицу
            while (resultSet.next()) {
                tableView.getItems().add(new User(resultSet.getInt("id"),
resultSet.getString("name")));
            }
        } catch (SQLException ex) {
            // Если произошла ошибка при работе с базой данных, она выводится
в консоль
            ex.printStackTrace();
        }
    }

}
```

4. Руководство пользователя

Когда пользователь запускает приложение, он видит окно авторизации, в котором предлагается выполнить вход, а если он ранее не был зарегистрирован, то предлагается зарегистрироваться.

Если пользователь был ранее зарегистрирован: необходимо ввести данные в поля «логин» и «пароль», после чего нужно нажать кнопку «войти».

Если пользователь не был ранее зарегистрирован в системе: нужно нажать «зарегистрироваться» и в открывшемся окне заполнить поля: имя, логин, пароль, которые нужно придумать. После ввода данных нужно нажать на кнопку «Зарегистрироваться».

После выполнения регистрации откроется снова окно входа, в котором нужно ввести пароль и логин, который вы создали раннее и нажать «Войти».

1. Руководство для клиента

Если вход был выполнен клиентом, то в окне Заказ будут отображаться заказы авторизированного клиента, если они есть. При нажатии на поле с заказом в таблице заказов, будет открываться новое окно, в котором выбранную запись можно удалить или редактировать. После чего список заказов или сам заказ изменятся.

В окне Заказ также есть кнопка «Создать запись», при нажатии на которую будет открыто новое окно с полями для создания нового заказа. В поле «Услуги» нужно выбрать на какую услугу вы хотите записаться, после чего увидите описание этой услуги. Также нужно выбрать подходящее вам время и указать при необходимости, есть ли у вас аллергия. После того как вы выберите услугу, вам будет показана сумма заказа. Когда все данные заполнены, нужно нажать кнопку «Создать» и ваш заказ будет добавлен в базу данных.

2. Руководство для сотрудника

Если вход был выполнен сотрудником, то в окне Заказ будут отображаться заказы всех клиентов, если они есть. При нажатии на поле с заказом в таблице заказов, будет открываться новое окно, в котором выбранную запись можно удалить или редактировать. После чего список заказов или сам заказ изменятся.

Если поступил запрос записать незарегистрированного пользователя, то его сначала нужно зарегистрировать, после чего авторизироваться под ролью сотрудника и уже в окне Заказ выполнять действия. В нём есть кнопка

«Создать запись», при нажатии на которую будет открыто новое окно, в котором необходимо выбрать пользователя, на которого будет оформляться заказ. Чтобы выбрать пользователя нужно выбрать соответствующее поле и нажать кнопку «Выбрать», после чего откроется окно для создания записи, где в поле «Услуги» нужно выбрать услугу, после чего увидите описание этой услуги. Также нужно выбрать подходящее время и указать при необходимости, есть ли аллергия. После того как вы выберите услугу, вам будет показана сумма заказа. Когда все данные заполнены, нужно нажать кнопку «Создать» и он будет добавлен в базу данных для клиента, которого вы выбрали. Чтобы просмотреть обновлённые заказы необходимо заново авторизироваться.

Заключение

Результатом работы курсового проекта является разработанное и функционирующие настольное приложение для маникюрного салона.

Приложение разработано и реализовано для оптимизации времени При записи на процедуру клиентом или сотрудником, с возможностью просмотра уже созданных записей.

Выполненный проект в полном объеме соответствует заданию курсового проекта.

В процессе выполнения проекта были разработаны пояснения к каждому функционалу всего проекта, для подробного ознакомления пользователей с работой в системе.

Из полученных результатов по завершению работы над курсовым проектом, можно сделать вывод, что полученная разработка удовлетворяет все поставленные цели при создании настольного приложения для маникюрного салона.

Список литературы

1. Шилдт, Г. Java. Полное руководство. [Текст] / Шилдт Герберт — 10-е изд. — : Диалектика (Вильямс), 2020 — 1488 с.
2. Прохоренок, Н. А. JavaFX — : БХВ, 2020 — 768 с