

МИНИСТЕРСТВО ОБРАЗОВАНИЯ НАУКИ И МОЛОДЕЖНОЙ ПОЛИТИКИ
НИЖЕГОРОДСКОЙ ОБЛАСТИ
Государственное бюджетное профессиональное образовательное учреждение

НИЖЕГОРОДСКИЙ РАДИОТЕХНИЧЕСКИЙ КОЛЛЕДЖ

Специальность 09.02.07 Информационные системы и программирование

Квалификация Специалист по тестированию в области информационных технологий

КУРСОВАЯ РАБОТА

МДК 02.01 Технология разработки программного обеспечения

Тема: Разработка настольного приложения для предметной области «Кредитный отдел»

Выполнил студент группы
ЗИСиП-21-1
Костякова Алёна Александровна

Проверил преподаватель
Гутянская Е.М.

Проект защищен с оценкой

Дата защиты _____

Подпись _____

Нижний Новгород, 2023

Оглавление

Введение.....	3
1. Анализ предметной области.....	4
2. Проектирование базы данных информационной системы.....	6
2.1 Информационно логическая модель базы данных.....	6
2.2 Словарь данных.....	7
2.3 Ограничения ссылочной целостности.....	8
2.4 Обоснование выбора СУБД.....	8
2.5 Триггеры и хранимые процедуры.....	8
3. Разработка информационной системы.....	12
3.1 Разработка начального окна, входа и регистрации.....	12
3.1.1 Разработка приветственного окна.....	12
3.1.2 Разработка окон входа.....	15
3.1.2.1 Окно входа для сотрудника.....	15
3.1.2.2 Окно входа для клиента.....	17
3.1.3 Окно регистрации для клиента.....	21
3.2 Разработка интерфейса для клиента.....	23
3.2.1 Разработка основного метода.....	24
3.2.2 Дополнительные классы.....	31
3.3.3 Разработка интерфейса для сотрудника.....	34
3.3.3.1 Основное окно сотрудника.....	34
3.3.3.2 Изменение и добавление заказов.....	41
3.3.3.2.1 Изменение.....	41
3.3.3.2.2 Добавление.....	45
4. Руководство пользователя.....	49
Заключение.....	51
Список литературы.....	52

Введение.

Современные гостиницы сталкиваются с рядом сложностей при управлении информацией о номерах, услугах и бронировании. Необходимо эффективно отслеживать доступность номеров, управлять ценами и сезонными акциями, а также оперативно обрабатывать запросы клиентов по бронированию и другим услугам. Все эти процессы требуют надежной и оптимизированной базы данных, которая будет обеспечивать хранение и обработку информации, а также поддержку многопользовательского доступа и защиту данных.

В рамках данной курсовой работы будет разработана структура базы данных для гостиницы. Будут определены основные сущности, такие как номера, гости, бронирования, услуги и т. д., и их взаимосвязи. Особое внимание будет уделено учету доступности номеров и изменениям в бронированиях. Также будет реализован многопользовательский доступ к базе данных с разделением прав доступа для разных ролей сотрудников гостиницы (администраторы, менеджеры, персонал ресепшн и т. д.) и разработаны соответствующие интерфейсы для клиентов и сотрудников.

Для достижения поставленной цели необходимо выполнить следующие задачи:

- Разработать структуру базы данных.
- Реализовать многопользовательский доступ к базе данных.
- Разработать интерфейс для клиентов.
- Разработать интерфейс для сотрудников гостиницы.

Предметом исследования данной работы является проектирование и разработка базы данных для гостиницы, которая будет обеспечивать эффективное управление информацией о номерах, услугах и бронировании. Основной упор делается на создание удобного и функционального интерфейса для клиентов и сотрудников гостиницы, а также на обеспечение многопользовательского доступа и разграничения функционала по ролям.

Таким образом, разработка базы данных для гостиницы является актуальной и востребованной темой в современном туристическом бизнесе, которая позволит оптимизировать процессы управления информацией и улучшить качество обслуживания клиентов.

1. Анализ предметной области.

Функционал, который будет реализован:

Управление номерами

- Хранение информации о номерах: Номера, их типы, характеристики, доступность, цены и статусы (свободен, забронирован).
- Учет доступности: Отслеживание свободных и забронированных номеров по датам.

Управление услугами

- Хранение данных о предоставляемых услугах: Описание, стоимость.

Управление бронированиями

- Система бронирования: Запросы на бронирование, подтверждения, отмены брони.

Многопользовательский доступ

- Разграничение доступа: Доступ будет доступен для зарегистрированных клиентов, а так же доступна регистрация для новых пользователей. Так же будет реализован доступ для сотрудников гостиницы со своими правами.

Интерфейс для новых пользователей

- Регистрация: Возможность зарегистрироваться, для последующего пользования.

Интерфейс для клиентов

- Бронирование номеров: Возможность клиентов выбирать номера, просматривать информацию о них, оформлять бронь.

Интерфейс для сотрудников

- Управление бронированиями: изменение/добавление/закрытие бронирований.

Формат хранения данных:

Номера

- Таблица номеров: ID номера, название номера, тип номера, описание, цена, доступность по датам.

Услуги

- Таблица услуг: ID услуги, название, стоимость.

Бронирования

- Таблица бронирований: ID брони, ID клиента, ID комнаты, даты бронирования, статус бронирования, полная цена.

Клиенты

- Таблица клиентов: ID клиента, ФИО, паспортные данные, номер телефон, пароль.

Администраторы

- Таблица администраторов: ID админа, имя, логин, пароль.

Журнал услуг

- Таблица услуг в заказах: доступ к просмотру купленных услуг к заказу, ID заказа, ID услуги.

2. Проектирование базы данных информационной системы.

2.1. Информационно-логическая модель базы данных.

Информационно-логическая модель является моделью данных, отображающей предметную область в виде совокупности информационных объектов (ИО) и структурных связей между ними.

Моя схема базы данных:

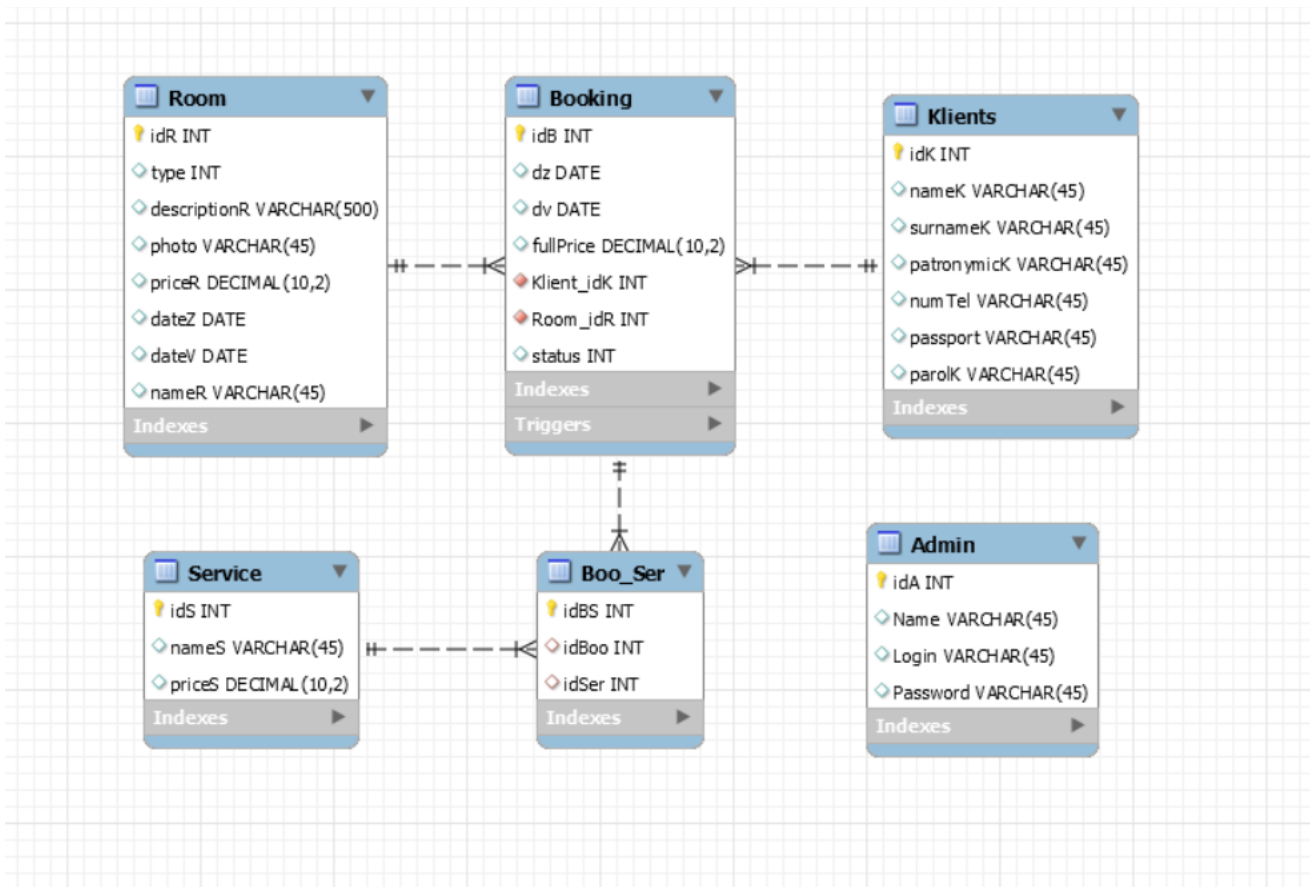


Рисунок 1. Схема базы данных.

Таблица Room содержит информацию о номерах, таблица Klient – это информация о клиентах, Booking – информация о заказах, Service – информация о дополнительных услугах, а таблица Admin содержит информацию о сотрудниках гостиницы. Так же есть таблица Boo_Ser это связующая таблица между Booking и Service, она нужна для записи того, какие услуги куплены в определенном заказе.

2.2. Словарь данных.

Словарь данных - это описание всех данных, используемых в информационной системе. Для данной информационной системы словарь данных будет содержать следующие данные:

Room:

idR (INT) - уникальный идентификатор номера;
type (INT) - тип номера;
descriptionR (VARCHAR) - описание номера;
photo (VARCHAR) - фото номера в формате изображения;
priceR (DECIMAL) - стоимость пребывания в номере;
dateZ (DATE) – дата заселения;
dateV (DATE) – дата выселения;
nameR (VARCHAR) – наименование комнаты.

Service:

idS (INT) - уникальный идентификатор услуги;
nameS (VARCHAR) - название услуги;
priceS (DECIMAL) - стоимость услуги.

Klients:

idK (INT) - уникальный идентификатор клиента;
nameK (VARCHAR) - имя клиента;
surnameK (VARCHAR) - фамилия клиента;
patronymicK (VARCHAR) - отчество клиента;
numTel (VARCHAR) - контактные данные клиента;
passport (VARCHAR) - паспортные данные клиента;
parolK (VARCHAR) – пароль клиента.

Booking:

idB (INT) - уникальный идентификатор бронирования;
dz (DATE) - дата заселения;
dv (DATE) - дата выселения;
Klient_idK (INT) - id клиента, на которого бронируется номер;
Room_idR (INT) - id выбранного номера для бронирования;
fullPrice (DECIMAL) - рассчитанная динамически стоимость заказа;
status (INT) — статус заказа (1 - действительный, 0 — закрытый).

Boo_Ser:

idBS (INT) – уникальный идентификатор связи услуг и бронирования;
idBoo (INT) – id бронирования;
idSer (INT) — id заказанных услуг.

Admin:

idA (INT) – уникальный идентификатор администратора;

Name (VARCHAR) – имя администратора;

Login (VARCHAR) – логин администратора;

Password (VARCHAR) – пароль администратора.

2.3. Ограничения ссылочной целостности.

Ограничения ссылочной целостности - это правила, которые гарантируют, что связи между таблицами базы данных будут соблюдаться. Для данной информационной системы будут установлены следующие ограничения ссылочной целостности:

Таблицы Room и Booking соединены связью один-ко-многим, idR из таблицы Room передается в таблицу Booking.

Таблицы Klient и Booking соединены связью один-ко-многим, idK из таблицы Klient передается в таблицу Booking.

Таблицы Service и Booking соединены между собой связью многие-ко-многим, тем самым образуя дополнительную таблицу Boo_Ser в которую передаются idS и idB

2.4. Обоснование выбора СУБД.

Для данной информационной системы будет использоваться реляционная СУБД MySQL. Она была выбрана из-за следующих причин:

MySQL является одной из самых популярных СУБД в мире и имеет большое сообщество разработчиков и пользователей.

MySQL обладает высокой производительностью и масштабируемостью, что позволит обеспечить работу системы при большом количестве пользователей и данных.

MySQL поддерживает многопользовательский доступ и разграничение прав доступа, что соответствует требованиям к информационной системе.

2.5. Триггеры и хранимые процедуры.

Триггеры и хранимые процедуры - это программные объекты, которые выполняются автоматически при определенных событиях в базе данных. Для данной информационной системы могут быть использованы следующие триггеры и хранимые процедуры:

В своей базе данных я реализовала три триггера:

1) Триггер на заполнение дат бронирования в таблице Room.


```

1 • use kyrsach;
2 • CREATE TRIGGER room_date
3   AFTER INSERT ON Booking
4   FOR EACH ROW
5   UPDATE Room
6   SET dateZ = NEW.dz, dateV = NEW.dv
7   WHERE idR = NEW.Room_idR;

```

Рисунок 2.Триггер№1.

Данный триггер(рис. 2) срабатывает после вставки новой записи в таблицу Booking.

Он обновляет таблицу Room, устанавливая для соответствующего номера дату заселения (dateZ) и дату выселения (dateV) из только что добавленного заказа (NEW.dz и NEW.dv соответственно).

Таким образом, при добавлении нового заказа в таблицу Booking, данный триггер автоматически изменяет информацию о датах заселения и выселения для соответствующего номера в таблице Room. Это позволяет поддерживать актуальные данные о занятости номеров и упрощает процесс управления и поиска свободных комнат.

2)Триггер на удаление дат бронирования у комнаты, если заказ пришел в статус 0 (закрылся).

```

1 • USE kyrsach;
2   DELIMITER //
3 • CREATE TRIGGER clear_dates_in_room
4   AFTER UPDATE ON Booking
5   FOR EACH ROW
6   BEGIN
7     IF NEW.status = 0 THEN
8       UPDATE Room
9       SET dateZ = NULL, dateV = NULL
10      WHERE idR = NEW.Room_idR;
11    END IF;
12  END;
13  //
14  DELIMITER ;

```

Рисунок 3.Триггер №2.

Данный триггер(рис. 3) срабатывает после обновления записи в таблице Booking.

Он проверяет, если у обновленного заказа новое значение статуса (NEW.status) равно 0, что означает закрытие заказа, то выполняется следующий блок кода.

В блоке кода происходит обновление таблицы Room для соответствующего номера. При закрытии заказа обновляются значения полей dateZ и dateV, устанавливая в них NULL (отсутствие значения). Это означает, что комната снова доступна для бронирования.

Таким образом, при закрытии заказа в таблице Booking, данный триггер автоматически обновляет информацию о датах заселения и выселения для соответствующего номера в таблице Room, делая его снова доступным для бронирования.

3) Триггер на изменение цены при изменении даты выезда.

```
DELIMITER //
CREATE TRIGGER update_fullPrice
BEFORE UPDATE ON Booking
FOR EACH ROW
BEGIN
    DECLARE price_change INT;
    DECLARE room_price INT;
    DECLARE days_diff INT;
    IF NEW.dv > OLD.dv THEN
        SELECT priceR INTO room_price
        FROM Room
        WHERE idR = NEW.Room_idR;
        SET days_diff = DATEDIFF(NEW.dv, OLD.dv);
        SET price_change = room_price * days_diff;
        IF (NEW.fullPrice + price_change) >= 0 THEN
            SET NEW.fullPrice = NEW.fullPrice + price_change;
        ELSE
            SET NEW.fullPrice = 0;
        END IF;
    ELSEIF NEW.dv < OLD.dv THEN
        SELECT priceR INTO room_price
        FROM Room
        WHERE idR = NEW.Room_idR;
        SET days_diff = DATEDIFF(OLD.dv, NEW.dv);
        SET price_change = room_price * days_diff;
        IF (NEW.fullPrice - price_change) >= 0 THEN
            SET NEW.fullPrice = NEW.fullPrice - price_change;
        ELSE
            SET NEW.fullPrice = 0;
        END IF;
    END IF;
END; //
DELIMITER ;
```

Этот триггер работает следующим образом:

1. Сначала он проверяет, изменился ли значение dv для конкретной записи в таблице Booking. Это делается с помощью оператора IF NEW.dv > OLD.dv и ELSEIF NEW.dv < OLD.dv. Если дата обновления (dv) больше предыдущей даты (OLD.dv), то выполняется первый блок кода. Если дата обновления меньше предыдущей даты, то выполняется второй блок кода.
2. Затем триггер выбирает цену комнаты (priceR) из таблицы Room, используя Room_idR из обновляемой записи в таблице Booking.
3. Далее вычисляется разница в днях между новой и старой датами (days_diff) и вычисляется изменение цены (price_change), умножая цену комнаты на количество дней.
4. Если новая полная цена (NEW.fullPrice) плюс изменение цены больше или равна нулю, то новая полная цена обновляется на сумму старой полной цены и изменения цены. Если новая полная цена плюс изменение цены меньше нуля, то новая полная цена устанавливается в ноль.
5. Если дата обновления меньше предыдущей даты, то происходит то же самое, но с вычитанием изменения цены из новой полной цены.

Таким образом, этот триггер автоматически обновляет столбец fullPrice в таблице Booking каждый раз, когда обновляется дата (dv), увеличивая или уменьшая полную цену в зависимости от изменения даты и цены комнаты.

Так же я реализовала одну хранимую процедуру:

```
1 CREATE DEFINER='root'@'%' PROCEDURE `add_user` (  
2     IN idK INT,  
3     IN nameK VARCHAR(50),  
4     IN surnameK VARCHAR(50),  
5     IN patronymicK VARCHAR(50),  
6     IN numTel VARCHAR(50),  
7     IN passport VARCHAR(50),  
8     IN parolK VARCHAR(50)  
9 )  
10 BEGIN  
11     INSERT INTO KlientS (idK, nameK, surnameK, patronymicK, numTel, passport,  
12     VALUES (idK, nameK, surnameK, patronymicK, numTel, passport, parolK);  
13 END
```

Рисунок 4. Хранимая процедура.

Эта процедура(рис. 4) добавляет нового пользователя, при регистрации.

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. Разработка начального окна, входа и регистрации.

3.1.1. Разработка приветственного окна.

```
public class Main extends Application {

    @Override
    public void start(Stage stage) throws IOException {
        // Установка иконки приложения
        stage.getIcons().add(new Image(s: "/photo/logo.png"));
        // Запрет изменения размеров окна
        stage.setResizable(false);
        // Загрузка FXML-файла для интерфейса входа
        FXMLLoader fxmlLoader = new FXMLLoader(Main.class.getResource(name: "login.fxml"));
        // Создание сцены и установка размеров окна
        Scene scene = new Scene(fxmlLoader.load(), w: 500, h: 500);
        // Установка заголовка окна
        stage.setTitle("Contemporary Oasis Hotel");
        // Установка сцены на окно
        stage.setScene(scene);
        // Отображение окна
        stage.show();
    }

    no usages
    public static void main(String[] args) {
        // Запуск приложения
        launch(args);
    }
}
```

Рисунок 5. Main.

Этот класс () осуществляет запуск оконного приложения. Он открывает FXML файл «login.fxml»

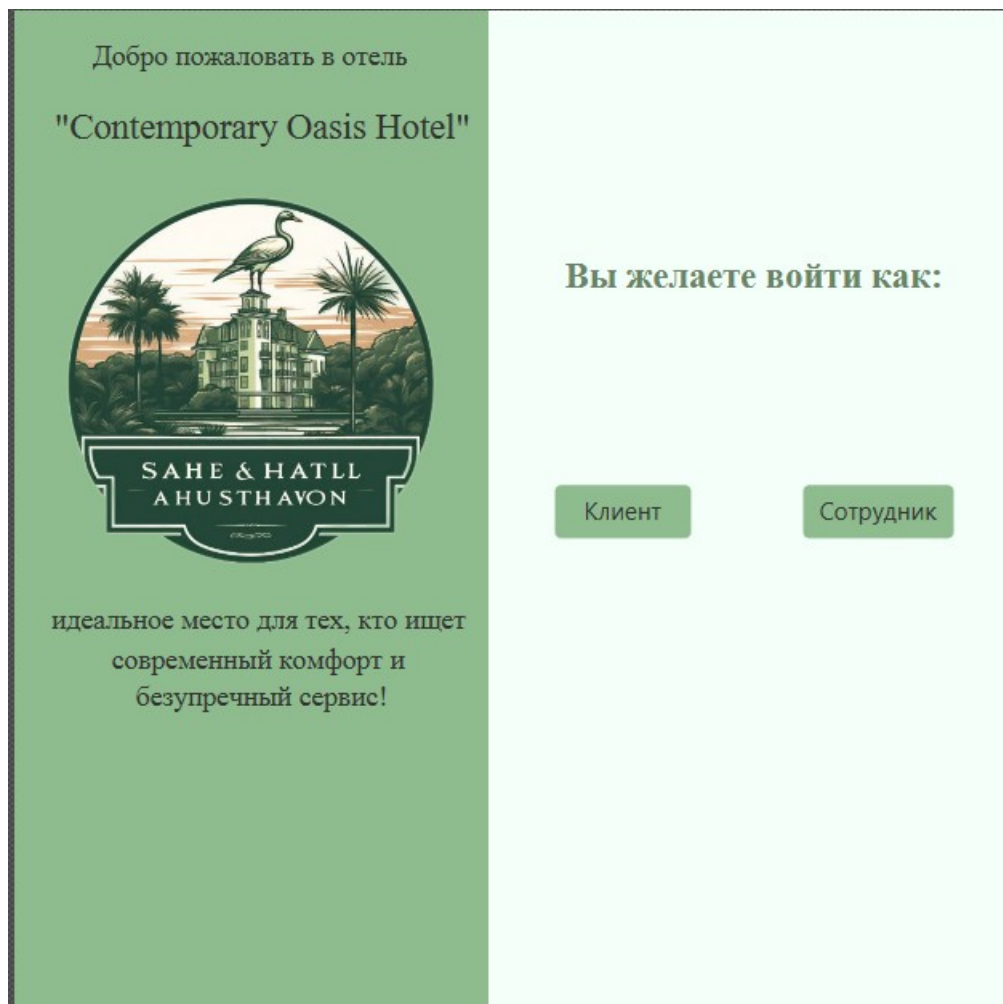


Рисунок 6. login.fxml

В этом FXML файле прописан зависимый контроллер — LoginController.

В коде реализован дизайн, подобран логотип, который соответствует теме предметной области и подобраны сочетаемые цвета.

Так же здесь находиться две кнопки «Клиент» и «Сотрудник», и к этим кнопкам подключены два действия «openKlientWindow» и «openSotrWindow» соответственно.

```

public class LoginController {
    1 usage
    @FXML
    private Button clientButton;
    1 usage
    @FXML
    private Button sotrButton;
    // Метод для открытия окна клиента
    1 usage
    public void openKlientWindow() throws IOException {
        // Создание нового окна
        Stage stage = new Stage();
        stage.getIcons().add(new Image(s: "/photo/logo.png"));
        stage.setResizable(false);
        // Загрузка FXML файла для окна клиента
        Parent root = FXMLLoader.load(getClass().getResource(name: "regKl.fxml"));
        stage.setTitle("Окно авторизации.");
        stage.setScene(new Scene(root, v: 500, v1: 500));
        stage.show();
    }
    // Метод для открытия окна сотрудника
    1 usage
    public void openSotrWindow() throws IOException{
        // Создание нового окна
        Stage stage = new Stage();
        stage.getIcons().add(new Image(s: "/photo/logo.png"));
        stage.setResizable(false);
        // Загрузка FXML файла для окна сотрудника
        Parent root = FXMLLoader.load(getClass().getResource(name: "sotr.fxml"));
        stage.setTitle("Окно авторизации.");
        stage.setScene(new Scene(root, v: 500, v1: 500));
        stage.show();
    }
}

```

Рисунок 6. LoginController.

LoginController (рис. 6), контроллер login.fxml. Здесь прописанно то, что при желании войти как клиент, открывается новое окно «regKl.fxml». Если же пользователь является сотрудником, то - «sotr.fxml».

3.1.2. Разработка окон входа.

3.1.2.1. Окно входа для сотрудника.

Рисунок 7. Sotr.fxml

Зависимый контроллер этого FXML – SotrController.

Здесь присутствуют два поля ввода для логина и пароля сотрудника. При неверном вводе одного или обоих полей, появляется сообщение о не правильном вводе данных. Если же вход выполнен правильно, то при нажатии на кнопку «Войти» срабатывает действие «vhodSotrClicked»(рис. 9) и открывается окно администратора(рис. 10).

```
public class SotrController {  
    2 usages  
    @FXML  
    private TextField logSotr; // Поле для ввода логина сотрудника  
    2 usages  
    @FXML  
    private TextField parolSotr; // Поле для ввода пароля сотрудника  
    2 usages  
    @FXML  
    private Button vhodSotr; // Кнопка для входа сотрудника  
    2 usages  
    @FXML  
    private Label message; // Метка для вывода сообщений об ошибке или статусе  
  
    // Поля для работы с базой данных  
    2 usages  
    private Connection connection; // Объект соединения с базой данных  
    4 usages  
    private PreparedStatement preparedStatement; // Подготовленное SQL-выражение  
    // Метод инициализации контроллера  
    no usages  
    @FXML  
    void initialize() {  
        connectToDatabase(); // Вызов метода для подключения к базе данных  
    }  
}
```

Рисунок 8. SortController

```

// Обработчик события нажатия кнопки "Вход сотрудника"
1 usage
@FXML
void vlnodSotrClicked(ActionEvent event) {
    String login = logSotr.getText(); // Получение введенного логина
    String password = parolSotr.getText(); // Получение введенного пароля

    // Попытка аутентификации пользователя по введенным данным
    if (authenticateUser(login, password)) { // Если аутентификация прошла успешно
        openAdminWindow(event); // Открыть окно администратора
    } else { // Если аутентификация не удалась
        message.setText("Неверный логин или пароль!"); // Вывести сообщение об ошибке
    }
}

// Метод для подключения к базе данных
1 usage
private void connectToDatabase() {
    try {
        String url = "jdbc:mysql://localhost:3306/kyrsach"; // URL базы данных
        String user = "root"; // Имя пользователя базы данных
        String password = ""; // Пароль пользователя

        // Установка соединения с базой данных
        connection = DriverManager.getConnection(url, user, password);
    } catch (SQLException e) {
        e.printStackTrace(); // Вывод сообщения об ошибке подключения
    }
}

// Метод для аутентификации пользователя в базе данных
1 usage
private boolean authenticateUser(String login, String password) {
    try {
        String query = "SELECT * FROM Admin WHERE Login = ? AND Password = ?";
        preparedStatement = connection.prepareStatement(query);
        preparedStatement.setString(parameterIndex: 1, login); // Установка параметра логина
        preparedStatement.setString(parameterIndex: 2, password); // Установка параметра пароля

        ResultSet resultSet = preparedStatement.executeQuery(); // Выполнение запроса

        return resultSet.next(); // Возвращение результата аутентификации
    } catch (SQLException e) {
        e.printStackTrace(); // Вывод сообщения об ошибке SQL
        return false; // Возвращение false в случае ошибки
    }
}
}

```

Рисунок 9. Методы класса SortController.


```

// Метод для открытия окна администратора
1 usage
private void openAdminWindow(ActionEvent event) {
    try {
        // Загрузка разметки из файла admin.fxml
        FXMLLoader loader = new FXMLLoader(getClass().getResource( name: "admin.fxml"));
        Parent root = loader.load(); // Загрузка корневого элемента из разметки

        // Создание и настройка нового окна
        Stage stage = new Stage();
        stage.setResizable(false); // Запрет изменения размеров окна
        stage.getIcons().add(new Image( "/photo/logo.png")); // Установка иконки окна
        stage.setTitle("Admin Window"); // Установка заголовка окна
        stage.setScene(new Scene(root)); // Установка сцены в окне
        stage.show(); // Отображение окна

        // Закрытие текущего окна
        Stage currentStage = (Stage) vhdSotr.getScene().getWindow();
        currentStage.close();
    } catch (Exception e) {
        e.printStackTrace(); // Вывод сообщения об ошибке при открытии окна
    }
}
}

```

Рисунок 10. Метод открытия нового окна.

3.1.2.2 Окно входа для клиента.

Авторизация

введите ваш номер телефона

введите ваш пароль

Войти

ещё не зарегистрированы? Зарегистрироваться

Рисунок 11. regKl.fxml.

Класс контроллер этого fxml-файла(рис. 11) – RegKl.

В этом fxml присутствуют поля ввода для телефона пользователя и его пароля. Кнопка «Войти» которая при верной авторизации открывается окно для создания заказа.

Если е пользователь ещё не зарегистрирован, то при нажатии на кнопку «Зарегистрироваться» открывается новое окно регистрации.

Вот код класса RegKl

```
3 usages
public class RegKl {
    3 usages
    public static int idK;
    2 usages
    @FXML
    private TextField tel;
    2 usages
    @FXML
    private TextField parol;
    3 usages
    @FXML
    private Button vhod;
    3 usages
    @FXML
    private Button reg;
    2 usages
    @FXML
    private Label messageLabel;
    // Объявление констант для подключения к базе данных
    1 usage
    private static final String url = "jdbc:mysql://localhost:3306/kyrsach";
    1 usage
    private static final String user = "root";
    1 usage
    private static final String password = "";
    // Объявление переменной для подключения к базе данных
    2 usages
    private Connection connection;
    // Конструктор класса, который устанавливает подключение к базе данных
    no usages
    public RegKl() {
        try {
            connection = DriverManager.getConnection(url, user, password);
        } catch (SQLException e) {
            e.printStackTrace();
        }
    }
}
```

Рисунок 12. RegKl.

```

// Метод инициализации, который настраивает обработчики действий для кнопок
no usages
@FXML
public void initialize() {
    vnod.setOnAction(event -> {
        String phoneNumber = tel.getText();
        String password = parol.getText();
        boolean isAuthorized = checkAuthorization(phoneNumber, password);

        if (isAuthorized) {
            openOrderWindow();
        } else {
            messageLabel.setText("Неправильно введены номер телефона или пароль!");
        }
    });
    reg.setOnAction(event -> {
        openRegistrationWindow();
    });
}

// Метод проверки авторизации пользователя
1 usage
private boolean checkAuthorization(String phoneNumber, String password) {
    try {
        String selectQuery = "SELECT * FROM Klients WHERE numTel = ? AND parolK = ?";
        PreparedStatement statement = connection.prepareStatement(selectQuery);
        statement.setString(parameterIndex: 1, phoneNumber);
        statement.setString(parameterIndex: 2, password);

        ResultSet resultSet = statement.executeQuery();

        if (resultSet.next()) {
            idK = resultSet.getInt(columnLabel: "idK"); // Получить idK
            return true;
        }
    } catch (SQLException e) {
        e.printStackTrace();
    }
    return false;
}

```

Рисунок 13. Методы RegKl.

```

// Метод открытия окна заказа
1 usage
private void openOrderWindow() {
    try {
        FXMLLoader loader = new FXMLLoader(getClass().getResource( name: "ordeer.fxml"));
        Parent root = loader.load();
        Stage stage = new Stage();
        stage.setResizable(false);
        stage.getIcons().add(new Image( s: "/photo/logo.png"));
        stage.setScene(new Scene(root));
        stage.show();

        Stage currentStage = (Stage) vmod.getScene().getWindow();
        currentStage.close();
    } catch (IOException e) {
        e.printStackTrace();
    }
}

// Метод открытия окна регистрации
1 usage
private void openRegistrationWindow() {
    try {
        FXMLLoader loader = new FXMLLoader(getClass().getResource( name: "zareg.fxml"));
        Parent root = loader.load();
        Stage stage = new Stage();
        stage.setResizable(false);
        stage.getIcons().add(new Image( s: "/photo/logo.png"));
        stage.setScene(new Scene(root));
        stage.show();

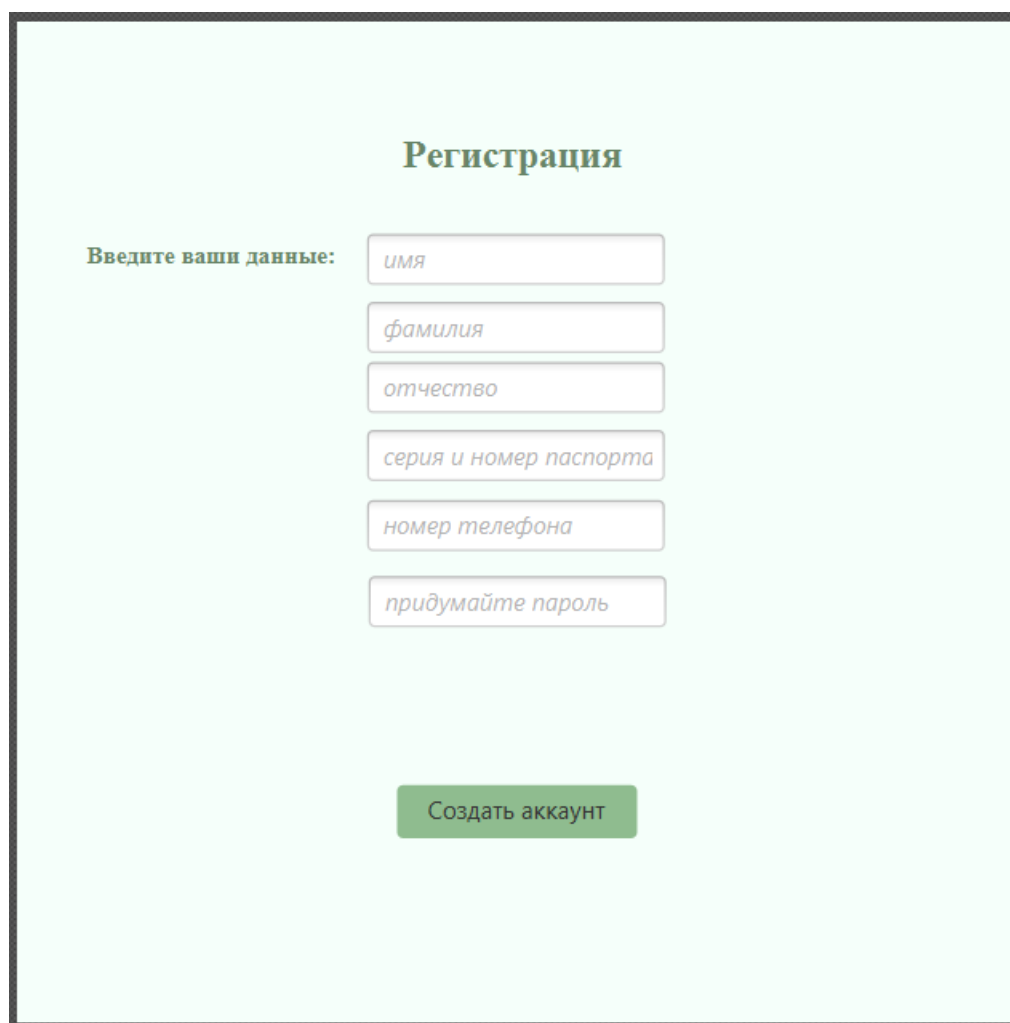
        Stage currentStage = (Stage) reg.getScene().getWindow();
        currentStage.close();
    } catch (IOException e) {
        e.printStackTrace();
    }
}
}

```

Рисунок 14. Методы для открытия окон.

Метод `openRegistrationWindow` открывает окно для регистрации `zareg.fxml`(рис. 15).

3.1.3. Окно регистрации для клиента



The image shows a registration window titled "Регистрация" (Registration) in a bold, dark green font. Below the title, the text "Введите ваши данные:" (Enter your data:) is displayed in a bold, dark green font. To the right of this text are six input fields, each with a light gray border and a light gray placeholder text. The fields are labeled: "имя" (name), "фамилия" (surname), "отчество" (patronymic), "серия и номер паспорта" (passport series and number), "номер телефона" (phone number), and "придумайте пароль" (create a password). Below the input fields is a green button with the text "Создать аккаунт" (Create account) in white.

Рисунок 15. zareg.fxml

У этого fxml(рис. 15) класс контроллер — Zareg. Здесь реализованы шесть полей ввода для данных клиента, которые при нажатии на кнопку «Создать аккаунт» записываются в базу данных.

```

public class Zareg {
    2 usages
    @FXML
    private TextField imya; // Поле для ввода имени пользователя
    2 usages
    @FXML
    private TextField fam; // Поле для ввода фамилии пользователя
    2 usages
    @FXML
    private TextField otch; // Поле для ввода отчества пользователя
    2 usages
    @FXML
    private TextField pass; // Поле для ввода паспортных данных пользователя
    2 usages
    @FXML
    private TextField tel; // Поле для ввода номера телефона пользователя
    2 usages
    @FXML
    private TextField parol; // Поле для ввода пароля пользователя
    1 usage
    @FXML
    private Label label; // Метка для вывода сообщений об ошибке или статусе
    2 usages
    @FXML
    private Button buttReg; // Кнопка для регистрации пользователя

```

Рисунок 16. Zareg


```

// Метод для добавления пользователя в базу данных
1 usage
@FXML
private void addUserToDatabase() {
    String url = "jdbc:mysql://localhost:3306/kyrsach"; // URL базы данных
    String user = "root"; // Имя пользователя базы данных
    String password = ""; // Пароль пользователя

    try (Connection connection = DriverManager.getConnection(url, user, password);
        CallableStatement statement = connection.prepareCall( sql: "{call add_user(?, ?, ?, ?, ?, ?, ?)}")) {
        // Установка параметров для вызова хранимой процедуры add_user
        statement.setNull( parameterIndex: 1, Types.INTEGER); // Первый параметр - идентификатор, пока не используется
        statement.setString( parameterIndex: 2, imya.getText()); // Установка имени пользователя
        statement.setString( parameterIndex: 3, fam.getText()); // Установка фамилии пользователя
        statement.setString( parameterIndex: 4, otch.getText()); // Установка отчества пользователя
        statement.setString( parameterIndex: 5, tel.getText()); // Установка номера телефона пользователя
        statement.setString( parameterIndex: 6, pass.getText()); // Установка паспортных данных пользователя
        statement.setString( parameterIndex: 7, parol.getText()); // Установка пароля пользователя
        statement.execute(); // Выполнение запроса

        Stage currentStage = (Stage) buttReg.getScene().getWindow();
        currentStage.close(); // Закрытие текущего окна

        FXMLLoader loader = new FXMLLoader(getClass().getResource( name: "regKl.fxml"));
        Parent root = loader.load(); // Загрузка разметки из файла regKl.fxml

        Stage regKlStage = new Stage(); // Создание нового окна
        regKlStage.getIcons().add(new Image( s: "/photo/logo.png")); // Установка иконки окна
        regKlStage.setResizable(false); // Запрет изменения размеров окна
        regKlStage.setScene(new Scene(root)); // Установка сцены в окне
        regKlStage.show(); // Отображение окна регистрации клиента
    } catch (SQLException | IOException e) {
        e.printStackTrace(); // Вывод сообщения об ошибке
        label.setText("Произошла ошибка при добавлении пользователя в базу данных."); // Вывод сообщения об ошибке в метке
    }
}
}

```

Рисунок 17. Метод класса Zareg.

При регистрации данные пользователя заносятся в базу данных и после нажатия на кнопку «Создать аккаунт» открывается окно regKl.fxml(рис. 11), где пользователь вводит уже свои зарегистрированные данные.

После того как клиент авторизовался открывается окно ordeer.fxml(рис. 18).

3.2. Разработка интерфейса для клиента.

Клиент должен иметь возможность создавать заказ. Что бы его создать надо выбрать подходящее время бронирования (причем дата заезда не может быть раньше сегодняшней даты, а дата выезда не должна быть меньше даты заезда), подходящую комнату(одноместную, двухместную, трехместную), так же должна быть возможность выбирать доп. слуги за доп. плату. Так же клиент должен видеть итоговую стоимость заказа.

3.2.1 Разработка основного метода.

Выберите комнату, подходящую под ваши требования



29.11.2023

30.11.2023

Одноместные номера

Двухместные номера

Трёхместные номера

Фото	Описание	Цена
		
	Наш небольшой и уютный номер эконом-класса. Номер продуманно спроектирован так, чтобы максимально увеличить пространство и в то же время предоставить все необходимые удобства, необходимые для приятного отдыха.	300.0

Выберите услуги:

Услуга	Цена
Доп. постельное бельё	100.0
Уборка в номере	230.0
Доставка ужина в номер	150.0
★all inclusive★	3000.0
Доп. спальное место	200.0

Добавить услуги

Итого: **3800,00**

Забронировать!

Рисунок 18. order.fxml

Класс контроллер этого fxml(рис. 18) — OrderController.

Здесь пользователь должен выбрать даты планируемого заезда и выезда, позже нажать на тот тип комнаты которую он хочет и после этого появляются подходящие свободные комнаты, с их названием, фото, описанием и ценой за сутки. При выборе одной комнаты при нажатии на контекстное меню «Забронировать» цена комнат умножается на количество дней, которые будет проживать в ней клиент и записываются в Lable как итоговая цена. Так же если пользователю необходимы дополнительные услуги, он может выбрать одну или несколько и на кнопку «Добавить услуги» их ценник складывается с итоговой стоимостью. При нажатии на кнопку «Забронировать» появляется такое окно (рис.19)

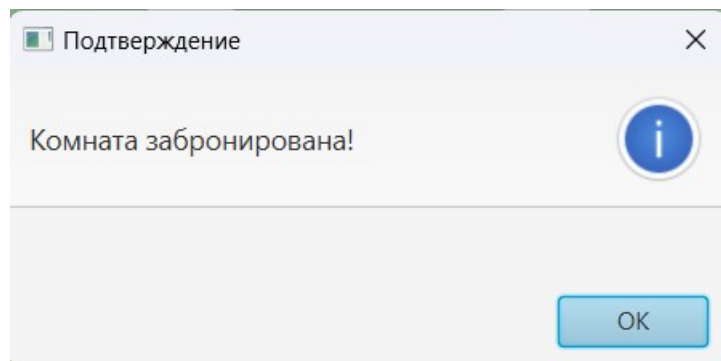


Рисунок 19. Окно подтверждения.

Вот код контроллера этого окна:

```
public class OrderController implements Initializable {  
    // Поля FXML, связанные с элементами пользовательского интерфейса  
    @FXML  
    private TextField tel; // Поле для ввода телефона  
    @FXML  
    private TextField parol; // Поле для ввода пароля  
    @FXML  
    private TableView<Room> roomTable; // Таблица для отображения доступных комнат  
    @FXML  
    private ContextMenu contextMenu; // Контекстное меню для комнат  
    @FXML  
    private TableColumn<Room, ImageView> photoColumn; // Столбец для отображения фотографий комнат  
    @FXML  
    private TableColumn<Room, String> descriptionColumn; // Столбец для отображения описания комнат  
    @FXML  
    private TableColumn<Room, Double> priceColumn; // Столбец для отображения цены комнат  
    @FXML  
    private Label label; // Метка для вывода сообщений об ошибке или статусе  
    @FXML  
    private DatePicker date1; // Выбор даты начала проживания  
    @FXML  
    private DatePicker date2; // Выбор даты окончания проживания  
    @FXML  
    private TableView<Service> tabYsl; // Таблица для отображения доступных услуг  
    @FXML  
    private TableColumn<Service, Integer> colID; // Столбец для отображения идентификатора услуги
```

```

@FXML
private TableColumn<Servicee, String> colYsl; // Столбец для отображения названия услуги
@FXML
private TableColumn<Servicee, Double> colPrice; // Столбец для отображения цены услуги
@FXML
private Label price; // Метка для отображения общей стоимости проживания и услуг
@FXML
private Button addYsl; // Кнопка для добавления выбранных услуг
@FXML
private Button bookButton; // Кнопка для бронирования комнаты
@FXML
private TableColumn<Room, String> nameColumn; // Столбец для отображения названия комнаты
private Stage stage; // Окно приложения
private ObservableList<Servicee> uslugi = FXCollections.observableArrayList(); // Список выбранных услуг

// Метод инициализации контроллера при загрузке FXML-файла
public void initialize(URL location, ResourceBundle resources) {
    // Настройка свойств для столбцов таблицы комнат
    descriptionColumn.setCellValueFactory(new PropertyValueFactory<>("description"));
    priceColumn.setCellValueFactory(new PropertyValueFactory<>("price"));
    nameColumn.setCellValueFactory(new PropertyValueFactory<>("nameR"));

    // Настройка отображения фотографий комнат в столбце
    photoColumn.setCellValueFactory(cellData -> {
        ImageView imageView = new ImageView();
        imageView.setFitWidth(150);
        imageView.setFitHeight(150);
        Room room = cellData.getValue();
        Image image = new Image(getClass().getResourceAsStream("/photo/" + room.getPhoto()));
        imageView.setImage(image);
        return new SimpleObjectProperty<>(imageView);
    });

    roomTable.setContextMenu(contextMenu); // Привязка контекстного меню к таблице

    // Настройка свойств для столбцов таблицы услуг
    colID.setCellValueFactory(new PropertyValueFactory<>("idS"));
    colYsl.setCellValueFactory(new PropertyValueFactory<>("name"));
    colPrice.setCellValueFactory(new PropertyValueFactory<>("price"));
    tabYsl.setItems(uslugi); // Привязка списка услуг к таблице
    tabYsl.getSelectionModel().setSelectionMode(SelectionMode.MULTIPLE);

    // Обработчик выбора комнаты в таблице
    roomTable.getSelectionModel().selectedItemProperty().addListener((obs, oldSelection, newSelection) -> {
        if (newSelection != null) {
            price.setText(String.format("%.2f", newSelection.getPrice()));
        }
    });
}

// Метод для получения данных о доступных комнатах
private ObservableList<Room> getRoomData(int type) {
    ObservableList<Room> data = FXCollections.observableArrayList();

    String url = "jdbc:mysql://localhost:3306/kyrsach";
    String user = "root";
    String password = "";

```

```

uslugi.clear(); // Очистка списка услуг

try (Connection connection = DriverManager.getConnection(url, user, password);
    PreparedStatement statement = connection.prepareStatement("SELECT idS, nameS, priceS FROM
Service"));
    ResultSet resultSet = statement.executeQuery() {

    // Заполнение списка услуг из базы данных
    while (resultSet.next()) {
        String name = resultSet.getString("nameS");
        double price = resultSet.getDouble("priceS");
        int idS = resultSet.getInt("idS");
        услуги.add(new Servicee(name, price, idS));
    }
} catch (SQLException e) {
    e.printStackTrace();
    label.setText("Произошла ошибка при получении данных из базы.");
}

LocalDate dateZ = date1.getValue();
LocalDate dateV = date2.getValue();

// Проверка корректности введенных дат
if (dateZ.isBefore(LocalDate.now())) {
    // Вывод ошибки, если дата начала проживания меньше текущей даты
    Alert alert = new Alert(Alert.AlertType.ERROR);
    alert.setTitle("Ошибка");
    alert.setHeaderText("Введите корректные даты");
    alert.setContentText("Дата начала не может быть меньше сегодняшней даты");
    alert.showAndWait();
    return data;
}

// Запрос к базе данных для получения доступных комнат
try (Connection connection = DriverManager.getConnection(url, user, password);
    PreparedStatement statement = connection.prepareStatement("SELECT idR, photo, descriptionR, priceR,
nameR FROM Room WHERE type = ? AND ((dateZ > ? AND dateZ > ?) OR (dateV < ? AND dateV < ?) OR
(dateZ IS NULL AND dateV IS NULL)))") {

    statement.setInt(1, type);
    statement.setDate(2, java.sql.Date.valueOf(dateV));
    statement.setDate(3, java.sql.Date.valueOf(dateZ));
    statement.setDate(4, java.sql.Date.valueOf(dateZ));
    statement.setDate(5, java.sql.Date.valueOf(dateV));

    ResultSet resultSet = statement.executeQuery();
    // Заполнение списка комнат из базы данных
    while (resultSet.next()) {
        // Извлекаем данные из результирующего набора
        String nameR = resultSet.getString("nameR");
        int idR = resultSet.getInt("idR");
        String photo = resultSet.getString("photo");
        String description = resultSet.getString("descriptionR");
        double price = resultSet.getDouble("priceR");

        // Создаем объект комнаты и добавляем его в список данных
        data.add(new Room(photo, description, price, idR, nameR));
    }
}

```

```

// Обработка исключения при получении данных из базы
} catch (SQLException e) {
    e.printStackTrace();
    label.setText("Произошла ошибка при получении данных из базы.");
}
// Возвращаем список данных
return data;
}

// Метод для отображения комнат в таблице в зависимости от типа
@FXML
private void showRooms(int type) {
    // Проверка корректности введенных дат
    if (areDatesValid()) {
        // Устанавливаем данные комнат в таблицу
        roomTable.setItems(getRoomData(type));
    }
}

// Методы для отображения комнат различных типов
@FXML
private void showSingleRooms() {
    showRooms(1); // одноместные номера
}
@FXML
private void showDoubleRooms() {
    showRooms(2); // двухместные номера
}
@FXML
private void showTripleRooms() {
    showRooms(3); // трехместные номера
}

// Методы, вызываемые при открытии окна заказа
@FXML
private void openZakazWindow() {
    updatePrice();
}
@FXML
private void addYsl() {
    updatePrice();
}

// Метод для обновления отображаемой цены
private void updatePrice() {
    // Получаем выбранную комнату
    Room selectedRoom = roomTable.getSelectionModel().getSelectedItem();
    // Проверка наличия выбранной комнаты и корректности введенных дат
    if (selectedRoom != null && areDatesValid()) {
        // Получаем значения выбранных дат
        LocalDate date1Value = date1.getValue();
        LocalDate date2Value = date2.getValue();

        // Рассчитываем разницу в днях между выбранными датами
        long daysDifference = ChronoUnit.DAYS.between(date1Value, date2Value) + 1;

        // Рассчитываем общую стоимость с учетом выбранной комнаты и дополнительных услуг
        double totalPrice = selectedRoom.getPrice() * daysDifference;
    }
}

```

```

        for (Servicee service : tabYsl.getSelectionModel().getSelectedItem()) {
            totalPrice += service.getPrice();
        }
        // Отображаем общую стоимость в текстовом поле
        price.setText(String.format("%.2f", totalPrice));
    }
}

// Метод для выполнения бронирования комнаты
@FXML
private void bron() {
    // Получаем выбранную комнату
    Room selectedRoom = roomTable.getSelectionModel().getSelectedItem();

    // Проверка наличия выбранной комнаты
    if (selectedRoom != null) {
        // Получаем значения выбранных дат и другие параметры
        double totalPrice = Double.parseDouble(price.getText().replaceAll(";", "."));
        LocalDate dateZ = date1.getValue();
        LocalDate dateV = date2.getValue();
        int Klient_idK = RegKl.idK;
        int Room_IdR = selectedRoom.getIdR();

        // Строка подключения к базе данных
        String url = "jdbc:mysql://localhost:3306/kyrsach";
        String user = "root";
        String password = "";

        try (Connection connection = DriverManager.getConnection(url, user, password);
            // Подготовленный запрос для проверки существования комнаты
            PreparedStatement checkRoomStatement = connection.prepareStatement("SELECT * FROM Room
WHERE idR = ?");
            // Подготовленный запрос для вставки данных о бронировании
            PreparedStatement statement = connection.prepareStatement("INSERT INTO Booking (idB, dz, dv,
fullPrice, Klient_idK, Room_IdR, status) VALUES (?, ?, ?, ?, ?, ?, ?)", Statement.RETURN_GENERATED_KEYS))
        {

            // Проверка существования комнаты с заданным идентификатором
            checkRoomStatement.setInt(1, Room_IdR);
            ResultSet checkRoomResultSet = checkRoomStatement.executeQuery();

            if (checkRoomResultSet.next()) {
                // Установка параметров для запроса по бронированию
                statement.setInt(1, 0);
                statement.setDate(2, java.sql.Date.valueOf(dateZ));
                statement.setDate(3, java.sql.Date.valueOf(dateV));
                statement.setDouble(4, totalPrice);
                statement.setInt(5, Klient_idK);
                statement.setInt(6, Room_IdR);
                statement.setInt(7, 1);

                // Выполнение запроса по бронированию
                statement.executeUpdate();

                // Получение сгенерированных ключей (в данном случае, идентификатор брони)
                ResultSet generatedKeys = statement.getGeneratedKeys();
            }
        }
    }
}

```

```

        if (generatedKeys.next()) {
            int newIdB = generatedKeys.getInt(1);

            // Подготовленный запрос для вставки данных о выбранных услугах
            try (PreparedStatement statement2 = connection.prepareStatement("INSERT INTO Boo_Ser (idBS,
idBoo, idSer) VALUES (?, ?, ?)")) {
                // Цикл для добавления записей о выбранных услугах
                for (Service service : tabYsl.getSelectionModel().getSelectedItem()) {
                    statement2.setInt(1, 0); // idBS - автоинкрементная
                    statement2.setInt(2, newIdB);
                    statement2.setInt(3, service.getIdS());
                    statement2.addBatch();
                }
                // Выполнение пакетной вставки данных
                statement2.executeBatch();

                // Если всё прошло успешно, отображаем окно "Комната забронирована!"
                stage = (Stage) tabYsl.getScene().getWindow();
                showConfirmationAlert("Комната забронирована!", stage);
            }
        }
    } else {
        // Обрабатываем ситуацию, когда комната не существует
        label.setText("Ошибка: комната не существует.");
    }
} catch (SQLException e) {
    e.printStackTrace();
    label.setText("Произошла ошибка при добавлении заказа.");
}
}
}

// Метод для отображения информационного диалогового окна подтверждения
private void showConfirmationAlert(String message) {
    // Создание объекта всплывающего диалогового окна типа INFORMATION
    Alert alert = new Alert(Alert.AlertType.INFORMATION);
    alert.setTitle("Подтверждение");
    alert.setHeaderText(message);
    // Отображение окна и ожидание закрытия пользователем
    alert.showAndWait();
}

// Перегруженный метод для отображения информационного диалогового окна подтверждения с
закрытием определенного окна
private void showConfirmationAlert(String message, Stage stage) {
    // Создание объекта всплывающего диалогового окна типа INFORMATION
    Alert alert = new Alert(Alert.AlertType.INFORMATION);
    alert.setTitle("Подтверждение");
    alert.setHeaderText(message);
    // Отображение окна и ожидание закрытия пользователем
    alert.showAndWait();

    // Закрытие указанного окна
    stage.close();
}

// Метод для проверки корректности введенных дат
private boolean areDatesValid() {
    // Получение значений выбранных дат

```

```

LocalDate date1Value = date1.getValue();
LocalDate date2Value = date2.getValue();

// Проверка наличия значений и соответствия порядка дат
if (date1Value == null || date2Value == null || date2Value.isBefore(date1Value)) {
    // В случае ошибки создается всплывающее диалоговое окно с сообщением об ошибке
    Alert alert = new Alert(Alert.AlertType.ERROR);
    alert.setTitle("Ошибка");
    alert.setHeaderText(null);
    alert.setContentText("Введите корректные даты");
    // Отображение окна и ожидание закрытия пользователем
    alert.showAndWait();
    return false; // Возвращаем false, чтобы указать, что даты не корректны
}
return true; // Возвращаем true, если даты корректны
}
}

```

3.2.2 Дополнительные классы.

Для того что бы в окне заказа(рис. 18) в таблице комнат у колонки «Описание» текст не выходил за поля колонки, а переносился на другие строки я реализовала такой дополнительный метод(рис. 20).

```

public class TextTableCellFactory<S, T> implements Callback<TableColumn<S, T>, TableCell<S, T>> {

    // Метод call вызывается при создании ячейки для столбца таблицы.
    @Override
    public TableCell<S, T> call(TableColumn<S, T> param) {
        // Создаем новую ячейку, расширяя базовый класс TableCell.
        return new TableCell<>() {
            // Переопределяем метод updateItem для обновления содержимого ячейки.
            1 usage
            @Override
            protected void updateItem(T item, boolean empty) {
                // Вызываем метод родительского класса.
                super.updateItem(item, empty);
                // Проверяем, пуста ли ячейка или содержит null.
                if (item == null || empty) {
                    setText(null); // Если да, устанавливаем текст ячейки в null.
                } else {
                    setText(item.toString()); // Иначе устанавливаем текст ячейки как строковое представление данных.
                    setWrapText(true); // Включаем перенос текста, если необходимо.
                }
            }
        };
    }
}

```

Рисунок 20. TextTableCellFactory

Дополнительный класс для вывода данных о комнатах в таблицу(рис. 18).

```
public class Room {  
    // Свойство для хранения фотографии комнаты.  
    private String photo;  
    // Свойство для хранения описания комнаты.  
    private String description;  
    // Свойство для хранения цены на проживание в комнате.  
    private double price;  
    // Уникальный идентификатор комнаты.  
    private int idR;  
    // Название комнаты.  
    private String nameR;  
    // Конструктор класса, инициализирующий все свойства комнаты.  
    public Room(String photo, String description, double price, int idR, String nameR) {  
        this.photo = photo;  
        this.description = description;  
        this.price = price;  
        this.idR = idR;  
        this.nameR = nameR;  
    }  
    // Метод для получения уникального идентификатора комнаты.  
    public int getIdR() { return idR; }  
  
    // Метод для получения фотографии комнаты.  
    public String getPhoto() { return photo; }  
  
    // Метод для установки фотографии комнаты.  
    public void setPhoto(String photo) { this.photo = photo; }  
  
    // Метод для получения описания комнаты.  
    public String getDescription() {return description;}  
  
    // Метод для установки описания комнаты.  
    public void setDescription(String description) {this.description = description;}  
  
    // Метод для получения цены на проживание в комнате.  
    public double getPrice() {return price;}  
  
    // Метод для установки цены на проживание в комнате.  
    public void setPrice(double price) { this.price = price;}  
  
    // Метод для получения названия комнаты.  
    public String getNameR() {return nameR;}  
  
    // Метод для установки названия комнаты.  
    public void setNameR(String nameR) { this.nameR = nameR;}  
}
```


Похожий класс(рис. 21) я так же написала для того, что бы отображать услуги в таблице(рис. 18)

```
public class Servicee {  
    // Свойство для хранения названия услуги.  
    2 usages  
    private String name;  
    // Свойство для хранения цены на предоставление услуги.  
    2 usages  
    private double price;  
    // Уникальный идентификатор услуги.  
    2 usages  
    private int idS;  
  
    // Конструктор класса, инициализирующий все свойства услуги.  
    1 usage  
    public Servicee(String name, double price, int idS) {  
        this.name = name;  
        this.price = price;  
        this.idS = idS;  
    }  
  
    // Метод для получения названия услуги.  
    public String getName() { return name; }  
  
    // Метод для получения уникального идентификатора услуги.  
    1 usage  
    public int getIdS() { return idS; }  
  
    // Метод для получения цены на предоставление услуги.  
    1 usage  
    public double getPrice() { return price; }  
}
```

Рисунок 21. Servicee

3.3.3. Разработка интерфейса для сотрудника.

Сотрудник гостиницы должен иметь возможность видеть все заказы, с информацией о них. Так же он будет способен изменять дату выселения(уменьшить её, если она не меньше сегодняшнего дня или увеличить), соответственно от этого будет меняться цена, увеличиваться или уменьшаться. Сотрудник сможет добавлять новые заказы или же закрывать уже созданные.

3.3.3.1 Основное окно сотрудника.

После авторизации как сотрудника(рис. 7) открывается новое окно admin.fxml.

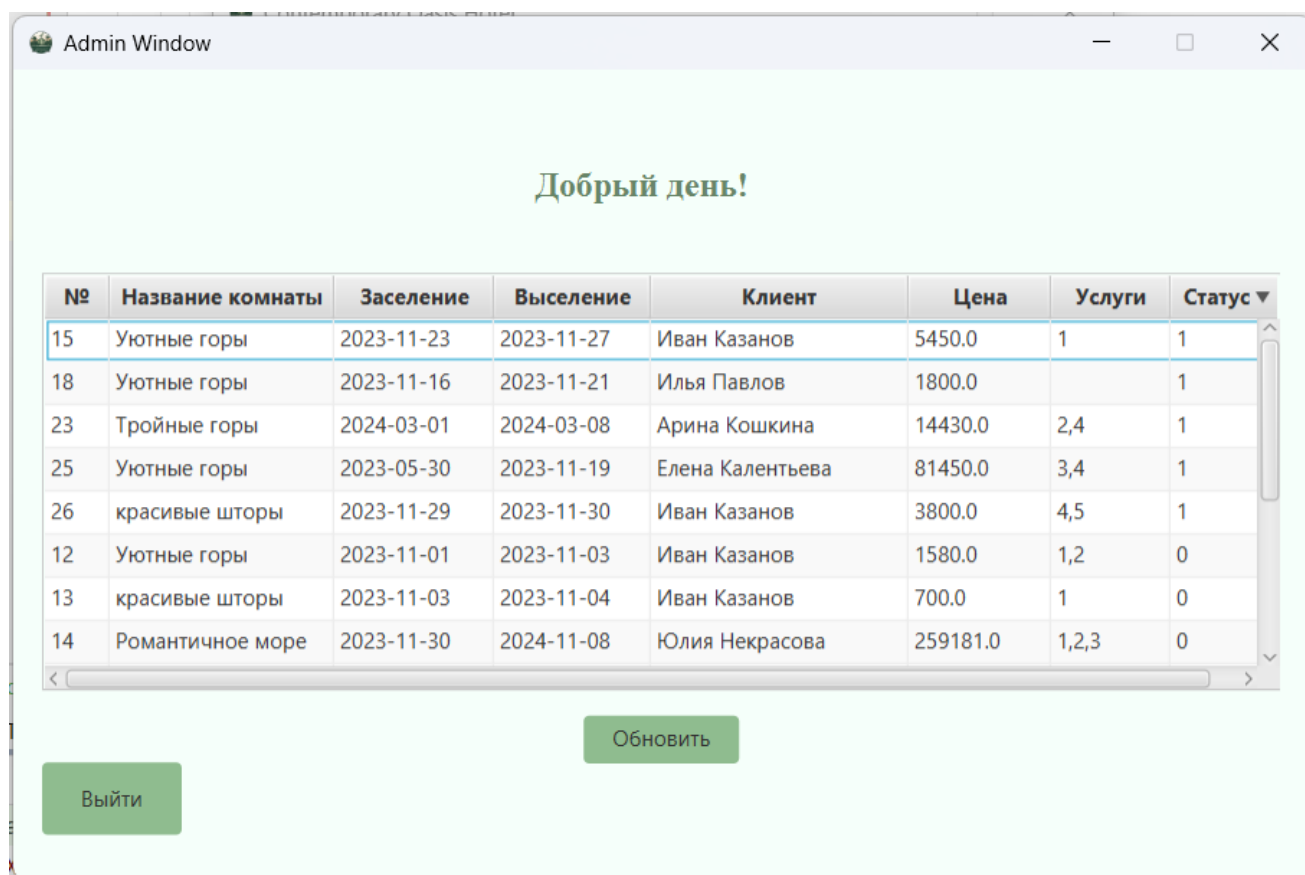


Рисунок 22. admin.fxml.

В этом окне(рис. 22) присутствуют: приветствие, которое меняется в зависимости от времени, таблица заказов с информацией о них. Кнопка «Обновить», которая при нажатии обновляет данные, если они вдруг изменились в базе данных. И кнопка «Выйти» которая возвращает окно входа(рис. 6). Услуги отсортированы по статусу, первые идут те, у которых статус = 1.

В таблице присутствует контекстное меню(рис. 23).

№	Название комнаты	Заселение	Выселение	Клиент	Цена	Услуги	Статус ▾
15	Уютные горы	2023-11-23	2023-11-27	Иван Казанов	5450.0	1	1
18	Уютные горы	2023-11-16	2023-11-21	Илья Павлов	1800.0		1
23	Тройные горы	2024-03-01	2024-03-08	Ирина	14430.0	2,4	1
25	Уютные горы	2023-05-30	2023-11-1	Ирина	81450.0	3,4	1
26	красивые шторы	2023-11-29	2023-11-3	Ирина	3800.0	4,5	1
12	Уютные горы	2023-11-01	2023-11-0	Ирина	1580.0	1,2	0
13	красивые шторы	2023-11-03	2023-11-04	Иван Казанов	700.0	1	0
14	Романтичное море	2023-11-30	2024-11-08	Юлия Некрасова	259181.0	1,2,3	0

Рисунок 23. Контекстное меню.

Здесь реализованы три кнопки «Изменить», которая изменяет дату выезда и обновляет итоговую стоимость, «Добавить заказ» которая дает возможность создать новый заказ в базу данных, «Закрыть заказ» она закрывает заказ(меняет его статус на 0). При этом, если сотрудник попробует изменить/закрыть уже закрытый заказ, будет выведена ошибка(рис. 24).

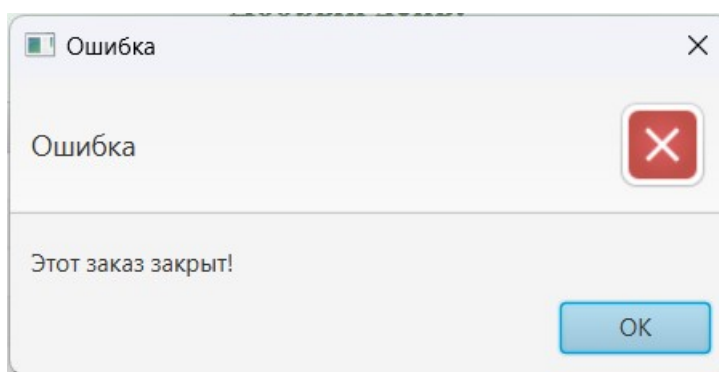


Рисунок 24. Выход.

Код класс AdminController:

```

public class AdminController {
    // Объявление и инициализация элементов интерфейса с использованием аннотации @FXML
    @FXML
    private Label idLabel;
    @FXML
    private TableView<BookingData> idTableZ;
    @FXML
    private TableColumn<BookingData, Integer> idComNum;
    @FXML
    private TableColumn<BookingData, String> idColName;
    @FXML
    private TableColumn<BookingData, Date> idColDateOne;
    @FXML
    private TableColumn<BookingData, Date> idColDateDva;

```

```

@FXML
private TableColumn<BookingData, Double> idColPrice;
@FXML
private TableColumn<BookingData, Integer> idColStatys;
@FXML
private TableColumn<BookingData, String> idColKlient;
@FXML
private TableColumn<BookingData, String> idColYsl;
@FXML
private Button idVon;
@FXML
private Button refreshButton;

// Параметры для подключения к базе данных
private static final String JDBC_URL = "jdbc:mysql://localhost:3306/kyrsach";
private static final String JDBC_USER = "root";
private static final String JDBC_PASSWORD = "";

@FXML
private void initialize() {
    // Вызов метода обновления времени приветствия
    updateTimeGreeting();

    // Заполнение таблицы данными и настройка кнопки обновления
    populateTable();
    setupRefreshButton();

    // Настройка сортировки таблицы по статусу заказа
    idColStatys.setSortType(TableColumn.SortType.DESCENDING);
    idTableZ.getSortOrder().add(idColStatys);

    // Создание контекстного меню для таблицы
    ContextMenu contextMenu = new ContextMenu();

    // Создание пунктов меню
    MenuItem editButton = new MenuItem("Изменить");
    MenuItem addOrderButton = new MenuItem("Добавить заказ");
    MenuItem closeOrderButton = new MenuItem("Закрыть заказ");

    // Обработчик события для пункта "Изменить"
    editButton.setOnAction(e -> {
        // Получение выбранного элемента из таблицы
        BookingData selectedItem = idTableZ.getSelectionModel().getSelectedItem();
        // Проверка, что элемент выбран и не закрыт
        if (selectedItem != null) {
            if (selectedItem.getStatus() == 0) {
                // Вывод сообщения об ошибке, если заказ закрыт
                Alert alert = new Alert(Alert.AlertType.ERROR);
                alert.setTitle("Ошибка");
                alert.setHeaderText("Ошибка");
                alert.setContentText("Этот заказ закрыт!");
                alert.showAndWait();
            } else {
                try {
                    // Открытие окна для изменения данных о заказе
                    FXMLLoader loader = new FXMLLoader(getClass().getResource("izmen.fxml"));
                    Parent root = loader.load();
                }
            }
        }
    });
}

```

```

        IzmenController controller = loader.getController();
        controller.setBookingData(selectedItem);
        controller.initialize();
        Stage stage = new Stage();
        stage.setScene(new Scene(root));
        stage.show();
    } catch (IOException ex) {
        ex.printStackTrace();
    }
}
});

// Обработчик события для пункта "Добавить заказ"
addOrderButton.setOnAction(e -> {
    // Получение выбранного элемента из таблицы
    BookingData selectedItem = idTableZ.getSelectionModel().getSelectedItem();
    // Проверка, что элемент выбран и не закрыт
    if (selectedItem != null) {
        if (selectedItem.getStatus() == 0) {
            // Вывод сообщения об ошибке, если заказ закрыт
            Alert alert = new Alert(Alert.AlertType.ERROR);
            alert.setTitle("Ошибка");
            alert.setHeaderText("Ошибка");
            alert.setContentText("Этот заказ закрыт!");
            alert.showAndWait();
        } else {
            try {
                // Открытие окна для добавления нового заказа
                FXMLLoader loader = new FXMLLoader(getClass().getResource("dob.fxml"));
                Parent root = loader.load();
                Stage stage = new Stage();
                stage.setScene(new Scene(root));
                stage.show();
            } catch (IOException ex) {
                ex.printStackTrace();
            }
        }
    }
});

// Обработчик события для пункта "Закрыть заказ"
closeOrderButton.setOnAction(e -> {
    // Получение выбранного элемента из таблицы
    BookingData selectedItem = idTableZ.getSelectionModel().getSelectedItem();
    // Проверка, что элемент выбран
    if (selectedItem != null) {
        if (selectedItem.getStatus() == 0) {
            // Вывод сообщения об ошибке, если заказ уже закрыт
            Alert alert = new Alert(Alert.AlertType.ERROR);
            alert.setTitle("Ошибка");
            alert.setHeaderText("Ошибка");
            alert.setContentText("Этот заказ закрыт!");
            alert.showAndWait();
        } else {
            selectedItem.setStatus(0);
            try (Connection connection = DriverManager.getConnection(JDBC_URL, JDBC_USER,

```

```

JDBC_PASSWORD)) {
    String query = "UPDATE Booking SET status = 0 WHERE idB = ?";
    PreparedStatement preparedStatement = connection.prepareStatement(query);
    preparedStatement.setInt(1, selectedItem.getIdB());
    preparedStatement.executeUpdate();
} catch (SQLException ex) {
    ex.printStackTrace();
}
}
} else {
    // Вывод сообщения об ошибке, если заказ не выбран
    Alert alert = new Alert(Alert.AlertType.ERROR);
    alert.setTitle("Ошибка");
    alert.setHeaderText("Ошибка");
    alert.setContentText("Необходимо выбрать заказ для закрытия.");
    alert.showAndWait();
}
});
// Добавление пунктов меню в контекстное меню
contextMenu.getItems().addAll(editButton, addOrderButton, closeOrderButton);
// Привязка контекстного меню к таблице
idTableZ.setContextMenu(contextMenu);
}

// Метод для обновления времени приветствия
private void updateTimeGreeting() {
    // Получение текущего времени
    java.time.LocalDateTime currentTime = java.time.LocalDateTime.now();
    // Установка приветствия в зависимости от времени суток
    if (currentTime.isAfter(java.time.LocalDateTime.of(4, 0)) && currentTime.isBefore(java.time.LocalDateTime.of(10, 0))) {
        idLable.setText("Доброе утро!");
    } else if (currentTime.isAfter(java.time.LocalDateTime.of(10, 0)) &&
currentTime.isBefore(java.time.LocalDateTime.of(16, 0))) {
        idLable.setText("Добрый день!");
    } else {
        idLable.setText("Добрый вечер!");
    }
}

// Метод для настройки кнопки обновления
private void setupRefreshButton() {
    refreshButton.setOnAction(e -> {
        // Обновление данных в таблице
        populateTable();
    });
}

// Метод для заполнения таблицы данными из базы данных
private void populateTable() {
    // Создание списка данных для таблицы
    ObservableList<BookingData> bookingDataList = FXCollections.observableArrayList();

    try (Connection connection = DriverManager.getConnection(JDBC_URL, JDBC_USER, JDBC_PASSWORD))
    {
        // Запрос к базе данных для получения данных о заказах
        String query = "SELECT Booking.idB, Room.nameR, Booking.dz, Booking.dv, Booking.fullPrice,

```

```

Booking.status, Klients.nameK, Klients.surnameK, GROUP_CONCAT(Boo_Ser.idSer) AS idSerList " +
    "FROM Booking " +
    "JOIN Room ON Booking.Room_idR = Room.idR " +
    "JOIN Klients ON Booking.Klient_idK = Klients.idK " +
    "LEFT JOIN Boo_Ser ON Booking.idB = Boo_Ser.idBoo " +
    "GROUP BY Booking.idB";
// Выполнение запроса и обработка результатов
try (PreparedStatement statement = connection.prepareStatement(query);
    ResultSet resultSet = statement.executeQuery()) {

    while (resultSet.next()) {
        // Извлечение данных о заказе из результата запроса
        int idB = resultSet.getInt("idB");
        String nameR = resultSet.getString("NameR");
        Date dz = resultSet.getDate("dz");
        Date dv = resultSet.getDate("dv");
        double fullPrice = resultSet.getDouble("fullPrice");
        int status = resultSet.getInt("status");
        String nameK = resultSet.getString("nameK");
        String surnameK = resultSet.getString("surnameK");
        String idSerList = resultSet.getString("idSerList");
        // Создание объекта BookingData и добавление в список
        BookingData bookingData = new BookingData(idB, nameR, dz, dv, fullPrice, new
SimpleIntegerProperty(status), nameK, surnameK, idSerList);
        bookingDataList.add(bookingData);
    }
}
} catch (SQLException e) {
    e.printStackTrace();
}
// Установка данных в таблицу и настройка соответствующих столбцов
idTableZ.setItems(bookingDataList);
idColDateOne.setCellValueFactory(new PropertyValueFactory<>("dz"));
idColDateDva.setCellValueFactory(new PropertyValueFactory<>("dv"));
idComNum.setCellValueFactory(new PropertyValueFactory<>("idB"));
idColName.setCellValueFactory(new PropertyValueFactory<>("nameR"));
idColKlient.setCellValueFactory(new PropertyValueFactory<>("fullName"));
idColPrice.setCellValueFactory(new PropertyValueFactory<>("fullPrice"));
idColYsl.setCellValueFactory(new PropertyValueFactory<>("idSerList"));
idColStatys.setCellValueFactory(new PropertyValueFactory<>("status"));
}

// Метод для закрытия окна
@FXML
private void closeWindow() {
    // Получение текущего окна и его закрытие
    Stage stage = (Stage) idVon.getScene().getWindow();
    stage.close();
}
}

```

Так же присутствует дополнительный класс для корректного отображения данных в таблице:

// Класс для хранения данных бронирования

```
public class BookingData {
    private int idB; // идентификатор бронирования
    private String nameR; // название ресурса
    private Date dz; // дата заезда
    private Date dv; // дата выезда
    private double fullPrice; // полная стоимость
    private SimpleIntegerProperty status; // статус
    private String nameK; // имя клиента
    private String surnameK; // фамилия клиента
    private String idSerList; // список идентификаторов услуг

    // Конструктор для инициализации полей
    public BookingData(int idB, String nameR, Date dz, Date dv, double fullPrice, SimpleIntegerProperty status,
String nameK, String surnameK, String idSerList) {
        this.idB = idB;
        this.nameR = nameR;
        this.dz = dz;
        this.dv = dv;
        this.fullPrice = fullPrice;
        this.status = status;
        this.nameK = nameK;
        this.surnameK = surnameK;
        this.idSerList = idSerList;
    }

    // Геттеры и сеттеры для доступа к полям
    public int getIdB() {
        return idB;
    }
    public String getNameR() {
        return nameR;
    }
    public Date getDz() {
        return dz;
    }
    public Date getDv() {
        return dv;
    }
    public void setDv(Date dv) {
        this.dv = dv;
    }
    public void setIdB(int idB) {
        this.idB = idB;
    }
    public void setNameR(String nameR) {
        this.nameR = nameR;
    }
    public void setDz(Date dz) {
        this.dz = dz;
    }
    public void setFullPrice(double fullPrice) {
        this.fullPrice = fullPrice;
    }
    public void setNameK(String nameK) {
```



```

    this.nameK = nameK;
}
public void setSurnameK(String surnameK) {
    this.surnameK = surnameK;
}
public void setIdSerList(String idSerList) {
    this.idSerList = idSerList;
}
public double getFullPrice() {
    return fullPrice;
}
public int getStatus() {
    return status.get();
}
public SimpleIntegerProperty statusProperty() {
    return status;
}
public void setStatus(int status) {
    this.status.set(status);
}
public String getNameK() {
    return nameK;
}
public String getSurnameK() {
    return surnameK;
}
public String getIdSerList() {
    return idSerList;
}
public String getFullName() {
    return nameK + " " + surnameK;
}
}

```

3.3.3.2 Изменение и добавление заказов.

3.3.3.2.1 Изменение.

Изменение даты выезда происходит в новом окне `izmen.fxml` (рис. 25), которое открывается на кнопку «Изменить» в контекстном меню (рис. 23).

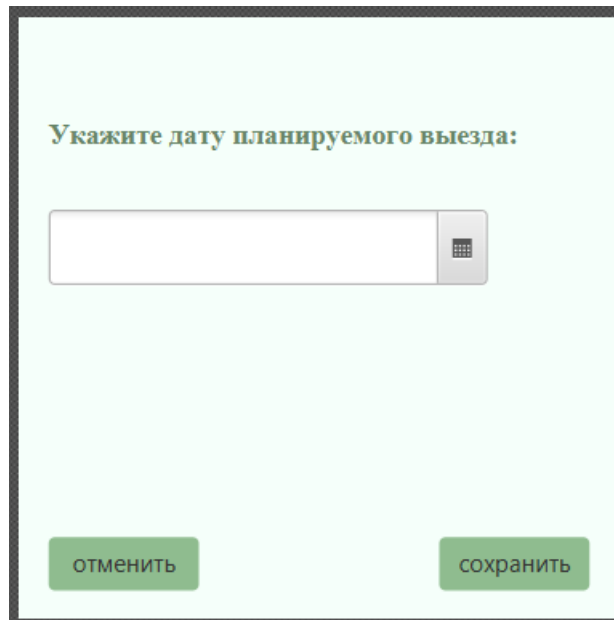


Рисунок 25. `izmen.fxml`.

В `DatePicker` изначально записывается указанная клиентом дата выезда. Её можно поменять, увеличить или уменьшить (не должна быть меньше даты заезда или меньше сегодняшней даты), на кнопку сохранить дата выезда изменяется в базе данных, так же изменяется цена. На кнопку отменить, изменения не сохраняются и окно просто закрывается.

Код класса `IzmenController`:

```
public class IzmenController {  
    // Привязки к элементам пользовательского интерфейса  
    @FXML  
    private DatePicker idDataVon;  
    @FXML  
    private Button idSoxr;  
    @FXML  
    private Button idOtmena;  
  
    // Объект для работы с данными бронирования  
    private BookingData bookingData;  
  
    // Метод для установки данных бронирования  
    public void setBookingData(BookingData bookingData) {  
        this.bookingData = bookingData;  
    }  
}
```

```

// Метод для инициализации компонентов и установки данных
@FXML
public void initialize() {
    // Установка значения DatePicker с текущим значением бронирования
    if (bookingData != null) {
        idDataVon.setValue(bookingData.getDv().toLocalDate());
    } else {
        idDataVon.setValue(null);
    }

    // Обработка события для кнопки "Отмена"
    idOtmena.setOnAction(e -> {
        // Закрытие окна
        Stage stage = (Stage) idOtmena.getScene().getWindow();
        stage.close();
    });
}

// Обработка события для кнопки "Сохранить"
@FXML
public void handleSoxrAction(ActionEvent event) {
    // Вывод сообщения в консоль
    System.out.println("handleSoxrAction method called");

    // Получение текущего значения DatePicker
    LocalDate newDate = idDataVon.getValue();

    // Проверка, является ли новая дата ранее или равна дате заезда
    if (newDate.isBefore(bookingData.getDz().toLocalDate())) {
        // Вывод сообщения об ошибке в консоль
        System.out.println("New date is before dz date");

        // Создание окна с сообщением об ошибке
        Alert alert = new Alert(Alert.AlertType.ERROR);
        alert.setTitle("Ошибка");
        alert.setHeaderText("Ошибка");
        alert.setContentText("Введите корректную дату!");
        alert.showAndWait();
    } else {
        // Обновление значения DatePicker в объекте BookingData
        bookingData.setDv(Date.valueOf(newDate));

        // Вывод сообщения в консоль
        System.out.println("BookingData object updated");

        // Сохранение изменений в базу данных
        BookingDataDAO dao = new BookingDataDAO();
        dao.updateBookingData(bookingData);

        // Закрытие окна
        Stage stage = (Stage) idSoxr.getScene().getWindow();
        stage.close();
    }
}

```

```

    }
}

```

Вспомогательный класс для записи обновлений в базу данных - BookingDataDAO

```

// Класс для работы с данными бронирования в базе данных
public class BookingDataDAO {
    // Метод для обновления данных бронирования
    public void updateBookingData(BookingData bookingData) {
        // Вывод сообщения в консоль
        System.out.println("updateBookingData method called");

        // Установка параметров для подключения к базе данных
        String DB_URL = "jdbc:mysql://localhost:3306/kyrsach";
        String USER = "root";
        String PASS = "";

        // SQL запрос для обновления данных бронирования
        String sql = "UPDATE booking SET dv = ? WHERE idB = ?";
        try (Connection conn = DriverManager.getConnection(DB_URL, USER, PASS);
            PreparedStatement pstmt = conn.prepareStatement(sql)) {
            // Установка значений в подготовленном запросе
            pstmt.setDate(1, new java.sql.Date(bookingData.getDv().getTime()));
            pstmt.setInt(2, bookingData.getIdB());
            pstmt.executeUpdate(); // Выполнение запроса на обновление
        } catch (SQLException e) {
            e.printStackTrace(); // Вывод стека вызовов в случае ошибки
        }
    }
}

```

За изменение цены отвечает триггер update_fullPrice.

3.3.3.2.2 Добавление.

Добавление нового заказа происходит с помощью окна dob.fxml

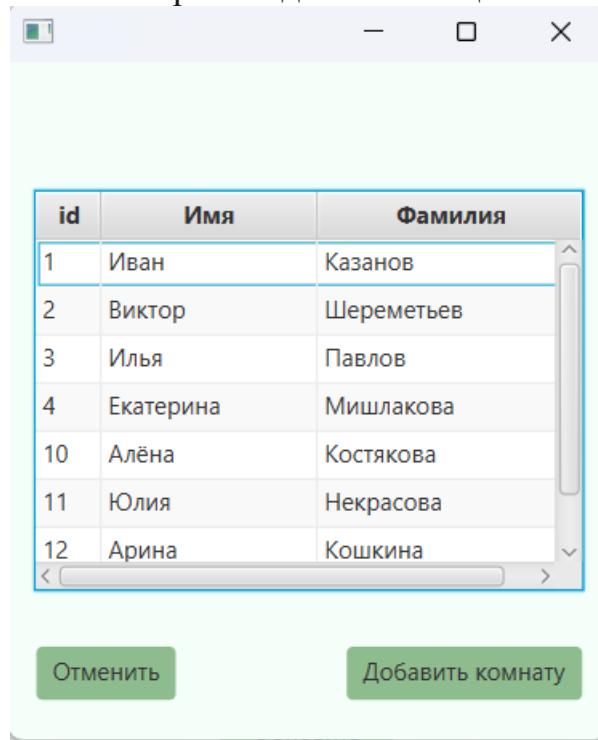


Рисунок 26. dob.fxml

Контроллер этого(рис. 26) окна – DobController.

В этом окне выводятся все зарегистрированные клиенты и сотрудник должен выбрать, на кого должен записаться заказ. При выборе клиента, сотрудник нажимает на кнопку «Добавить комнату» и открывается окно ordeer.fxml(рис. 18). На кнопку «Отменить» окно просто закрывается.

Код DobController:

```
// Контроллер для добавления клиентов
public class DobController implements Initializable {
    // Привязки к элементам пользовательского интерфейса
    @FXML
    private TableView<Klient> idTableKl;
    @FXML
    private TableColumn<Klient, Integer> idID;
    @FXML
    private TableColumn<Klient, String> idNameKl;
    @FXML
    private TableColumn<Klient, String> idFamKl;
    @FXML
    private Button IdDob;
    @FXML
    private Button idOtm;
    private Connection connection;
```

```

// Метод инициализации контроллера
@Override
public void initialize(URL location, ResourceBundle resources) {
    connectToDatabase(); // Подключение к базе данных

    // Установка фабрик значений для столбцов таблицы
    idID.setCellValueFactory(new PropertyValueFactory<>("idK"));
    idNameKl.setCellValueFactory(new PropertyValueFactory<>("nameK"));
    idFamKl.setCellValueFactory(new PropertyValueFactory<>("surnameK"));
    loadData(); // Загрузка данных в таблицу

    // Обработка события для кнопки "Отмена"
    idOtm.setOnAction(e -> {
        ((Node) (e.getSource())).getScene().getWindow().hide(); // Закрытие окна
    });

    // Обработка события для кнопки "Добавить"
    idDob.setOnAction(e -> {
        Klient selectedKlient = idTableKl.getSelectionModel().getSelectedItem(); // Получение выбранного
        клиента
        if (selectedKlient != null) {
            RegKl.idK = selectedKlient.getIdK(); // Установка id выбранного клиента

            try {
                // Загрузка нового окна для оформления заказа
                FXMLLoader loader = new FXMLLoader(getClass().getResource("ordeer.fxml"));
                Parent root = loader.load();
                Scene scene = new Scene(root);
                Stage stage = new Stage();
                stage.setScene(scene);
                stage.show();
            } catch (IOException ex) {
                ex.printStackTrace(); // Вывод стека вызовов в случае ошибки
            }
        }
    });
}

// Метод для подключения к базе данных
private void connectToDatabase() {
    try {
        connection = DriverManager.getConnection("jdbc:mysql://localhost:3306/kyrsach", "root", "");
    } catch (SQLException ex) {
        ex.printStackTrace(); // Вывод стека вызовов в случае ошибки
    }
}

// Метод для загрузки данных в таблицу
private void loadData() {
    try {
        Statement statement = connection.createStatement();
        ResultSet resultSet = statement.executeQuery("SELECT * FROM Klient");
        while (resultSet.next()) {
            idTableKl.getItems().add(new Klient(resultSet.getInt("idK"), resultSet.getString("nameK"),
            resultSet.getString("surnameK")));
        }
    }
}

```

```

    } catch (SQLException ex) {
        ex.printStackTrace(); // Вывод стека вызовов в случае ошибки
    }
}

// Внутренний класс для представления клиента
public static class Klient {
    private final Integer idK;
    private final String nameK;
    private final String surnameK;

    // Конструктор для инициализации полей клиента
    public Klient(Integer idK, String nameK, String surnameK) {
        this.idK = idK;
        this.nameK = nameK;
        this.surnameK = surnameK;
    }

    // Геттеры для доступа к полям клиента
    public Integer getIdK() {
        return idK;
    }
    public String getNameK() {
        return nameK;
    }
    public String getSurnameK() {
        return surnameK;
    }
}
}

```

Вспомогательный класс, который запоминает id выбранного клиента:

```

// Класс, представляющий клиента
public class Client {
    private String idK; // Идентификатор клиента
    private String nameK; // Имя клиента
    private String surnameK; // Фамилия клиента

    // Конструктор для инициализации полей клиента
    public Client(String idK, String nameK, String surnameK) {
        this.idK = idK;
        this.nameK = nameK;
        this.surnameK = surnameK;
    }

    // Геттеры для доступа к полям клиента
    public String getIdK() {
        return idK;
    }
    public String getNameK() {
        return nameK;
    }
    public String getSurnameK() {
        return surnameK;
    }
}

```

```
// Переопределение метода toString для представления объекта в виде строки  
@Override  
public String toString() {  
    return idK + " - " + nameK + " " + surnameK;  
}  
}
```

После того как клиент выбран, открывается order.fxml(рис. 18) и выполняются те же методы как и для создания заказа со стороны клиента.

4. Руководство пользователя.

Первым делом пользователь должен авторизоваться как клиент или как сотрудник.

Если это клиент, открывается окно авторизации, куда пользователь может указать свой номер телефона и пароль. Если же пользователь не зарегистрирован, то он должен нажать на кнопку «Зарегистрироваться» и внести туда свои данные. После создания аккаунта опять открывается окно авторизации.

Когда пользователь авторизовался открывается окно оформления заказа. Сначала он должен выбрать планируемую дату заезда, она не должна быть меньше сегодняшнего дня и дату выезда, она не должна быть меньше даты заезда. После того как даты заполнены пользователю надо выбрать какого типа комнату он хочет заказать(посмотреть можно все), нажав на одну из кнопок «Одноместные комнаты», «Двухместные комнаты» или «Трехместные комнаты». Как только пользователь выбирает понравившуюся комнату он должен ПКЛ нажать на неё и нажать на кнопку «Забронировать», после чего в итоговой стоимости будет рассчитана цена номера на то количество дней, которое будет проживать клиент. Если клиенту нужны дополнительные услуги, он может выбрать одну или несколько (через Ctrl) услуг из таблицы и нажать на кнопку «Добавить услуги», после чего их стоимость складывается с итоговой стоимостью. Если клиента всё устраивает, то он нажимает на кнопку «Забронировать», после чего появляется сообщение, что заказ успешно создан. На этом действия со стороны клиента закончились.

Если пользователь является сотрудником, то он должен ввести свой логин и пароль, после чего нажимает на кнопку «Авторизоваться». После этого открывается окно для просмотра заказов. Сотрудник может выбрать какойнибудь заказ и изменить/закрыть его, если же заказ уже закрыт, то при попытке сделать с ним какое-либо действие, то появляется окно с ошибкой. Если сотруднику надо изменить дату выезда заказа, то он должен выбрать нужную ему строчку, нажать на неё ПКЛ и выбрать кнопку «Изменить». Откроется окно с первоначальной датой выезда, которую можно изменить в большую или меньшую сторону, после чего надо нажать на кнопку «Сохранить», тогда изменения сохранятся в базу данных. Новая дата не может быть меньше даты заезда. Если сотруднику надо закрыть заказ, то он так же должен нажать на нужную строчку ПКЛ и выбрать кнопку «Закрыть заказ», после чего статус заказа меняется на 0 и комната становится свободной на эти даты. Что бы сотруднику добавить новую запись, он должен нажать ПКЛ на таблице и выбрать «Добавить заказ», после чего появляется окно со списком зарегистрированных клиентов. Сотрудник выбирает на кого должен быть оформлен заказ и нажимает на кнопку «Создать комнату». Далее он так же как и клиент должен выбрать нужные дату и тип комнаты, после забронировать комнату на ПКЛ и по желанию добавить доп. услуги, после чего он нажимает на кнопку «Забронировать» следом за которой появляется сообщение, что заказ успешно создан. В окне сотрудника так же присутствует

кнопка «Обновить» которая позволяет обновить базу данных, если там произошли какие-либо изменения.

Заключение.

В ходе выполнения данной курсовой работы была проведена обширная работа по проектированию и разработке базы данных для гостиницы. Современные гостиничные предприятия сталкиваются с разнообразными вызовами в управлении информацией о номерах, бронированиях, и услугах. Моя работа нацелена на решение этих проблем через создание эффективной структуры базы данных.

Основное внимание уделялось определению основных сущностей, таких как номера, гости, бронирования, и услуги, а также их взаимосвязям. Разработанная структура базы данных предоставляет надежное хранение и оперативную обработку информации. Важным элементом стало внедрение многопользовательского доступа с разделением прав доступа для различных ролей, что обеспечивает эффективное функционирование гостиничного предприятия.

Кроме того, разработаны интерфейсы для клиентов и сотрудников гостиницы, что создает удобные инструменты для взаимодействия с системой. Успешная реализация проекта предоставляет возможность эффективного управления доступностью номеров, адекватного реагирования на изменения в бронированиях, и улучшения обслуживания клиентов.

Таким образом, данная работа подчеркивает актуальность и значимость разработки базы данных для гостиниц в современном туристическом бизнесе. Эффективное управление информацией о номерах, услугах, и бронированиях, а также создание удобного интерфейса для клиентов и персонала гостиницы, способствует оптимизации бизнес-процессов и повышению уровня обслуживания, что является ключевым фактором для успешной деятельности в данной индустрии.

Список литературы.

1. Шилдт Г. Java. Полное руководство / Шилдт Г. — 12-е издание. — : Диалектика-Вильямс, 2022 — 1344 с.
2. Прохоренок, Н. А. JavaFX — : БХВ, 2020 — 768 с.
3. Фиайли К. SQL: Руководство по изучению языка / — : ДМК Пресс, 2010 — 345 с.
4. Ванстрас К., Чэнсон С./ GitHub : — URL: <https://github.com/> (дата обращения: 24.11.2023).