**Курсовой проект на тему : «Система аренды недвижимости»»**

Веб-приложение на Java с использованием Spring Boot, предназначенное для управления недвижимостью, пользователями, риэлторами и сделками.

В нем реализованы функции регистрации, аутентификации, работы с объектами недвижимости, запросами клиентов, профилями риэлторов и пользователей, а также управление сделками и платежами.

**Цели и задачи**

Разработка реляционной базы данных, обеспечивающей хранение и управление данными о недвижимости, пользователях, риэлторов. База данных должна минимизировать избыточность данных, обеспечивать целостность информации и поддерживать возможность масштабирования.

Для осуществления данной работы следует выполнить следующие задачи:

1. Спроектировать структуру базы данных.
2. Обеспечить взаимосвязь таблиц для эффективного управления данными.
3. Реализовать функции управления данными: создание, обновление и удаление записей.

Реализованная база данных станет основой серверной части информационной системы автошколы, обеспечивая быстрый доступ к данным, согласованное взаимодействие всех компонентов.

**Структура базы данных**

Структура базы данных, включающая следующие сущности:

* + Недвижимость (apartaments)

Хранит информацию об объектах недвижимости, доступных для покупки или продажи. Это может быть квартира, дом, коммерческое помещение и т.д. (id, цена, кв метров, тип недвижимости, расположение)

* + Сделки (payments)

Фиксирует факт сделки между клиентом и риэлтором по определённому объекту недвижимости.. (id, id риэлтора, id клиента, id недвижимости, Дата сделки)

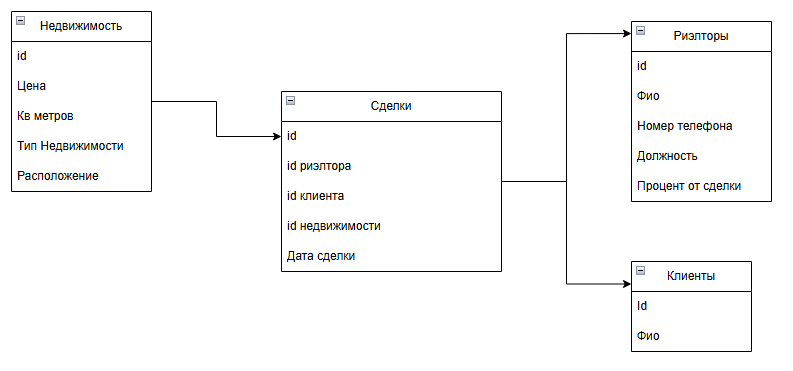
* + Клиенты (users)

Содержит данные о клиентах, которые участвуют в сделках. (id, Фио)

* + Риелторы (rental\_requests)

Хранит информацию о сотрудниках агентства недвижимости, которые занимаются сопровождением сделок. (id, Фио, номер телефонп, должность, процент от сделки)

**ER диаграмма**



**Анализ связей между таблицами**

Блок-схема показывает следующие отношения:

1. Сделки **→** Недвижимость

Каждая сделка привязана к одному объекту недвижимости..

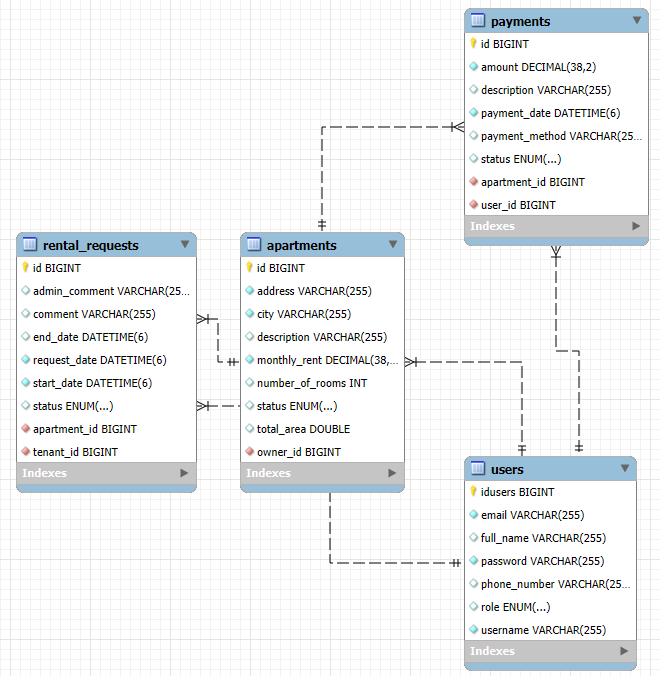
1. Сделки **→** Риэлторы

Сделку сопровождает один риэлтор.

1. Сделки **→** Клиенты

Сделку совершает один клиент.

**Физическая модель**



Использование нормальных форм

Таблицы и их связи:

1. users

Основной справочник пользователей.

Поля: idusers, email, full\_name, password, phone\_number, role, username.

Связи:

* apartments.owner\_id → users.idusers — владелец квартиры.
* rental\_requests.tenant\_id → users.idusers — арендатор (запрос аренды).
* payments.user\_id → users.idusers — пользователь, совершивший платёж.

2. apartments

Хранит данные о квартирах.

Поля: id, address, city, description, monthly\_rent, status, number\_of\_rooms, total\_area, owner\_id.

Связи:

* owner\_id → users.idusers — кто владеет квартирой.
* rental\_requests.apartment\_id → apartments.id — на какую квартиру подан запрос.
* payments.apartment\_id → apartments.id — за какую квартиру сделан платёж.

3. rental\_requests

Хранит информацию о запросах на аренду.

Поля: id, admin\_comment, comment, end\_date, request\_date, start\_date, status, apartment\_id, tenant\_id.

Связи:

* apartment\_id → apartments.id — арендуемая квартира.
* tenant\_id → users.idusers — пользователь, сделавший запрос.

4. payments

Хранит информацию об оплатах.

Поля: id, amount, description, payment\_date, payment\_method, status, apartment\_id, user\_id.

Связи:

* apartment\_id → apartments.id — платёж за квартиру.
* user\_id → users.idusers — пользователь, оплативший.

1. Таблица apartments (Квартиры)

Назначение:  
Хранит информацию о сдаваемых квартирах.

Основные поля:

* id — уникальный идентификатор квартиры.
* address — точный адрес квартиры.
* city — город, в котором расположена квартира.
* description — описание квартиры (удобства, особенности).
* monthly\_rent — стоимость аренды в месяц.
* number\_of\_rooms — количество комнат.
* total\_area — общая площадь (в м²).
* status — текущий статус квартиры (например, доступна / занята).
* owner\_id — ID пользователя (из таблицы users), который владеет квартирой.

2. Таблица users (Пользователи)

Назначение:  
Содержит данные всех пользователей системы: арендаторов, владельцев квартир, администраторов и т.д.

Основные поля:

* idusers — уникальный ID пользователя.
* email, username, password — данные для авторизации.
* full\_name — полное имя пользователя.
* phone\_number — контактный номер.
* role — роль пользователя в системе (например, tenant, owner, admin).

3. Таблица rental\_requests (Запросы на аренду)

Назначение:  
Отражает заявки пользователей на аренду квартиры.

Основные поля:

* id — уникальный ID запроса.
* request\_date — дата подачи заявки.
* start\_date — дата предполагаемого начала аренды.
* end\_date — дата окончания аренды.
* status — статус запроса (например, "в ожидании", "одобрен", "отклонён").
* comment — комментарий от пользователя.
* admin\_comment — комментарий администратора.
* apartment\_id — ID квартиры, которую хотят арендовать.
* tenant\_id — ID пользователя, подавшего запрос (из таблицы users).

4. Таблица payments (Платежи)

Назначение:  
Содержит информацию обо всех платежах, связанных с арендой квартир.

Основные поля:

* id — уникальный ID платежа.
* amount — сумма платежа.
* description — назначение платежа.
* payment\_date — дата и время платежа.
* payment\_method — способ оплаты (например, карта, наличные, перевод).
* status — статус платежа (например, "оплачен", "отменён").
* apartment\_id — ID квартиры, за которую произведён платёж.
* user\_id — ID пользователя, совершившего платёж.

Примеры кода

1.Модель пользователя (User.java)

@Entity

public class User {

@Id

@GeneratedValue(strategy = GenerationType.IDENTITY)

private Long id;

private String username;

private String password;

@Enumerated(EnumType.STRING)

private UserRole role;

// геттеры и сеттеры

}

1. Сервис регистрации пользователя (UserService.java)

@Service

public class UserService {

@Autowired

private UserRepository userRepository;

@Autowired

private PasswordEncoder passwordEncoder;

public void registerNewUser(User user) {

if (userRepository.findByUsername(user.getUsername()).isPresent()) {

throw new RuntimeException("User already exists");

}

user.setPassword(passwordEncoder.encode(user.getPassword()));

userRepository.save(user);

}

}

1. Контроллер аутентификации (AuthController.java)

@Controller

@RequestMapping("/auth")

public class AuthController {

@Autowired

private UserService userService;

@PostMapping("/register")

public String register(@ModelAttribute User user) {

userService.registerNewUser(user);

return "redirect:/auth/login";

}

}

1. Репозиторий недвижимости (PropertyRepository.java)

public interface PropertyRepository extends JpaRepository<Property, Long> {

List<Property> findByStatus(PropertyStatus status);

}

5. Тест на регистрацию пользователя (UserServiceTest.java)

@Test

void registerNewUser\_success() {

User user = new User();

user.setUsername("testuser");

user.setPassword("password");

when(userRepository.findByUsername("testuser")).thenReturn(Optional.empty());

when(passwordEncoder.encode("password")).thenReturn("hashed");

when(userRepository.save(any(User.class))).thenReturn(user);

userService.registerNewUser(user);

assertEquals("hashed", user.getPassword());

verify(userRepository, times(1)).save(user);

}

**Кратко о возможностях проекта**

* Регистрация и вход пользователей (клиенты, риэлторы, администраторы)
* Просмотр, создание, редактирование и удаление объектов недвижимости
* Оформление и обработка запросов на аренду/покупку недвижимости
* Управление сделками и платежами
* Профили пользователей и риэлторов
* Защита данных с помощью Spring Security

**Итог**

Разработанная база данных успешно решает задачи учета и управления процессами аренды недвижимости.