Частное учреждение

профессиональная образовательная организация

ТЕХНИКУМ «БИЗНЕС И ПРАВО»

ЗАЩИЩЕНО С ОЦЕНКОЙ

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(подпись и ФИО руководителя)

«\_\_\_\_\_» \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ 20\_\_\_\_г.

**ОТЧЕТ О ВЫПОЛНЕНИИ ИНДИВИДУАЛЬНОГО ЗАДАНИЯ**

**ПО УЧЕБНОЙ ПРАКТИКЕ**

|  |  |
| --- | --- |
| по ПМ.05 | Проектирование и разработка информационных систем |
| Выполнил | Баранцев И.И. |
| Курс, группа | 3 курс, группа ИС-3 |
| Специальность | 09.02.07 Информационные системы и программирование |
| Руководитель практики | Гонгадзе А.С. |

г. Белореченск

2024

СОДЕРЖАНИЕ

|  |  |
| --- | --- |
| ВВЕДЕНИЕ | 3 |
| 1 ПРОЕКТИРОВАНИЕ ИНФОРМАЦИОННОЙ СИСТЕМЫ | 5 |
| 1.1 Анализ предметной области | 5 |
| 1.2 Техническое задание | 6 |
| 1.3 Эскизный проект | 23 |
| 2 РАЗРАБОТКА И ДОКУМЕНТИРОВАНИЕ ИНФОРМАЦИОННОЙ СИСТЕМЫ | 31 |
| 2.1 Тестирование клиентской части | 31 |
| 2.2 Руководство пользователя | 36 |
| 2.3 Описание страниц | 41 |
| ЗАКЛЮЧЕНИЕ | 51 |
| СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ | 52 |
| ПРИЛОЖЕНИЕ | 53 |

ВВЕДЕНИЕ

В настоящее время, в эпоху цифровых технологий, автоматизация процессов в различных сферах деятельности становится необходимостью для повышения эффективности и конкурентоспособности. Отельный бизнес не является исключением. Агентство недвижимости «TerraNova» стремится предоставить своим клиентам комфортное обслуживание и качественный сервис, а также упростить работу своего персонала. Для этого необходимо разработать и внедрить информационную систему, способную автоматизировать процессы покупки недвижимости, учета клиентов и владельцев, а также обработки сделок.

Объектом исследования является агентство недвижимости «TerraNova», его структура, бизнес-процессы и информационные потоки

Предметом исследования является разработка и внедрение информационной системы для агентства недвижимости «TerraNova», способной автоматизировать процессы недвижимости, учета клиентов и владельцев, а также обработки сделок.

Целью исследования является разработка и внедрение информационной системы для агентства недвижимости «TerraNova», способной автоматизировать процессы недвижимости, учета клиентов и владельцев, а также обработки сделок, с учетом требований заказчика и особенностей объекта исследования.

Для достижения этой цели были сформулированы следующие задачи исследования:

‑ Анализ структуры, бизнес-процессов и информационных потоков агентства недвижимости «TerraNova».

- Определение требований заказчика к информационной системе.

- Разработка концептуальной модели информационной системы.

- Разработка архитектуры информационной системы.

- Разработка базы данных информационной системы.

- Разработка модулей адреса, недвижимости, владельца, сделки, клиента, сотрудника.

- Тестирование и отладка информационной системы.

- Внедрение информационной системы в агентство недвижимости «TerraNova».

- Подготовка документации к информационной системе.

Для достижения поставленных задач были использованы следующие методы:

- Анализ документов и материалов, относящихся к агентству недвижимости «TerraNova».

- Интервьюирование сотрудников агентства недвижимости «TerraNova» для выявления бизнес-процессов и информационных потоков.

- Моделирование бизнес-процессов и информационных потоков с помощью специализированного программного обеспечения.

- Разработка концептуальной модели и архитектуры информационной системы с помощью UML-диаграмм.

- Разработка базы данных информационной системы с помощью SQL-языка.

- Разработка модулей адреса, недвижимости, владельца, сделки, клиента, сотрудника с помощью Java-языка.

- Тестирование и отладка информационной системы с помощью специализированного программного обеспечения.

- Внедрение информационной системы в агентство недвижимости «TerraNova» с учетом требований заказчика и особенностей объекта исследования.

Подготовка документации к информационной системе с учетом требований заказчика и нормативных документов.

1 ПРОЕКТИРОВАНИЕ ИНФОРМАЦИОННОЙ СИСТЕМЫ

1.1 Анализ предметной области

Агентство недвижимости «TerraNova»

Агентство недвижимости «TerraNova» занимается покупкой, продажей, сдачей в аренду объектов недвижимости по договорам с их собственниками. Агентство управляет объектами недвижимости как физических, так и юридических лиц. Собственник может иметь несколько объектов. В случае покупки или аренды клиент может произвести осмотр объекта. В качестве одной из услуг, предлагаемых агентством, является проведение инспектирования текущего состояния объекта для адекватного определения его рыночной цены. По результатам своей деятельности агентство производит отчисления в налоговые органы и предоставляет отчетность в органы государственной статистики.

Администратор системы может активно взаимодействовать с собственниками недвижимости, который он обслуживает. Также он должен предоставлять консультации по вопросам, которые связаны с услугами, предоставляемых компанией. Кроме того, в обязанности администратора входит и учет обращений собственников - и устных, и письменных, которые касаются обслуживания жилищного комплекса и контроля по их выполнению.

Для упрощения работы администратора будет разработана система, которая позволит автоматизировать работу по ведению учета о свободных недвижимостях, о занятых недвижимостях, об особенностях каждой недвижимости и предоставляемых услугах, что позволит уменьшить трудоемкость работы.

Виды запросов в информационной системе:

1. Показать список всех адресов недвижимости

2. Показать недвижимость для продажи

3. Показать перечень объектов недвижимости

4. Показать все типы недвижимости

5. Показать владельца по недвижимости

6. Показать все типы владельцев

7. Показать сведенья о сотруднике

8. Показать сделки клиентов

9. Показать сведения о клиентах

10. Показать всех сотрудников агентства

11. Показать список всех адресов недвижимости определённого района

12. Показать количество свободных объектов

13. Показать все квартиры определенного количества комнат

14. Показать недвижимость определенных размеров

15. Показать сделки сотрудников

16. Реализовать добавление, редактирование, поиск, фильтрацию, удаление информации по адресу

17. Реализовать добавление, редактирование, поиск, фильтрацию, удаление информации по недвижимости

18. Реализовать добавление, редактирование, поиск, фильтрацию, удаление информации по сделкам

19. Реализовать добавление, редактирование, поиск, фильтрацию, удаление информации по клиентам

20. Реализовать добавление, редактирование, поиск, фильтрацию, удаление информации по сотрудникам

1.2 Техническое задание

1 Общие сведенья

1.1 Полное наименование программы и ее условное обозначение

Полное наименование системы: Автоматизирования информационная система агентства недвижимости «TerraNova»

Краткое наименование системы: АИС «TerraNova»

1.2 Шифр темы или шифр (номер) договора

Шифр темы: АИС-ГК-20

Номер договора: №2/34-56-78-002 от 15.04.2024.

1.3 Наименование предприятий разработчика и заказчика системы, их реквизиты

Полное наименование заказчика: Частное учреждение профессиональная образовательная организация ТЕХНИКУМ «БИЗНЕС И ПРАВО».

Сокращённое название: ЧУПОО ТЕХНИКУМ «БИЗНЕС И ПРАВО».

Юридический и почтовый адрес: [352632, Краснодарский Край, г Белореченск, ул. Чапаева, д. 48](https://yandex.ru/maps/?source=exp-counterparty_entity&text=352632,%20%D0%9A%D1%80%D0%B0%D1%81%D0%BD%D0%BE%D0%B4%D0%B0%D1%80%D1%81%D0%BA%D0%B8%D0%B9%20%D0%9A%D1%80%D0%B0%D0%B9,%20%D0%BC.%D1%80-%D0%BD%20%D0%91%D0%B5%D0%BB%D0%BE%D1%80%D0%B5%D1%87%D0%B5%D0%BD%D1%81%D0%BA%D0%B8%D0%B9,%20%D0%B3.%D0%BF.%20%D0%91%D0%B5%D0%BB%D0%BE%D1%80%D0%B5%D1%87%D0%B5%D0%BD%D1%81%D0%BA%D0%BE%D0%B5,%20%D0%B3%20%D0%91%D0%B5%D0%BB%D0%BE%D1%80%D0%B5%D1%87%D0%B5%D0%BD%D1%81%D0%BA,%20%D1%83%D0%BB%20%D0%A7%D0%B0%D0%BF%D0%B0%D0%B5%D0%B2%D0%B0,%20%D0%B4.%2048).

Контактный телефон: +7 (861) 553 39 12.

ИНН: 2310029652

КПП: 236801001.

БИК: 040349722

ОГРН: 1022301620791

Расчетный счет: 407038106.

Полное наименование разработчика: Баранцев Иван Иванович

Юридический и почтовый адрес: Краснодарский край, г Белореченск, ул. Благодатная 64

Контактный телефон: +7 (906) 434 62 48

2 Назначение и цели создания системы

Разрабатываемая система предназначена для автоматизации деятельности агентства недвижимости «TerraNova». Она позволит оптимизировать процессы учета и управления объектами недвижимости, а также взаимодействия с клиентами. Система будет предоставлять информацию о характеристиках объектов, их наличии, ценах и дополнительных услугах.

Цели создания информационной системы:

- Упрощение работы агентов по поиску и подбору объектов недвижимости

- Улучшение сервиса для клиентов за счет быстрого и удобного доступа к информации об объектах

- Повышение эффективности управления агентством недвижимости

Задачи информационной системы:

- Учет всех объектов недвижимости, доступных в агентстве

- Ведение базы данных о клиентах и их запросах

- Автоматизация процесса поиска и подбора объектов в соответствии с критериями клиента

- Контроль за выполнением сделок и фиксация финансовых операций

- Анализ данных о сделках и клиентах для оптимизации работы агентства

3 Характеристика объектов автоматизации

В ходе разработки информационной системы для агентства недвижимости «TerraNova» автоматизируются процессы учета: недвижимости, типов владельцев, сделок, клиентов и сотрудников.

Автоматизируемыми процессами будут такие операции: недвижимость, сделка, клиент, сотрудник. Система будет предоставлять возможность быстрого доступа к информации о недвижимости, клиентов, сделках и клиентах, что позволит упростить работу администратора, уменьшить вероятность ошибок и повысить качество обслуживания.

При внедрении системы будет достигнуто:

- Упрощение процессов сделок

- Повышение эффективности работы персонала

- Увеличение удовлетворенности владельцев

Объекты автоматизации характеризуются:

- Необходимостью доступа к базе данных о номерах, владельцев

- Возможностью быстрого доступа к информации для принятия оперативных решений

- Потребностью к постоянному доступу к сети интернет

4 Требования к системе

4.1 Требования к структуре и функционированию системы

Система должная предполагать наличие следующих модулей, представленных на рисунке 1:

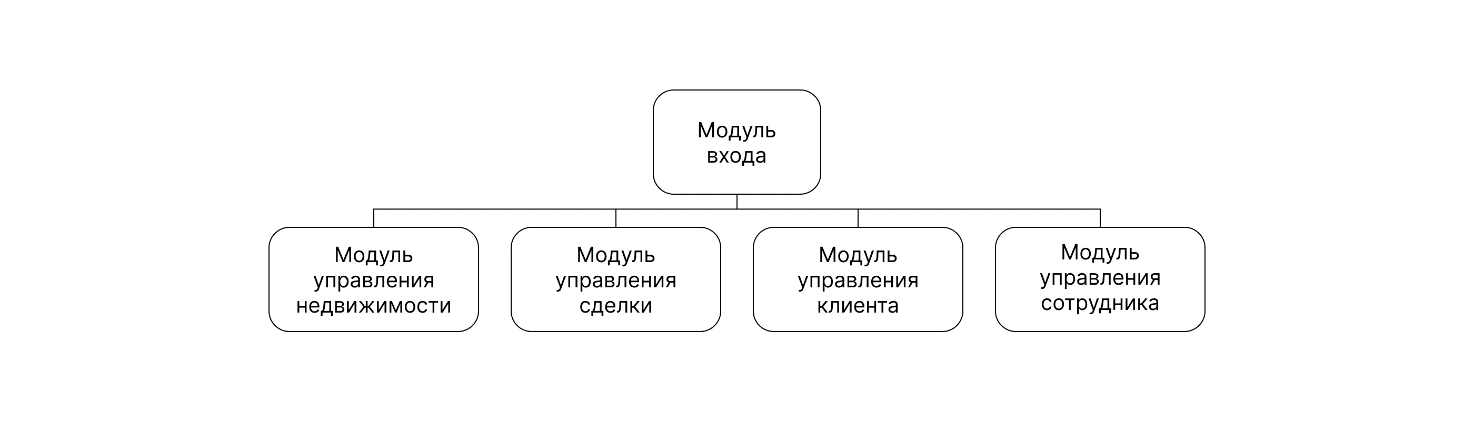


Рисунок 1 - Модули разрабатываемой системы

Серверная часть:

- Модуль авторизации

- Модуль управления недвижимостью

- Модуль управления клиентами

- Модуль управления сделками

- Модуль управления сотрудниками

Клиентская часть:

- Интерфейс администратора (доступ ко всем модулям системы)

- Интерфейс для клиентов (поиска недвижимости, оформления сделок)

На данном этапе необходимо описать примерную архитектуру разрабатываемой системы. Для того, чтобы сделать информационную систему агентства недвижимости, мы будем использовать **Model-View-Controller** (MVC) архитектуру. Чтобы убедиться, что она нам точно подходит рассмотрим ее подробнее.

В **Model-View-Controller** (MVC) архитектуре имеется три звена:

**- Модель (Model)**:

**- Представление (View)**:

**- Контроллер (Controller)**:

Мы выбрали эту архитектуру, так как у нее есть ряд преимуществ перед другими архитектурами:

- это основная логика приложения. Отвечает за данные, методы работы с ними и структуру программы. Модель реагирует на команды из контроллера и выдаёт информацию или изменяет своё состояние.

- передаёт данные в представление.

- задача компонента ‑ визуализация информации, которую он получает от модели. View отображает данные на уровне пользовательского интерфейса.

- обеспечивает взаимодействие с системой. Обрабатывает действия пользователя, проверяет полученную информацию и передаёт её модели. Контроллер определяет, как приложение будет реагировать на действия пользователя. Также контроллер может отвечать за фильтрацию данных и авторизацию.

Для агентства недвижимости подойдет **Model-View-Controller** (MVC), так как она обеспечивает изменять каждый компонент независимо друг от друга для простой разработки и поддержки веб-приложений.

На рисунке 2 представлена структура **Model-View-Controller** архитектуры:

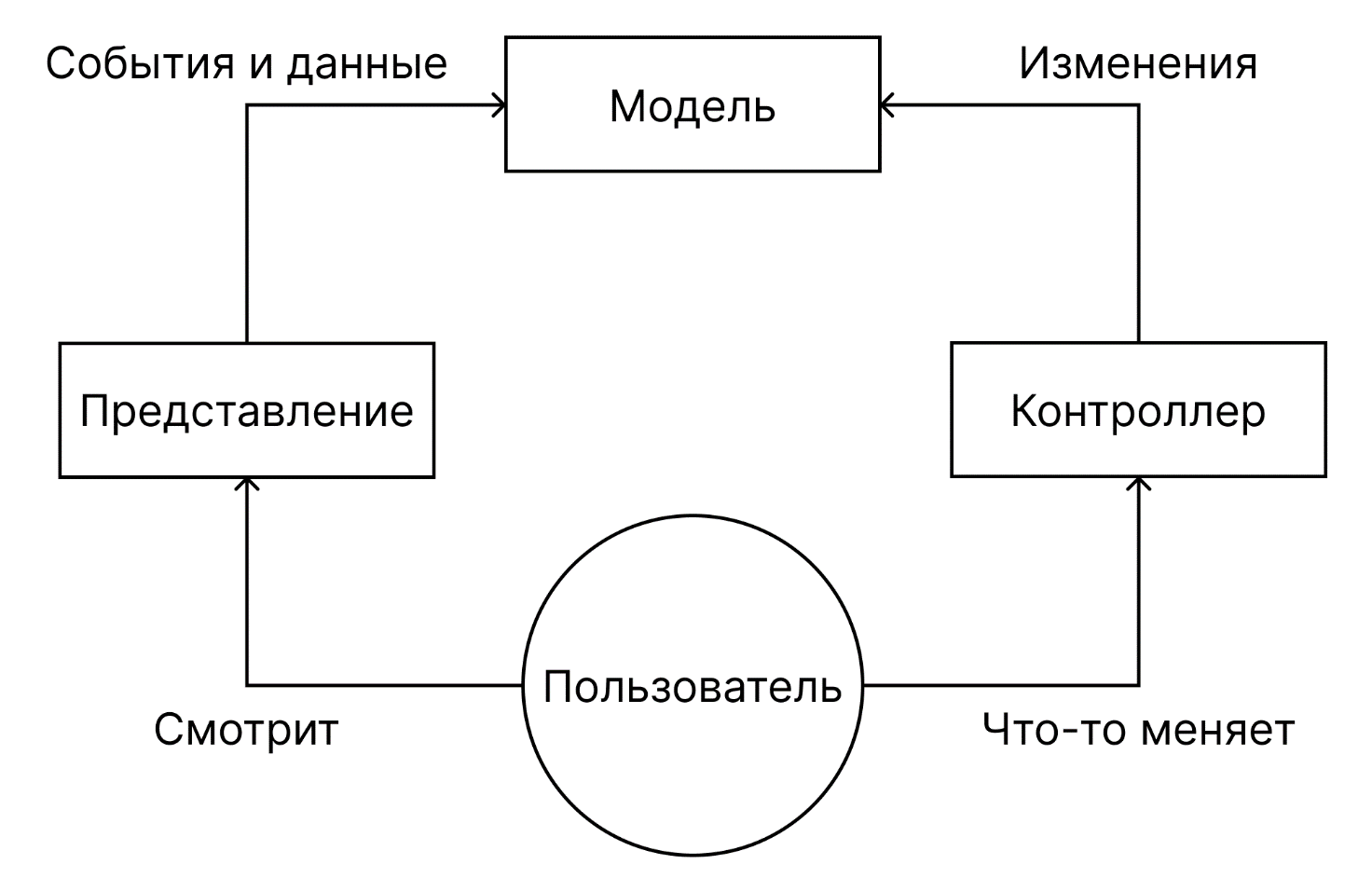


Рисунок 2 - Структура **Model-View-Controller** архитектуры

4.2 Требования к видам обеспечения

Для функционирования информационной системы агентства недвижимости «TerraNova», работа клиента и сервера может быть организована следующим образом:

Клиентская часть:

- На компьютерах сотрудников агентства требуется наличие сервера и клиентских компьютеров, на которых будет производится работа с системой.

- Приложение будет иметь удобный графический интерфейс пользователя, разработанный с учетом требований и ролей пользователей.

- В зависимости от роли, сотрудники смогут выполнять различные операции в приложении, такие как:

- Агенты недвижимости: поиск предложений недвижимости, оформление сделок, управление данными о объектах недвижимости.

Серверная часть:

- Сервер базы данных будет хранить всю информацию о недвижимостях, клиентов, сделках и сотрудниках.

- Серверная часть будет реализована с использованием технологий Java с использованием фреймворка Spring Boot.

- Обеспечение безопасности данных и авторизации пользователей.

В данном случае клиентская часть системы будет представлять собой отдельное приложение, установленное на компьютерах сотрудников агентства недвижимости. Приложение будет иметь графический интерфейс и функционал в зависимости от роли пользователя.

База данных будет содержать информацию о недвижимости, клиентах, сделках и сотрудниках.

При анализе предметной области агентства недвижимости были выделены следующие сущности, на основе которых будет разрабатываться база данных системы:

- Недвижимость

- Сделка

- Клиент

- Сотрудник

База данных информационной системы должна состоять из следующих связанных таблиц:

1. Realty – таблица, содержащая информацию о недвижимостях

2. Deal – таблица, содержащая информацию о сделках

3. Client – таблица, содержащая информацию о клиентах

4. Employee – таблица, содержащая информацию о сотрудниках

На таблицах 1-4 предоставлены эскизы страниц сайта:

Таблица 1 - Таблица Realty

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Имя поля | Тип | Размер | Комментарий | Ограничение |
| 1 | 2 | 3 | 4 | 5 |
| id | INT | auto\_increment | Идентификатор недвижимости | Первичный ключ |
| region | VARCHAR | 100 | Субъект РФ | NOT NULL |
| locality | VARCHAR | 100 | Населённый пункт | NOT NULL |
| street | VARCHAR | 100 | Название улицы | NOT NULL |
| rooms | INT |  | Количество комнат | NOT NULL |
| floors | INT |  | Количество этажей | NOT NULL |
| square | DECIMAL | 10, 2 | Площадь | NOT NULL |

Продолжение таблицы 1

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 1 | 2 | 3 | 4 | 5 |
| repair | BOOLEAN |  | Ремонт | NOT NULL |
| cost | DECIMAL | 10, 2 | Стоимость | NOT NULL |
| elevator | BOOLEAN |  | Лифт | NOT NULL |
| playground | BOOLEAN |  | Детская площадка | NOT NULL |
| trashChute | BOOLEAN |  | Мусоропровод | NOT NULL |
| parking | BOOLEAN |  | Автомобильная стоянка | NOT NULL |
| balcony | BOOLEAN |  | Балкон | NOT NULL |
| deal\_id | INT |  | Номер сделки | NOT NULL |
| client\_id | INT |  | Номер клиента | NOT NULL |

Таблица 2. Таблица Deal

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Имя поля | Тип | Размер | Комментарий | Ограничение |
| id | INT | auto\_increment | Идентификатор сделки | Первичный ключ |
| amount | DECIMAL | 10, 2 | Сумма сделки | NOT NULL |
| date | DATE |  | Дата сделки | NOT NULL |
| status | VARCHAR | 50 | Статус сделки | NOT NULL |
| number | INT |  | Номер сделки | NOT NULL |
| client\_id | INT |  | Номер клиента | NOT NULL |
| employee\_id | INT |  | Номер сотрудника | NOT NULL |

Таблица 3 - Таблица Client

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Имя поля | Тип | Размер | Комментарий | Ограничение |
| 1 | 2 | 3 | 4 | 5 |
| id | INT | auto\_increment | Идентификатор клиента | Первичный ключ |
| passport | VARCHAR | 20 | Паспортные данные | NOT NULL |
| name | VARCHAR | 40 | Имя | NOT NULL |
| surname | VARCHAR | 40 | Фамилия | NOT NULL |
| patronymic | VARCHAR | 40 | Отчество | NULL |
| phone | VARCHAR | 15 | Номер телефона | NOT NULL |
| ownerType | VARCHAR | 50 | Тип владельца | NOT NULL |
| company | VARCHAR | 100 | Название компании | NULL |
| license | VARCHAR | 50 | Лицензия компании | NULL |

Таблица 4 - Таблица Employee

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Имя поля | Тип | Размер | Комментарий | Ограничение |
| id | INT | auto\_increment | Идентификатор сотрудника | Первичный ключ |
| name | VARCHAR | 40 | Имя | NOT NULL |
| surname | VARCHAR | 40 | Фамилия | NOT NULL |
| patronymic | VARCHAR | 40 | Отчество | NULL |
| position | VARCHAR | 50 | Должность сотрудника | NOT NULL |
| deals | INT |  | Количество совершенных сделок | NOT NULL |
| email | VARCHAR | 50 | Электронная почта | NOT NULL |
| phone | VARCHAR | 15 | Номер телефона | NOT NULL |

4.3 - Требования к эргономике и технической эстетике

Общие требования:

- Фоновый цвет белый.

- При наведении указателя мышки на элементы кнопок, они должны выделяться, например, путем изменения цвета или появления подсветки экрана.

- Шрифты и размеры текста должны быть четкими и хорошо читаемыми.

- Адаптивный интерфейс

системы (Недвижимость, Сделки, Клиенты, Сотрудники, О системе, Выход). Это обеспечивает удобный и интуитивно понятный доступ к возможностям системы

- Все кнопки должны иметь одинаковую высоту

Элементы интерфейса:

- Таблицы имеют прозрачные границы ячеек и заголовков для более гладкого визуального восприятия.

- При фокусировке на строке таблицы, она выделяется цветом

- Кнопки имеют закругленные углы, что придает им более современный и привлекательный вид.

Экран авторизации:

- Все элементы страницы должны быть отцентрированы одновременно по вертикали и горизонтали.

- Заголовок страницы - «Авторизация».

- Над элементами ввода текст - Кнопка «Войти» и элементы ввода текста (логин и пароль) должны иметь одинаковую ширину.

- Кнопка «Войти» - светло-серый цвет. Цвет шрифта - черный. При наведении на кнопку ее цвет становится светлее.

4.4 Требования к содержанию и оформлению выводимых сообщений

- Отображение всплывающей подсказки, в случае ввода некорректного формата, уведомление автоматически исчезает, когда пользователь начнет заново вводить данные

- Если пользователь вводит неверные данные и нажимает на кнопку войти, то сформируется окно с надписью: «Неверный логин или пароль. Попробуйте заново»

- Сообщения об ошибках при вводе некорректных данных (например, название должно начинаться с заглавной буквы и содержать только русские буквы)

- Автоматическое исчезновение подсказки при начале нового ввода данных в соответствующее поле

- Сообщения об ошибках при незаполненных обязательных полях (например, «Недопустимое заполнение» или «Пожалуйста, заполните все поля»)

- Отображение ошибки при добавлении, редактировании, удалении данных

- Сообщение об ошибке при проблемах с сервером (например, «Произошла ошибка при запуске сервера», «Пожалуйста, проверьте соединение к сети и попробуйте снова»)

4.5 Требования к видам обеспечения

Для функционирования системы требуется сервер и клиентских компьютеры, на которых будет производится запуск системы.

Требования для сервера базе данных:

- Windows 10 и выше;

- Процессор 2 ядра (8 логических потоков), частота - 2-2,53 - Оперативная память 8 Гб и выше

- Свободное дисковое пространство не менее 2 Гб

- HDD (жесткий диск), тип - SATA2, скорость вращения 7200 об/мин

Требования к клиентским устройствам:

- Операционная система: Windows 10 и выше

- Браузер версия: Google Chrome 131.0.6778.69, Opera 115.0.5322.109, Brave 1.71.114.

- Оперативная память менее 4 ГБ и выше

4.6 Эскизы основных окон системы

На рисунках 3-8 предоставлены эскизы страниц сайта:

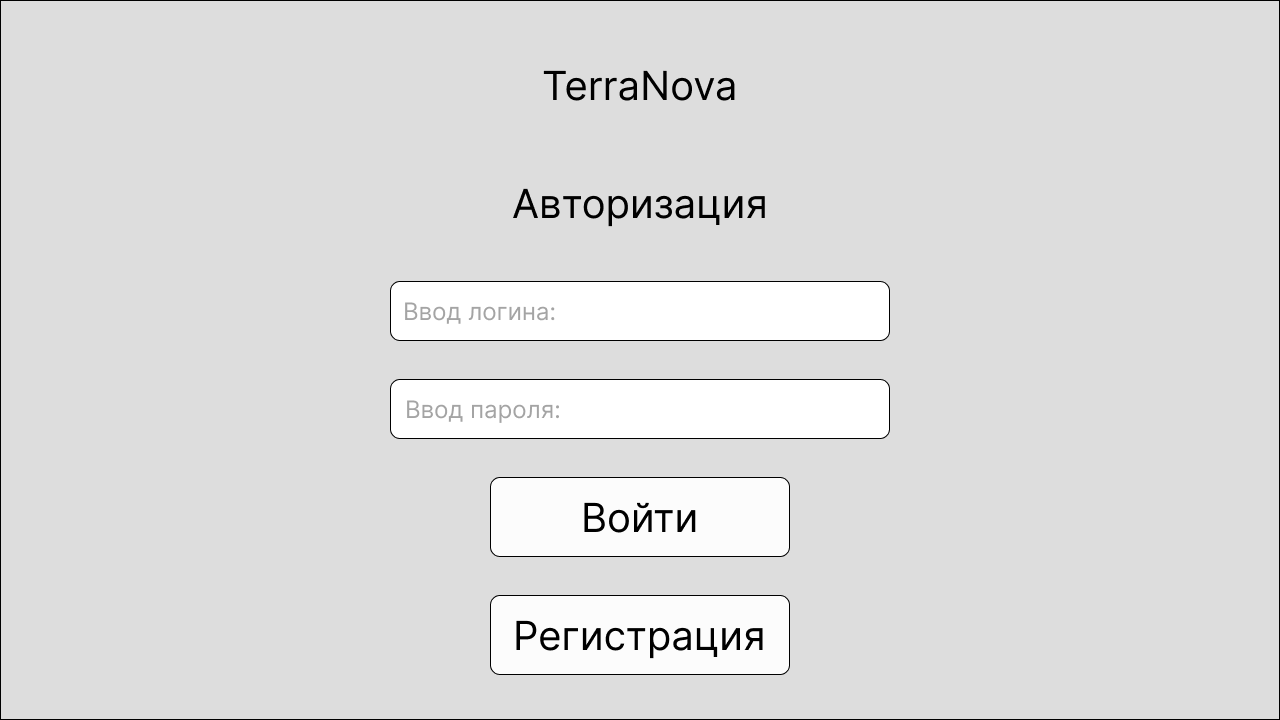


Рисунок 3 - Страница авторизации

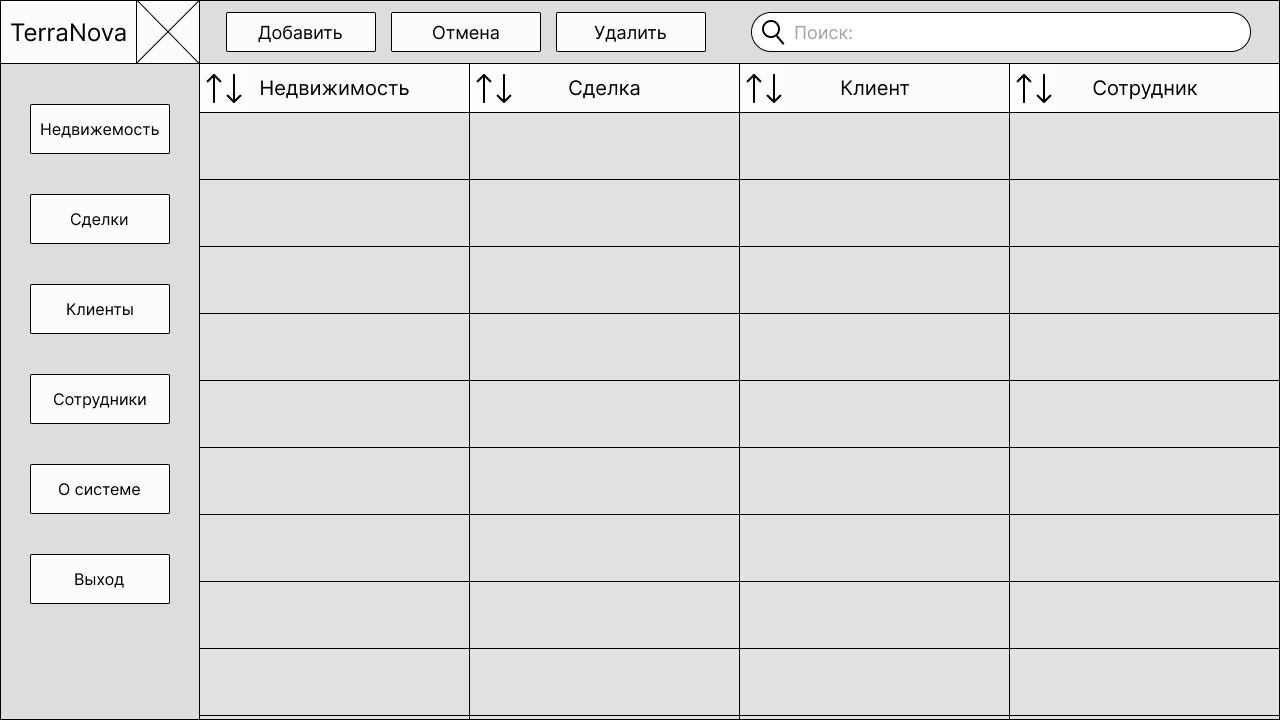


Рисунок 4 - Главная страница

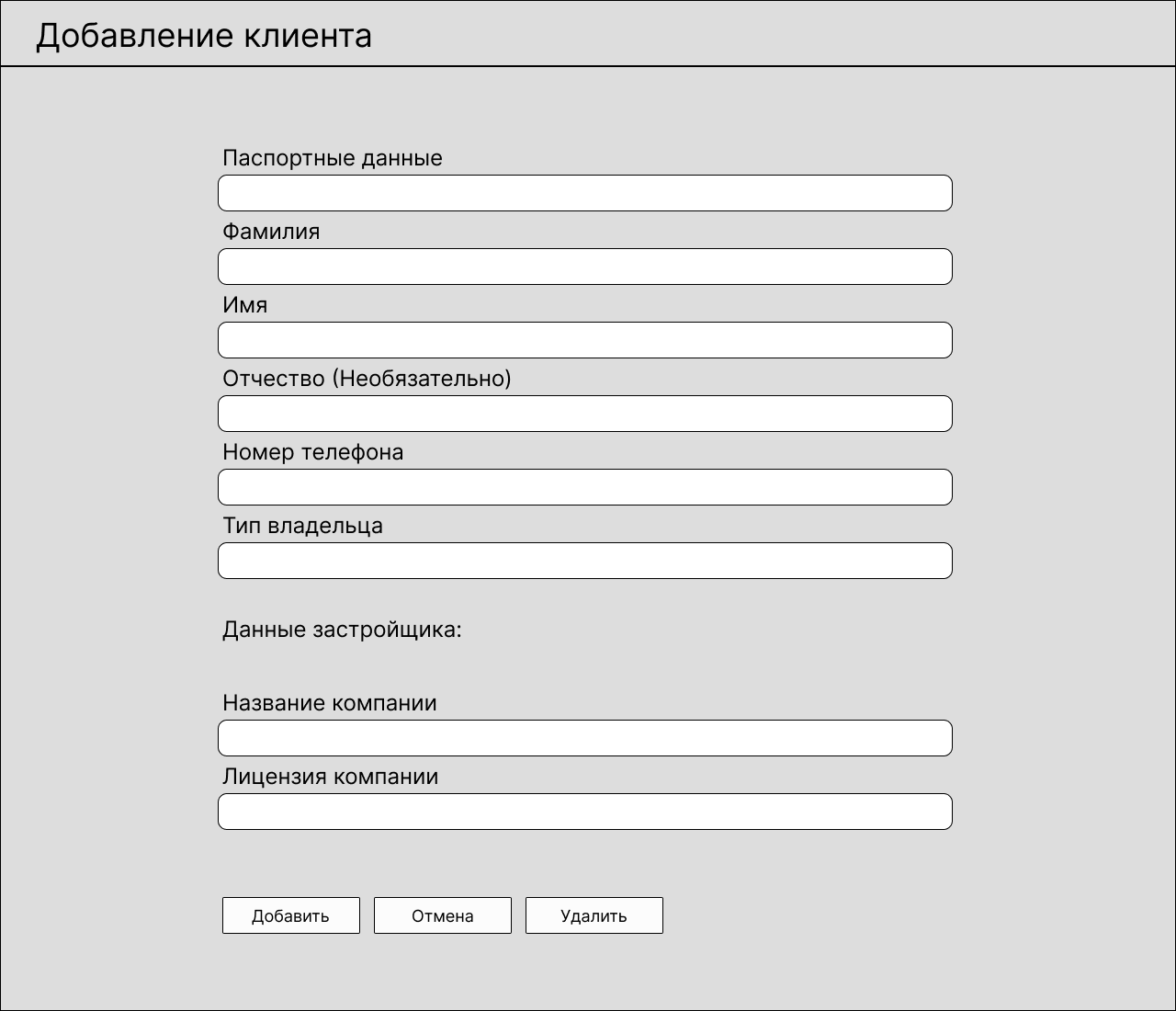


Рисунок 5 - Страница добавления клиента

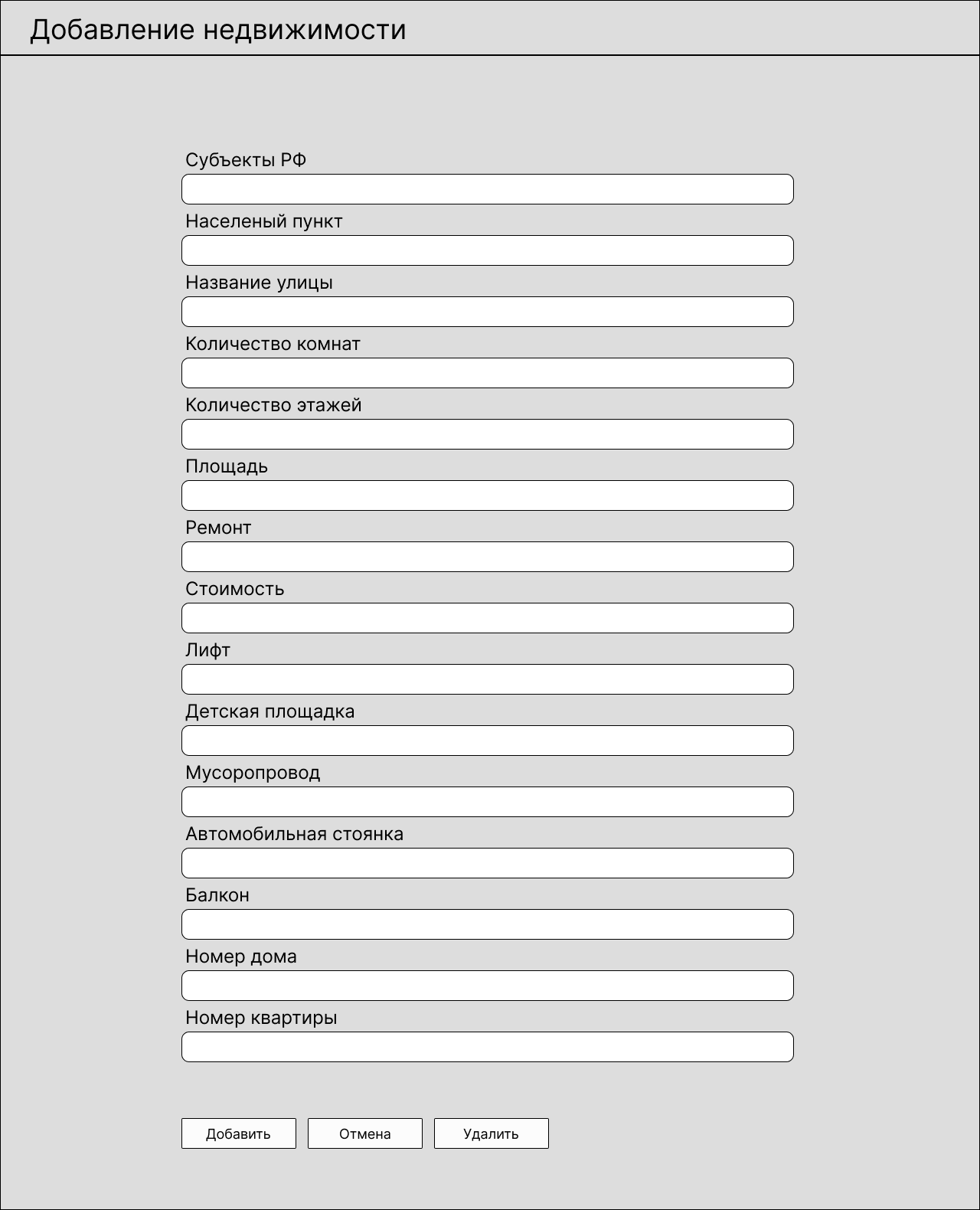


Рисунок 6 - Страница добавления недвижимости

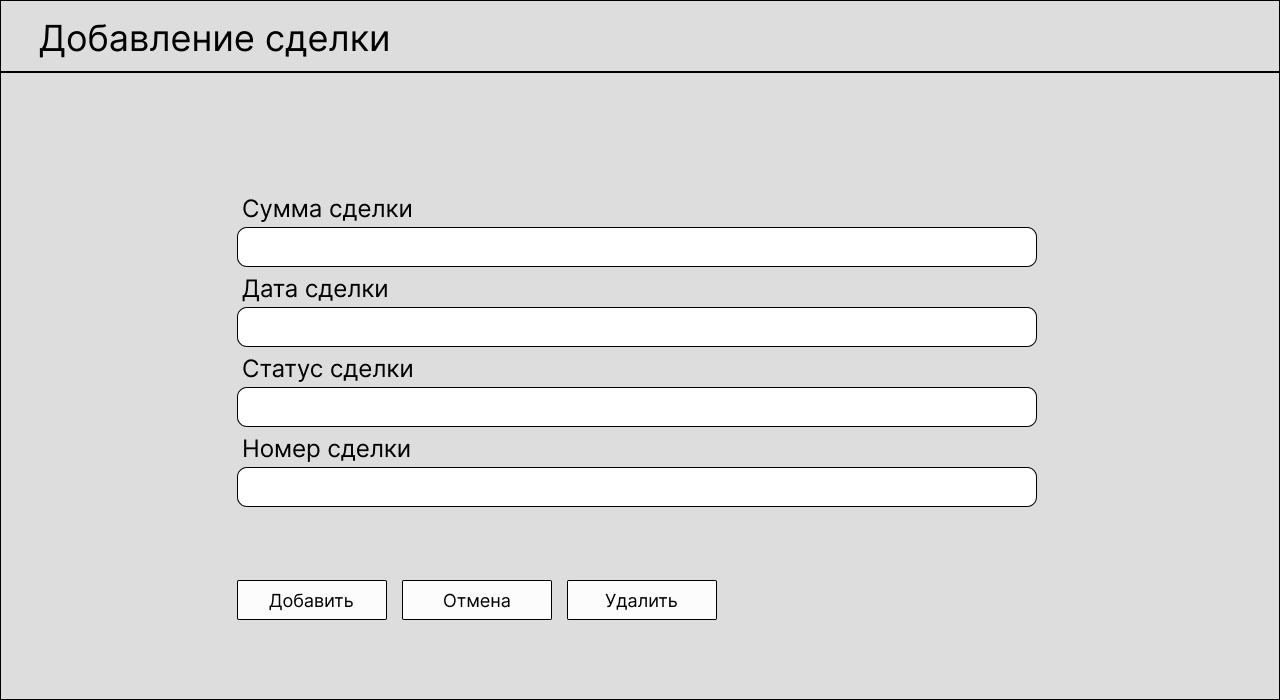


Рисунок 7 - Страница добавления сделки

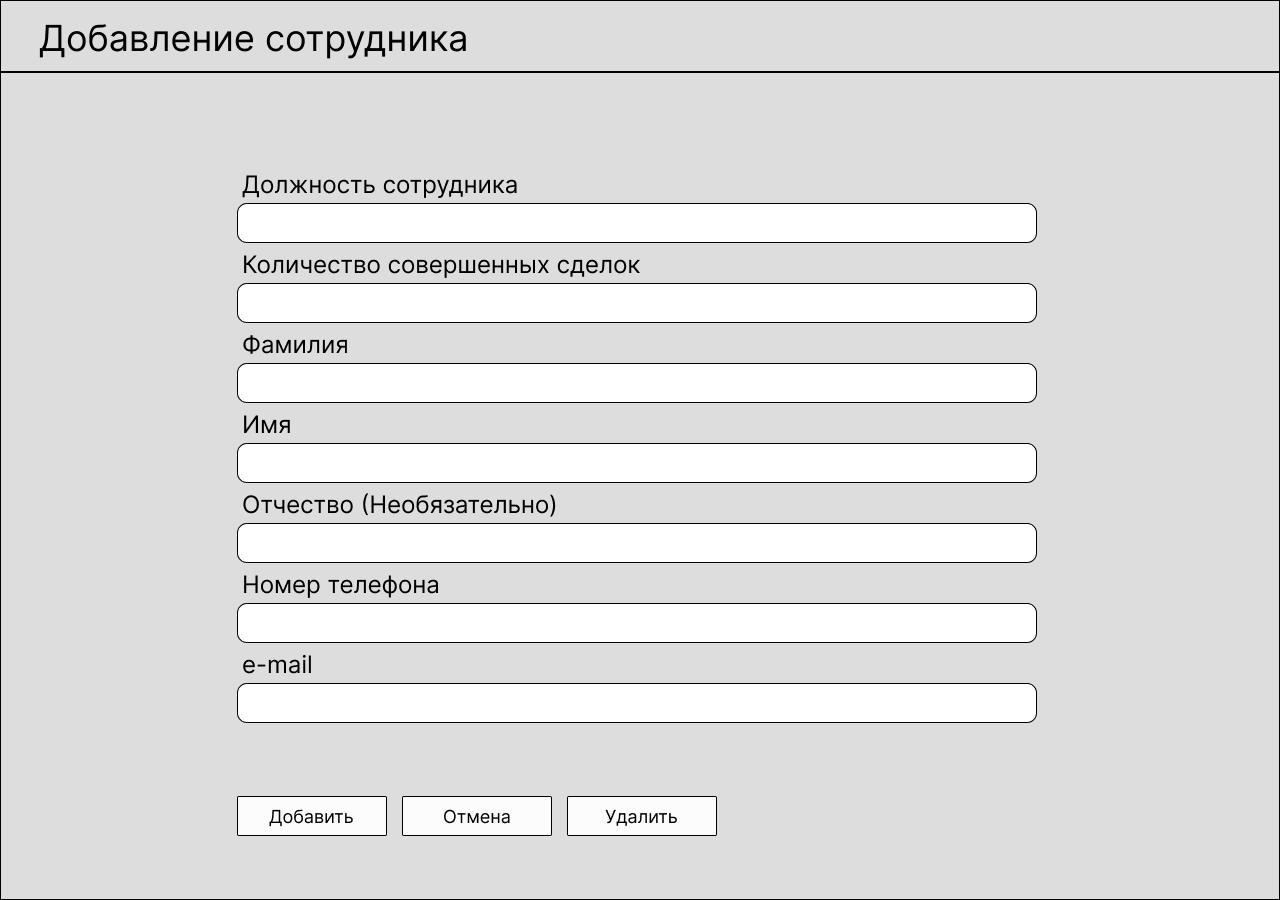


Рисунок 8 - Страница добавления сотрудника

5. Состав и содержание работ по созданию системы

Комплекс работ по созданию АИС для агентства недвижимости «TerraNova» должен осуществляться в несколько этапов:

Анализ требований:

- Сбор и анализ требований к системе со стороны агентства недвижимости и клиентов

- Определение функциональных и нефункциональных требований к системе

- Изучение бизнес-процессов агентства недвижимости

Проектирование системы:

- Разработка концептуальной модели системы

- Проектирование архитектуры системы (MVC, веб-приложение и т.д.)

- Проектирование базы данных для хранения информации о недвижимости, клиентах, сделках и т.д.

- Проектирование пользовательского интерфейса и навигации

Разработка системы:

- Создание серверной части системы (Backend)

- Разработка клиентской части системы (Frontend)

- Реализация модулей для управления недвижимостью, клиентами, сделками и т.д.

Тестирование системы:

- Модульное тестирование отдельных компонентов системы

- Интеграционное тестирование взаимодействия компонентов

- Системное тестирование функциональности системы

Развертывание и внедрение системы:

- Подготовка серверной и клиентской инфраструктуры

- Установка и настройка системы на серверах и клиентских устройствах

- Миграция и загрузка данных из существующих систем (если применимо)

- Обучение персонала работе с новой системой

Опытная эксплуатация и доработка системы:

- Опытная эксплуатация системы в рабочей среде

- Устранение выявленных недочетов и ошибок

- Доработка системы по результатам опытной эксплуатации

Ввод системы в промышленную эксплуатацию:

- Окончательный переход на использование новой системы

- Вывод из эксплуатации старых систем (если применимо)

Сопровождение и развитие системы:

- Техническая поддержка и устранение возникающих ошибок

- Обновление системы в соответствии с изменяющимися требованиями

- Добавление новых функций и возможностей по мере необходимости

Приемка работ:

- Проведение предварительных испытаний на стенде исполнителя

6 Порядок контроля и приемки системы

Система передается в виде полностью функционирующего комплекса на базе средств вычислительной техники заказчика и исполнителя в сроки, установленные договором. Приемка системы осуществляется комиссией в составе уполномоченных представителей заказчика и исполнителя. Весь комплект документации на АИС, разработанный исполнителем, передается заказчику на бумажных (в двух экземплярах) и на USB-носитель (флешка), один экземпляр после подписания передается исполнителю.

7 Требования к составу и содержанию работ по подготовке объекта автоматизации к вводу системы в действие

Заказчик обязан предоставить сервер и персональные компьютеры, требования к которым указаны в разделе 4.2 «Требования к видам обеспечения».

Необходимое программное обеспечение для сервера:

- Операционная система: Windows 10

- СУБД: MySQL.

Необходимое программное обеспечение для персонального компьютера:

- Операционная система: Windows 10;

- Google Chrome(x64)/ Opera(x64)

8 Требования к документированию

Перечень подлежащих разработке комплектов и видов документов:

- Описание анализа предметной области

- Техническое задание

- Эскизный проект

9 Источники разработки

Источниками разработки настоящего Технического Задания являются:

- Перечень документации, представленный в разделе 1.2 «Основания для проведения работ»;

- ГОСТ 34.602-89 Техническое задание на создание автоматизированной системы;

- ГОСТ 24.601-86 Автоматизированные системы. Стадии создания;

- ГОСТ Р ИСО/МЭК 9126-93. Информационная технология. Оценка программной продукции. Характеристики качества и руководства по их применению;

- ГОСТ 24.601 Исследование и обоснование создания АС.

1.3 Эскизный проект

3.1 Декомпозиция системы на комплексы и проработка архитектурных решений

Назначение системы АИС «TerraNova» предназначена для комплексного информационно управленческого обеспечения процессов в части исполнения процессов.

Комплексное информационно-управленческое обеспечение процессов в сфере недвижимости, включая:

- Управление объектами недвижимости

- Работа с клиентами (покупателями и продавцами)

- Заключение сделок

- Учет и анализ данных

Информационная система управления «TerraNova» разработана для повышения эффективности и качества услуг агентства недвижимости.

Система позволит централизованно хранить и обрабатывать всю информацию о недвижимости, клиентах и сделках. Для сотрудников агентства система обеспечит удобный доступ ко всем необходимым данным и инструментам управления.

Основными целями являются:

- Повышение эффективности управления недвижимостью

- Улучшение качества обслуживания клиентов

- Увеличение прибыльности агентства

Достижение поставленных целей направлено на увеличение прибыльности и конкурентоспособности агентства недвижимости путем оптимизации бизнес-процессов и повышения качества предоставляемых услуг клиентам.

Для достижения поставленных целей информационная система управления гостиничным комплексом «TerraNova» должна решать следующие основные задачи:

- Ведение базы данных объектов недвижимости с подробной информацией

- Учет запросов клиентов на покупку или продажу недвижимости

- Организация просмотров объектов и встреч с клиентами

- Подготовка и сопровождение сделок по купле-продаже недвижимости

- Учет оказанных услуг и финансовых операций

На данном этапе рассмотрим, как будет вести себя система при выполнении выделенных функций. Для этого будем использовать диаграммы состояний в нотации UML.

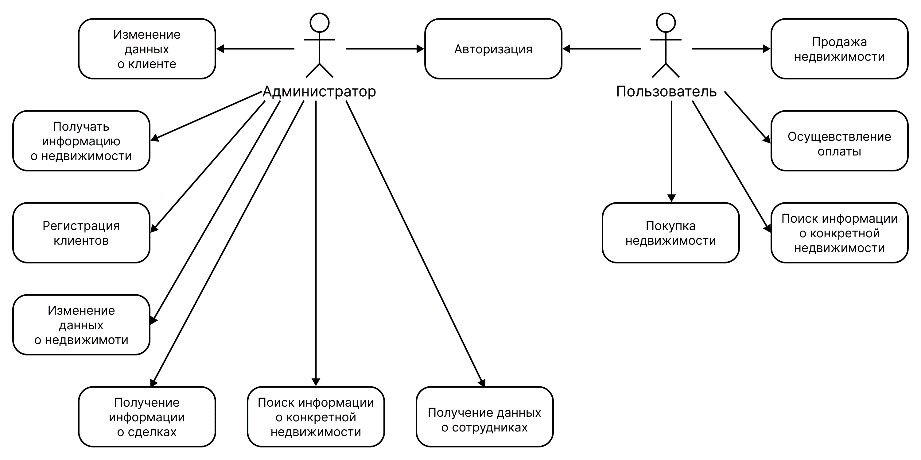


Рисунок 9 - Диаграмма варианта использования

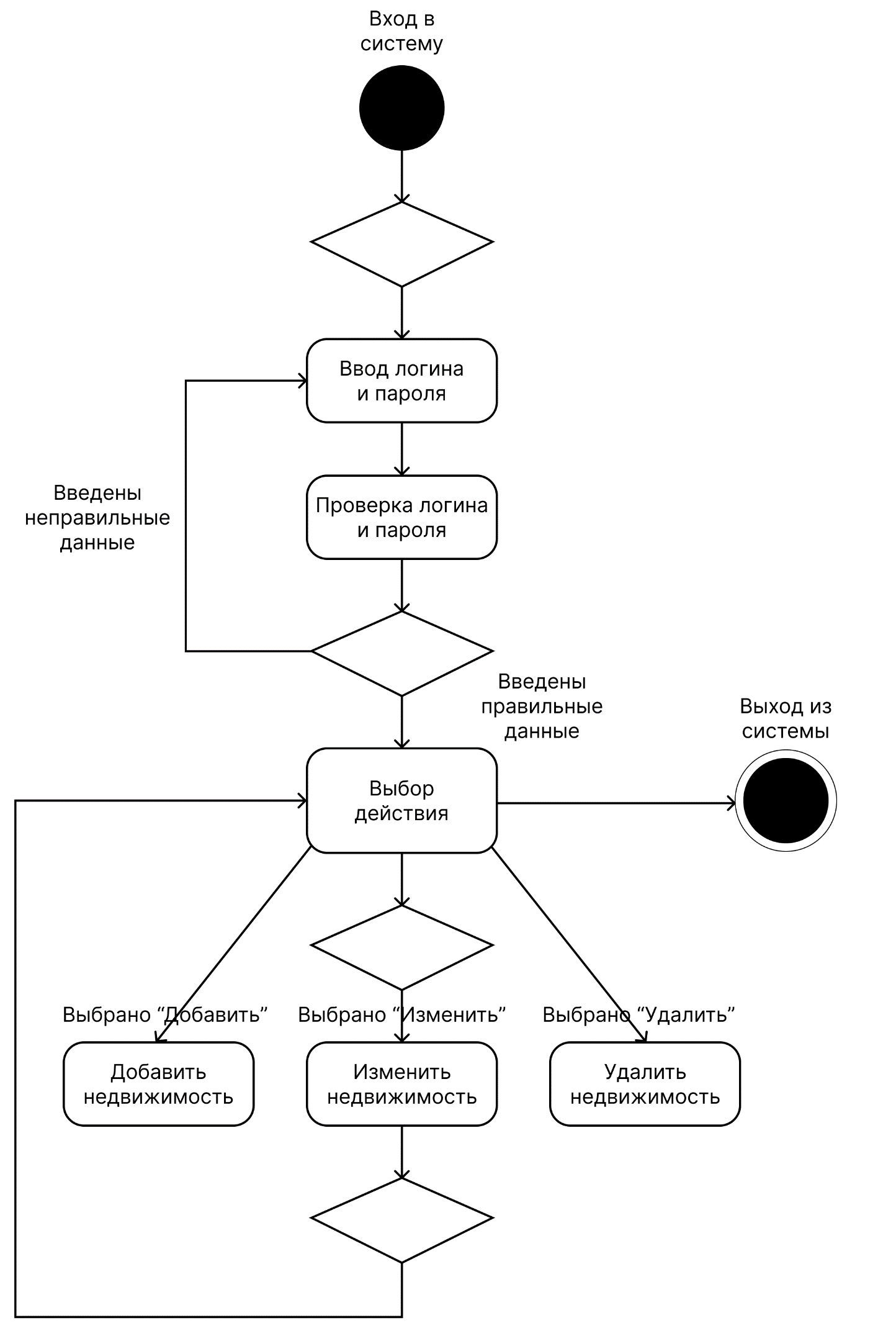


Рисунок 10 - Диаграмма состояния

На данном этапе необходимо описать примерную архитектуру разрабатываемой системы. Для того, чтобы сделать информационную систему агентства недвижимости, мы будем использовать **Model-View-Controller** (MVC) архитектуру. Чтобы убедиться, что она нам точно подходит рассмотрим ее подробнее. В **Model-View-Controller** (MVC) архитектуре имеется три звена:

**- Модель (Model)**:

**- Представление (View)**:

**- Контроллер (Controller)**:

Мы выбрали эту архитектуру, так как у нее есть ряд преимуществ перед другими архитектурами:

- это основная логика приложения. Отвечает за данные, методы работы с ними и структуру программы. Модель реагирует на команды из контроллера и выдаёт информацию или изменяет своё состояние.

- передаёт данные в представление.

- задача компонента - визуализация информации, которую он получает от модели. View отображает данные на уровне пользовательского интерфейса.

- обеспечивает взаимодействие с системой.

Обрабатывает действия пользователя, проверяет полученную информацию и передаёт её модели. Контроллер определяет, как приложение будет реагировать на действия пользователя. Также контроллер может отвечать за фильтрацию данных и авторизацию.

Для агентства недвижимости подойдет **Model-View-Controller** (MVC), так как она обеспечивает изменять каждый компонент независимо друг от друга для простой разработки и поддержки веб-приложений.

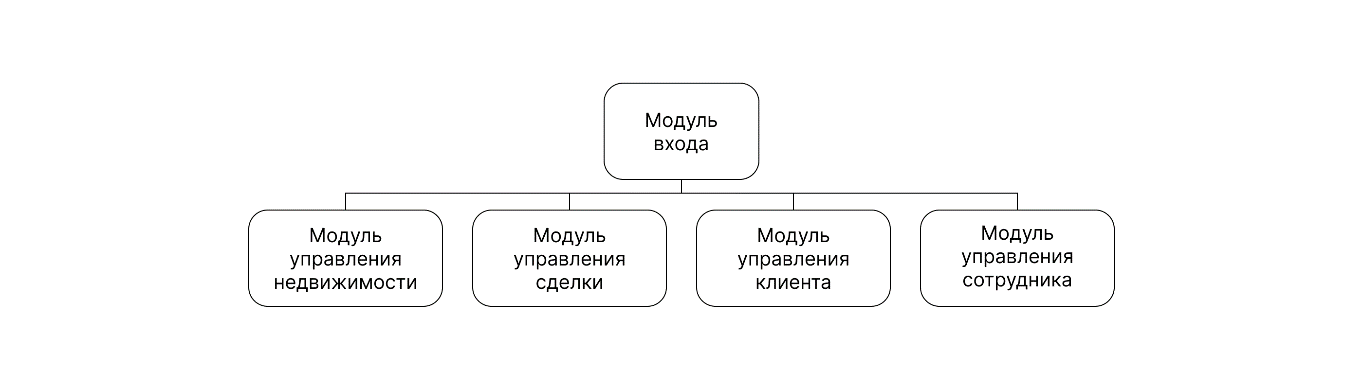


Рисунок 11 - Модули разрабатываемой системы

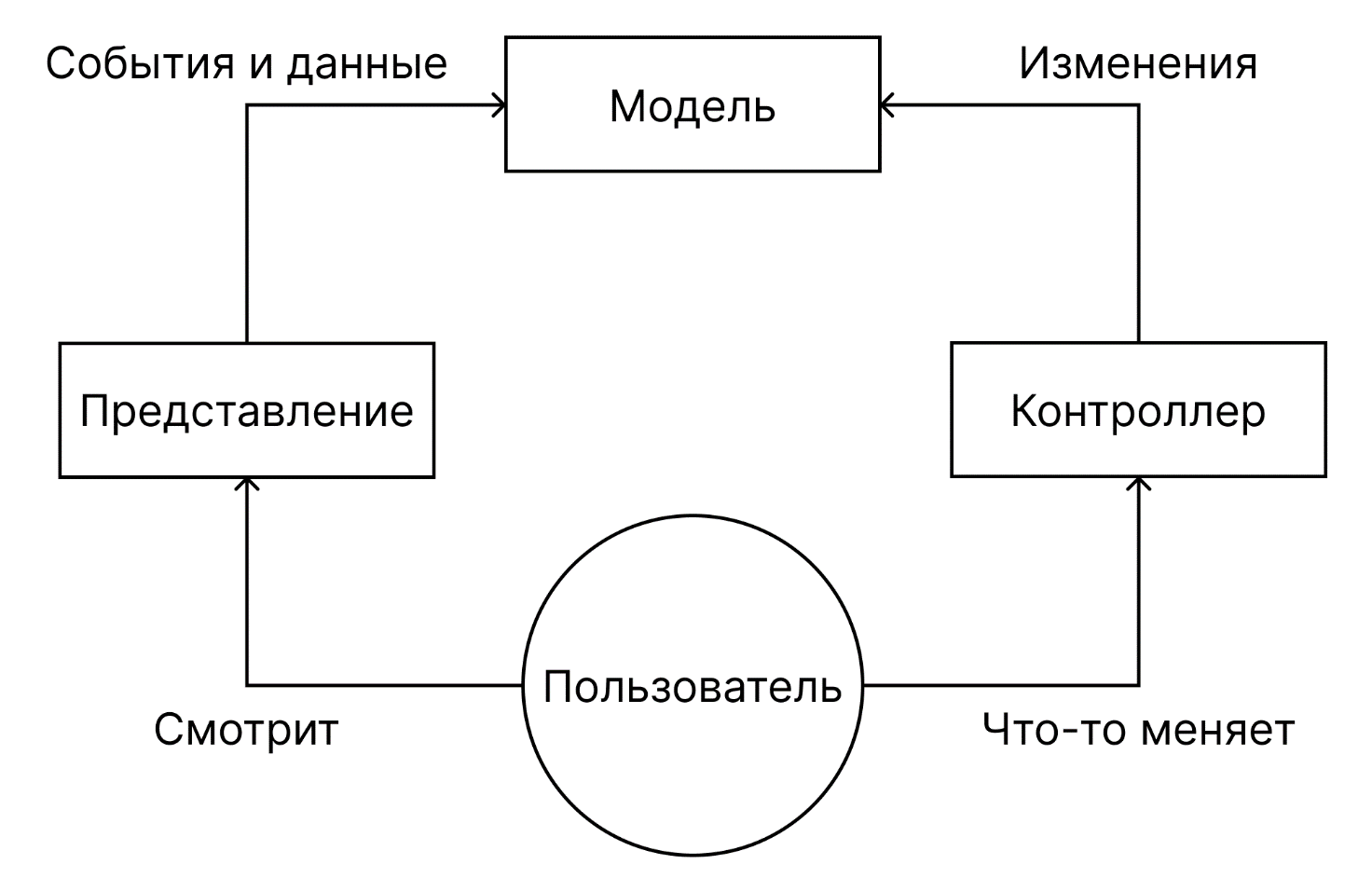


Рисунок 12 - Структура **Model-View-Controller** архитектуры

Серверная часть:

- Модуль авторизации

- Модуль управления недвижимостью

- Модуль управления клиентами

- Модуль управления сделками

- Модуль управления сотрудниками

В состав персонала, для обеспечения эксплуатации АИС «TerraNova» в рамках соответствующих подразделений, необходимо выделение следующих ответственных лиц. Состав персонала, необходимого для обеспечения эксплуатации АИС «TerraNova»

Таблица 5 - Таблица состав персонала

|  |  |  |
| --- | --- | --- |
| Должность | Количество | Подсистема |
| Администратор | 1 | Администрирование БД |
| Сотрудники | 1-5 | Управление недвижимостью |

Данные лица должны выполнять следующие функциональные обязанности:

Пользователь:

- Управление статусами недвижимости (свободно, ремонт и т.д.)

Администратор:

- Внесение правок в базу данных

База данных информационной системы должна состоять из следующих связанных таблиц:

Realty - таблица, содержащая информацию о недвижимостях

Deal - таблица, содержащая информацию о сделках

Client - таблица, содержащая информацию о клиентах

Employee - таблица, содержащая информацию о сотрудниках

На таблицах 6-9 предоставлены эскизы страниц сайта:

Таблица 6 - Таблица Realty

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Имя поля | Тип | Размер | Комментарий | Ограничение |
| id | INT | auto\_increment | Идентификатор недвижимости | Первичный ключ |
| region | VARCHAR | 100 | Субъект РФ | NOT NULL |
| locality | VARCHAR | 100 | Населённый пункт | NOT NULL |
| street | VARCHAR | 100 | Название улицы | NOT NULL |
| rooms | INT |  | Количество комнат | NOT NULL |
| floors | INT |  | Количество этажей | NOT NULL |
| square | DECIMAL | 10, 2 | Площадь | NOT NULL |
| repair | BOOLEAN |  | Ремонт | NOT NULL |
| cost | DECIMAL | 10, 2 | Стоимость | NOT NULL |
| elevator | BOOLEAN |  | Лифт | NOT NULL |
| playground | BOOLEAN |  | Детская площадка | NOT NULL |
| trashChute | BOOLEAN |  | Мусоропровод | NOT NULL |
| parking | BOOLEAN |  | Автомобильная стоянка | NOT NULL |
| balcony | BOOLEAN |  | Балкон | NOT NULL |
| deal\_id | INT |  | Номер сделки | NOT NULL |
| client\_id | INT |  | Номер клиента | NOT NULL |

Таблица 7 - Таблица Deal

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Имя поля | Тип | Размер | Комментарий | Ограничение |
| id | INT | auto\_increment | Идентификатор сделки | Первичный ключ |
| amount | DECIMAL | 10, 2 | Сумма сделки | NOT NULL |
| date | DATE |  | Дата сделки | NOT NULL |
| status | VARCHAR | 50 | Статус сделки | NOT NULL |
| number | INT |  | Номер сделки | NOT NULL |
| client\_id | INT |  | Номер клиента | NOT NULL |
| employee\_id | INT |  | Номер сотрудника | NOT NULL |

Таблица 8 - Таблица Client

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Имя поля | Тип | Размер | Комментарий | Ограничение |
| id | INT | auto\_increment | Идентификатор клиента | Первичный ключ |
| passport | VARCHAR | 20 | Паспортные данные | NOT NULL |
| name | VARCHAR | 40 | Имя | NOT NULL |
| surname | VARCHAR | 40 | Фамилия | NOT NULL |
| patronymic | VARCHAR | 40 | Отчество | NULL |
| phone | VARCHAR | 15 | Номер телефона | NOT NULL |
| ownerType | VARCHAR | 50 | Тип владельца | NOT NULL |
| company | VARCHAR | 100 | Название компании | NULL |
| license | VARCHAR | 50 | Лицензия компании | NULL |

Таблица 9 - Таблица Employee

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Имя поля | Тип | Размер | Комментарий | Ограничение |
| id | INT | auto\_increment | Идентификатор сотрудника | Первичный ключ |
| name | VARCHAR | 40 | Имя | NOT NULL |
| surname | VARCHAR | 40 | Фамилия | NOT NULL |
| patronymic | VARCHAR | 40 | Отчество | NULL |
| position | VARCHAR | 50 | Должность сотрудника | NOT NULL |
| deals | INT |  | Количество совершенных сделок | NOT NULL |
| email | VARCHAR | 50 | Электронная почта | NOT NULL |
| phone | VARCHAR | 15 | Номер телефона | NOT NULL |

4.2 Разработка операционной среды или требований к ней

Серверная часть: ОС Windows 10, СУБД MySQL

Клиентские рабочие места: ОС Windows 10

Использование сети Интернет для удаленного доступа.

4.3 Разработка методов анализа качества системы и метрик оценки

Метрики показателей качественной оценки:

- Надежность: Количество сбоев и их частота.

- Производительность: Время отклика системы на запросы.

- Масштабируемость: Возможность системы обрабатывать увеличивающиеся объемы данных и пользователей.

- Безопасность: Уровень защиты данных, количество инцидентов безопасности.

- Удобство использования: Оценка удобства интерфейса пользователями.

Методы анализа качества:

- Тестирование на надежность и отказоустойчивость.

- Производительность и нагрузочное тестирование.

- Безопасное тестирование, включая тестирование на проникновение.

- Оценка удобства использования на основе обратной связи пользователей.

4.4 Планирование перспектив создания АСУ

Основные концепции технологий разработки:

- Используемые технологии: Framework Vaadin для клиентской части, Spring Framework для серверной части, MySQL для базы данных.

- Подход к разработке: с итеративным развитием и частыми релизами.

- Требования к программным средствам:

- Среда разработки: IntelliJ IDEA для разработки на Java.

База данных по автоматизации и контролю работ:

- Разработка базы данных для отслеживания задач и прогресса разработки.

- Использование систем мониторинга и отчетности для контроля качества кода и системы в целом.

Оформление пояснительной записки:

- Подробное описание архитектурных решений и технологий.

- Технические требования и спецификации.

- План внедрения и тестирования системы.

- Документация по эксплуатации и поддержке системы.

4.5 Эскиз пользовательского интерфейса АИС

 Эскизы пользовательского интерфейса представлены в разделе 4.6 «Эскизы основных окон системы»

2 РАЗРАБОТКА И ДОКУМЕНТИРОВАНИЕ ИНФОРМАЦИОННОЙ СИСТЕМЫ

2.1 Тестирование клиентской части

1. Тестирование контроллера ClientController

1.1 Проверка добавления нового клиента:

- Переходим на страницу /client/add

- Вводим данные клиента в соответствующие поля формы (ФИО, паспорт, телефон, и т.п.)

- Нажимаем кнопку Сохранить

- Проверяем, что новый клиент появился в списке на странице /client

1.2 Проверка редактирования существующего клиента:

- Переходим на страницу /client

- Нажимаем кнопку Изменить рядом с нужным клиентом

- Изменяем данные в форме

- Нажимаем кнопку Сохранить

- Проверяем, что в списке клиентов отображаются обновлённые данные

1.3 Проверка удаления клиента:

- Переходим на страницу /client

- Нажимаем кнопку Удалить рядом с нужным клиентом

- Подтверждаем удаление (если реализовано)

- Проверяем, что клиент исчез из списка

1.4 Проверка сортировки клиентов:

- Переходим на страницу /client?sortField=fullName&sortDir=asc

- Проверяем, что список отсортирован по ФИО по возрастанию

- Повторяем с sortDir=desc - проверяем сортировку по убыванию

2. Тестирование контроллера DealController

2.1 Проверка добавления новой сделки:

- Переходим на страницу /deal/add

- Заполняем форму: выбираем клиента, сотрудника и недвижимость, при этом:

- Проверяем, что при выборе недвижимости автоматически проставляется стоимость и владелец

- Проверяем, что при выборе клиента автоматически подставляется его номер

- Нажимаем кнопку Сохранить

- Проверяем, что новая сделка отображается в списке /deal

2.2 Проверка редактирования сделки:

- Переходим на страницу /deal

- Нажимаем кнопку Изменить рядом с нужной сделкой

- Изменяем поля (например, выбираем другую недвижимость)

- Нажимаем Сохранить

- Проверяем, что данные сделки обновились в списке

2.3 Проверка удаления сделки:

- Переходим на страницу /deal

- Нажимаем кнопку Удалить рядом с нужной сделкой

- Проверяем, что сделка исчезла из списка

2.4 Проверка сортировки сделок:

- Переходим на страницу /deal?sortField=date&sortDir=asc

- Проверяем, что список отсортирован по дате сделки по возрастанию

- Меняем параметр sortDir=desc — убеждаемся, что сортировка обратная

3. Тестирование контроллера EmployeeController

3.1 Проверка отображения списка сотрудников с сортировкой:

- Открыть страницу /employee без параметров сортировки.

- Проверить, что отображается список всех сотрудников.

- Открыть страницу /employee?sortBy=lastName&sortDir=asc.

- Проверить, что список сотрудников отсортирован по фамилии в порядке возрастания.

- Открыть страницу /employee?sortBy=lastName&sortDir=desc.

- Проверить, что список сотрудников отсортирован по фамилии в порядке убывания.

3.2 Проверка добавления нового сотрудника:

­- Перейти на страницу /employee/add.

- Заполнить поля формы для нового сотрудника (Фамилия, Имя, Отчество, - привязать к пользователю из списка).

- Нажать кнопку "Сохранить".

- Проверить, что сотрудник появился в списке /employee.

3.3 Проверка валидации при добавлении сотрудника:

- Перейти на страницу /employee/add.

- Оставить обязательные поля пустыми.

- Нажать "Сохранить".

- Проверить, что на странице отображаются сообщения об ошибках валидации и форма не отправляется.

3.4 Проверка редактирования сотрудника:

- На странице списка /employee выбрать сотрудника и перейти по ссылке редактирования /employee/edit/{id}.

- Изменить данные сотрудника.

- Нажать "Сохранить".

- Проверить, что изменения отобразились на странице списка сотрудников.

3.5 Проверка удаления одного сотрудника:

- На странице списка /employee выбрать сотрудника.

- Нажать кнопку "Удалить" рядом с сотрудником.

- Проверить, что сотрудник удалён из списка.

3.6 Проверка массового удаления сотрудников:

- Перейти на страницу /employee/deleteAll.

- Проверить, что список сотрудников полностью очищен.

4. Тестирование контроллера RealtyController

4.1 Проверка отображения списка недвижимости с сортировкой:

- Открыть страницу /realty без параметров сортировки.

- Проверить, что отображается полный список объектов недвижимости.

- Открыть страницу /realty?sortField=cost&sortDir=asc.

- Проверить, что список отсортирован по стоимости по возрастанию.

- Открыть страницу /realty?sortField=region&sortDir=desc.

- Проверить, что список отсортирован по региону в порядке убывания.

- Открыть страницу /realty?sortField=street&sortDir=asc.

- Проверить сортировку по улице в порядке возрастания.

4.2 Проверка добавления нового объекта недвижимости:

- Перейти на страницу /realty/add.

- Заполнить форму, включая поля: адрес, регион, стоимость, привязать к пользователю, сделке и клиенту.

- Нажать кнопку "Сохранить".

- Проверить, что новый объект недвижимости появился в списке /realty.

4.3 Проверка редактирования объекта недвижимости:

- На странице списка выбрать объект недвижимости и перейти по ссылке редактирования /realty/edit/{id}.

- Изменить поля объекта.

- Нажать кнопку "Сохранить".

- Проверить, что изменения корректно отображаются на странице списка.

4.4 Проверка удаления объекта недвижимости по одному:

- На странице списка нажать кнопку удаления рядом с выбранным объектом.

- Проверить, что объект недвижимости удалён из списка.

4.5 Проверка массового удаления объектов недвижимости:

- Перейти по ссылке /realty/delete.

- Проверить, что список объектов недвижимости полностью очищен.

4.6 Проверка обработки ошибки при попытке редактирования несуществующего объекта:

- Открыть страницу /realty/edit/{id} с несуществующим id.

- Проверить, что выбрасывается исключение NoSuchElementException с соответствующим сообщением.

5. Тестирование контроллера RegistrationController

5.1. Проверка успешной регистрации нового пользователя:

- Вводим корректные данные в поля username, password, confirmPassword.

- Нажимаем кнопку "Зарегистрироваться".

- Проверяем, что происходит редирект на страницу входа /include/login.

5.2. Проверка регистрации с уже занятым логином:

- Вводим существующее имя пользователя.

- Нажимаем кнопку "Зарегистрироваться".

- Проверяем, что отображается сообщение об ошибке: "Логин уже занят".

5.3. Проверка регистрации при несовпадении паролей:

- Вводим разные значения в поля password и confirmPassword.

- Нажимаем кнопку "Зарегистрироваться".

- Проверяем, что отображается сообщение: "Пароли не совпадают".

5.4. Проверка регистрации с пустыми полями или недопустимыми значениями:

- Оставляем обязательные поля пустыми или вводим недопустимые символы.

- Проверяем, что остаёмся на странице регистрации и отображаются валидационные ошибки.

6. Тестирование контроллера ProfileController

6.1. Проверка отображения профиля пользователя:

- Входим под зарегистрированным пользователем.

- Переходим на /profile.

- Проверяем, что отображаются:

- Информация о пользователе;

- Список его объектов недвижимости;

6.4. Проверка отсутствия аватара:

- У пользователя не загружен аватар.

- Проверяем, что отображается изображение по умолчанию.

2.2 Руководство пользователя

1. Регистрация и авторизация

1.1 Регистрация нового пользователя

Для начала работы в системе необходимо зарегистрироваться:

- Перейдите на страницу регистрации, выбрав соответствующую ссылку в интерфейсе.

- Заполните все обязательные поля формы:

- Логин (уникальное имя пользователя)

‑ Пароль (рекомендуется использовать сложный пароль с цифрами и буквами)

‑ Подтверждение пароля

- Другие запрашиваемые данные

- Нажмите кнопку "Зарегистрироваться".

В случае успешной регистрации система автоматически перенаправит вас на страницу авторизации. Если регистрация не удалась (например, логин уже занят), появится сообщение с ошибкой.

1.2 Авторизация (вход в систему)

- На странице авторизации введите логин и пароль в соответствующие поля.

- Нажмите кнопку "Войти".

В случае успешного входа откроется ваш профиль. При неверном вводе данных появится уведомление с просьбой повторить попытку.

2. Главное окно системы

После успешной авторизации откроется главное окно, которое является центральным пунктом управления системой. В нем расположены следующие элементы:

2.1 Таблицы с данными

- Недвижимость: отображает список всех объектов недвижимости, доступных в базе. В таблице представлены ключевые параметры: адрес, тип, площадь, стоимость и статус.

- Сделки: содержит информацию о заключённых сделках, включая клиента, сотрудника, недвижимость и сумму.

- Клиенты: список всех клиентов агентства с их контактными данными.

- Сотрудники: отображение всех сотрудников с должностями и контактами.

- Пользователи: учетные записи пользователей системы с указанием ролей и статусов.

2.2 Панель управления сущностями

Для каждой таблицы расположены кнопки управления:

- Добавить - открывает форму для создания новой записи.

- Изменить - позволяет редактировать выбранную запись.

- Удалить - удаляет выбранную запись из базы данных.

2.3 Общие кнопки

- Профиль - переход к странице редактирования личных данных текущего пользователя.

- О системе - отображение информации о версии и разработчиках.

- Выход - завершение сеанса и возврат на страницу входа.

3. Управление недвижимостью

3.1 Добавление нового объекта недвижимости

- Нажмите кнопку "Добавить недвижимость".

В открывшейся форме заполните обязательные поля:

- Адрес (строка с точным указанием расположения объекта)

- Тип недвижимости (например, квартира, дом, офис)

- Площадь (в квадратных метрах)

- Стоимость (в валюте системы)

- Дополнительные характеристики (этажность, состояние и пр.)

- Проверьте корректность введенных данных и нажмите "Добавить".

- Если данные корректны, объект появится в таблице недвижимости.

3.2 Изменение информации об объекте

- Выберите нужный объект в таблице.

- Нажмите кнопку "Изменить недвижимость".

- Внесите необходимые изменения в форму и нажмите "Изменить".

- Обновленные данные отобразятся в таблице.

3.3 Удаление объекта недвижимости

- Выберите объект недвижимости в списке.

- Нажмите кнопку "Удалить недвижимость".

- Подтвердите удаление в появившемся диалоговом окне.

- Запись будет удалена из базы.

4. Управление сделками

4.1 Добавление новой сделки

- Нажмите кнопку "Добавить сделку".

Заполните поля формы:

- Клиент (выберите из существующих клиентов)

- Недвижимость (выберите объект)

- Сотрудник (агент, ответственный за сделку)

- Сумма сделки

- Дата заключения сделки

- Нажмите "Добавить" для сохранения. Сделка появится в списке.

4.2 Изменение сделки

- Выберите сделку из таблицы.

- Нажмите кнопку "Изменить сделку".

- Измените необходимые параметры и сохраните изменения кнопкой "Изменить".

4.3 Удаление сделки

- Выберите сделку в списке.

- Нажмите "Удалить сделку".

- Подтвердите удаление.

5. Управление клиентами

5.1 Добавление клиента

- Нажмите кнопку "Добавить клиента".

- Введите данные клиента: ФИО, контактный телефон, адрес, электронную почту и другие сведения.

- Нажмите "Добавить".

5.2 Изменение данных клиента

- Выберите клиента в таблице.

- Нажмите кнопку "Изменить клиента".

- Отредактируйте поля и сохраните.

5.3 Удаление клиента

- Выберите клиента.

- Нажмите "Удалить клиента" и подтвердите действие.

6. Управление сотрудниками

6.1 Добавление сотрудника

- Кнопка "Добавить сотрудника" открывает форму для заполнения данных: ФИО, должность, контактная информация, электронная почта.

- Нажмите "Добавить" для сохранения.

6.2 Редактирование сотрудника

- Выберите сотрудника и нажмите "Изменить сотрудника".

- Внесите изменения и сохраните.

6.3 Удаление сотрудника

- Выберите сотрудника и нажмите "Удалить сотрудника". Подтвердите удаление.

7. Управление пользователями системы

7.1 Добавление пользователя

- Нажмите "Добавить пользователя".

- Заполните поля: логин, пароль, роль (например, администратор, агент), контактные данные.

- Нажмите "Добавить".

7.2 Редактирование пользователя

- Выберите пользователя.

- Нажмите "Изменить пользователя", измените необходимые поля и сохраните.

7.3 Удаление пользователя

- Выберите запись и нажмите "Удалить пользователя". Подтвердите действие.

8. Профиль пользователя

8.1 Просмотр и редактирование данных профиля

- Перейдите в раздел "Профиль" через главное меню.

- В профиле отображаются ваши личные данные и роль в системе.

- Можно изменить пароль, обновить контактные данные и загрузить фотографию (аватар).

- После внесения изменений нажмите "Сохранить".

8.2 Просмотр связанной информации

- В профиле отображается список сделок и недвижимости, связанных с вами как с сотрудником или клиентом.

9. Информация о системе

- Нажмите кнопку "О системе" в главном меню.

- Откроется окно с информацией: версия ПО, дата релиза, данные разработчика, контактные данные службы поддержки.

10. Завершение работы (Выход)

- Для выхода из системы нажмите кнопку "Выход" в главном меню.

- Сеанс будет завершён, и вы будете перенаправлены на страницу входа.

- Рекомендуется завершать работу через кнопку "Выход" для обеспечения безопасности данных.

2.3 Описание страниц

1. Главная страница

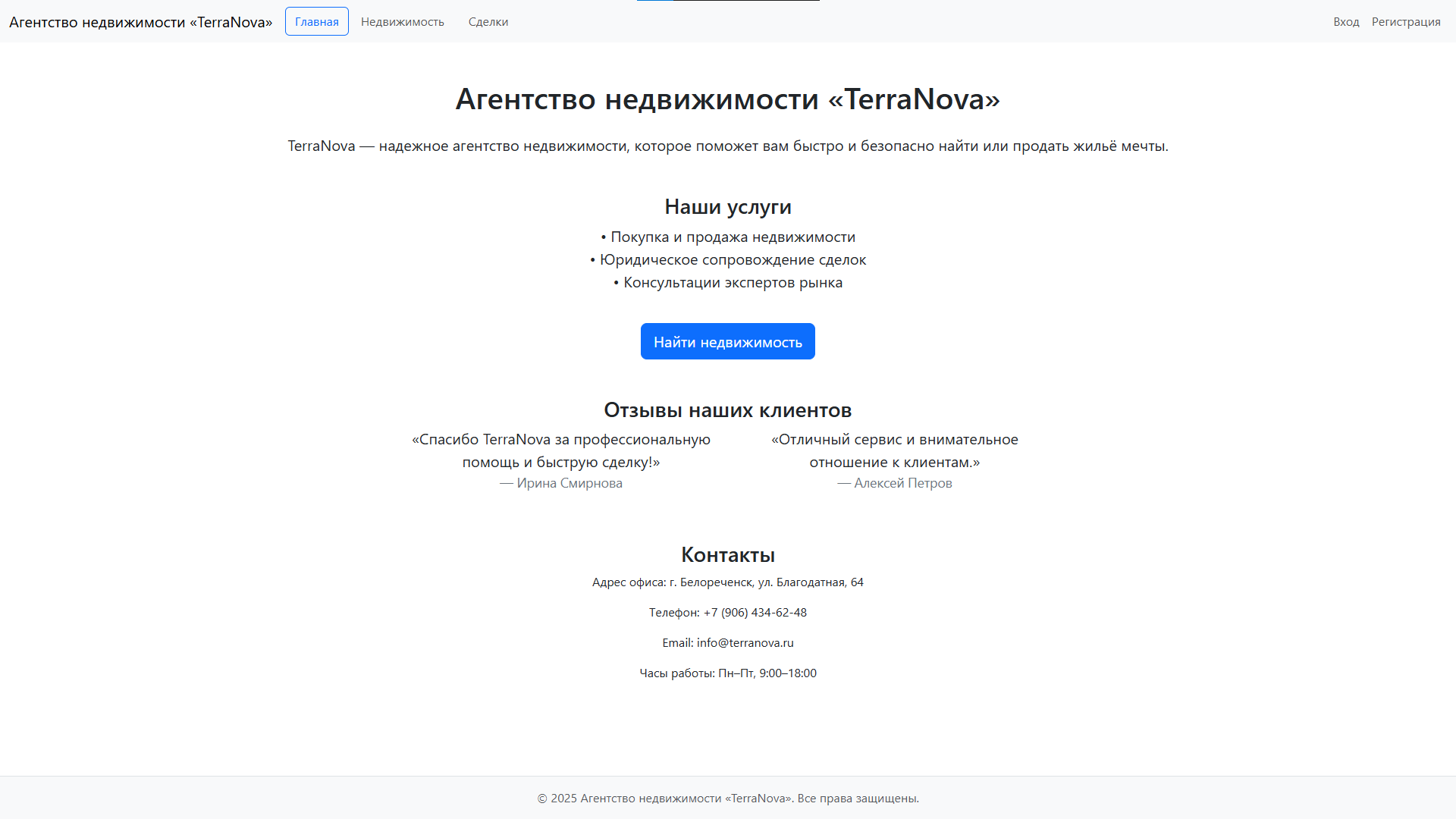


Рисунок 13 - Главная страница

На главной странице отображается основная информация о риэлторском агентстве «TerraNova». Посетителям представлено краткое описание агентства, перечень предоставляемых услуг (покупка и продажа недвижимости, юридическое сопровождение, консультации экспертов), а также отзывы довольных клиентов.

Раздел «Контакты» содержит адрес, телефон, электронную почту и режим работы офиса. Также на странице расположена кнопка для перехода к выбору недвижимости.

Для доступа к данной странице можно использовать один из следующих URL: /, /index, /main.

2. Страница регистрации

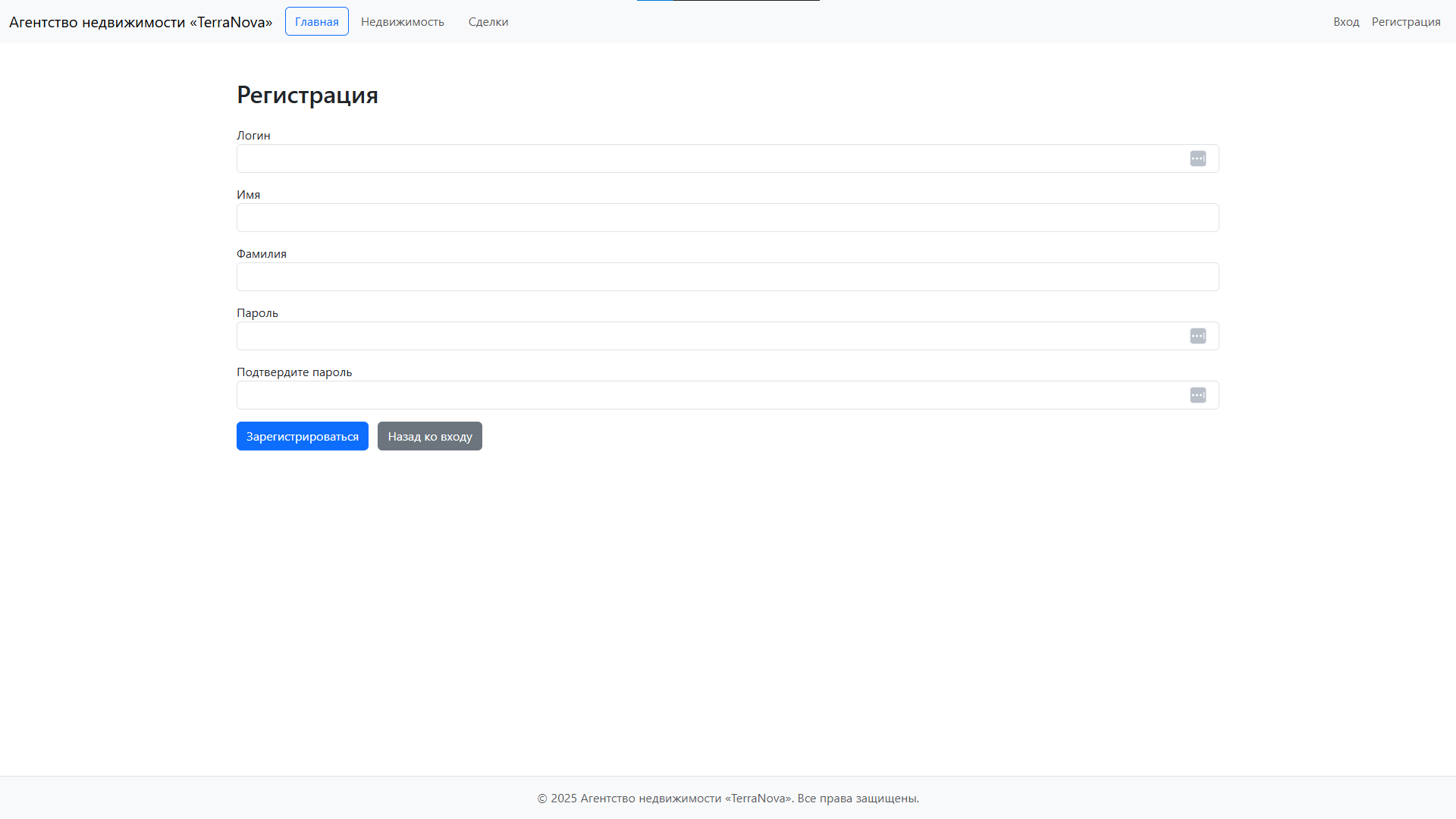


Рисунок 14 - Страница регистрации

Страница регистрации предназначена для создания новой учётной записи пользователя на платформе агентства недвижимости «TerraNova». Форма регистрации включает следующие обязательные поля:

- Логин (уникальное имя пользователя),

- Имя,

- Фамилия,

- Пароль,

- Подтверждение пароля (для исключения ошибок при вводе).

После заполнения формы пользователь может нажать кнопку «Зарегистрироваться» для отправки данных. При необходимости вернуться на страницу входа предусмотрена кнопка «Назад ко входу».

Адрес страницы регистрации: /registration.

3. Страница авторизации

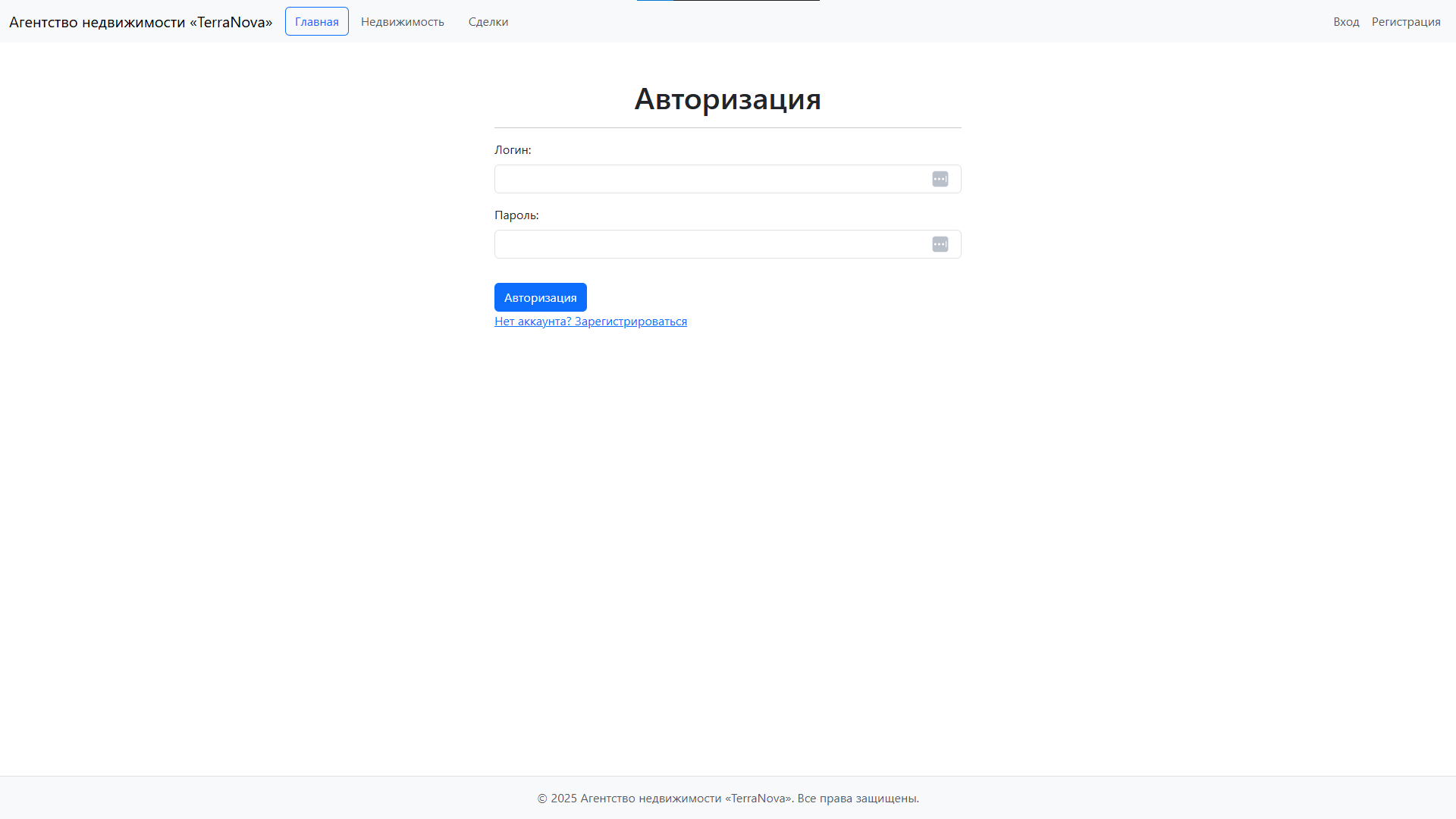


Рисунок 14 - Страница авторизации

Страница авторизации предназначена для входа пользователей в систему «TerraNova». Она включает поля для ввода логина и пароля, кнопку «Авторизация» для подтверждения данных, а также ссылку «Забыли пароль?» для восстановления доступа. Интерфейс минималистичен, с четким расположением элементов и фирменным стилем компании, обеспечивая удобство и безопасность при входе.

4. Страница профиля

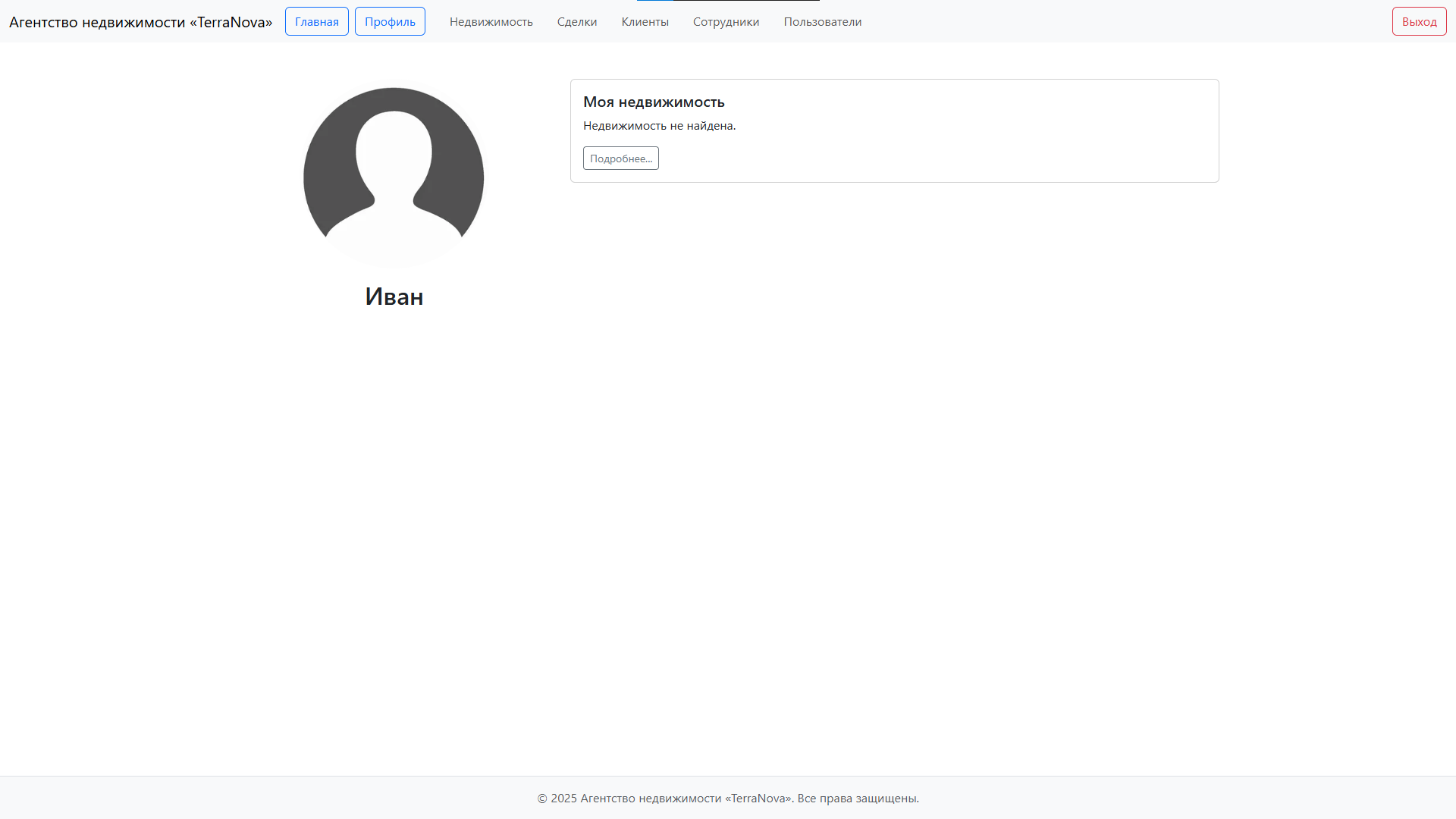


Рисунок 15 - Страница профиля

Страница профиля в системе «TerraNova» отображает основную информацию о пользователе. В центре представлена аватарка и имя пользователя. Справа находится раздел «Моя недвижимость», где отображаются недвижимости, с кнопкой «Подробнее» для перехода к дополнительной информации. В верхней части страницы расположено меню навигации с разделами «Профиль», «Недвижимость», «Ссылки», «Клиенты», «Сотрудники» и «Подробней», а также кнопка «Выход» для завершения сеанса. Дизайн страницы прост и функционален, обеспечивая удобный доступ к ключевым функциям.

5. Страница недвижимости

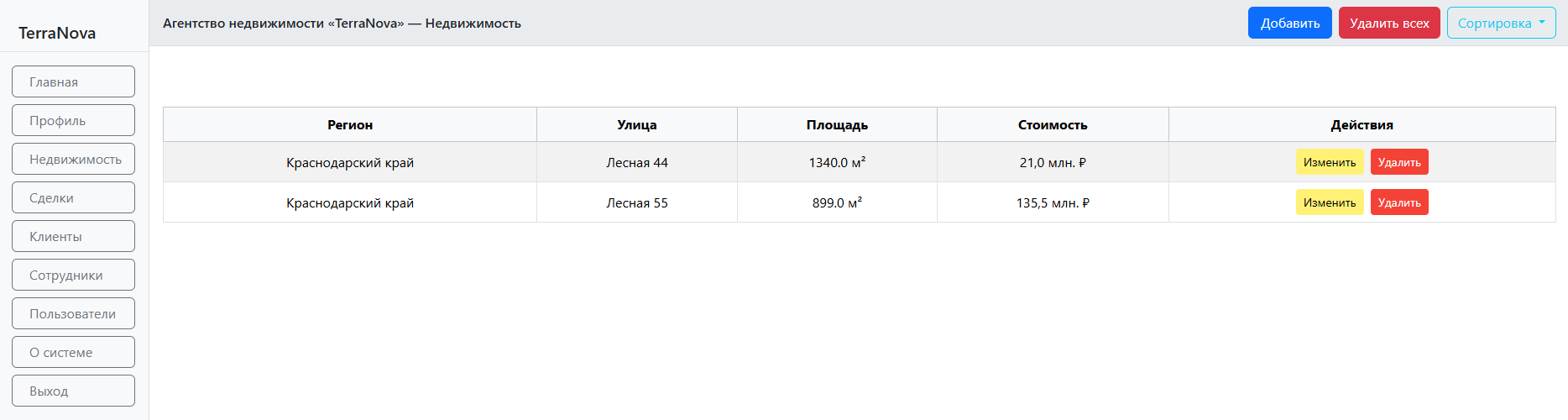


Рисунок 16 - Страница недвижимости

Страница недвижимости в системе «TerraNova» предназначена для управления объектами недвижимости. Она содержит таблицу с колонками: «Регион», «Улица», «Площадь», «Стоимость» и «Действия». Для каждого объекта доступны действия: «Изменить» (желтая кнопка) и «Удалить» (красная кнопка). Слева расположено меню навигации с разделами, включая «Недвижимость», «Клиенты», «Сотрудники» и другие. Вверху страницы есть кнопки «Добавить» и «Удалить», а также иконка для дополнительных настроек. Интерфейс удобен для быстрого просмотра и редактирования данных об объектах.

6. Страница формы недвижимости

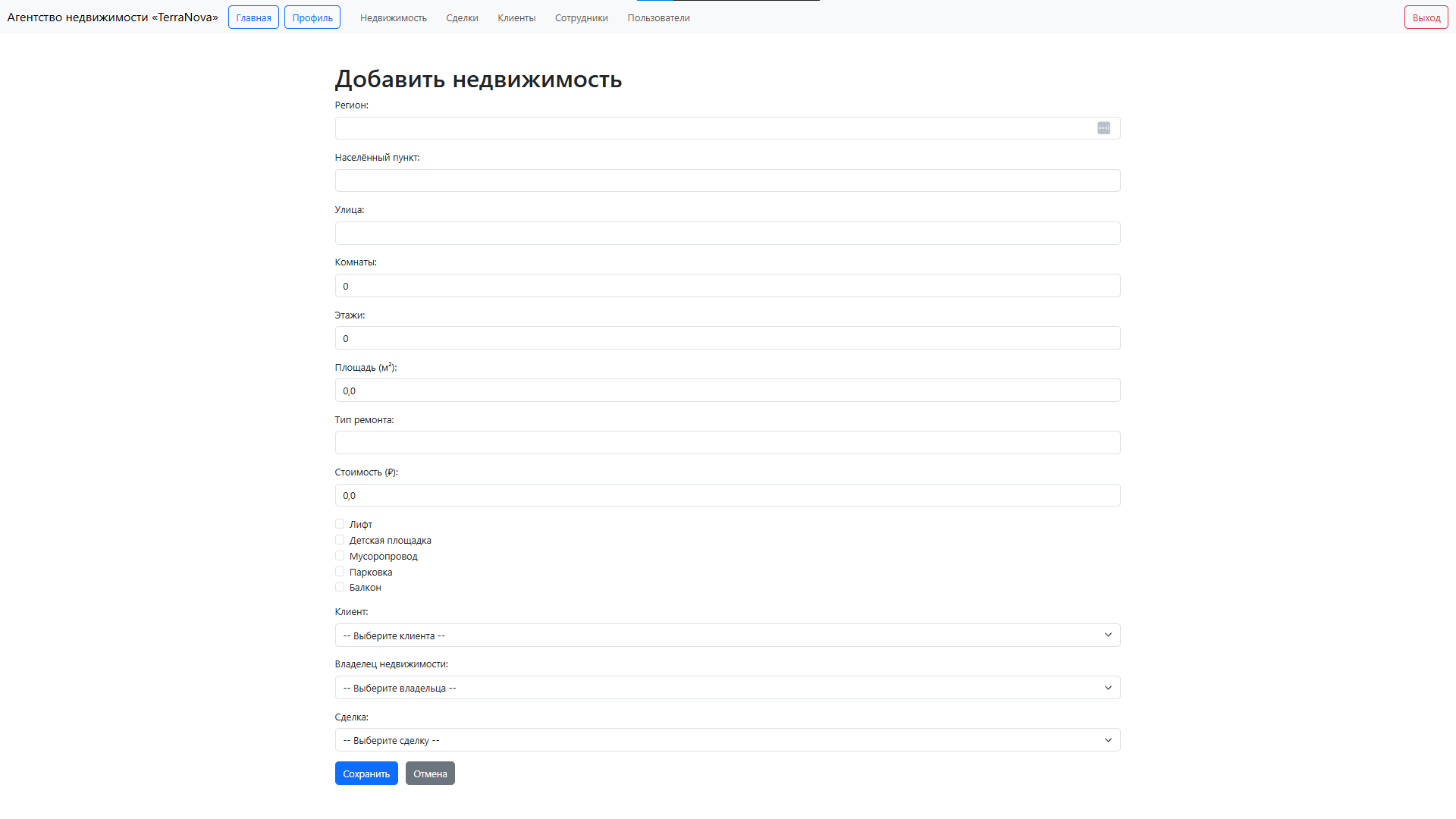


Рисунок 17 - Страница формы недвижимости

Страница формы недвижимости в системе «TerraNova» предназначена для добавления или редактирования данных об объектах недвижимости. Она включает поля для указания региона, наименование пункта, улицы, комментариев, этажа, площади, стоимости, типа ремонта, состояния, наличия лифта, а также выпадающие списки для выбора вида сделки. Пользователь может сохранить изменения с помощью кнопки «Сохранить» или отменить действие кнопкой «Отмена». Интерфейс организован в виде таблицы с четким расположением полей, обеспечивая удобство ввода и проверки информации.

7. Страница сделок

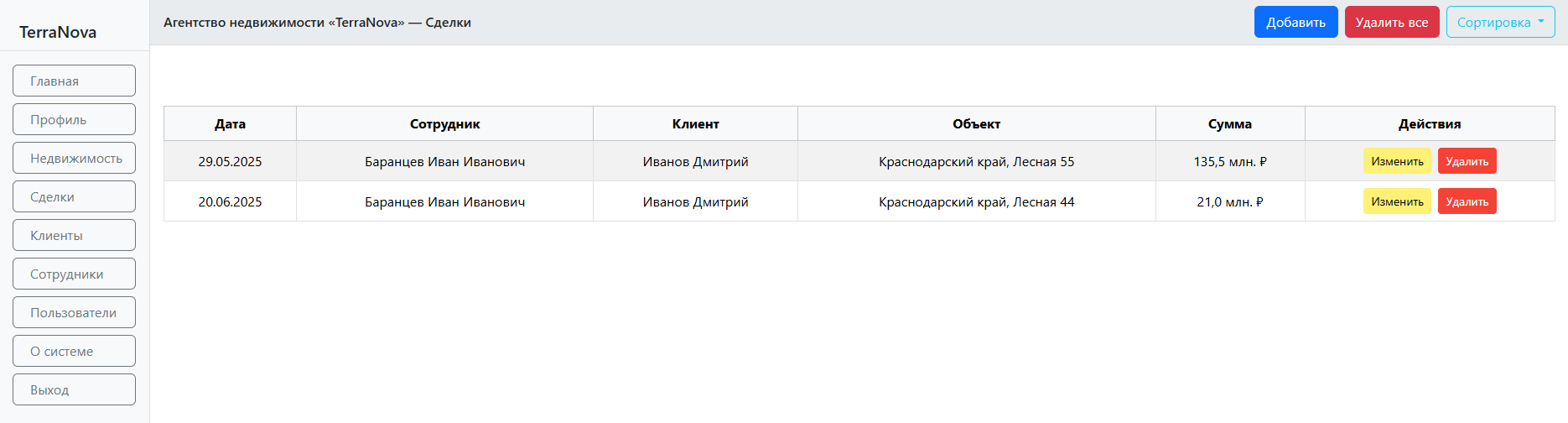


Рисунок 18 - Страница сделок

Страница сделок в системе «TerraNova» предназначена для управления и просмотра информации о совершенных сделках. Она включает таблицу с колонками: «Дата», «Сотрудник», «Клиент», «Объект», «Сумма» и «Действия». Для каждой сделки доступны действия: «Изменить» (желтая кнопка) и «Удалить» (красная кнопка). Слева расположено меню навигации с разделами, включая «Сделки», «Недвижимость», «Клиенты» и другие. Вверху страницы есть кнопки «Добавить» и «Удалить все», а также иконка для дополнительных настроек. Интерфейс удобен для мониторинга и управления сделками.

8. Страница форма сделок

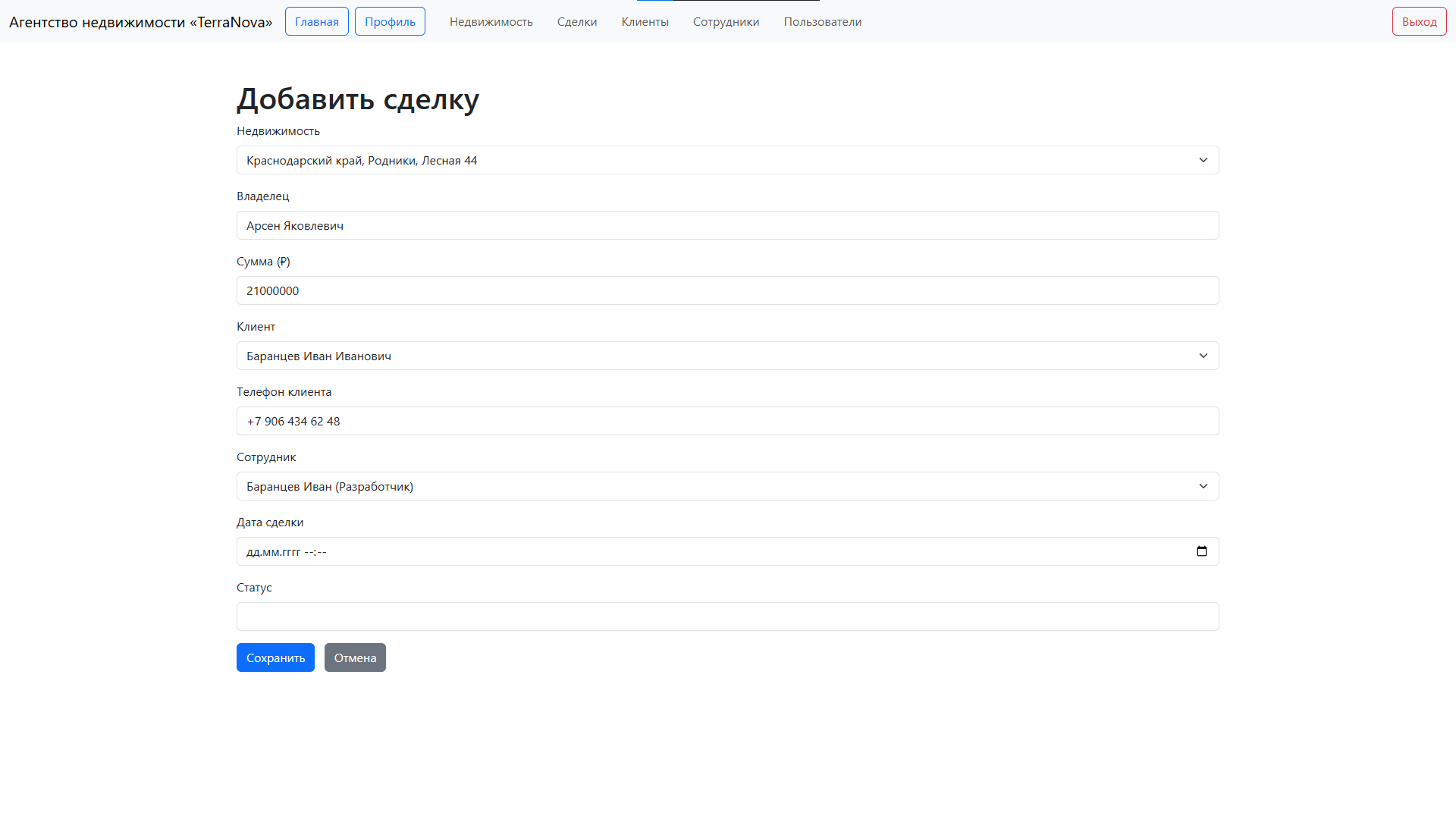


Рисунок 19 - Страница формы сделок

Страница формы сделок в системе «TerraNova» предназначена для добавления или редактирования информации о сделках. Она включает поля для выбора недвижимости, владельца, суммы сделки, клиента, телефона клиента, сотрудника, даты сделки (ДД.ММ.ГГГГ), а также статуса сделки. Пользователь может сохранить данные с помощью кнопки «Сохранить» или отменить изменения кнопкой «Отмена». Интерфейс организован в виде таблицы с четким расположением полей, обеспечивая удобство ввода и проверки информации.

9. Страница клиентов

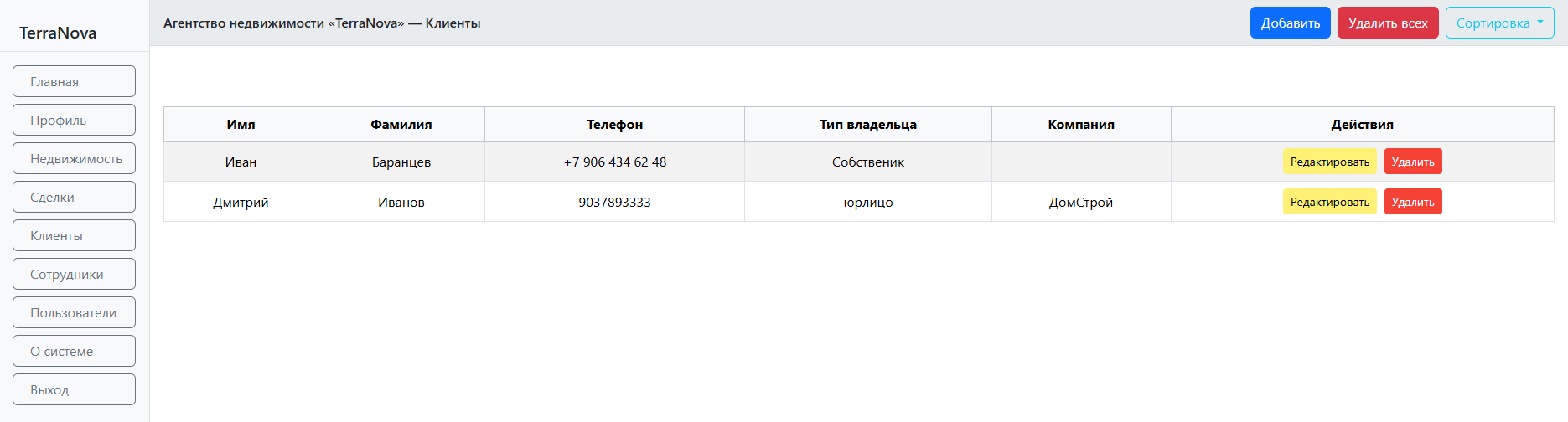


Рисунок 20 - Страница клиентов

Страница клиентов в системе «TerraNova» предназначена для управления и просмотра информации о клиентах. Она включает таблицу с колонками: «Имя», «Фамилия», «Телефон», «Тип владельца», «Компания» и «Действия».

Для каждого клиента доступны действия: «Редактировать» (желтая кнопка) и «Удалить» (красная кнопка). Слева расположено меню навигации с разделами, включая «Клиенты», «Недвижимость», «Сделки» и другие. Вверху страницы есть кнопки «Добавить» и «Удалить все», а также иконка для дополнительных настроек. Интерфейс удобен для работы с клиентской базой.

10. Страница формы клиента

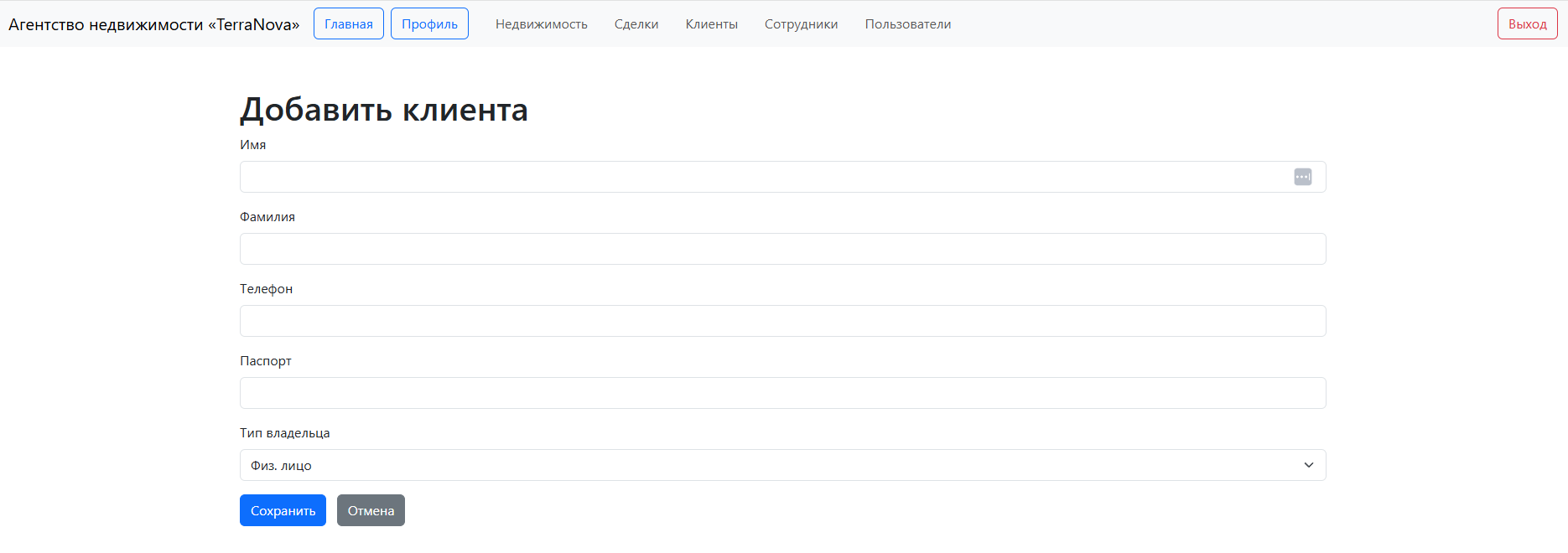


Рисунок 21 - Страница формы клиента

Страница формы клиента в системе «TerraNova» предназначена для добавления или редактирования данных о клиентах. Она включает поля для ввода имени, фамилии, телефона, паспорта, типа владельца («физ. лицо» или «юр. лицо»), если это «юр. лицо», то появляется еще два поля ввода компания и лицензия. Пользователь может сохранить изменения с помощью кнопки «Сохранить» или отменить действие кнопкой «Отмена». Интерфейс организован в виде таблицы с четким расположением полей, обеспечивая удобство ввода и проверки информации о клиентах.

11. Страница сотрудников

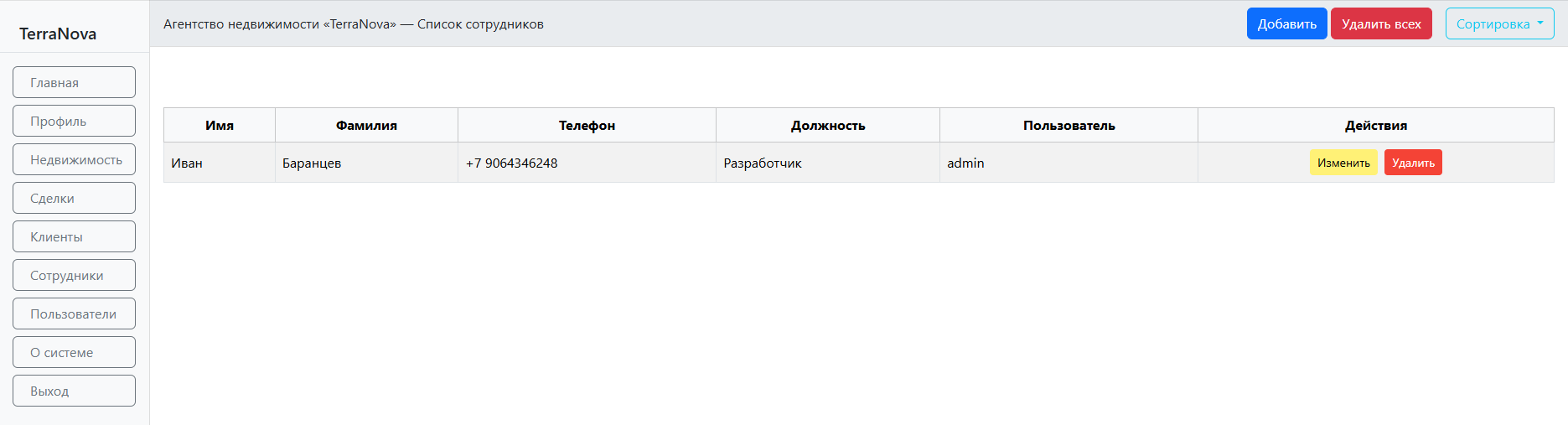


Рисунок 22 – Страница сотрудников

Страница сотрудников в системе «TerraNova» предназначена для управления и просмотра информации о сотрудниках. Она включает таблицу с колонками: «Имя», «Фамилия», «Телефон», «Должность», «Пользователь» и «Действия». Для каждого сотрудника доступны действия: «Изменить» (желтая кнопка) и «Удалить» (красная кнопка). Слева расположено меню навигации с разделами, включая «Сотрудники», «Клиенты», «Недвижимость» и другие. Вверху страницы есть кнопки «Добавить» и «Удалить все», а также иконка для дополнительных настроек. Интерфейс удобен для работы с данными сотрудников.

12. Страница формы сотрудников

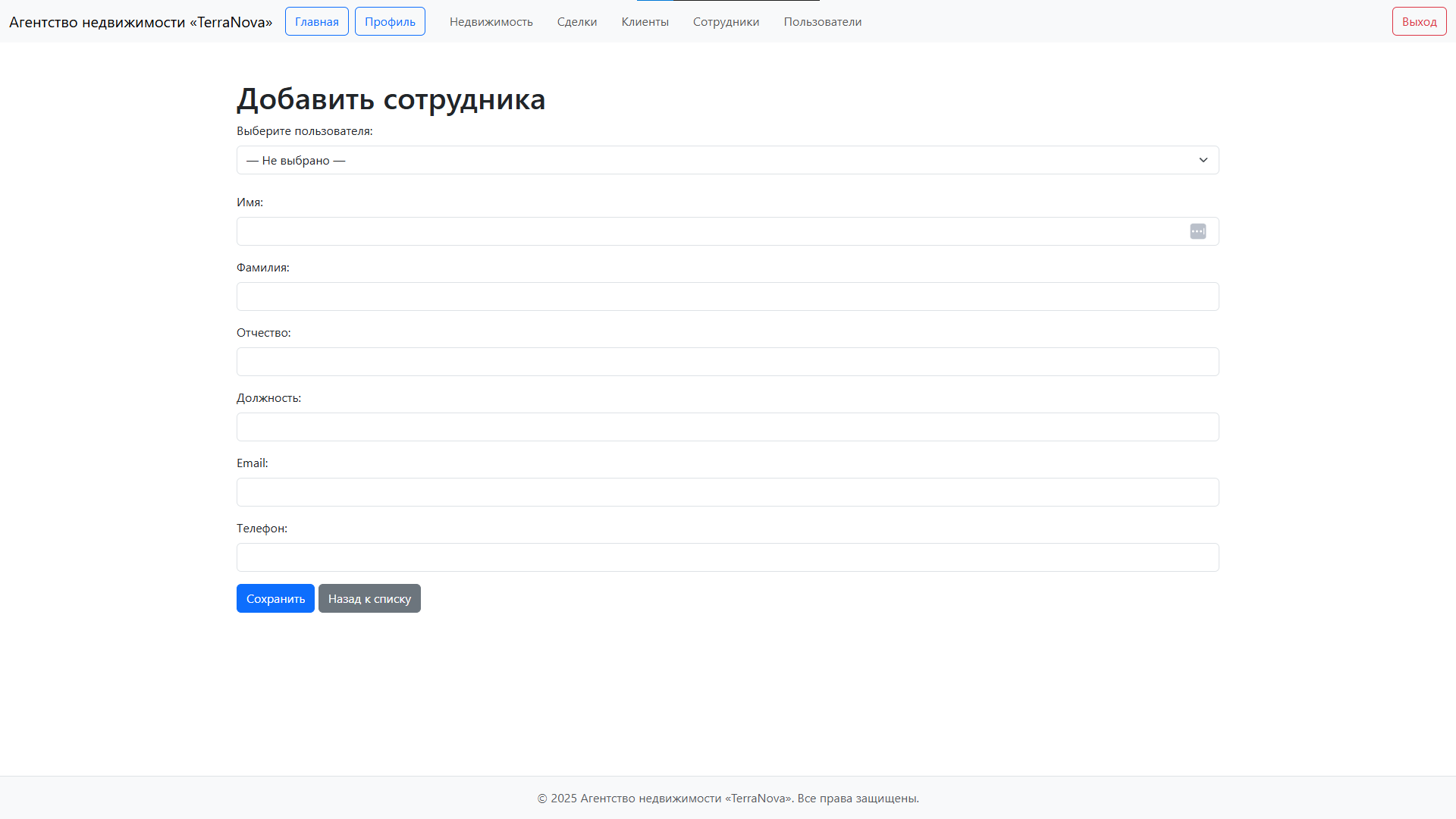


Рисунок 23 – Страница формы сотрудников

Страница формы сотрудников в системе «TerraNova» предназначена для добавления или редактирования данных о сотрудниках. Она включает поля для выбора пользователя, ввода имени, фамилии, отчества, должности, email и телефона. Пользователь может сохранить изменения с помощью кнопки «Сохранить» или отменить действие кнопкой «Отмена». Также доступна кнопка «Назад к списку» для возврата к списку сотрудников. Интерфейс организован в виде таблицы с четким расположением полей, обеспечивая удобство ввода и проверки информации о сотрудниках.

13. Страница пользователей

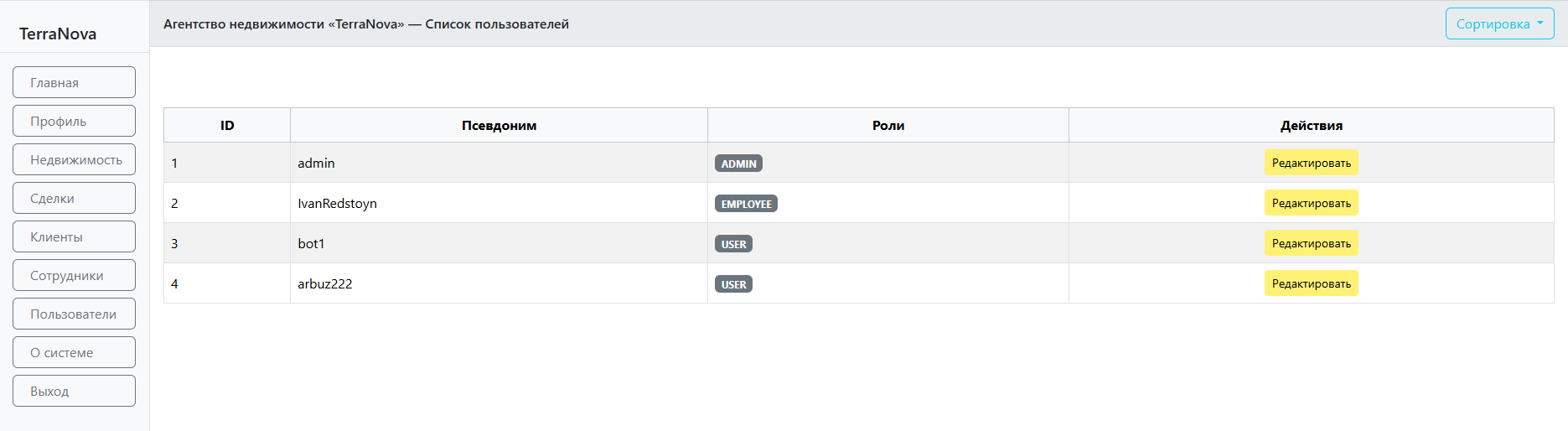


Рисунок 24 – Страница пользователей

Страница пользователей в системе «TerraNova» предназначена для управления и просмотра списка пользователей. Она включает таблицу с колонками: «ID», «Псевдоним», «Роли» и «Действия». Для каждого пользователя доступно действие «Редактировать» (желтая кнопка). Слева расположено меню навигации с разделами, включая «Пользователи», «Сотрудники», «Клиенты» и другие. Вверху страницы есть иконка для дополнительных настроек. Интерфейс прост и удобен для администрирования пользователей.

14. Страница редактирования пользователей

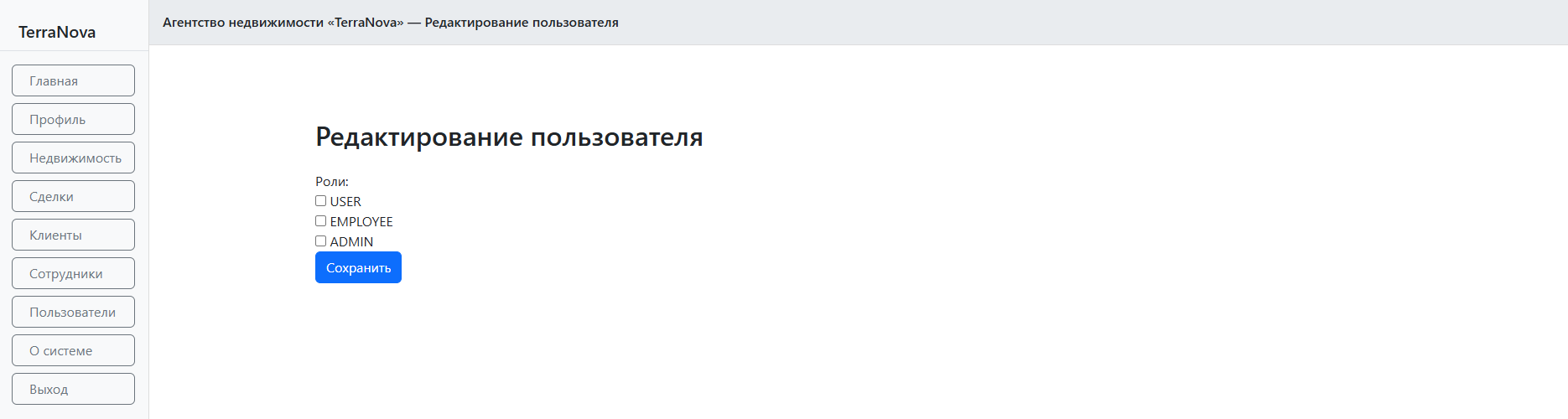


Рисунок 25 – Страница редактирования пользователей

Страница редактирования пользователей в системе «TerraNova» предназначена для редактирования данных о пользователях. Она включает поля для выбора роли пользователя админ, сотрудник, пользователь. Пользователь может сохранить изменения с помощью кнопки «Сохранить». Интерфейс организован в виде таблицы с четким расположением, обеспечивая удобство выбора роли и проверки информации о пользователе.

ЗАКЛЮЧЕНИЕ

В ходе выполнения работы была создана информационная система для агентства недвижимости «TerraNova», направленная на автоматизацию ключевых процессов, связанных с управлением недвижимостью, клиентами и сделками.

Была разработана и внедрена база данных, которая включает информацию о недвижимости, клиентах, сотрудниках и заключённых сделках. Это позволяет сотрудникам агентства оперативно получать необходимую информацию для эффективного обслуживания клиентов и ведения учета.

Система обеспечивает удобное управление объектами недвижимости, регистрацию и обработку сделок, а также учёт клиентов и сотрудников. Благодаря этому достигается повышение прозрачности и контроля над бизнес-процессами агентства.

Реализованные функциональные возможности способствуют улучшению качества обслуживания клиентов, сокращению времени на обработку заявок и сделок, а также оптимизации работы персонала.

Рекомендуется регулярно обновлять и модернизировать систему с учётом развития рынка недвижимости и потребностей агентства, а также расширять функциональность для дальнейшего повышения эффективности работы.

Таким образом, разработанная информационная система для агентства недвижимости «TerraNova» является надёжным и эффективным инструментом для управления деятельностью агентства и повышения уровня сервиса для клиентов.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Java. Полное руководство, 12-е изд. : Пер. с англ. - СПб. "Диалектика"; 2023. - 1344 с.: ил. - Парал. тит. англ.
2. Spring в действии. 6-е изд. / пер. с англ. А. Н. Киселева. – М.: ДМКПресс, 2022. – 544 с.: ил.
3. Hibernate для начинающих [Электронный ресурс]. Режим доступа: <https://habr.com/ru/articles/320542/>
4. Spring Framework Reference Documentation [Электронный ресурс]. Режим доступа: <https://docs.spring.io/spring-framework/docs/5.3.9/reference/html/>
5. Руководство по Spring Data JPA [Электронный ресурс]. Режим доступа: <https://spring.io/projects/spring-data-jpa>
6. Thymeleaf Documentation [Электронный ресурс]. Режим доступа: https://www.thymeleaf.org/documentation.html
7. PostgreSQL Documentation [Электронный ресурс]. Режим доступа: <https://www.postgresql.org/docs/>
8. Maven: руководство пользователя [Электронный ресурс]. Режим доступа: <https://maven.apache.org/guides/index.html>
9. ГОСТ 34.602-89 Техническое задание на создание автоматизированной системы
10. ГОСТ 19.201-78 Автоматизированные системы. Техническая документация. Общие требования

ПРИЛОЖЕНИЕ

ПРИЛОЖЕНИЕ

package org.example.terranova.config;  
  
import org.springframework.context.annotation.Bean;  
import org.springframework.context.annotation.Configuration;  
import org.springframework.security.config.annotation.web.builders.HttpSecurity;  
import org.springframework.security.config.annotation.web.configuration.EnableWebSecurity;  
import org.springframework.security.crypto.bcrypt.BCryptPasswordEncoder;  
import org.springframework.security.web.SecurityFilterChain;  
  
@Configuration  
@EnableWebSecurity  
public class SecurityConfig {  
  
 @Bean  
 public BCryptPasswordEncoder passwordEncoder() {  
 return new BCryptPasswordEncoder();  
 }  
  
 @Bean  
 public SecurityFilterChain securityFilterChain(HttpSecurity http) throws Exception {  
 http  
 .authorizeHttpRequests(requests -> requests  
 .requestMatchers(  
 "/login", "/include/registration/\*\*", "/logout",  
 "/webjars/\*\*", "/css/\*\*", "/js/\*\*", "/images/\*\*", "/fonts/\*\*",  
 "/header", "/footer", "/about", "/include/about",  
 "/uploads/\*\*", "/img/\*\*"  
 ).permitAll()  
 .requestMatchers("/", "/realty/realty\_list", "/include/profile", "/include/profile/\*\*", "/deal/deal\_list")  
 .hasAnyRole("USER", "EMPLOYEE", "ADMIN")  
 .requestMatchers("/realty/realty\_form", "/deal/deal\_form", "/client/client\_list", "/client/client\_form")  
 .hasAnyRole("EMPLOYEE", "ADMIN")  
 .requestMatchers("/employee/employee\_form", "/employee/employee\_list", "/user/user\_list", "/user/user\_form")  
 .hasRole("ADMIN")  
 .anyRequest().authenticated()  
 )  
 .formLogin(form -> form  
 .loginPage("/login")  
 .defaultSuccessUrl("/profile", true)  
 .permitAll()  
 )  
 .logout(logout -> logout  
 .logoutUrl("/logout")  
 .logoutSuccessUrl("/")  
 .permitAll()  
 );  
  
 return http.build();  
 }  
}

package org.example.terranova.config;  
  
import org.springframework.context.annotation.Configuration;  
import org.springframework.web.servlet.config.annotation.ResourceHandlerRegistry;  
import org.springframework.web.servlet.config.annotation.ViewControllerRegistry;  
import org.springframework.web.servlet.config.annotation.WebMvcConfigurer;  
import org.springframework.beans.factory.annotation.Value;  
  
@Configuration  
public class WebConfig implements WebMvcConfigurer {  
  
 private final String uploadDir;  
  
 public WebConfig(@Value("${upload.path:uploads}") String uploadDir) {  
 this.uploadDir = uploadDir.endsWith("/") ? uploadDir : uploadDir + "/";  
 }  
  
 @Override  
 public void addViewControllers(ViewControllerRegistry registry) {  
 registry.addViewController("/login").setViewName("/include/login");  
 registry.addViewController("/").setViewName("/include/index");  
 }  
  
 @Override  
 public void addResourceHandlers(ResourceHandlerRegistry registry) {  
 *// Абсолютный путь с префиксом "file:" для отдачи файлов* registry.addResourceHandler("/uploads/\*\*")  
 .addResourceLocations("file:" + uploadDir);  
 }  
}

package org.example.terranova.controller;  
  
import org.springframework.stereotype.Controller;  
import org.springframework.web.bind.annotation.GetMapping;  
  
@Controller  
public class AboutController {  
  
 @GetMapping("/about")  
 public String aboutPage() {  
 return "include/about";  
 }  
}

package org.example.terranova.controller;  
  
import jakarta.validation.Valid;  
import org.example.terranova.model.Client;  
import org.example.terranova.service.ClientService;  
import org.springframework.stereotype.Controller;  
import org.springframework.ui.Model;  
import org.springframework.validation.BindingResult;  
import org.springframework.web.bind.annotation.\*;  
  
import java.util.List;  
import java.util.Optional;  
  
@Controller  
@RequestMapping("/client")  
public class ClientController {  
  
 private final ClientService clientService;  
  
 public ClientController(ClientService clientService) {  
 this.clientService = clientService;  
 }  
  
 @GetMapping  
 public String listClients(Model model,  
 @RequestParam(required = false) String sortField,  
 @RequestParam(required = false) String sortDir) {  
  
 List<Client> clients;  
 String direction = (sortDir != null && sortDir.equalsIgnoreCase("desc")) ? "desc" : "asc";  
  
 if (sortField == null || sortField.isBlank()) {  
 clients = clientService.findAll();  
 sortField = null;  
 direction = null;  
 } else {  
 clients = clientService.findAllSortedByField(sortField.toLowerCase(), direction);  
 }  
  
 model.addAttribute("clients", clients);  
 model.addAttribute("sortField", sortField);  
 model.addAttribute("sortDir", direction);  
  
 return "client/client\_list";  
 }  
  
 @GetMapping("/add")  
 public String addClientForm(Model model) {  
 model.addAttribute("client", new Client());  
 return "client/client\_form";  
 }  
  
 @GetMapping("/edit/{id}")  
 public String editClientForm(@PathVariable Long id, Model model) {  
 Optional<Client> clientOpt = clientService.findById(id);  
 if (clientOpt.isEmpty()) {  
 return "redirect:/client";  
 }  
 model.addAttribute("client", clientOpt.get());  
 return "client/client\_form";  
 }  
  
 @PostMapping("/save")  
 public String saveClient(@Valid @ModelAttribute("client") Client client,  
 BindingResult bindingResult) {  
 if (bindingResult.hasErrors()) {  
 return "client/client\_form";  
 }  
 clientService.save(client);  
 return "redirect:/client";  
 }  
  
 @PostMapping("/delete/{id}")  
 public String deleteClient(@PathVariable Long id) {  
 boolean deleted = clientService.deleteByIdIfPossible(id);  
 return deleted ? "redirect:/client" : "redirect:/client?error=true";  
 }  
  
 @PostMapping("/deleteAll")  
 public String deleteAllClients() {  
 int notDeletedCount = clientService.deleteAllExceptWithDeals();  
 if (notDeletedCount > 0) {  
 return "redirect:/client?partialError=" + notDeletedCount;  
 }  
 return "redirect:/client";  
 }  
}

package org.example.terranova.controller;  
  
import com.fasterxml.jackson.core.JsonProcessingException;  
import com.fasterxml.jackson.databind.ObjectMapper;  
import lombok.RequiredArgsConstructor;  
import org.example.terranova.model.Client;  
import org.example.terranova.model.Deal;  
import org.example.terranova.model.Employee;  
import org.example.terranova.model.Realty;  
import org.example.terranova.service.ClientService;  
import org.example.terranova.service.DealService;  
import org.example.terranova.service.EmployeeService;  
import org.example.terranova.service.RealtyService;  
import org.springframework.data.domain.Sort;  
import org.springframework.stereotype.Controller;  
import org.springframework.ui.Model;  
import org.springframework.web.bind.annotation.\*;  
import org.springframework.web.servlet.mvc.support.RedirectAttributes;  
  
import java.util.HashMap;  
import java.util.List;  
import java.util.Map;  
import java.util.stream.Collectors;  
  
@Controller  
@RequestMapping("/deal")  
@RequiredArgsConstructor  
public class DealController {  
  
 private final DealService dealService;  
 private final ClientService clientService;  
 private final RealtyService realtyService;  
 private final EmployeeService employeeService;  
  
 @GetMapping  
 public String listDeals(  
 Model model,  
 @RequestParam(value = "sortField", required = false) String sortField,  
 @RequestParam(value = "sortDir", required = false) String sortDir  
 ) {  
 Sort sort;  
  
 if (sortField == null || sortField.isEmpty()) {  
 sort = Sort.unsorted();  
 } else {  
 try {  
 *// Защита от null и неправильных значений* Sort.Direction direction = (sortDir == null || (!sortDir.equalsIgnoreCase("asc") && !sortDir.equalsIgnoreCase("desc")))  
 ? Sort.Direction.ASC  
 : Sort.Direction.fromString(sortDir);  
 sort = Sort.by(direction, sortField);  
 } catch (IllegalArgumentException ex) {  
 sort = Sort.by(Sort.Direction.ASC, "date");  
 sortField = "date";  
 sortDir = "asc";  
 }  
 }  
  
 List<Deal> deals = dealService.getAllDeals(sort);  
  
 model.addAttribute("deals", deals);  
 model.addAttribute("sortField", sortField);  
 model.addAttribute("sortDir", sortDir);  
 model.addAttribute("reverseSortDir", (sortDir != null && sortDir.equalsIgnoreCase("asc")) ? "desc" : "asc");  
  
 return "deal/deal\_list";  
 }  
  
 @PostMapping("/save")  
 public String saveDeal(@ModelAttribute Deal deal) {  
 dealService.saveDeal(deal);  
 return "redirect:/deal";  
 }  
  
 @GetMapping({"/add", "/edit/{id}"})  
 public String showDealForm(@PathVariable(required = false) Long id, Model model) throws JsonProcessingException {  
 Deal deal = (id != null) ? dealService.findById(id).orElse(new Deal()) : new Deal();  
  
 List<Realty> realties = realtyService.findAll();  
 List<Client> clients = clientService.findAll();  
 List<Employee> employees = employeeService.findAll();  
  
 model.addAttribute("deal", deal);  
 model.addAttribute("realties", realties);  
 model.addAttribute("clients", clients);  
 model.addAttribute("employees", employees);  
  
 ObjectMapper objectMapper = new ObjectMapper();  
  
 *// JSON-данные по недвижимости* List<Map<String, Object>> realtyJsonList = realties.stream().map(r -> {  
 Map<String, Object> map = new HashMap<>();  
 map.put("id", r.getId());  
 map.put("cost", r.getCost());  
 map.put("ownerName", r.getOwner() != null ? r.getOwner().getFullName() : "—");  
 return map;  
 }).collect(Collectors.toList());  
  
 *// JSON-данные по клиентам* List<Map<String, Object>> clientJsonList = clients.stream().map(c -> {  
 Map<String, Object> map = new HashMap<>();  
 map.put("id", c.getId());  
 map.put("phone", c.getPhone());  
 return map;  
 }).collect(Collectors.toList());  
  
 model.addAttribute("realtyJson", objectMapper.writeValueAsString(realtyJsonList));  
 model.addAttribute("clientJson", objectMapper.writeValueAsString(clientJsonList));  
  
 return "deal/deal\_form";  
 }  
  
 @PostMapping("/delete/{id}")  
 public String deleteDeal(@PathVariable Long id, RedirectAttributes redirectAttributes) {  
 try {  
 dealService.deleteDealById(id);  
 redirectAttributes.addFlashAttribute("successMessage", "Сделка успешно удалена");  
 } catch (Exception e) {  
 redirectAttributes.addFlashAttribute("errorMessage", "Ошибка при удалении сделки");  
 }  
 return "redirect:/deal";  
 }  
  
 @PostMapping("/deleteAll")  
 public String deleteAllDeals(RedirectAttributes redirectAttributes) {  
 try {  
 dealService.deleteAllDeals();  
 redirectAttributes.addFlashAttribute("successMessage", "Все сделки успешно удалены");  
 } catch (Exception e) {  
 redirectAttributes.addFlashAttribute("errorMessage", "Ошибка при удалении всех сделок");  
 }  
 return "redirect:/deal";  
 }  
  
}

package org.example.terranova.controller;  
  
import jakarta.validation.Valid;  
import org.example.terranova.model.Employee;  
import org.example.terranova.service.EmployeeService;  
import org.example.terranova.service.UserService;  
import org.springframework.data.domain.Sort;  
import org.springframework.http.ResponseEntity;  
import org.springframework.stereotype.Controller;  
import org.springframework.ui.Model;  
import org.springframework.validation.BindingResult;  
import org.springframework.web.bind.annotation.\*;  
import org.springframework.web.servlet.mvc.support.RedirectAttributes;  
  
import java.util.List;  
import java.util.Map;  
  
@Controller  
@RequestMapping("/employee")  
public class EmployeeController {  
  
 private final EmployeeService employeeService;  
 private final UserService userService;  
  
 public EmployeeController(EmployeeService employeeService, UserService userService) {  
 this.employeeService = employeeService;  
 this.userService = userService;  
 }  
  
 @GetMapping  
 public String listEmployees(@RequestParam(value = "sortBy", required = false) String sortBy,  
 @RequestParam(value = "sortDir", required = false) String sortDir,  
 Model model) {  
 List<Employee> employees;  
  
 if (sortBy == null || sortBy.isEmpty()) {  
 employees = employeeService.getAllEmployeesWithUsers();  
 } else {  
 Sort.Direction direction = "desc".equalsIgnoreCase(sortDir) ? Sort.Direction.DESC : Sort.Direction.ASC;  
 employees = employeeService.getAllEmployeesSorted(Sort.by(direction, sortBy));  
 }  
  
 model.addAttribute("employees", employees);  
 model.addAttribute("sortBy", sortBy);  
 model.addAttribute("sortDir", sortDir);  
 return "employee/employee\_list";  
 }  
  
 @GetMapping("/add")  
 public String showAddForm(Model model) {  
 model.addAttribute("employee", new Employee());  
 model.addAttribute("users", userService.findAll());  
 return "employee/employee\_form";  
 }  
  
 @PostMapping("/save")  
 public String saveEmployee(@Valid @ModelAttribute("employee") Employee employee,  
 BindingResult result,  
 Model model) {  
 if (result.hasErrors()) {  
 model.addAttribute("users", userService.findAll());  
 return "employee/employee\_form";  
 }  
  
 try {  
 employeeService.saveEmployee(employee);  
 return "redirect:/employee";  
 } catch (Exception e) {  
 model.addAttribute("validationErrors", List.of("Ошибка при сохранении сотрудника: " + e.getMessage()));  
 model.addAttribute("users", userService.findAll());  
 return "employee/employee\_form";  
 }  
 }  
  
 @GetMapping("/edit/{id}")  
 public String showEditForm(@PathVariable Long id, Model model) {  
 Employee employee = employeeService.getEmployeeById(id);  
 if (employee == null) {  
 return "redirect:/employee";  
 }  
 model.addAttribute("employee", employee);  
 model.addAttribute("users", userService.findAll());  
 return "employee/employee\_form";  
 }  
  
 @GetMapping("/delete/{id}")  
 public String deleteEmployee(@PathVariable Long id, RedirectAttributes redirectAttributes) {  
 try {  
 employeeService.deleteEmployee(id);  
 } catch (IllegalStateException e) {  
 redirectAttributes.addAttribute("error", true);  
 }  
 return "redirect:/employee";  
 }  
  
 @GetMapping("/deleteAll")  
 public String deleteAllEmployees(RedirectAttributes redirectAttributes) {  
 List<Employee> notDeleted = employeeService.deleteAllExceptWithDeals();  
  
 if (!notDeleted.isEmpty()) {  
 redirectAttributes.addAttribute("partialError", notDeleted.size());  
 }  
  
 return "redirect:/employee";  
 }  
  
 @GetMapping("/patronymic/{userId}")  
 @ResponseBody  
 public ResponseEntity<Map<String, String>> getPatronymicByUserId(@PathVariable Long userId) {  
 Employee employee = employeeService.getEmployeeByUserId(userId);  
 if (employee != null && employee.getPatronymic() != null) {  
 return ResponseEntity.ok(Map.of("patronymic", employee.getPatronymic()));  
 } else {  
 return ResponseEntity.ok(Map.of("patronymic", ""));  
 }  
 }  
}

package org.example.terranova.controller;  
  
import jakarta.servlet.ServletContext;  
import lombok.Setter;  
import org.example.terranova.model.Deal;  
import org.example.terranova.model.Realty;  
import org.example.terranova.model.User;  
import org.example.terranova.service.DealService;  
import org.example.terranova.service.RealtyService;  
import org.example.terranova.service.UserService;  
import org.springframework.beans.factory.annotation.Autowired;  
import org.springframework.beans.factory.annotation.Value;  
import org.springframework.core.env.Environment;  
import org.springframework.security.core.userdetails.UsernameNotFoundException;  
import org.springframework.stereotype.Controller;  
import org.springframework.ui.Model;  
import org.springframework.web.bind.annotation.GetMapping;  
import org.springframework.web.bind.annotation.PostMapping;  
import org.springframework.web.bind.annotation.RequestParam;  
import org.springframework.web.multipart.MultipartFile;  
  
import java.io.File;  
import java.io.IOException;  
import java.security.Principal;  
import java.util.List;  
import java.util.Optional;  
  
@Controller  
public class ProfileController {  
  
 @Autowired  
 private Environment env; *// Правильный импорт!* @Autowired  
 private final UserService userService;  
 private final RealtyService realtyService;  
 private final DealService dealService;  
 @Autowired  
 private ServletContext servletContext;  
  
 *// Значение можно получить из application.properties (fallback — "uploads")* @Setter  
 @Value("${upload.path:uploads}")  
 private String uploadPath;  
  
 public ProfileController(UserService userService, RealtyService realtyService, DealService dealService) {  
 this.userService = userService;  
 this.realtyService = realtyService;  
 this.dealService = dealService;  
 }  
  
 @GetMapping("/profile")  
 public String userProfile(Principal principal, Model model) {  
 User user = userService.findByUsername(principal.getName())  
 .orElseThrow(() -> new UsernameNotFoundException("Пользователь не найден"));  
  
 List<Realty> userRealties = realtyService.findAllByOwner(user);  
 List<Deal> userDeals = dealService.findAllByUser(user);  
  
 model.addAttribute("user", user);  
 model.addAttribute("realtyList", userRealties);  
 model.addAttribute("dealList", userDeals);  
  
 return "include/profile";  
 }  
  
 @PostMapping("/profile/avatar")  
 public String handleAvatarUpload(@RequestParam("avatar") MultipartFile avatar, Principal principal) throws IOException {  
 if (avatar != null && !avatar.isEmpty()) {  
 String relativePath = uploadPath; *// например, "/uploads" или "/terranova/uploads"* String realPath = servletContext.getRealPath(relativePath);  
 if (realPath == null) {  
 throw new IOException("Не удалось определить реальный путь для " + relativePath);  
 }  
  
 File uploadDir = new File(realPath);  
 if (!uploadDir.exists() && !uploadDir.mkdirs()) {  
 throw new IOException("Не удалось создать директорию для загрузки файлов: " + realPath);  
 }  
  
 String filename = System.currentTimeMillis() + "\_" + principal.getName() + ".jpg";  
 File file = new File(uploadDir, filename);  
 avatar.transferTo(file);  
  
 Optional<User> optionalUser = userService.findByUsername(principal.getName());  
 if (optionalUser.isPresent()) {  
 User user = optionalUser.get();  
 user.setAvatarUrl(relativePath + "/" + filename); *// сохраняем относительный URL* userService.save(user);  
 } else {  
 throw new IOException("Пользователь не найден: " + principal.getName());  
 }  
 }  
 return "redirect:/profile";  
 }  
}

package org.example.terranova.controller;  
  
import org.example.terranova.model.Realty;  
import org.example.terranova.model.User;  
import org.example.terranova.service.ClientService;  
import org.example.terranova.service.DealService;  
import org.example.terranova.service.RealtyService;  
import org.example.terranova.service.UserService;  
import org.springframework.stereotype.Controller;  
import org.springframework.ui.Model;  
import org.springframework.web.bind.annotation.\*;  
import org.springframework.web.servlet.mvc.support.RedirectAttributes;  
  
import java.util.Comparator;  
import java.util.List;  
import java.util.NoSuchElementException;  
  
@Controller  
@RequestMapping("/realty")  
public class RealtyController {  
  
 private final RealtyService realtyService;  
 private final UserService userService;  
 private final DealService dealService;  
 private final ClientService clientService;  
  
 public RealtyController(RealtyService realtyService, UserService userService, DealService dealService, ClientService clientService) {  
 this.realtyService = realtyService;  
 this.userService = userService;  
 this.dealService = dealService;  
 this.clientService = clientService;  
 }  
  
 @GetMapping  
 public String listRealty(  
 @RequestParam(required = false) String sortField,  
 @RequestParam(required = false) String sortDir,  
 Model model) {  
  
 List<Realty> realties = realtyService.findAll();  
  
 if (sortField != null && sortDir != null) {  
 Comparator<Realty> comparator = null;  
  
 switch (sortField) {  
 case "cost":  
 comparator = Comparator.comparing(Realty::getCost);  
 break;  
 case "region":  
 comparator = Comparator.comparing(Realty::getRegion, String.CASE\_INSENSITIVE\_ORDER);  
 break;  
 case "street":  
 comparator = Comparator.comparing(Realty::getStreet, String.CASE\_INSENSITIVE\_ORDER);  
 break;  
 }  
  
 if (comparator != null) {  
 if ("desc".equalsIgnoreCase(sortDir)) {  
 comparator = comparator.reversed();  
 }  
 realties.sort(comparator);  
 }  
 }  
  
 model.addAttribute("realties", realties);  
 model.addAttribute("sortField", sortField);  
 model.addAttribute("sortDir", sortDir);  
  
 return "realty/realty\_list";  
 }  
  
 @GetMapping("/add")  
 public String showAddForm(Model model) {  
 model.addAttribute("realty", new Realty());  
 model.addAttribute("users", userService.findAll());  
 model.addAttribute("deals", dealService.findAll());  
 model.addAttribute("clients", clientService.findAll());  
 return "realty/realty\_form";  
 }  
  
 @PostMapping("/save")  
 public String saveRealty(@ModelAttribute Realty realty) {  
 realtyService.saveRealty(realty);  
 return "redirect:/realty";  
 }  
  
 @PostMapping("/delete/{id}")  
 public String deleteRealty(@PathVariable Long id, RedirectAttributes redirectAttributes) {  
 if (dealService.existsByRealtyId(id)) {  
 redirectAttributes.addFlashAttribute("errorMessage", "Невозможно удалить: недвижимость участвует в сделке.");  
 } else {  
 realtyService.deleteById(id);  
 redirectAttributes.addFlashAttribute("successMessage", "Недвижимость успешно удалена.");  
 }  
 return "redirect:/realty";  
 }  
  
 @GetMapping("/edit/{id}")  
 public String showEditForm(@PathVariable Long id, Model model) {  
 Realty realty = realtyService.getRealtyById(id);  
 if (realty == null) {  
 throw new NoSuchElementException("Недвижимость с id " + id + " не найдена");  
 }  
 model.addAttribute("realty", realty);  
 model.addAttribute("users", userService.findAll());  
 model.addAttribute("deals", dealService.findAll());  
 model.addAttribute("clients", clientService.findAll());  
 return "realty/realty\_form";  
 }  
  
 @PostMapping("/delete")  
 public String deleteAllExceptWithDeals(RedirectAttributes redirectAttributes) {  
 List<Realty> deleted = realtyService.deleteAllExceptWithDeals();  
 if (deleted.isEmpty()) {  
 redirectAttributes.addFlashAttribute("warningMessage", "Удаление невозможно: все объекты участвуют в сделках.");  
 } else {  
 int deletedCount = deleted.size();  
 redirectAttributes.addFlashAttribute("warningMessage",  
 "Были удалены все недвижимости, которые не находятся в сделках. Количество удалённых: " + deletedCount);  
 }  
 return "redirect:/realty";  
 }  
  
}

package org.example.terranova.controller;  
  
import jakarta.validation.Valid;  
import lombok.RequiredArgsConstructor;  
import org.example.terranova.dto.UserDTO;  
import org.example.terranova.service.UserService;  
import org.springframework.stereotype.Controller;  
import org.springframework.ui.Model;  
import org.springframework.validation.BindingResult;  
import org.springframework.web.bind.annotation.\*;  
  
@Controller  
@RequiredArgsConstructor  
public class RegistrationController {  
  
 private final UserService userService;  
  
 @GetMapping("/include/registration")  
 public String showRegistrationForm(Model model) {  
 model.addAttribute("userDTO", new UserDTO());  
 return "include/registration";  
 }  
  
 @PostMapping("/include/registration/save")  
 public String registerUser(@Valid @ModelAttribute("userDTO") UserDTO userDTO,  
 BindingResult result,  
 Model model) {  
 if (result.hasErrors()) {  
 return "include/registration";  
 }  
  
 if (!userService.isUsernameAvailable(userDTO.getUsername())) {  
 model.addAttribute("usernameError", "Логин уже занят");  
 return "include/registration";  
 }  
  
 if (!userDTO.getPassword().equals(userDTO.getConfirmPassword())) {  
 model.addAttribute("passwordMismatch", "Пароли не совпадают");  
 return "include/registration";  
 }  
  
 userService.save(userDTO);  
 return "redirect:/include/login";  
 }  
}

package org.example.terranova.controller;  
  
import jakarta.validation.Valid;  
import lombok.RequiredArgsConstructor;  
import org.example.terranova.dto.UserDTO;  
import org.example.terranova.impl.RealtyServiceImpl;  
import org.example.terranova.model.Realty;  
import org.example.terranova.model.Role;  
import org.example.terranova.model.User;  
import org.example.terranova.service.UserService;  
import org.springframework.security.core.annotation.AuthenticationPrincipal;  
import org.springframework.security.core.userdetails.UserDetails;  
import org.springframework.stereotype.Controller;  
import org.springframework.ui.Model;  
import org.springframework.validation.BindingResult;  
import org.springframework.web.bind.annotation.\*;  
  
import java.util.HashSet;  
import java.util.List;  
import java.util.Set;  
  
@Controller  
@RequiredArgsConstructor  
@RequestMapping("/user")  
public class UserController {  
  
 private final UserService userService;  
 private RealtyServiceImpl realtyService;  
  
 @PostMapping("/update")  
 public String updateUser(@ModelAttribute("user") User userForm,  
 @RequestParam(value = "roles", required = false) List<String> roleNames) {  
 Set<Role> roles = new HashSet<>();  
 if (roleNames != null) {  
 for (String roleName : roleNames) {  
 try {  
 roles.add(Role.*valueOf*(roleName));  
 } catch (IllegalArgumentException e) {  
 *// логирование* }  
 }  
 }  
 userService.updateUserRoles(userForm.getId(), roles);  
 return "redirect:/user";  
 }  
   
 @GetMapping  
 public String listUsers(Model model,  
 @RequestParam(name = "sort", required = false) String sort) {  
 List<User> users = switch (sort != null ? sort.toLowerCase() : "") {  
 case "username\_asc" -> userService.findAllOrderByUsernameAsc();  
 case "username\_desc" -> userService.findAllOrderByUsernameDesc();  
 case "name\_asc" -> userService.findAllOrderByNameAsc();  
 case "name\_desc" -> userService.findAllOrderByNameDesc();  
 default -> userService.findAll();  
 };  
 model.addAttribute("users", users);  
 model.addAttribute("sort", sort);  
 return "user/user\_list";  
 }  
  
 @GetMapping("/create")  
 public String showCreateForm(Model model) {  
 model.addAttribute("user", new UserDTO());  
 model.addAttribute("allRoles", Role.*values*());  
 return "user/user\_form";  
 }  
  
 @GetMapping("/edit/{id}")  
 public String editUserForm(@PathVariable Long id, Model model) {  
 User user = userService.findById(id);  
 UserDTO userDTO = UserDTO.*fromEntity*(user);  
 model.addAttribute("user", userDTO);  
 model.addAttribute("allRoles", Role.*values*());  
 return "user/user\_form";  
 }  
  
 @PostMapping("/save")  
 public String saveUser(@Valid @ModelAttribute("user") UserDTO userDTO, BindingResult bindingResult, Model model) {  
 if (bindingResult.hasErrors()) {  
 model.addAttribute("allRoles", Role.*values*());  
 return "user/user\_form";  
 }  
 userService.save(userDTO);  
 return "redirect:/user";  
 }  
  
 @PostMapping("/delete/{id}")  
 public String deleteUser(@PathVariable Long id) {  
 userService.deleteUser(id);  
 return "redirect:/user";  
 }  
  
 @GetMapping("/realty")  
 public String userRealty(Model model, @AuthenticationPrincipal UserDetails userDetails) {  
 User currentUser = (User) userDetails;  
 List<Realty> realties = realtyService.findAllByOwner(currentUser);  
 model.addAttribute("realties", realties);  
 return "profile/realty\_list";  
 }  
  
 @GetMapping("/users")  
 public String getAllUsers(@RequestParam(name = "sort", required = false) String sort, Model model) {  
 List<User> users;  
  
 switch (sort != null ? sort : "") {  
 case "username\_asc":  
 users = userService.findAllOrderByUsernameAsc();  
 break;  
 case "username\_desc":  
 users = userService.findAllOrderByUsernameDesc();  
 break;  
 case "name\_asc":  
 users = userService.findAllOrderByNameAsc();  
 break;  
 case "name\_desc":  
 users = userService.findAllOrderByNameDesc();  
 break;  
 default:  
 users = userService.findAll();  
 }  
  
 model.addAttribute("user", users);  
 model.addAttribute("sort", sort);  
 return "users/user\_list";  
 }  
  
}

package org.example.terranova.dto;  
  
import jakarta.validation.constraints.NotEmpty;  
import jakarta.validation.constraints.Size;  
import lombok.\*;  
  
@NoArgsConstructor  
@AllArgsConstructor  
@Getter  
@Setter  
@Data  
public class ClientDTO {  
 @NotEmpty(message = "Паспорт не может быть пустым")  
 private String passport;  
 @NotEmpty(message = "Имя не может быть пустым")  
 @Size(min = 3, max = 30, message = "От 3 до 20 символов")  
 private String name;  
 @NotEmpty(message = "Фамилия не может быть пустым")  
 @Size(min = 3, max = 30, message = "От 3 до 30 символов")  
 private String surname;  
 *// Отчество - может быть пустым* private String patronymic;  
 @NotEmpty(message = "Номер телефона не может быть пустым")  
 @Size(min = 3, max = 10, message = "От 3 до 10 символов")  
 private String phone;  
 *// ownerType - может быть пустым* private String ownerType;  
 *// company - может быть пустым* private String company;  
 *// license - может быть пустым* private String license;  
  
}

package org.example.terranova.dto;  
  
import jakarta.validation.constraints.NotEmpty;  
import jakarta.validation.constraints.Size;  
import lombok.\*;  
import java.time.LocalDate;  
  
@NoArgsConstructor  
@AllArgsConstructor  
@Getter  
@Setter  
@Data  
public class DealDTO {  
 @NotEmpty(message = "Цена не может быть пустым")  
 private double amount;  
 @NotEmpty(message = "Дата не может быть пустым")  
 private LocalDate date;  
 @NotEmpty(message = "Статус сделки не может быть пустым")  
 private String status;  
 @NotEmpty(message = "Номер телефона не может быть пустым")  
 @Size(min = 3, max = 10, message = "От 3 до 10 символов")  
 private String number;  
  
}

package org.example.terranova.dto;  
  
import jakarta.validation.constraints.NotEmpty;  
import jakarta.validation.constraints.Size;  
import lombok.\*;  
  
@Data  
@NoArgsConstructor  
@AllArgsConstructor  
@Getter  
@Setter  
public class EmployeeDTO {  
 @NotEmpty(message = "Имя не может быть пустым")  
 @Size(min = 3, max = 30, message = "От 3 до 30 символов")  
 private String name;  
 @NotEmpty(message = "Фамилия не может быть пустым")  
 @Size(min = 3, max = 30, message = "От 3 до 30 символов")  
 private String surname;  
 *// Отчество - может быть пустым* private String patronymic;  
 @NotEmpty(message = "Должность не может быть пустым")  
 private String position;  
 @NotEmpty(message = "email не может быть пустым")  
 private String email;  
 @NotEmpty(message = "Номер телефона не может быть пустым")  
 @Size(min = 3, max = 10, message = "От 3 до 10 символов")  
 private String phone;  
  
}

package org.example.terranova.dto;  
  
import jakarta.validation.constraints.NotEmpty;  
import jakarta.validation.constraints.Size;  
import lombok.\*;  
  
@Data  
@NoArgsConstructor  
@AllArgsConstructor  
@Getter  
@Setter  
public class RealtyDTO {  
 @NotEmpty(message = "Регион не может быть пустым")  
 @Size(min = 3, max = 50, message = "От 3 до 20 символов")  
 private String region;  
 @NotEmpty(message = "Местонахождение не может быть пустым")  
 @Size(min = 3, max = 50, message = "От 3 до 20 символов")  
 private String locality;  
 @NotEmpty(message = "Улица не может быть пустым")  
 @Size(min = 3, max = 50, message = "От 3 до 20 символов")  
 private String street;  
 @NotEmpty(message = "Количество комнат не может быть пустым")  
 private int rooms;  
 @NotEmpty(message = "floors комнат не может быть пустым")  
 private int floors;  
 @NotEmpty(message = "square комнат не может быть пустым")  
 private double square;  
 @NotEmpty(message = "repair комнат не может быть пустым")  
 private String repair;  
 @NotEmpty(message = "cost комнат не может быть пустым")  
 private double cost;  
 @NotEmpty(message = "elevator комнат не может быть пустым")  
 private boolean elevator;  
 @NotEmpty(message = "playgraund комнат не может быть пустым")  
 private boolean playground;  
 @NotEmpty(message = "trashChute комнат не может быть пустым")  
 private boolean trashChute;  
 @NotEmpty(message = "parking комнат не может быть пустым")  
 private boolean parking;  
 @NotEmpty(message = "balcony комнат не может быть пустым")  
 private boolean balcony;  
  
}

package org.example.terranova.dto;  
  
import jakarta.validation.constraints.NotBlank;  
import jakarta.validation.constraints.Size;  
import lombok.AllArgsConstructor;  
import lombok.Getter;  
import lombok.NoArgsConstructor;  
import lombok.Setter;  
import org.example.terranova.model.Role;  
import org.example.terranova.model.User;  
  
import java.util.HashSet;  
import java.util.Set;  
  
@NoArgsConstructor  
@AllArgsConstructor  
@Getter  
@Setter  
public class UserDTO {  
  
 private Long id;  
  
 @NotBlank(message = "Логин не может быть пустым")  
 @Size(min = 3, max = 20, message = "От 3 до 20 символов")  
 private String username;  
  
 @NotBlank(message = "Имя не может быть пустым")  
 private String name;  
  
 @NotBlank(message = "Фамилия не может быть пустой")  
 private String surname;  
  
 @NotBlank(message = "Пароль не может быть пустым")  
 @Size(min = 8, max = 20, message = "От 8 до 20 символов")  
 private String password;  
  
 @NotBlank(message = "Пароль не может быть пустым")  
 @Size(min = 8, max = 20, message = "От 8 до 20 символов")  
 private String confirmPassword;  
  
 private Set<Role> roles = new HashSet<>();  
  
 *// Статический метод преобразования из сущности User в DTO* public static UserDTO fromEntity(User user) {  
 UserDTO dto = new UserDTO();  
 dto.setId(user.getId());  
 dto.setUsername(user.getUsername());  
 dto.setName(user.getName());  
 dto.setSurname(user.getSurname());  
 *// Пароль и confirmPassword, как правило, не копируются из сущности (безопасность)* dto.setRoles(user.getRoles() != null ? new HashSet<>(user.getRoles()) : new HashSet<>());  
 return dto;  
 }  
}

package org.example.terranova.impl;  
  
import lombok.\*;  
import org.example.terranova.model.Client;  
import org.example.terranova.repo.ClientRepo;  
import org.example.terranova.repo.DealRepo;  
import org.example.terranova.service.ClientService;  
import org.springframework.data.domain.Sort;  
import org.springframework.stereotype.Service;  
import java.util.List;  
import java.util.Optional;  
  
@Service  
@RequiredArgsConstructor  
public class ClientServiceImpl implements ClientService {  
  
 private final ClientRepo clientRepository;  
 private final DealRepo dealRepository;  
 private static final List<String> ALLOWED\_FIELDS = List.of("name", "surname", "passport", "phone", "ownertype", "company");  
  
 @Override  
 public List<Client> findAll() {  
 return clientRepository.findAll();  
 }  
  
 @Override  
 public List<Client> findAllSortedByField(String fieldName, String sortDirection) {  
 if (fieldName == null || !ALLOWED\_FIELDS.contains(fieldName.toLowerCase())) {  
 throw new IllegalArgumentException("Недопустимое поле для сортировки: " + fieldName);  
 }  
  
 Sort.Direction direction;  
 try {  
 direction = Sort.Direction.fromString(sortDirection);  
 } catch (IllegalArgumentException e) {  
 throw new IllegalArgumentException("Недопустимое направление сортировки: " + sortDirection, e);  
 }  
  
 String actualFieldName = switch (fieldName.toLowerCase()) {  
 case "name" -> "name";  
 case "surname" -> "surname";  
 case "passport" -> "passport";  
 case "phone" -> "phone";  
 case "ownertype" -> "ownerType";  
 case "company" -> "company";  
 default -> fieldName;  
 };  
  
 return clientRepository.findAll(Sort.by(direction, actualFieldName));  
 }  
  
 @Override  
 public Optional<Client> findById(Long id) {  
 return clientRepository.findById(id);  
 }  
  
 @Override  
 public void save(Client client) {  
 clientRepository.save(client);  
 }  
  
 @Override  
 public void deleteById(Long id) {  
 clientRepository.deleteById(id);  
 }  
  
 @Override  
 public void deleteAll() {  
 clientRepository.deleteAll();  
 }  
  
 @Override  
 public boolean deleteByIdIfPossible(Long id) {  
 Optional<Client> optionalClient = clientRepository.findById(id);  
 if (optionalClient.isPresent()) {  
 Client client = optionalClient.get();  
 if (!dealRepository.existsByClient(client)) {  
 clientRepository.delete(client);  
 return true;  
 }  
 }  
 return false;  
 }  
  
 @Override  
 public int deleteAllExceptWithDeals() {  
 List<Client> allClients = clientRepository.findAll();  
 int notDeletedCount = 0;  
 for (Client client : allClients) {  
 if (!dealRepository.existsByClient(client)) {  
 clientRepository.delete(client);  
 } else {  
 notDeletedCount++;  
 }  
 }  
 return notDeletedCount;  
 }  
  
}

package org.example.terranova.impl;  
  
import lombok.RequiredArgsConstructor;  
import org.example.terranova.model.\*;  
import org.example.terranova.repo.ClientRepo;  
import org.example.terranova.repo.DealRepo;  
import org.example.terranova.repo.EmployeeRepo;  
import org.example.terranova.repo.RealtyRepo;  
import org.example.terranova.service.DealService;  
import org.springframework.data.domain.Sort;  
import org.springframework.stereotype.Service;  
import java.util.List;  
import java.util.Optional;  
  
@Service  
@RequiredArgsConstructor  
public class DealServiceImpl implements DealService {  
  
 private final DealRepo dealRepository;  
 private final RealtyRepo realtyRepository;  
 private final ClientRepo clientRepository;  
 private final EmployeeRepo employeeRepository;  
  
 public List<Deal> getAllDeals(Sort sort) {  
 return dealRepository.findAll(sort);  
 }  
  
 @Override  
 public void saveDeal(Deal deal) {  
 if (deal.getRealty() != null && deal.getRealty().getId() != null) {  
 Realty realty = realtyRepository.findById(deal.getRealty().getId()).orElse(null);  
 deal.setRealty(realty);  
 }  
  
 if (deal.getClient() != null && deal.getClient().getId() != null) {  
 Client client = clientRepository.findById(deal.getClient().getId()).orElse(null);  
 deal.setClient(client);  
 }  
  
 if (deal.getEmployee() != null && deal.getEmployee().getId() != null) {  
 Employee employee = employeeRepository.findById(deal.getEmployee().getId()).orElse(null);  
 deal.setEmployee(employee);  
 }  
  
 dealRepository.save(deal);  
 }  
  
 @Override  
 public void deleteDeal(Long id) {  
 if (dealRepository.existsById(id)) {  
 dealRepository.deleteById(id);  
 }  
 }  
  
 @Override  
 public List<Deal> findAll() {  
 return dealRepository.findAll();  
 }  
  
 @Override  
 public List<Deal> findAllByUser(User user) {  
 return dealRepository.findAllByUser(user);  
 }  
  
 @Override  
 public Optional<Deal> findById(Long id) {  
 return dealRepository.findById(id);  
 }  
  
 @Override  
 public void deleteAllDeals() {  
 dealRepository.deleteAll();  
 }  
  
 @Override  
 public boolean existsByRealtyId(Long realtyId) {  
 return dealRepository.existsByRealtyId(realtyId);  
 }  
  
 @Override  
 public void deleteDealById(Long id) {  
 dealRepository.deleteById(id);  
 }  
  
 @Override  
 public void deleteDealsByIds(List<Long> ids) {  
 List<Deal> dealsToDelete = dealRepository.findAllById(ids);  
 dealRepository.deleteAll(dealsToDelete);  
 }  
  
}

package org.example.terranova.impl;  
  
import lombok.RequiredArgsConstructor;  
import org.example.terranova.model.Employee;  
import org.example.terranova.model.User;  
import org.example.terranova.repo.DealRepo;  
import org.example.terranova.repo.EmployeeRepo;  
import org.example.terranova.service.EmployeeService;  
import org.springframework.data.domain.Sort;  
import org.springframework.stereotype.Service;  
  
import java.util.ArrayList;  
import java.util.Arrays;  
import java.util.List;  
  
@Service  
@RequiredArgsConstructor  
public class EmployeeServiceImpl implements EmployeeService {  
  
 private final EmployeeRepo employeeRepo;  
 private final DealRepo dealRepo;  
  
 @Override  
 public List<Employee> getAllEmployeesSortedBy(String sortBy) {  
 List<String> allowedFields = Arrays.*asList*("surname", "name", "position", "email");  
  
 if (!allowedFields.contains(sortBy)) {  
 sortBy = "surname";  
 }  
  
 return employeeRepo.findAll(Sort.*by*(Sort.Direction.*ASC*, sortBy));  
 }  
  
 @Override  
 public Employee getEmployeeById(Long id) {  
 return employeeRepo.findById(id).orElse(null);  
 }  
  
 @Override  
 public void saveEmployee(Employee employee) {  
 employeeRepo.save(employee);  
 }  
  
 @Override  
 public void deleteAllEmployees() {  
 employeeRepo.deleteAll();  
 }  
  
 @Override  
 public List<Employee> getAllEmployeesSorted(Sort sort) {  
 return employeeRepo.findAll(sort);  
 }  
  
 @Override  
 public List<Employee> getAllEmployees() {  
 return employeeRepo.findAll();  
 }  
  
 @Override  
 public Employee findByUser(User user) {  
 return employeeRepo.findByUser(user)  
 .orElseThrow(() -> new IllegalArgumentException("Сотрудник не найден"));  
 }  
  
 @Override  
 public List<Employee> findAll() {  
 return employeeRepo.findAll();  
 }  
  
 @Override  
 public Employee getEmployeeByUserId(Long userId) {  
 return employeeRepo.findByUserId(userId);  
 }  
  
 @Override  
 public List<Employee> getAllEmployeesWithUsers() {  
 return employeeRepo.findAllWithUser();  
 }  
  
 @Override  
 public void deleteEmployee(Long id) {  
 Employee employee = employeeRepo.findById(id).orElse(null);  
 if (employee == null) return;  
  
 if (dealRepo.existsByEmployee(employee)) {  
 throw new IllegalStateException("Employee is involved in a deal.");  
 }  
  
 employeeRepo.delete(employee);  
 }  
  
 @Override  
 public List<Employee> deleteAllExceptWithDeals() {  
 List<Employee> allEmployees = employeeRepo.findAll();  
 List<Employee> undeleted = new ArrayList<>();  
  
 for (Employee employee : allEmployees) {  
 if (dealRepo.existsByEmployee(employee)) {  
 undeleted.add(employee);  
 } else {  
 employeeRepo.delete(employee);  
 }  
 }  
  
 return undeleted;  
 }  
}

package org.example.terranova.impl;  
  
import lombok.RequiredArgsConstructor;  
import org.example.terranova.model.Employee;  
import org.example.terranova.model.Realty;  
import org.example.terranova.model.User;  
import org.example.terranova.repo.DealRepo;  
import org.example.terranova.repo.EmployeeRepo;  
import org.example.terranova.repo.RealtyRepo;  
import org.example.terranova.service.RealtyService;  
import org.springframework.stereotype.Service;  
  
import java.util.List;  
import java.util.stream.Collectors;  
  
@Service  
@RequiredArgsConstructor  
public class RealtyServiceImpl implements RealtyService {  
  
 private final RealtyRepo realtyRepo;  
 private final EmployeeRepo employeeRepo;  
 private final DealRepo dealRepo;  
  
 @Override  
 public Realty getRealtyById(Long id) {  
 return realtyRepo.findById(id).orElse(null);  
 }  
  
 @Override  
 public Realty saveRealty(Realty realty) {  
 realtyRepo.save(realty);  
 return realty;  
 }  
  
 @Override  
 public void deleteRealty(Long id) {  
 realtyRepo.deleteById(id);  
 }  
  
 @Override  
 public List<Realty> findAll() {  
 return realtyRepo.findAll();  
 }  
  
 @Override  
 public void deleteAll() {  
 realtyRepo.deleteAll();  
 }  
  
 @Override  
 public Employee findByUser(User user) {  
 return employeeRepo.findByUser(user)  
 .orElseThrow(() -> new IllegalArgumentException("Сотрудник не найден"));  
 }  
  
 @Override  
 public List<Realty> findAllByOwner(User owner) {  
 return realtyRepo.findAllByOwner(owner);  
 }  
  
 @Override  
 public void deleteById(Long id) {  
 realtyRepo.deleteById(id);  
 }  
  
 @Override  
 public List<Realty> findAllByClientUser(User user) {  
 return realtyRepo.findAllByClientUser(user);  
 }  
  
 @Override  
 public int deleteAllExceptInDeals() {  
 List<Long> realtyIdsInDeals = dealRepo.findAll()  
 .stream()  
 .map(deal -> deal.getRealty().getId())  
 .distinct()  
 .toList();  
  
 if (realtyIdsInDeals.isEmpty()) {  
 int count = (int) realtyRepo.count();  
 realtyRepo.deleteAll();  
 return count;  
 }  
  
 return realtyRepo.deleteByIdNotIn(realtyIdsInDeals);  
 }  
  
 @Override  
 public int deleteUnlinkedRealties() {  
 List<Realty> all = realtyRepo.findAll();  
 int deleted = 0;  
 for (Realty realty : all) {  
 if (dealRepo.findByRealty(realty).isEmpty()) {  
 realtyRepo.delete(realty);  
 deleted++;  
 }  
 }  
 return deleted;  
 }  
  
 @Override  
 public int countLinkedRealties() {  
 return (int) realtyRepo.findAll().stream()  
 .filter(realty -> !dealRepo.findByRealty(realty).isEmpty())  
 .count();  
 }  
  
 @Override  
 public List<Realty> deleteAllExceptWithDeals() {  
 List<Realty> all = realtyRepo.findAll();  
 List<Realty> toDelete = all.stream()  
 .filter(realty -> dealRepo.findByRealty(realty).isEmpty())  
 .collect(Collectors.toList());  
 realtyRepo.deleteAll(toDelete);  
 return toDelete;  
 }  
  
}

package org.example.terranova.impl;  
  
import lombok.RequiredArgsConstructor;  
import org.example.terranova.model.User;  
import org.example.terranova.service.UserService;  
import org.springframework.security.core.authority.SimpleGrantedAuthority;  
import org.springframework.security.core.userdetails.UserDetails;  
import org.springframework.security.core.userdetails.UserDetailsService;  
import org.springframework.security.core.userdetails.UsernameNotFoundException;  
import org.springframework.stereotype.Service;  
  
import java.util.List;  
  
@Service  
@RequiredArgsConstructor  
public class UserDetailsServiceImpl implements UserDetailsService {  
  
 private final UserService userService;  
  
 @Override  
 public UserDetails loadUserByUsername(String username) throws UsernameNotFoundException {  
 User user = userService.findByUsername(username)  
 .orElseThrow(() -> new UsernameNotFoundException("Пользователь не найден: " + username));  
  
 if (user.getRoles() == null || user.getRoles().isEmpty()) {  
 throw new UsernameNotFoundException("У пользователя нет назначенных ролей: " + username);  
 }  
  
 List<SimpleGrantedAuthority> authorities = user.getRoles().stream()  
 .map(role -> new SimpleGrantedAuthority("ROLE\_" + role.name()))  
 .toList();  
  
 return new org.springframework.security.core.userdetails.User(  
 user.getUsername(),  
 user.getPassword(),  
 authorities  
 );  
 }  
}

package org.example.terranova.impl;  
  
import lombok.\*;  
import org.example.terranova.dto.UserDTO;  
import org.example.terranova.model.Role;  
import org.example.terranova.model.User;  
import org.example.terranova.repo.UserRepo;  
import org.example.terranova.service.UserService;  
import org.springframework.data.domain.Sort;  
import org.springframework.security.core.userdetails.UsernameNotFoundException;  
import org.springframework.security.crypto.bcrypt.BCryptPasswordEncoder;  
import org.springframework.stereotype.Service;  
import org.springframework.web.multipart.MultipartFile;  
  
import java.io.IOException;  
import java.nio.file.Files;  
import java.nio.file.Path;  
import java.nio.file.Paths;  
import java.nio.file.StandardCopyOption;  
import java.util.\*;  
  
@Service  
@RequiredArgsConstructor  
public class UserServiceImpl implements UserService {  
  
 private final UserRepo userRepo;  
 private final BCryptPasswordEncoder encoder;  
  
 @Override  
 public User save(UserDTO userDTO) {  
 if (!isUsernameAvailable(userDTO.getUsername())) {  
 throw new IllegalArgumentException("Пользователь с таким именем уже существует");  
 }  
  
 User user = new User();  
 user.setUsername(userDTO.getUsername());  
 user.setPassword(encoder.encode(userDTO.getPassword()));  
 user.setName(userDTO.getName());  
 user.setSurname(userDTO.getSurname());  
 user.setRoles(Collections.singleton(Role.USER));  
 return userRepo.save(user);  
 }  
  
 @Override  
 public boolean isUsernameAvailable(String username) {  
 return !userRepo.existsByUsername(username);  
 }  
  
 @Override  
 public User findById(Long id) {  
 return userRepo.findById(id).orElseThrow(() -> new RuntimeException("Пользователь не найден"));  
 }  
  
 @Override  
 public void deleteUser(Long id) {  
 userRepo.deleteById(id);  
 }  
  
 @Override  
 public List<User> findAll() {  
 return userRepo.findAll();  
 }  
  
 @Override  
 public List<User> findAllOrderByUsernameAsc() {  
 return userRepo.findAll(Sort.by(Sort.Direction.ASC, "username"));  
 }  
  
 @Override  
 public List<User> findAllOrderByUsernameDesc() {  
 return userRepo.findAll(Sort.by(Sort.Direction.DESC, "username"));  
 }  
  
 @Override  
 public List<User> findAllOrderByNameAsc() {  
 return userRepo.findAll(Sort.by(Sort.Direction.ASC, "name"));  
 }  
  
 @Override  
 public List<User> findAllOrderByNameDesc() {  
 return userRepo.findAll(Sort.by(Sort.Direction.DESC, "name"));  
 }  
  
 @Override  
 public void updateUserRoles(Long userId, Set<Role> roles) {  
 User user = userRepo.findById(userId)  
 .orElseThrow(() -> new RuntimeException("Пользователь не найден"));  
 user.setRoles(roles);  
 userRepo.save(user);  
 }  
  
 @Override  
 public Optional<User> findByUsername(String username) {  
 return userRepo.findByUsername(username);  
 }  
  
 public String saveAvatar(User user, MultipartFile file) {  
 try {  
 String contentType = file.getContentType();  
 if (contentType == null || !contentType.startsWith("image/")) {  
 throw new IllegalArgumentException("Можно загружать только изображения.");  
 }  
  
 String originalFilename = Paths.get(file.getOriginalFilename()).getFileName().toString();  
 String filename = UUID.randomUUID() + "\_" + originalFilename;  
 Path uploadPath = Paths.get("src/main/resources/static/uploads");  
  
 if (!Files.exists(uploadPath)) {  
 Files.createDirectories(uploadPath);  
 }  
  
 Files.copy(file.getInputStream(), uploadPath.resolve(filename), StandardCopyOption.REPLACE\_EXISTING);  
 return filename;  
 } catch (IOException e) {  
 throw new RuntimeException("Ошибка загрузки файла", e);  
 }  
 }  
  
 @Override  
 public User save(User user) {  
 return userRepo.save(user);  
 }  
  
 @Override  
 public String saveAvatarFile(MultipartFile file) {  
 try {  
 String contentType = file.getContentType();  
 if (contentType == null || !contentType.startsWith("image/")) {  
 throw new IllegalArgumentException("Можно загружать только изображения.");  
 }  
  
 String originalFilename = Paths.get(file.getOriginalFilename()).getFileName().toString();  
 String filename = UUID.randomUUID() + "\_" + originalFilename;  
 Path uploadPath = Paths.get("src/main/resources/static/uploads");  
  
 if (!Files.exists(uploadPath)) {  
 Files.createDirectories(uploadPath);  
 }  
  
 Files.copy(file.getInputStream(), uploadPath.resolve(filename), StandardCopyOption.REPLACE\_EXISTING);  
 return filename;  
 } catch (IOException e) {  
 throw new RuntimeException("Ошибка загрузки файла", e);  
 }  
 }  
  
}

package org.example.terranova.model;  
  
import jakarta.persistence.\*;  
import lombok.\*;  
  
@Getter  
@Setter  
@NoArgsConstructor  
@AllArgsConstructor  
@Entity  
@Table(name = "client")  
public class Client {  
  
 @Id  
 @GeneratedValue(strategy = GenerationType.IDENTITY)  
 private Long id;  
  
 private String passport;  
 private String name;  
 private String surname;  
 private String patronymic;  
 private String phone;  
 private String ownerType;  
 private String company;  
 private String license;  
  
 *// Связь OneToOne с User — владелец клиента* @OneToOne(fetch = FetchType.LAZY, optional = false)  
 @JoinColumn(name = "user\_id", unique = true, nullable = false)  
 private User user;  
  
 *// Вспомогательный метод для отображения полного имени клиента* public String getFullName() {  
 return surname + " " + name + (patronymic != null ? " " + patronymic : "");  
 }  
}

package org.example.terranova.model;  
  
import jakarta.persistence.\*;  
import lombok.\*;  
import org.springframework.format.annotation.DateTimeFormat;  
  
import java.time.LocalDateTime;  
import java.time.format.DateTimeFormatter;  
import java.util.Locale;  
  
@Getter  
@Setter  
@NoArgsConstructor  
@AllArgsConstructor  
@Entity  
@Table(name = "deal")  
public class Deal {  
  
 @Id  
 @GeneratedValue(strategy = GenerationType.IDENTITY)  
 private Long id;  
  
 private Long amount;  
  
 @DateTimeFormat(iso = DateTimeFormat.ISO.DATE\_TIME)  
 private LocalDateTime date;  
  
 private String status;  
  
 private String number;  
  
 @ManyToOne(fetch = FetchType.LAZY)  
 @JoinColumn(name = "realty\_id")  
 private Realty realty;  
  
 @ManyToOne(fetch = FetchType.LAZY)  
 @JoinColumn(name = "client\_id")  
 private Client client;  
  
 @ManyToOne(fetch = FetchType.LAZY)  
 @JoinColumn(name = "employee\_id")  
 private Employee employee;  
  
 @Transient  
 public String getDateForInput() {  
 if (date == null) return "";  
 return date.format(DateTimeFormatter.ofPattern("yyyy-MM-dd'T'HH:mm"));  
 }  
  
 @Transient  
 public String getFormattedDate() {  
 if (date != null) {  
 DateTimeFormatter formatter = DateTimeFormatter.ofPattern("dd.MM.yyyy", new Locale("ru"));  
 return date.format(formatter);  
 }  
 return "—";  
 }  
  
 @Transient  
 public String getFormattedCost() {  
 if (amount == null) return "—";  
  
 long value = amount;  
 if (value < 100\_000) {  
 return value + " ₽";  
 } else if (value < 1\_000\_000) {  
 return (value / 1\_000) + " тыс. ₽";  
 } else if (value < 1\_000\_000\_000) {  
 return String.format("%.1f млн. ₽", value / 1\_000\_000.0);  
 } else {  
 return String.format("%.1f млрд ₽", value / 1\_000\_000\_000.0);  
 }  
 }  
}

package org.example.terranova.model;  
  
import jakarta.persistence.\*;  
import lombok.AllArgsConstructor;  
import lombok.Getter;  
import lombok.NoArgsConstructor;  
import lombok.Setter;  
  
import java.util.ArrayList;  
import java.util.List;  
  
@Getter  
@Setter  
@AllArgsConstructor  
@NoArgsConstructor  
@Entity  
@Table(name = "employee")  
public class Employee {  
  
 @Id  
 @GeneratedValue(strategy = GenerationType.IDENTITY)  
 private Long id;  
  
 @Column(nullable = false)  
 private String name;  
  
 @Column(nullable = false)  
 private String surname;  
  
 private String patronymic;  
  
 private String position;  
  
 private String email;  
  
 private String phone;  
  
 @OneToMany(mappedBy = "employee", fetch = FetchType.LAZY)  
 private List<Deal> deals = new ArrayList<>();  
  
 @OneToOne(fetch = FetchType.LAZY)  
 @JoinColumn(name = "user\_id")  
 private User user;  
  
 public String getFirstName() {  
 return name;  
 }  
  
 public String getLastName() {  
 return surname;  
 }  
  
 public String getFullName() {  
 StringBuilder fullName = new StringBuilder();  
 if (surname != null) fullName.append(surname).append(" ");  
 if (name != null) fullName.append(name).append(" ");  
 if (patronymic != null) fullName.append(patronymic);  
 return fullName.toString().trim();  
 }  
}

package org.example.terranova.model;  
  
import jakarta.persistence.\*;  
import lombok.\*;  
  
import java.time.LocalDateTime;  
  
@Getter  
@Setter  
@NoArgsConstructor  
@AllArgsConstructor  
@Entity  
@Table(name = "realty")  
public class Realty {  
  
 @Id  
 @GeneratedValue(strategy = GenerationType.IDENTITY)  
 private Long id;  
  
 private String region;  
 private String locality;  
 private String street;  
 private int rooms;  
 private int floors;  
 private double square;  
 private String repair;  
 private double cost;  
  
 private boolean elevator;  
 private boolean playground;  
 private boolean trashChute;  
 private boolean parking;  
 private boolean balcony;  
  
 @Column(name = "created\_at", updatable = false)  
 private LocalDateTime createdAt;  
  
 @PrePersist  
 public void prePersist() {  
 this.createdAt = LocalDateTime.now();  
 }  
  
 @ManyToOne(fetch = FetchType.LAZY)  
 @JoinColumn(name = "deal\_id")  
 private Deal deal;  
  
 @ManyToOne(fetch = FetchType.LAZY)  
 @JoinColumn(name = "client\_id")  
 private Client client;  
  
 @ManyToOne(fetch = FetchType.LAZY)  
 @JoinColumn(name = "owner\_id")  
 private User owner;  
  
 @Transient  
 public String getFormattedCost() {  
 if (cost < 100\_000) {  
 return cost + " ₽";  
 } else if (cost < 1\_000\_000) {  
 return ((int) (cost / 1\_000)) + " тыс. ₽";  
 } else if (cost < 1\_000\_000\_000) {  
 return String.format("%.1f млн. ₽", cost / 1\_000\_000.0);  
 } else {  
 return String.format("%.1f млрд ₽", cost / 1\_000\_000\_000.0);  
 }  
 }  
}

package org.example.terranova.model;  
  
public enum Role {  
 USER("Пользователь"),  
 EMPLOYEE("Сотрудник"),  
 ADMIN("Администратор");  
  
 private final String displayName;  
  
 Role(String displayName) {  
 this.displayName = displayName;  
 }  
  
 public String getDisplayName() {  
 return displayName;  
 }  
}

package org.example.terranova.model;  
  
import jakarta.persistence.\*;  
import lombok.\*;  
  
import java.util.Set;  
  
@Getter  
@Setter  
@NoArgsConstructor  
@AllArgsConstructor  
@Entity  
@Table(name = "users")  
public class User {  
  
 @Id  
 @GeneratedValue(strategy = GenerationType.IDENTITY)  
 private Long id;  
  
 private String username;  
 private String name;  
 private String surname;  
 private String password;  
  
 @Transient  
 private String confirmPassword;  
  
 *// Поле для хранения имени файла аватара (URL или имя файла)* private String avatarUrl;  
  
 @ElementCollection(targetClass = Role.class, fetch = FetchType.EAGER)  
 @CollectionTable(name = "users\_roles", joinColumns = @JoinColumn(name = "user\_id"))  
 @Enumerated(EnumType.STRING)  
 private Set<Role> roles;  
  
 *// Связь OneToOne с Client (обратная сторона)* @OneToOne(mappedBy = "user", cascade = CascadeType.ALL, fetch = FetchType.LAZY, optional = true)  
 private Client client;  
  
 public String getFullName() {  
 return name + " " + surname;  
 }  
}

package org.example.terranova.repo;  
  
import org.example.terranova.model.Client;  
import org.springframework.data.jpa.repository.JpaRepository;  
import org.springframework.stereotype.Repository;  
  
import java.util.List;  
  
@Repository  
public interface ClientRepo extends JpaRepository<Client, Long> {  
 List<Client> findAllByOrderBySurnameAsc();  
 List<Client> findAllByOrderBySurnameDesc();  
 List<Client> findAllByOrderByNameAsc();  
 List<Client> findAllByOrderByNameDesc();  
  
}

package org.example.terranova.repo;  
  
import org.example.terranova.model.\*;  
import org.springframework.data.domain.Sort;  
import org.springframework.data.jpa.repository.JpaRepository;  
import org.springframework.data.jpa.repository.Query;  
import org.springframework.data.repository.query.Param;  
  
import java.util.Collection;  
import java.util.List;  
  
public interface DealRepo extends JpaRepository<Deal, Long> {  
  
 List<Deal> findAll(Sort sort);  
 @Query("SELECT d FROM Deal d WHERE d.client.user = :user")  
 List<Deal> findAllByUser(@Param("user") User user);  
 boolean existsByClient(Client client);  
 boolean existsByEmployee(Employee employee);  
 boolean existsByRealtyId(Long realtyId);  
 List<Deal> findByRealty(Realty realty);  
  
}

package org.example.terranova.repo;  
  
import org.example.terranova.model.Employee;  
import org.example.terranova.model.User;  
import org.springframework.data.domain.Sort;  
import org.springframework.data.jpa.repository.JpaRepository;  
import org.springframework.data.jpa.repository.Query;  
import org.springframework.stereotype.Repository;  
  
import java.util.List;  
import java.util.Optional;  
  
@Repository  
public interface EmployeeRepo extends JpaRepository<Employee, Long> {  
 Optional<Employee> findByUser(User user);  
 List<Employee> findAll(Sort sort);  
 Employee findByUserId(Long userId);  
 @Query("SELECT e FROM Employee e LEFT JOIN FETCH e.user")  
 List<Employee> findAllWithUser();  
}

package org.example.terranova.repo;  
  
import jakarta.transaction.Transactional;  
import org.example.terranova.model.Client;  
import org.example.terranova.model.Employee;  
import org.example.terranova.model.Realty;  
import org.example.terranova.model.User;  
import org.springframework.data.jpa.repository.JpaRepository;  
import org.springframework.data.jpa.repository.Modifying;  
import org.springframework.data.jpa.repository.Query;  
import org.springframework.data.repository.query.Param;  
import org.springframework.stereotype.Repository;  
  
import java.util.List;  
import java.util.Optional;  
  
@Repository  
public interface RealtyRepo extends JpaRepository<Realty, Long> {  
 List<Realty> findByClient(Client client);  
 List<Realty> findAllByOwner(User owner);  
 @Query("SELECT r FROM Realty r WHERE r.client.user = :user")  
 List<Realty> findAllByClientUser(@Param("user") User user);  
 *// Удалить все объекты, ID которых не входят в переданный список* @Modifying  
 @Transactional  
 @Query("DELETE FROM Realty r WHERE r.id NOT IN :ids")  
 int deleteByIdNotIn(@Param("ids") List<Long> ids);  
}

package org.example.terranova.repo;  
  
import org.example.terranova.model.User;  
import org.springframework.data.jpa.repository.JpaRepository;  
  
import java.util.Optional;  
  
public interface UserRepo extends JpaRepository<User, Long> {  
 boolean existsByUsername(String username);  
 Optional<User> findByUsername(String username);  
  
  
}

package org.example.terranova.service;  
  
import org.example.terranova.model.Client;  
import java.util.List;  
import java.util.Optional;  
  
public interface ClientService {  
 List<Client> findAll();  
 List<Client> findAllSortedByField(String fieldName, String sortDirection);  
 Optional<Client> findById(Long id);  
 void save(Client client);  
 void deleteById(Long id);  
 void deleteAll();  
 boolean deleteByIdIfPossible(Long id);  
 int deleteAllExceptWithDeals();  
  
}

package org.example.terranova.service;  
  
import org.example.terranova.model.\*;  
import org.springframework.data.domain.Sort;  
  
import java.util.List;  
import java.util.Optional;  
  
public interface DealService {  
 List<Deal> getAllDeals(Sort sort);  
 void saveDeal(Deal deal);  
 void deleteDeal(Long id);  
 List<Deal> findAll();  
 List<Deal> findAllByUser(User user);  
 Optional<Deal> findById(Long id);  
 void deleteAllDeals();  
 boolean existsByRealtyId(Long id);  
 void deleteDealById(Long id);  
 void deleteDealsByIds(List<Long> ids);  
  
}

package org.example.terranova.service;  
  
import org.example.terranova.model.Employee;  
import org.example.terranova.model.User;  
import org.springframework.data.domain.Sort;  
  
import java.util.List;  
  
public interface EmployeeService {  
 List<Employee> getAllEmployeesSortedBy(String sortBy);  
 Employee getEmployeeById(Long id);  
 void saveEmployee(Employee employee);  
 void deleteEmployee(Long id);  
 void deleteAllEmployees();  
 List<Employee> getAllEmployees();  
 List<Employee> getAllEmployeesSorted(Sort sort);  
 Employee findByUser(User user);  
 List<Employee> findAll();  
 Employee getEmployeeByUserId(Long userId);  
 List<Employee> getAllEmployeesWithUsers();  
 List<Employee> deleteAllExceptWithDeals();  
  
}

package org.example.terranova.service;  
  
import org.example.terranova.model.Employee;  
import org.example.terranova.model.Realty;  
import org.example.terranova.model.User;  
  
import java.util.List;  
  
public interface RealtyService {  
 Realty getRealtyById(Long id);  
 Realty saveRealty(Realty realty);  
 void deleteRealty(Long id);  
 List<Realty> findAll();  
 void deleteAll();  
 Employee findByUser(User user);  
 List<Realty> findAllByOwner(User owner);  
 List<Realty> findAllByClientUser(User user);  
 void deleteById(Long id);  
 int deleteAllExceptInDeals();  
  
 int deleteUnlinkedRealties();  
  
 int countLinkedRealties();  
  
 List<Realty> deleteAllExceptWithDeals();  
}

package org.example.terranova.service;  
  
import org.example.terranova.dto.UserDTO;  
import org.example.terranova.model.Role;  
import org.example.terranova.model.User;  
import org.springframework.web.multipart.MultipartFile;  
  
import java.util.List;  
import java.util.Optional;  
import java.util.Set;  
  
public interface UserService {  
  
 User save(UserDTO userDTO);  
 boolean isUsernameAvailable(String username);  
 User findById(Long id);  
 void deleteUser(Long id);  
 List<User> findAll();  
 List<User> findAllOrderByUsernameAsc();  
 List<User> findAllOrderByUsernameDesc();  
 List<User> findAllOrderByNameAsc();  
 List<User> findAllOrderByNameDesc();  
 void updateUserRoles(Long userId, Set<Role> roles);  
 Optional<User> findByUsername(String username);  
 User save(User user);  
 String saveAvatarFile(MultipartFile file);  
  
}

package org.example.terranova;  
  
import org.springframework.boot.SpringApplication;  
import org.springframework.boot.autoconfigure.SpringBootApplication;  
  
@SpringBootApplication  
public class TerraNovaApplication {  
  
 public static void main(String[] args) {  
 SpringApplication.*run*(TerraNovaApplication.class, args);  
 }  
  
}

<!DOCTYPE html>  
<html xmlns:th="http://www.thymeleaf.org" lang="ru">  
<head>  
 <meta charset="UTF-8" />  
 <meta name="viewport" content="width=device-width, initial-scale=1" />  
 <title>Агентство недвижимости «TerraNova» — Форма клиента</title>  
 <link href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.3/dist/css/bootstrap.min.css" rel="stylesheet"  
 integrity="sha384-QWTKZyjpPEjISv5WaRU9OFeRpok6YctnYmDr5pNlyT2bRjXh0JMhjY6hW+ALEwIH" crossorigin="anonymous" />  
</head>  
<body class="d-flex flex-column min-vh-100">  
  
<div th:replace="~{/include/header}"></div>  
  
<main class="flex-grow-1">  
 <div class="container mt-5">  
 <h1 th:text="${#strings.isEmpty(client.id)} ? 'Добавить клиента' : 'Редактировать клиента'"></h1>  
  
 <form th:action="@{/client/save}" th:object="${client}" method="post" novalidate>  
 <input type="hidden" th:field="\*{id}" />  
  
 <div class="mb-3">  
 <label for="name" class="form-label">Имя</label>  
 <input type="text" class="form-control" id="name" th:field="\*{name}" required />  
 </div>  
  
 <div class="mb-3">  
 <label for="surname" class="form-label">Фамилия</label>  
 <input type="text" class="form-control" id="surname" th:field="\*{surname}" required />  
 </div>  
  
 <div class="mb-3">  
 <label for="phone" class="form-label">Телефон</label>  
 <input type="tel" class="form-control" id="phone" th:field="\*{phone}" required  
 pattern="^\+?[0-9\s\-]{7,15}$" title="Введите корректный номер телефона" />  
 </div>  
  
 <div class="mb-3">  
 <label for="passport" class="form-label">Паспорт</label>  
 <input type="text" class="form-control" id="passport" th:field="\*{passport}" required  
 pattern="[0-9]{10}" title="Введите 10 цифр паспорта" />  
 </div>  
  
 <div class="mb-3">  
 <label for="ownerType" class="form-label">Тип владельца</label>  
 <select class="form-select" id="ownerType" th:field="\*{ownerType}">  
 <option value="физлицо">Физ. лицо</option>  
 <option value="юрлицо">Юр. лицо</option>  
 </select>  
 </div>  
  
 <div class="mb-3">  
 <label for="company" class="form-label">Компания</label>  
 <input type="text" class="form-control" id="company" th:field="\*{company}" />  
 </div>  
  
 <div class="mb-3">  
 <label for="license" class="form-label">Лицензия</label>  
 <input type="text" class="form-control" id="license" th:field="\*{license}" />  
 </div>  
  
 <button type="submit" class="btn btn-primary">Сохранить</button>  
 <a th:href="@{/client/list}" class="btn btn-secondary ms-2">Отмена</a>  
 </form>  
 </div>  
</main>  
  
<script src="https://cdn.jsdelivr.net/npm/bootstrap@5.3.3/dist/js/bootstrap.bundle.min.js"  
 integrity="sha384-YvpcrYf0tY3lHB60NNkmXc5s9fDVZLESaAA55NDzOxhy9GkcIdslK1eN7N6jIeHz"  
 crossorigin="anonymous"></script>  
  
<script>  
 document.addEventListener('DOMContentLoaded', function () {  
 const ownerType = document.getElementById('ownerType');  
 const companyField = document.getElementById('company').closest('.mb-3');  
 const licenseField = document.getElementById('license').closest('.mb-3');  
  
 function toggleFields() {  
 const isLegal = ownerType.value === 'юрлицо';  
 companyField.style.display = isLegal ? 'block' : 'none';  
 licenseField.style.display = isLegal ? 'block' : 'none';  
 }  
  
 ownerType.addEventListener('change', toggleFields);  
 toggleFields();  
 });  
</script>  
</body>  
</html>

<!DOCTYPE html>  
<html xmlns:th="http://www.thymeleaf.org"  
 xmlns:sec="http://www.thymeleaf.org/extras/spring-security"  
 lang="ru">  
<head>  
 <meta charset="UTF-8" />  
 <meta name="viewport" content="width=device-width, initial-scale=1" />  
 <title>Агентство недвижимости «TerraNova» — Клиенты</title>  
 <link href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.3/dist/css/bootstrap.min.css" rel="stylesheet"  
 integrity="sha384-QWTKZyjpPEjISv5WaRU9OFeRpok6YctnYmDr5pNlyT2bRjXh0JMhjY6hW+ALEwIH" crossorigin="anonymous" />  
 <style>  
 body, html {  
 height: 100%;  
 margin: 0;  
 }  
 .sidebar {  
 width: 180px;  
 background-color: #f8f9fa;  
 border-right: 1px solid #ddd;  
 height: 100vh;  
 position: fixed;  
 padding-top: 1rem;  
 }  
 .sidebar .nav-link {  
 margin: 0.3rem 0;  
 }  
 .header {  
 height: 56px;  
 padding: 0 1rem;  
 background-color: #e9ecef;  
 border-bottom: 1px solid #ddd;  
 margin-left: 180px;  
 display: flex;  
 align-items: center;  
 justify-content: space-between;  
 font-weight: 500;  
 font-size: 1rem;  
 }  
 .content {  
 margin-left: 180px;  
 margin-top: 56px;  
 padding: 1rem;  
 }  
 .btn-edit {  
 background-color: #fff176;  
 border-color: #fff176;  
 color: #000;  
 }  
 .btn-edit:hover {  
 background-color: #f9f871;  
 border-color: #f9f871;  
 }  
 .btn-delete {  
 background-color: #f44336;  
 border-color: #f44336;  
 color: #fff;  
 }  
 .btn-delete:hover {  
 background-color: #d32f2f;  
 border-color: #d32f2f;  
 }  
 table thead th {  
 text-align: center;  
 vertical-align: middle;  
 }  
 table tbody td {  
 vertical-align: middle;  
 }  
 .delete-confirmation {  
 font-size: 0.9rem;  
 color: #b71c1c;  
 user-select: none;  
 }  
 </style>  
</head>  
<body>  
  
*<!-- Боковое меню -->*<div th:replace="~{/include/sidebar :: sidebar}"></div>  
  
<header class="header">  
 <div>Агентство недвижимости «TerraNova» — Клиенты</div>  
 <div class="d-flex gap-2 align-items-center">  
  
 *<!-- Кнопки "Добавить" и "Удалить всех" только для EMPLOYEE и ADMIN -->* <div sec:authorize="hasAnyRole('EMPLOYEE','ADMIN')" class="d-flex align-items-center gap-2">  
 <a th:href="@{/client/add}" class="btn btn-primary">Добавить</a>  
 <form th:action="@{/client/deleteAll}" method="post" style="display: inline;"  
 sec:authorize="hasAnyRole('EMPLOYEE','ADMIN')"  
 onsubmit="return confirm('Удалить всех клиентов?');">  
 <button type="submit" class="btn btn-danger">Удалить всех</button>  
 </form>  
 </div>  
  
 *<!-- Сортировка -->* <div class="dropdown">  
 <button class="btn btn-outline-info dropdown-toggle" type="button" id="sortDropdown"  
 data-bs-toggle="dropdown" aria-expanded="false">  
 Сортировка  
 </button>  
 <ul class="dropdown-menu" aria-labelledby="sortDropdown">  
 <li><a class="dropdown-item" th:href="@{/client(sortField='surname', sortDir='asc')}"  
 th:classappend="${sortField == 'surname' and sortDir == 'asc'} ? ' active' : ''">Фамилия ↑</a></li>  
 <li><a class="dropdown-item" th:href="@{/client(sortField='surname', sortDir='desc')}"  
 th:classappend="${sortField == 'surname' and sortDir == 'desc'} ? ' active' : ''">Фамилия ↓</a></li>  
 <li><a class="dropdown-item" th:href="@{/client(sortField='name', sortDir='asc')}"  
 th:classappend="${sortField == 'name' and sortDir == 'asc'} ? ' active' : ''">Имя ↑</a></li>  
 <li><a class="dropdown-item" th:href="@{/client(sortField='name', sortDir='desc')}"  
 th:classappend="${sortField == 'name' and sortDir == 'desc'} ? ' active' : ''">Имя ↓</a></li>  
 <li><a class="dropdown-item" th:href="@{/client(sortField='ownerType', sortDir='asc')}"  
 th:classappend="${sortField == 'ownerType' and sortDir == 'asc'} ? ' active' : ''">Тип владельца ↑</a></li>  
 <li><a class="dropdown-item" th:href="@{/client(sortField='ownerType', sortDir='desc')}"  
 th:classappend="${sortField == 'ownerType' and sortDir == 'desc'} ? ' active' : ''">Тип владельца ↓</a></li>  
 <li><a class="dropdown-item" th:href="@{/client(sortField='company', sortDir='asc')}"  
 th:classappend="${sortField == 'company' and sortDir == 'asc'} ? ' active' : ''">Компания ↑</a></li>  
 <li><a class="dropdown-item" th:href="@{/client(sortField='company', sortDir='desc')}"  
 th:classappend="${sortField == 'company' and sortDir == 'desc'} ? ' active' : ''">Компания ↓</a></li>  
 <li><hr class="dropdown-divider"></li>  
 <li><a class="dropdown-item" th:href="@{/client}"  
 th:classappend="${sortField == null} ? ' active' : ''">Без сортировки</a></li>  
 </ul>  
 </div>  
 </div>  
</header>  
  
<main class="content">  
 <div th:if="${param.error}" class="alert alert-danger text-center" role="alert">  
 Невозможно удалить клиента — он участвует в сделке.  
 </div>  
  
 <div th:if="${param.partialError}" class="alert alert-warning text-center" role="alert">  
 Удалены все клиенты, не участвующие в сделках. <br />  
 <span th:text="'Невозможно удалить ' + ${param.partialError} + ' клиент(ов), так как они участвуют в сделках.'"></span>  
 </div>  
 <table class="table table-bordered table-hover table-striped align-middle text-center">  
 <thead class="table-light">  
 <tr>  
 <th>Имя</th>  
 <th>Фамилия</th>  
 <th>Телефон</th>  
 <th>Тип владельца</th>  
 <th>Компания</th>  
 <th sec:authorize="hasAnyRole('EMPLOYEE','ADMIN')">Действия</th>  
 </tr>  
 </thead>  
 <tbody>  
 <tr th:each="client : ${clients}">  
 <td th:text="${client.name}"></td>  
 <td th:text="${client.surname}"></td>  
 <td th:text="${client.phone}"></td>  
 <td th:text="${client.ownerType}"></td>  
 <td th:text="${client.company}"></td>  
 <td sec:authorize="hasAnyRole('EMPLOYEE','ADMIN')">  
 <a th:href="@{/client/edit/{id}(id=${client.id})}" class="btn btn-sm btn-edit me-1">Редактировать</a>  
  
 <form th:id="'deleteForm\_' + ${client.id}"  
 th:action="@{/client/delete/{id}(id=${client.id})}"  
 method="post" style="display: inline-block;"  
 onsubmit="return confirm('Удалить клиента?');">  
 <input type="hidden" th:name="${\_csrf.parameterName}" th:value="${\_csrf.token}" />  
 <button type="submit" class="btn btn-delete btn-sm">Удалить</button>  
 </form>  
  
 <div th:id="'deleteConfirm\_' + ${client.id}" class="delete-confirmation" style="display:none; margin-top: 0.3rem;">  
 <span>Удалить клиента?</span>  
 <button type="button" class="btn btn-sm btn-danger ms-2" th:attr="onclick=|submitDeleteForm(${client.id})|">Подтвердить</button>  
 <button type="button" class="btn btn-sm btn-secondary ms-1" th:attr="onclick=|hideDeleteConfirm(${client.id})|">Отмена</button>  
 </div>  
 </td>  
 </tr>  
 <tr th:if="${#lists.isEmpty(clients)}">  
 <td colspan="6" class="text-center text-muted">Клиенты отсутствуют</td>  
 </tr>  
 </tbody>  
 </table>  
</main>  
  
<script src="https://cdn.jsdelivr.net/npm/bootstrap@5.3.3/dist/js/bootstrap.bundle.min.js"  
 integrity="sha384-YvpcrYf0tY3lHB60NNkmXc5s9fDVZLESaAA55NDzOxhy9GkcIdslK1eN7N6jIeHz"  
 crossorigin="anonymous"></script>  
  
<script>  
 function showDeleteConfirm(id) {  
 document.getElementById('deleteConfirm\_' + id).style.display = 'block';  
 }  
  
 function hideDeleteConfirm(id) {  
 document.getElementById('deleteConfirm\_' + id).style.display = 'none';  
 }  
  
 function submitDeleteForm(id) {  
 document.getElementById('deleteForm\_' + id).submit();  
 }  
  
 function showDeleteAllConfirm() {  
 document.getElementById('deleteAllConfirm').style.display = 'block';  
 }  
  
 function hideDeleteAllConfirm() {  
 document.getElementById('deleteAllConfirm').style.display = 'none';  
 }  
</script>  
  
</body>  
</html>

<!DOCTYPE html>  
<html xmlns:th="http://www.thymeleaf.org" lang="ru">  
<head>  
 <meta charset="UTF-8" />  
 <meta name="viewport" content="width=device-width, initial-scale=1" />  
 <title th:text="${deal.id == null} ? 'Создание сделки' : 'Редактирование сделки'">Сделка</title>  
 <link href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.3/dist/css/bootstrap.min.css" rel="stylesheet"  
 integrity="sha384-QWTKZyjpPEjISv5WaRU9OFeRpok6YctnYmDr5pNlyT2bRjXh0JMhjY6hW+ALEwIH" crossorigin="anonymous" />  
</head>  
<body class="d-flex flex-column min-vh-100">  
  
<div th:replace="~{/include/header}"></div>  
  
<main class="flex-grow-1">  
 <div class="container mt-5">  
 <h1 th:text="${deal.id == null} ? 'Добавить сделку' : 'Редактировать сделку'"></h1>  
  
 <form th:action="@{/deal/save}" th:object="${deal}" method="post" novalidate>  
 <input type="hidden" th:field="\*{id}" />  
 *<!-- Недвижимость -->* <div class="mb-3">  
 <label for="realtySelect" class="form-label">Недвижимость</label>  
 <select th:field="\*{realty.id}" class="form-select" id="realtySelect" onchange="updateRealtyFields()" required>  
 <option value="" disabled th:selected="\*{realty == null}">-- Выберите недвижимость --</option>  
 <option th:each="realty : ${realties}"  
 th:value="${realty.id}"  
 th:text="${realty.region + ', ' + realty.locality + ', ' + realty.street}">  
 </option>  
 </select>  
 <div class="invalid-feedback">Пожалуйста, выберите недвижимость.</div>  
 </div>  
  
 *<!-- Владелец -->* <div class="mb-3">  
 <label for="ownerField" class="form-label">Владелец</label>  
 <input type="text" id="ownerField" class="form-control" readonly />  
 </div>  
  
 *<!-- Сумма -->* <div class="mb-3">  
 <label for="amountField" class="form-label">Сумма (₽)</label>  
 <input type="text" th:field="\*{amount}" id="amountField" class="form-control" readonly />  
 </div>  
  
 *<!-- Клиент -->* <div class="mb-3">  
 <label for="clientSelect" class="form-label">Клиент</label>  
 <select th:field="\*{client.id}" class="form-select" id="clientSelect" onchange="updateClientFields()" required>  
 <option value="" disabled th:selected="\*{client == null}">-- Выберите клиента --</option>  
 <option th:each="client : ${clients}"  
 th:value="${client.id}"  
 th:text="${client.fullName}">  
 </option>  
 </select>  
 <div class="invalid-feedback">Пожалуйста, выберите клиента.</div>  
 </div>  
  
 *<!-- Телефон клиента -->* <div class="mb-3">  
 <label for="clientNumberField" class="form-label">Телефон клиента</label>  
 <input type="text" id="clientNumberField" class="form-control" readonly />  
 </div>  
  
 *<!-- Скрытое поле для телефона клиента -->* <input type="hidden" th:field="\*{number}" />  
  
 *<!-- Сотрудник -->* <div class="mb-3">  
 <label for="employeeSelect" class="form-label">Сотрудник</label>  
 <select th:field="\*{employee.id}" class="form-select" id="employeeSelect" required>  
 <option value="" disabled th:selected="\*{employee == null}">-- Выберите сотрудника --</option>  
 <option th:each="employee : ${employees}"  
 th:value="${employee.id}"  
 th:text="${employee.lastName + ' ' + employee.firstName + ' (' + employee.position + ')'}">  
 </option>  
 </select>  
 <div class="invalid-feedback">Пожалуйста, выберите сотрудника.</div>  
 </div>  
  
 *<!-- Дата -->* <div class="mb-3">  
 <label for="dateField" class="form-label">Дата сделки</label>  
 <input type="datetime-local" th:field="\*{date}" id="dateField" class="form-control" required />  
 <div class="invalid-feedback">Пожалуйста, укажите дату сделки.</div>  
 </div>  
  
 *<!-- Статус -->* <div class="mb-3">  
 <label for="statusField" class="form-label">Статус</label>  
 <input type="text" th:field="\*{status}" id="statusField" class="form-control" required />  
 <div class="invalid-feedback">Пожалуйста, укажите статус сделки.</div>  
 </div>  
  
 <button type="submit" class="btn btn-primary">Сохранить</button>  
 <a th:href="@{/deal}" class="btn btn-secondary ms-2">Отмена</a>  
 </form>  
 </div>  
</main>  
  
*<!-- JSON данные -->*<div id="realtyJson" th:attr="data-json=${realtyJson}" hidden></div>  
<div id="clientJson" th:attr="data-json=${clientJson}" hidden></div>  
  
<script>  
 function updateRealtyFields() {  
 const realtyElement = document.getElementById("realtyJson");  
 const data = realtyElement ? realtyElement.dataset.json : "[]";  
 const realtyData = JSON.parse(data || "[]");  
 const selectedId = document.getElementById("realtySelect").value;  
 const realty = realtyData.find(r => r.id == selectedId);  
 document.getElementById("ownerField").value = realty?.ownerName || "";  
 document.getElementById("amountField").value = realty?.cost || "";  
 }  
  
 function updateClientFields() {  
 const clientElement = document.getElementById("clientJson");  
 const data = clientElement ? clientElement.dataset.json : "[]";  
 const clientData = JSON.parse(data || "[]");  
 const selectedId = document.getElementById("clientSelect").value;  
 const client = clientData.find(c => c.id == selectedId);  
 document.getElementById("clientNumberField").value = client?.phone || "";  
 document.querySelector("[name='number']").value = client?.phone || "";  
 }  
  
 document.addEventListener("DOMContentLoaded", function () {  
 updateRealtyFields();  
 updateClientFields();  
  
 const form = document.querySelector('form');  
 form.addEventListener('submit', function(event) {  
 if (!form.checkValidity()) {  
 event.preventDefault();  
 event.stopPropagation();  
 }  
 form.classList.add('was-validated');  
 }, false);  
 });  
</script>  
  
<script src="https://cdn.jsdelivr.net/npm/bootstrap@5.3.3/dist/js/bootstrap.bundle.min.js"  
 integrity="sha384-YvpcrYf0tY3lHB60NNkmXc5s9fDVZLESaAA55NDzOxhy9GkcIdslK1eN7N6jIeHz"  
 crossorigin="anonymous"></script>  
</body>  
</html>

<!DOCTYPE html>  
<html xmlns:th="http://www.thymeleaf.org"  
 xmlns:sec="http://www.thymeleaf.org/extras/spring-security"  
 lang="ru">  
<head>  
 <meta charset="UTF-8" />  
 <meta name="viewport" content="width=device-width, initial-scale=1" />  
 <title>Агентство недвижимости «TerraNova» — Сделки</title>  
 <link href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.3/dist/css/bootstrap.min.css" rel="stylesheet"  
 integrity="sha384-QWTKZyjpPEjISv5WaRU9OFeRpok6YctnYmDr5pNlyT2bRjXh0JMhjY6hW+ALEwIH" crossorigin="anonymous" />  
 <style>  
 body, html {  
 height: 100%;  
 margin: 0;  
 }  
 .sidebar {  
 width: 180px;  
 background-color: #f8f9fa;  
 border-right: 1px solid #ddd;  
 height: 100vh;  
 position: fixed;  
 padding-top: 1rem;  
 }  
 .sidebar .nav-link {  
 margin: 0.3rem 0;  
 }  
 .header {  
 height: 56px;  
 padding: 0 1rem;  
 background-color: #e9ecef;  
 border-bottom: 1px solid #ddd;  
 margin-left: 180px;  
 display: flex;  
 align-items: center;  
 justify-content: space-between;  
 font-weight: 500;  
 font-size: 1rem;  
 }  
 .header .btn {  
 min-width: 100px;  
 }  
 .content {  
 margin-left: 180px;  
 margin-top: 56px;  
 padding: 1rem;  
 }  
 .btn-edit {  
 background-color: #fff176;  
 border-color: #fff176;  
 color: #000;  
 }  
 .btn-edit:hover {  
 background-color: #f9f871;  
 border-color: #f9f871;  
 }  
 .btn-delete {  
 background-color: #f44336;  
 border-color: #f44336;  
 color: #fff;  
 }  
 .btn-delete:hover {  
 background-color: #d32f2f;  
 border-color: #d32f2f;  
 }  
 table thead th {  
 text-align: center;  
 vertical-align: middle;  
 }  
 table tbody td {  
 vertical-align: middle;  
 }  
 </style>  
</head>  
<body>  
  
*<!-- Боковое меню -->*<div th:replace="~{/include/sidebar :: sidebar}"></div>  
  
<header class="header">  
 <div>Агентство недвижимости «TerraNova» — Сделки</div>  
 <div class="d-flex gap-2">  
 *<!-- Кнопка "Добавить" -->* <div sec:authorize="hasAnyRole('EMPLOYEE','ADMIN')" class="align-self-center">  
 <a href="/deal/add" class="btn btn-primary">Добавить</a>  
 </div>  
 *<!-- Кнопка "Удалить все" -->* <div sec:authorize="hasAnyRole('EMPLOYEE','ADMIN')" class="align-self-center">  
 <form th:action="@{/deal/deleteAll}" method="post"  
 onsubmit="return confirm('Удалить все сделки?');" style="margin: 0; display:inline-block;">  
 <input type="hidden" th:name="${\_csrf.parameterName}" th:value="${\_csrf.token}" />  
 <button type="submit" class="btn btn-danger">Удалить все</button>  
 </form>  
 </div>  
  
 *<!-- Выпадающее меню сортировки -->* <div class="dropdown">  
 <button class="btn btn-outline-info dropdown-toggle" type="button" id="sortDropdown"  
 data-bs-toggle="dropdown" aria-expanded="false">  
 Сортировка  
 </button>  
 <ul class="dropdown-menu" aria-labelledby="sortDropdown">  
 <li>  
 <a class="dropdown-item" th:href="@{/deal(sortField='amount', sortDir='asc')}"  
 th:classappend="${(sortField == 'amount' and sortDir == 'asc') ? ' active' : ''}">Сумма ↑</a>  
 </li>  
 <li>  
 <a class="dropdown-item" th:href="@{/deal(sortField='amount', sortDir='desc')}"  
 th:classappend="${(sortField == 'amount' and sortDir == 'desc') ? ' active' : ''}">Сумма ↓</a>  
 </li>  
 <li>  
 <a class="dropdown-item" th:href="@{/deal(sortField='date', sortDir='asc')}"  
 th:classappend="${(sortField == 'date' and sortDir == 'asc') ? ' active' : ''}">Дата ↑</a>  
 </li>  
 <li>  
 <a class="dropdown-item" th:href="@{/deal(sortField='date', sortDir='desc')}"  
 th:classappend="${(sortField == 'date' and sortDir == 'desc') ? ' active' : ''}">Дата ↓</a>  
 </li>  
 <li><hr class="dropdown-divider"></li>  
 <li>  
 <a class="dropdown-item" th:href="@{/deal}"  
 th:classappend="${sortField == null} ? ' active' : ''">Без сортировки</a>  
 </li>  
 </ul>  
 </div>  
 </div>  
</header>  
  
<main class="content">  
  
 <div th:if="${errorMessage}" class="alert alert-danger" th:text="${errorMessage}"></div>  
 <div th:if="${warningMessage}" class="alert alert-warning" th:text="${warningMessage}"></div>  
  
 <table class="table table-bordered table-hover table-striped align-middle text-center">  
 <thead class="table-light">  
 <tr>  
 <th>Дата</th>  
 <th>Сотрудник</th>  
 <th>Клиент</th>  
 <th>Объект</th>  
 <th>Сумма</th>  
 <th sec:authorize="hasAnyRole('EMPLOYEE','ADMIN')">Действия</th>  
 </tr>  
 </thead>  
 <tbody>  
 <tr th:each="deal : ${deals}">  
 <td th:text="${#temporals.format(deal.date, 'dd.MM.yyyy')}"></td>  
 <td th:text="${deal.employee.fullName}"></td>  
 <td th:text="${deal.client.fullName}"></td>  
 <td th:text="${deal.realty.region + ', ' + deal.realty.street}"></td>  
 <td th:text="${deal.formattedCost}"></td>  
 <td sec:authorize="hasAnyRole('EMPLOYEE','ADMIN')">  
 <a th:href="@{/deal/edit/{id}(id=${deal.id})}" class="btn btn-edit btn-sm me-1">Изменить</a>  
 <form th:action="@{/deal/delete/{id}(id=${deal.id})}" method="post"  
 onsubmit="return confirm('Удалить эту сделку?');" style="display:inline-block; margin: 0;">  
 <input type="hidden" th:name="${\_csrf.parameterName}" th:value="${\_csrf.token}" />  
 <button type="submit" class="btn btn-delete btn-sm">Удалить</button>  
 </form>  
 </td>  
 </tr>  
 <tr th:if="${#lists.isEmpty(deals)}">  
 <td colspan="6" class="text-center text-muted">Сделки не найдены.</td>  
 </tr>  
 </tbody>  
 </table>  
  
</main>  
  
<script src="https://cdn.jsdelivr.net/npm/bootstrap@5.3.3/dist/js/bootstrap.bundle.min.js"  
 integrity="sha384-YvpcrYf0tY3lHB60NNkmXc5s9fDVZLESaAA55NDzOxhy9GkcIdslK1eN7N6jIeHz"  
 crossorigin="anonymous"></script>  
</body>  
</html>

<!DOCTYPE html>  
<html xmlns:th="http://www.thymeleaf.org" lang="ru">  
<head>  
 <meta charset="UTF-8" />  
 <meta name="viewport" content="width=device-width, initial-scale=1" />  
 <title>Агентство недвижимости «TerraNova» — Форма для сотрудников</title>  
 <link href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.3/dist/css/bootstrap.min.css" rel="stylesheet"  
 integrity="sha384-QWTKZyjpPEjISv5WaRU9OFeRpok6YctnYmDr5pNlyT2bRjXh0JMhjY6hW+ALEwIH" crossorigin="anonymous" />  
</head>  
<body class="d-flex flex-column min-vh-100">  
  
*<!-- Подключение шапки -->*<div th:replace="~{/include/header}"></div>  
  
<main class="flex-grow-1">  
 <div class="container mt-5">  
 <h1 th:text="${employee.id == null ? 'Добавить сотрудника' : 'Редактировать сотрудника'}">Форма сотрудника</h1>  
  
 <form th:action="@{/employee/save}" th:object="${employee}" method="post" novalidate>  
 <input type="hidden" th:field="\*{id}" />  
  
 *<!-- Выбор пользователя -->* <div class="mb-4">  
 <label for="userSelect" class="form-label">Выберите пользователя:</label>  
 <select class="form-select" id="userSelect" th:field="\*{user.id}">  
 <option value="">— Не выбрано —</option>  
 <option th:each="user : ${users}"  
 th:value="${user.id}"  
 th:selected="${employee.user != null and user.id == employee.user.id}"  
 th:data-name="${user.name}"  
 th:data-surname="${user.surname}"  
 th:text="|${user.username} (${user.name} ${user.surname})|">  
 </option>  
 </select>  
 </div>  
  
 *<!-- Остальные поля -->* <div class="mb-3">  
 <label for="name" class="form-label">Имя:</label>  
 <input type="text" class="form-control" id="name" th:field="\*{name}" required />  
 </div>  
  
 <div class="mb-3">  
 <label for="surname" class="form-label">Фамилия:</label>  
 <input type="text" class="form-control" id="surname" th:field="\*{surname}" required />  
 </div>  
  
 <div class="mb-3">  
 <label for="patronymic" class="form-label">Отчество:</label>  
 <input type="text" class="form-control" id="patronymic" th:field="\*{patronymic}" />  
 </div>  
  
 <div class="mb-3">  
 <label for="position" class="form-label">Должность:</label>  
 <input type="text" class="form-control" id="position" th:field="\*{position}" required />  
 </div>  
  
 <div class="mb-3">  
 <label for="email" class="form-label">Email:</label>  
 <input type="email" class="form-control" id="email" th:field="\*{email}" required />  
 </div>  
  
 <div class="mb-3">  
 <label for="phone" class="form-label">Телефон:</label>  
 <input type="text" class="form-control" id="phone" th:field="\*{phone}" />  
 </div>  
  
 <button type="submit" class="btn btn-primary">Сохранить</button>  
 <a th:href="@{/employee}" class="btn btn-secondary">Назад к списку</a>  
 </form>  
  
 *<!-- Блок ошибок валидации -->* <th:block th:if="${validationErrors != null and #lists.isNotEmpty(validationErrors)}">  
 <ul class="alert alert-danger mt-3">  
 <li th:each="error : ${validationErrors}" th:text="${error}"></li>  
 </ul>  
 </th:block>  
 </div>  
</main>  
  
<script>  
 document.addEventListener("DOMContentLoaded", function () {  
 const userSelect = document.getElementById("userSelect");  
 const nameInput = document.getElementById("name");  
 const surnameInput = document.getElementById("surname");  
  
 userSelect.addEventListener("change", function () {  
 const selectedOption = userSelect.options[userSelect.selectedIndex];  
 const name = selectedOption.getAttribute("data-name") || "";  
 const surname = selectedOption.getAttribute("data-surname") || "";  
  
 nameInput.value = name;  
 surnameInput.value = surname;  
 });  
 });  
</script>  
  
<script src="https://cdn.jsdelivr.net/npm/bootstrap@5.3.3/dist/js/bootstrap.bundle.min.js"  
 integrity="sha384-YvpcrYf0tY3lHB60NNkmXc5s9fDVZLESaAA55NDzOxhy9GkcIdslK1eN7N6jIeHz"  
 crossorigin="anonymous"></script>  
  
*<!-- Подключение подвала -->*<div th:replace="~{/include/footer}"></div>  
  
</body>  
</html>

<!DOCTYPE html>  
<html xmlns:th="http://www.thymeleaf.org"  
 xmlns:sec="http://www.thymeleaf.org/extras/spring-security"  
 lang="ru">  
<head>  
 <meta charset="UTF-8"/>  
 <meta name="viewport" content="width=device-width, initial-scale=1"/>  
 <title>Агентство недвижимости «TerraNova» — Список сотрудников</title>  
 <link href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.3/dist/css/bootstrap.min.css" rel="stylesheet"  
 integrity="sha384-QWTKZyjpPEjISv5WaRU9OFeRpok6YctnYmDr5pNlyT2bRjXh0JMhjY6hW+ALEwIH" crossorigin="anonymous"/>  
 <style>  
 body, html {  
 height: 100%;  
 margin: 0;  
 }  
 .sidebar {  
 width: 180px;  
 background-color: #f8f9fa;  
 border-right: 1px solid #ddd;  
 height: 100vh;  
 position: fixed;  
 padding-top: 1rem;  
 }  
 .sidebar .nav-link {  
 margin: 0.3rem 0;  
 }  
 .header {  
 height: 56px;  
 padding: 0 1rem;  
 background-color: #e9ecef;  
 border-bottom: 1px solid #ddd;  
 margin-left: 180px;  
 display: flex;  
 align-items: center;  
 justify-content: space-between;  
 }  
 .content {  
 margin-left: 180px;  
 margin-top: 56px;  
 padding: 1rem;  
 }  
 .btn-edit {  
 background-color: #fff176;  
 border-color: #fff176;  
 color: #000;  
 }  
 .btn-edit:hover {  
 background-color: #f9f871;  
 border-color: #f9f871;  
 }  
 .btn-delete {  
 background-color: #f44336;  
 border-color: #f44336;  
 color: #fff;  
 }  
 .btn-delete:hover {  
 background-color: #d32f2f;  
 border-color: #d32f2f;  
 }  
 table thead th {  
 text-align: center;  
 vertical-align: middle;  
 }  
 table tbody td {  
 vertical-align: middle;  
 }  
 </style>  
</head>  
<body>  
  
*<!-- Боковое меню -->*<div th:replace="~{/include/sidebar :: sidebar}"></div>  
  
*<!-- Шапка -->*<header class="header">  
 <div>Агентство недвижимости «TerraNova» — Список сотрудников</div>  
 <div class="d-flex gap-2">  
 <div sec:authorize="hasAnyRole('EMPLOYEE', 'ADMIN')">  
 <a th:href="@{/employee/add}" class="btn btn-primary">Добавить</a>  
 <a th:href="@{/employee/deleteAll}" class="btn btn-danger"  
 onclick="return confirm('Вы уверены, что хотите удалить всех сотрудников?');">Удалить всех</a>  
 </div>  
  
 *<!-- Сортировка -->* <div class="dropdown ms-2">  
 <button class="btn btn-outline-info dropdown-toggle" type="button" id="sortDropdown"  
 data-bs-toggle="dropdown" aria-expanded="false">  
 Сортировка  
 </button>  
 <ul class="dropdown-menu" aria-labelledby="sortDropdown">  
 <li><a class="dropdown-item" th:href="@{/employee(sortBy='surname', sortDir='asc')}"  
 th:classappend="${sortBy == 'surname' and sortDir == 'asc'} ? ' active' : ''">Фамилия ↑</a></li>  
 <li><a class="dropdown-item" th:href="@{/employee(sortBy='surname', sortDir='desc')}"  
 th:classappend="${sortBy == 'surname' and sortDir == 'desc'} ? ' active' : ''">Фамилия ↓</a></li>  
 <li><a class="dropdown-item" th:href="@{/employee(sortBy='name', sortDir='asc')}"  
 th:classappend="${sortBy == 'name' and sortDir == 'asc'} ? ' active' : ''">Имя ↑</a></li>  
 <li><a class="dropdown-item" th:href="@{/employee(sortBy='name', sortDir='desc')}"  
 th:classappend="${sortBy == 'name' and sortDir == 'desc'} ? ' active' : ''">Имя ↓</a></li>  
 <li><a class="dropdown-item" th:href="@{/employee(sortBy='position', sortDir='asc')}"  
 th:classappend="${sortBy == 'position' and sortDir == 'asc'} ? ' active' : ''">Должность ↑</a></li>  
 <li><a class="dropdown-item" th:href="@{/employee(sortBy='position', sortDir='desc')}"  
 th:classappend="${sortBy == 'position' and sortDir == 'desc'} ? ' active' : ''">Должность ↓</a></li>  
 <li><hr class="dropdown-divider"></li>  
 <li><a class="dropdown-item" th:href="@{/employee}"  
 th:classappend="${sortBy == null} ? ' active' : ''">Без сортировки</a></li>  
 </ul>  
 </div>  
 </div>  
</header>  
  
*<!-- Контент -->*<main class="content">  
 *<!-- Сообщение при ошибке удаления одного сотрудника -->* <div th:if="${param.error}" class="alert alert-danger text-center" role="alert">  
 Невозможно удалить сотрудника — он участвует в сделке.  
 </div>  
  
 *<!-- Сообщение при частичном удалении -->* <div th:if="${param.partialError}" class="alert alert-warning text-center" role="alert">  
 Удалены все сотрудники, не участвующие в сделках. <br/>  
 <span th:text="'Невозможно удалить ' + ${param.partialError} + ' сотрудник(ов), так как они участвуют в сделках.'"></span>  
 </div>  
  
 <table class="table table-bordered table-hover table-striped">  
 <thead class="table-light">  
 <tr>  
 <th>Имя</th>  
 <th>Фамилия</th>  
 <th>Телефон</th>  
 <th>Должность</th>  
 <th>Пользователь</th>  
 <th sec:authorize="hasAnyRole('EMPLOYEE', 'ADMIN')">Действия</th>  
 </tr>  
 </thead>  
  
 <tbody>  
 <tr th:each="employee : ${employees}">  
 <td th:text="${employee.name}">Имя</td>  
 <td th:text="${employee.surname}">Фамилия</td>  
 <td th:text="${employee.phone}">Телефон</td>  
 <td th:text="${employee.position}">Должность</td>  
 <td th:text="${employee.user != null ? employee.user.username : '-'}">Пользователь</td>  
 <td class="text-center" sec:authorize="hasAnyRole('EMPLOYEE', 'ADMIN')">  
 <a th:href="@{/employee/edit/{id}(id=${employee.id})}" class="btn btn-edit btn-sm me-1">Изменить</a>  
 <a th:href="@{/employee/delete/{id}(id=${employee.id})}"  
 class="btn btn-delete btn-sm"  
 onclick="return confirm('Удалить сотрудника?');">Удалить</a>  
 </td>  
 </tr>  
 <tr th:if="${#lists.isEmpty(employees)}">  
 <td colspan="6" class="text-center text-muted">Сотрудники отсутствуют.</td>  
 </tr>  
 </tbody>  
  
 </table>  
</main>  
  
<script src="https://cdn.jsdelivr.net/npm/bootstrap@5.3.3/dist/js/bootstrap.bundle.min.js"  
 integrity="sha384-YvpcrYf0tY3lHB60NNkmXc5s9fDVZLESaAA55NDzOxhy9GkcIdslK1eN7N6jIeHz"  
 crossorigin="anonymous"></script>  
</body>  
</html>

<!DOCTYPE html>  
<html lang="ru" xmlns:th="http://www.thymeleaf.org">  
<head>  
 <meta charset="UTF-8" />  
 <meta name="viewport" content="width=device-width, initial-scale=1" />  
 <title>Агентство недвижимости «TerraNova» — О системе</title>  
 <link href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.3/dist/css/bootstrap.min.css" rel="stylesheet"  
 integrity="sha384-QWTKZyjpPEjISv5WaRU9OFeRpok6YctnYmDr5pNlyT2bRjXh0JMhjY6hW+ALEwIH" crossorigin="anonymous" />  
 <style>  
 body, html {  
 height: 100%;  
 }  
 body {  
 display: flex;  
 flex-direction: column;  
 min-height: 100vh;  
 }  
 main {  
 flex-grow: 1;  
 max-width: 900px;  
 margin: 2rem auto;  
 padding: 0 1rem;  
 }  
 </style>  
</head>  
<body class="bg-light">  
  
<div th:replace="~{/include/header}"></div>  
  
<main role="main" class="bg-white rounded shadow-sm p-4">  
 <h1 class="mb-4">О системе TerraNova</h1>  
  
 <p>Добро пожаловать в информационную систему агентства недвижимости <strong>«TerraNova»</strong>. Наша платформа предназначена для эффективного управления объектами недвижимости, сделками и клиентской базой.</p>  
  
 <h2 class="h4 mt-4 mb-3">Основные возможности системы:</h2>  
 <ul class="list-group list-group-flush mb-4">  
 <li class="list-group-item">Управление списком объектов недвижимости с возможностью добавления, редактирования и удаления.</li>  
 <li class="list-group-item">Ведение и контроль сделок с недвижимостью.</li>  
 <li class="list-group-item">Работа с базой клиентов и сотрудников агентства.</li>  
 <li class="list-group-item">Безопасная авторизация и управление правами доступа.</li>  
 </ul>  
  
 <h2 class="h4 mt-4 mb-3">Технологии и архитектура:</h2>  
 <p>Система построена с использованием современных технологий:</p>  
 <ul class="list-group list-group-flush mb-4">  
 <li class="list-group-item">Backend на <strong>Spring Boot</strong> с использованием JPA и Hibernate для работы с базой данных.</li>  
 <li class="list-group-item">Frontend с использованием <strong>Thymeleaf</strong> и <strong>Bootstrap 5</strong> для удобного и адаптивного интерфейса.</li>  
 <li class="list-group-item">Безопасность реализована через Spring Security с поддержкой регистрации и аутентификации пользователей.</li>  
 </ul>  
  
 <h2 class="h4 mt-4 mb-3">Контакты для поддержки:</h2>  
 <p>Если у вас возникли вопросы или предложения по работе системы, пожалуйста, свяжитесь с нашей службой поддержки:</p>  
 <ul class="list-unstyled mb-0">  
 <li>Email: <a href="mailto:support@terranova.example">support@terranova.example</a></li>  
 <li>Телефон: +7 (906) 434-62-48</li>  
 <li>Адрес офиса: г. Белореченск, ул. Благодатная, 64</li>  
 </ul>  
</main>  
  
<div th:replace="~{/include/footer}"></div>  
  
<script src="https://cdn.jsdelivr.net/npm/bootstrap@5.3.3/dist/js/bootstrap.bundle.min.js"  
 integrity="sha384-YvpcrYf0tY3lHB60NNkmXc5s9fDVZLESaAA55NDzOxhy9GkcIdslK1eN7N6jIeHz"  
 crossorigin="anonymous"></script>  
  
</body>  
</html>

<div th:fragment="footer" xmlns:th="http://www.w3.org/1999/xhtml">  
 <footer class="footer py-3 bg-light border-top">  
 <div class="container text-center">  
 <span class="text-muted">© 2025 Агентство недвижимости «TerraNova». Все права защищены.</span>  
 </div>  
 </footer>  
</div>

<div th:fragment="header" xmlns:sec="http://www.thymeleaf.org" xmlns:th="http://www.thymeleaf.org">  
  
 <link href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.3/dist/css/bootstrap.min.css" rel="stylesheet"  
 integrity="sha384-QWTKZyjpPEjISv5WaRU9OFeRpok6YctnYmDr5pNlyT2bRjXh0JMhjY6hW+ALEwIH" crossorigin="anonymous">  
  
 <nav class="navbar navbar-expand-lg navbar-light bg-light">  
 <div class="container-fluid">  
  
 <a class="navbar-brand" href="/">Агентство недвижимости «TerraNova»</a>  
  
 <div class="d-flex align-items-center">  
 *<!-- Кнопки "Главная" и "Профиль" -->* <a class="btn btn-outline-primary me-2" href="/" role="button">Главная</a>  
 <a class="btn btn-outline-primary me-4" th:href="@{/profile}" role="button" sec:authorize="isAuthenticated()">Профиль</a>  
 </div>  
  
 *<!-- Ссылки на разделы -->* <ul class="navbar-nav flex-row flex-wrap">  
 <li class="nav-item me-3">  
 <a class="nav-link" th:href="@{/realty}">Недвижимость</a>  
 </li>  
 <li class="nav-item me-3">  
 <a class="nav-link" th:href="@{/deal}">Сделки</a>  
 </li>  
 <li class="nav-item me-3" sec:authorize="hasAnyRole('EMPLOYEE', 'ADMIN')">  
 <a class="nav-link" th:href="@{/client}">Клиенты</a>  
 </li>  
 <li class="nav-item me-3" sec:authorize="hasAnyRole( 'ADMIN')">  
 <a class="nav-link" th:href="@{/employee}">Сотрудники</a>  
 </li>  
 <li class="nav-item" sec:authorize="hasRole('ADMIN')">  
 <a class="nav-link" th:href="@{/user}">Пользователи</a>  
 </li>  
 </ul>  
  
 *<!-- Панель авторизации -->* <ul class="navbar-nav ms-auto mb-2 mb-lg-0">  
 <li class="nav-item" sec:authorize="!isAuthenticated()">  
 <a class="nav-link" th:href="@{/include/login}">Вход</a>  
 </li>  
 <li class="nav-item" sec:authorize="!isAuthenticated()">  
 <a class="nav-link" th:href="@{/include/registration}">Регистрация</a>  
 </li>  
 <li class="nav-item" sec:authorize="isAuthenticated()">  
 <form th:action="@{/logout}" method="post" class="d-inline">  
 <button type="submit" class="btn btn-outline-danger ms-2">Выход</button>  
 </form>  
 </li>  
 </ul>  
  
 </div>  
 </nav>  
  
 <script src="https://cdn.jsdelivr.net/npm/bootstrap@5.3.3/dist/js/bootstrap.bundle.min.js"  
 integrity="sha384-YvpcrYf0tY3lHB60NNkmXc5s9fDVZLESaAA55NDzOxhy9GkcIdslK1eN7N6jIeHz"  
 crossorigin="anonymous"></script>  
</div>

<!DOCTYPE html>  
<html lang="ru" xmlns:th="http://www.thymeleaf.org" xmlns:sec="http://www.thymeleaf.org/extras/spring-security">  
<head>  
 <meta charset="UTF-8">  
 <meta name="viewport" content="width=device-width, initial-scale=1">  
 <title>Агентство недвижимости «TerraNova» — Главная</title>  
 <link href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.3/dist/css/bootstrap.min.css" rel="stylesheet" integrity="sha384-QWTKZyjpPEjISv5WaRU9OFeRpok6YctnYmDr5pNlyT2bRjXh0JMhjY6hW+ALEwIH" crossorigin="anonymous">  
</head>  
<body class="d-flex flex-column min-vh-100">  
<div th:replace="~{/include/header}"></div>  
  
<main class="flex-grow-1">  
 <div class="container mt-5 text-center">  
 <h1>Агентство недвижимости «TerraNova»</h1>  
  
 <h5 class="mt-4" sec:authorize="isAuthenticated()">Ваше имя:</h5>  
 <p class="fw-normal fs-5" th:text="${#authentication.name}" sec:authorize="isAuthenticated()"></p>  
  
 <p class="mt-4 fs-5">  
 TerraNova — надежное агентство недвижимости, которое поможет вам быстро и безопасно найти или продать жильё мечты.  
 </p>  
  
 <h3 class="mt-5">Наши услуги</h3>  
 <ul class="list-unstyled fs-5">  
 <li>• Покупка и продажа недвижимости</li>  
 <li>• Юридическое сопровождение сделок</li>  
 <li>• Консультации экспертов рынка</li>  
 </ul>  
  
 <a href="/realty" class="btn btn-primary btn-lg mt-4">Найти недвижимость</a>  
  
 <section class="mt-5">  
 <h3>Отзывы наших клиентов</h3>  
 <div class="row justify-content-center">  
 <div class="col-md-4">  
 <blockquote class="blockquote">  
 <p>«Спасибо TerraNova за профессиональную помощь и быструю сделку!»</p>  
 <footer class="blockquote-footer">Ирина Смирнова</footer>  
 </blockquote>  
 </div>  
 <div class="col-md-4">  
 <blockquote class="blockquote">  
 <p>«Отличный сервис и внимательное отношение к клиентам.»</p>  
 <footer class="blockquote-footer">Алексей Петров</footer>  
 </blockquote>  
 </div>  
 </div>  
 </section>  
  
 <section class="mt-5">  
 <h3>Контакты</h3>  
 <p>Адрес офиса: г. Белореченск, ул. Благодатная, 64</p>  
 <p>Телефон: +7 (906) 434-62-48</p>  
 <p>Email: info@terranova.ru</p>  
 <p>Часы работы: Пн–Пт, 9:00–18:00</p>  
 </section>  
 </div>  
</main>  
  
<div th:replace="~{/include/footer}"></div>  
<script src="https://cdn.jsdelivr.net/npm/bootstrap@5.3.3/dist/js/bootstrap.bundle.min.js" integrity="sha384-ZGJQH6C7+C7DkOJgpFrw20FdLcdvrU3HkakK0ZAdPWJ/wVG51731EbPY6beOOA4F" crossorigin="anonymous"></script>  
</body>  
</html>

<!DOCTYPE html>  
<html lang="ru" xmlns:th="http://www.thymeleaf.org/">  
<head>  
 <meta charset="UTF-8">  
 <title>Агентство недвижимости «TerraNova» — Авторизация</title>  
 <link href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.3/dist/css/bootstrap.min.css" rel="stylesheet" integrity="sha384-QWTKZyjpPEjISv5WaRU9OFeRpok6YctnYmDr5pNlyT2bRjXh0JMhjY6hW+ALEwIH" crossorigin="anonymous">  
</head>  
<body class="d-flex flex-column min-vh-100">  
<div th:replace="~{/include/header}"></div>  
  
<main class="flex-grow-1">  
 <div class="container col-4 mt-5">  
 <h1 class="text-center">Авторизация</h1>  
 <hr>  
 <form th:action="@{/login}" method="post">  
 <input type="hidden" th:name="${\_csrf.parameterName}" th:value="${\_csrf.token}"/>  
 <div class="mb-3">  
 <label for="username" class="form-label">Логин:</label>  
 <input type="text" class="form-control" id="username" name="username" required autofocus />  
 </div>  
 <div class="mb-3">  
 <label for="password" class="form-label">Пароль:</label>  
 <input type="password" class="form-control" id="password" name="password" required />  
 </div>  
 <div th:if="${param.error}" class="alert alert-danger" role="alert">  
 Указаны неверные данные. Попробуйте еще раз.  
 </div>  
 <button class="btn btn-primary mt-3" type="submit">Авторизация</button>  
 <p><a th:href="@{/include/registration}">Нет аккаунта? Зарегистрироваться</a></p>  
 </form>  
 </div>  
</main>  
  
<div th:replace="~{/include/footer}"></div>  
<script src="https://cdn.jsdelivr.net/npm/bootstrap@5.3.3/dist/js/bootstrap.bundle.min.js" integrity="sha384-ZGJQH6C7+C7DkOJgpFrw20FdLcdvrU3HkakK0ZAdPWJ/wVG51731EbPY6beOOA4F" crossorigin="anonymous"></script>  
</body>  
</html>

<!DOCTYPE html>  
<html lang="ru" xmlns:th="http://www.thymeleaf.org" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"  
 xsi:schemaLocation="http://www.thymeleaf.org ">  
<head>  
 <meta charset="UTF-8"/>  
 <meta name="viewport" content="width=device-width, initial-scale=1"/>  
 <title>Профиль пользователя</title>  
 <link href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.3/dist/css/bootstrap.min.css"  
 rel="stylesheet"  
 integrity="sha384-QWTKZyjpPEjISv5WaRU9OFeRpok6YctnYmDr5pNlyT2bRjXh0JMhjY6hW+ALEwIH"  
 crossorigin="anonymous"/>  
</head>  
<body class="d-flex flex-column min-vh-100">  
  
<div th:replace="~{/include/header}"></div>  
  
<main class="flex-grow-1">  
 <div class="container mt-5">  
 <div class="row">  
 *<!-- Левая колонка: информация пользователя -->* <div class="col-md-4 text-center">  
 <div class="text-center">  
 <img th:src="@{/img/default-avatar.png}"  
 class="rounded-circle img-fluid"  
 alt="Аватар пользователя"  
 style="width: 250px; height: 250px; object-fit: cover;"/>  
 </div>  
 <h2 class="mt-3" th:text="${user.name}">Имя пользователя</h2>  
 </div>  
  
 *<!-- Правая колонка: недвижимость и сделки -->* <div class="col-md-8">  
  
 *<!-- Недвижимость -->* <div class="card mb-4">  
 <div class="card-body">  
 <h5 class="card-title">Моя недвижимость</h5>  
 <ul th:if="${realtyList != null and !#lists.isEmpty(realtyList)}" class="list-group">  
 <li th:each="realty : ${realtyList}" class="list-group-item">  
 <span th:text="${realty.region + ', ' + realty.locality + ', ' + realty.street}"></span><br/>  
 <span>Добавлено: <span  
 th:text="${#temporals.format(realty.createdAt, 'dd.MM.yyyy')}"></span></span>  
 </li>  
 </ul>  
 <div th:if="${realtyList == null or #lists.isEmpty(realtyList)}">  
 <p>Недвижимость не найдена.</p>  
 </div>  
 <div class="mt-3">  
 <a href="/realty" class="btn btn-outline-secondary btn-sm">Подробнее...</a>  
 </div>  
 </div>  
 </div>  
  
 *<!-- Здесь можно добавить блок для сделок, если нужно -->* </div>  
 </div>  
 </div>  
</main>  
  
<script src="/webjars/bootstrap/5.3.3/js/bootstrap.bundle.min.js"></script>  
  
<div th:replace="~{/include/footer}"></div>  
  
</body>  
</html>

<!DOCTYPE html>  
<html xmlns:th="http://www.thymeleaf.org" lang="ru">  
<head>  
 <meta charset="UTF-8">  
 <title>Агентство недвижимости «TerraNova» — Регистрация</title>  
 <link href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.3/dist/css/bootstrap.min.css" rel="stylesheet" integrity="sha384-QWTKZyjpPEjISv5WaRU9OFeRpok6YctnYmDr5pNlyT2bRjXh0JMhjY6hW+ALEwIH" crossorigin="anonymous">  
</head>  
<body class="d-flex flex-column min-vh-100"> *<!-- d-flex flex-column min-vh-100 -->*<div th:replace="~{/include/header}"></div>  
  
<main class="flex-grow-1"> *<!-- flex-grow-1 -->* <div class="container mt-5">  
 <h2 class="mb-4">Регистрация</h2>  
 <form th:action="@{/include/registration/save}" method="post" th:object="${userDTO}" class="needs-validation" novalidate> *<!-- Путь изменен -->* <input type="hidden" th:name="${\_csrf.parameterName}" th:value="${\_csrf.token}"/>  
  
 <div class="form-group mb-3">  
 <label for="username">Логин</label>  
 <input type="text" class="form-control" th:field="\*{username}" id="username" required>  
 <div class="text-danger" th:if="${#fields.hasErrors('username')}" th:errors="\*{username}"></div>  
 </div>  
  
 <div class="form-group mb-3">  
 <label for="name">Имя</label>  
 <input type="text" class="form-control" th:field="\*{name}" id="name" required>  
 <div class="text-danger" th:if="${#fields.hasErrors('name')}" th:errors="\*{name}"></div>  
 </div>  
  
 <div class="form-group mb-3">  
 <label for="surname">Фамилия</label>  
 <input type="text" class="form-control" th:field="\*{surname}" id="surname" required>  
 <div class="text-danger" th:if="${#fields.hasErrors('surname')}" th:errors="\*{surname}"></div>  
 </div>  
  
 <div class="form-group mb-3">  
 <label for="password">Пароль</label>  
 <input type="password" class="form-control" th:field="\*{password}" id="password" required>  
 <div class="text-danger" th:if="${#fields.hasErrors('password')}" th:errors="\*{password}"></div>  
 </div>  
  
 <div class="form-group mb-3">  
 <label for="confirmPassword">Подтвердите пароль</label>  
 <input type="password" class="form-control" th:field="\*{confirmPassword}" id="confirmPassword" required>  
 <div class="text-danger" th:if="${#fields.hasErrors('confirmPassword')}" th:errors="\*{confirmPassword}"></div>  
 </div>  
  
 <button type="submit" class="btn btn-primary">Зарегистрироваться</button>  
 <a th:href="@{/include/login}" class="btn btn-secondary ms-2">Назад ко входу</a> *<!-- Изменили ссылку на логин -->* </form>  
 </div>  
</main>  
  
<div th:replace="~{/include/footer}"></div>  
<script src="https://cdn.jsdelivr.net/npm/bootstrap@5.3.3/dist/js/bootstrap.bundle.min.js" integrity="sha384-ZGJQH6C7+C7DkOJgpFrw20FdLcdvrU3HkakK0ZAdPWJ/wVG51731EbPY6beOOA4F" crossorigin="anonymous"></script>  
</body>  
</html>

<div th:fragment="sidebar" class="sidebar d-flex flex-column bg-light vh-100"  
 xmlns:th="http://www.thymeleaf.org"  
 xmlns:sec="http://www.thymeleaf.org/extras/spring-security">  
 <div class="px-3 mb-3 d-flex align-items-center border-bottom py-2">  
 <i class="bi bi-house-door-fill me-2 fs-5"></i>  
 <span class="fs-5 fw-semibold">TerraNova</span>  
 </div>  
 <nav class="nav flex-column px-3">  
 <a class="btn btn-outline-secondary mb-2 text-start" th:href="@{/}">  
 <i class="bi bi-house-door me-2"></i>Главная  
 </a>  
 <a class="btn btn-outline-secondary mb-2 text-start" th:href="@{/profile}">  
 <i class="bi bi-person-circle me-2"></i>Профиль  
 </a>  
 <a class="btn btn-outline-secondary mb-2 text-start" th:href="@{/realty}">  
 <i class="bi bi-geo-alt me-2"></i>Недвижимость  
 </a>  
 <a class="btn btn-outline-secondary mb-2 text-start" th:href="@{/deal}">  
 <i class="bi bi-file-earmark-text me-2"></i>Сделки  
 </a>  
  
 <a class="btn btn-outline-secondary mb-2 text-start" th:href="@{/client}"  
 sec:authorize="hasAnyRole('EMPLOYEE', 'ADMIN')">  
 <i class="bi bi-people me-2"></i>Клиенты  
 </a>  
  
 <a class="btn btn-outline-secondary mb-2 text-start" th:href="@{/employee}"  
 sec:authorize="hasRole('ADMIN')">  
 <i class="bi bi-person-workspace me-2"></i>Сотрудники  
 </a>  
  
 <a class="btn btn-outline-secondary mb-2 text-start" th:href="@{/user}"  
 sec:authorize="hasRole('ADMIN')">  
 <i class="bi bi-person-lines-fill me-2"></i>Пользователи  
 </a>  
  
 <a class="btn btn-outline-secondary mb-2 text-start" th:href="@{/about}">  
 <i class="bi bi-info-circle me-2"></i>О системе  
 </a>  
 <a href="#" class="btn btn-outline-secondary text-start"  
 onclick="event.preventDefault(); document.getElementById('logout-form').submit();">  
 <i class="bi bi-box-arrow-right me-2"></i>Выход  
 </a>  
  
 <form id="logout-form" th:action="@{/logout}" method="post" style="display:none;">  
 <input type="hidden" name="${\_csrf.parameterName}" value="${\_csrf.token}" />  
 </form>  
 </nav>  
</div>

<!DOCTYPE html>  
<html xmlns:th="http://www.thymeleaf.org" lang="ru">  
<head>  
 <meta charset="UTF-8" />  
 <meta name="viewport" content="width=device-width, initial-scale=1" />  
 <title>Агентство недвижимости «TerraNova» — Форма для недвижимости</title>  
 <link href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.3/dist/css/bootstrap.min.css" rel="stylesheet"  
 integrity="sha384-QWTKZyjpPEjISv5WaRU9OFeRpok6YctnYmDr5pNlyT2bRjXh0JMhjY6hW+ALEwIH" crossorigin="anonymous" />  
</head>  
<body class="d-flex flex-column min-vh-100">  
  
<div th:replace="~{/include/header}"></div>  
  
<main class="flex-grow-1">  
 <div class="container mt-5">  
 <h1 th:text="${#strings.isEmpty(realty.id)} ? 'Добавить недвижимость' : 'Редактировать недвижимость'"></h1>  
  
 <form th:action="@{/realty/save}" th:object="${realty}" method="post" novalidate>  
 <input type="hidden" th:field="\*{id}" />  
  
 <div class="mb-3">  
 <label for="region" class="form-label">Регион:</label>  
 <input type="text" class="form-control" id="region" th:field="\*{region}" required />  
 </div>  
  
 <div class="mb-3">  
 <label for="locality" class="form-label">Населённый пункт:</label>  
 <input type="text" class="form-control" id="locality" th:field="\*{locality}" required />  
 </div>  
  
 <div class="mb-3">  
 <label for="street" class="form-label">Улица:</label>  
 <input type="text" class="form-control" id="street" th:field="\*{street}" required />  
 </div>  
  
 <div class="mb-3">  
 <label for="rooms" class="form-label">Комнаты:</label>  
 <input type="number" class="form-control" id="rooms" th:field="\*{rooms}" required min="1" />  
 </div>  
  
 <div class="mb-3">  
 <label for="floors" class="form-label">Этажи:</label>  
 <input type="number" class="form-control" id="floors" th:field="\*{floors}" min="0" />  
 </div>  
  
 <div class="mb-3">  
 <label for="square" class="form-label">Площадь (м²):</label>  
 <input type="number" step="0.01" class="form-control" id="square" th:field="\*{square}" required min="0" />  
 </div>  
  
 <div class="mb-3">  
 <label for="repair" class="form-label">Тип ремонта:</label>  
 <input type="text" class="form-control" id="repair" th:field="\*{repair}" />  
 </div>  
  
 <div class="mb-3">  
 <label for="cost" class="form-label">Стоимость (₽):</label>  
 <input type="number" step="0.01" class="form-control" id="cost" th:field="\*{cost}" required min="0" />  
 </div>  
  
 *<!-- Чекбоксы -->* <div class="form-check">  
 <input class="form-check-input" type="checkbox" id="elevator" th:field="\*{elevator}" />  
 <label class="form-check-label" for="elevator">Лифт</label>  
 </div>  
 <div class="form-check">  
 <input class="form-check-input" type="checkbox" id="playground" th:field="\*{playground}" />  
 <label class="form-check-label" for="playground">Детская площадка</label>  
 </div>  
 <div class="form-check">  
 <input class="form-check-input" type="checkbox" id="trashChute" th:field="\*{trashChute}" />  
 <label class="form-check-label" for="trashChute">Мусоропровод</label>  
 </div>  
 <div class="form-check">  
 <input class="form-check-input" type="checkbox" id="parking" th:field="\*{parking}" />  
 <label class="form-check-label" for="parking">Парковка</label>  
 </div>  
 <div class="form-check mb-3">  
 <input class="form-check-input" type="checkbox" id="balcony" th:field="\*{balcony}" />  
 <label class="form-check-label" for="balcony">Балкон</label>  
 </div>  
  
 <div class="mb-3">  
 <label for="client" class="form-label">Клиент:</label>  
 <select class="form-select" id="client" th:field="\*{client}">  
 <option value="">-- Выберите клиента --</option>  
 <option th:each="client : ${clients}" th:value="${client.id}" th:text="${client.fullName}"></option>  
 </select>  
 </div>  
  
 <div class="mb-3">  
 <label for="owner" class="form-label">Владелец недвижимости:</label>  
 <select class="form-select" id="owner" th:field="\*{owner}" required>  
 <option value="">-- Выберите владельца --</option>  
 <option th:each="user : ${users}" th:value="${user.id}" th:text="${user.fullName}"></option>  
 </select>  
 </div>  
  
 <div class="mb-3">  
 <label for="deal" class="form-label">Сделка:</label>  
 <select class="form-select" id="deal" th:field="\*{deal}">  
 <option value="">-- Выберите сделку --</option>  
 <option th:each="deal : ${deals}" th:value="${deal.id}" th:text="${deal.id} + ' — ' + deal.formattedDate + ' — ' + deal.formattedCost"></option>  
 </select>  
 </div>  
  
 <button type="submit" class="btn btn-primary">Сохранить</button>  
 <a th:href="@{/realty}" class="btn btn-secondary ms-2">Отмена</a>  
 </form>  
 </div>  
</main>  
  
<script src="https://cdn.jsdelivr.net/npm/bootstrap@5.3.3/dist/js/bootstrap.bundle.min.js"  
 integrity="sha384-YvpcrYf0tY3lHB60NNkmXc5s9fDVZLESaAA55NDzOxhy9GkcIdslK1eN7N6jIeHz"  
 crossorigin="anonymous"></script>  
</body>  
</html>

<!DOCTYPE html>  
<html xmlns:th="http://www.thymeleaf.org"  
 xmlns:sec="http://www.thymeleaf.org/extras/spring-security"  
 lang="ru">  
<head>  
 <meta charset="UTF-8" />  
 <meta name="viewport" content="width=device-width, initial-scale=1" />  
 <title>Агентство недвижимости «TerraNova» — Недвижимость</title>  
 <link href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.3/dist/css/bootstrap.min.css" rel="stylesheet"  
 integrity="sha384-QWTKZyjpPEjISv5WaRU9OFeRpok6YctnYmDr5pNlyT2bRjXh0JMhjY6hW+ALEwIH" crossorigin="anonymous" />  
 <style>  
 body, html {  
 height: 100%;  
 margin: 0;  
 }  
 .sidebar {  
 width: 180px;  
 background-color: #f8f9fa;  
 border-right: 1px solid #ddd;  
 height: 100vh;  
 position: fixed;  
 padding-top: 1rem;  
 }  
 .sidebar .nav-link {  
 margin: 0.3rem 0;  
 }  
 .header {  
 height: 56px;  
 padding: 0 1rem;  
 background-color: #e9ecef;  
 border-bottom: 1px solid #ddd;  
 margin-left: 180px;  
 display: flex;  
 align-items: center;  
 justify-content: space-between;  
 font-weight: 500;  
 font-size: 1rem;  
 }  
 .header .btn {  
 min-width: 100px;  
 }  
 .content {  
 margin-left: 180px;  
 margin-top: 56px;  
 padding: 1rem;  
 }  
 .btn-edit {  
 background-color: #fff176;  
 border-color: #fff176;  
 color: #000;  
 }  
 .btn-edit:hover {  
 background-color: #f9f871;  
 border-color: #f9f871;  
 }  
 .btn-delete {  
 background-color: #f44336;  
 border-color: #f44336;  
 color: #fff;  
 }  
 .btn-delete:hover {  
 background-color: #d32f2f;  
 border-color: #d32f2f;  
 }  
 table thead th {  
 text-align: center;  
 vertical-align: middle;  
 }  
 table tbody td {  
 vertical-align: middle;  
 }  
 </style>  
</head>  
<body>  
  
*<!-- Боковое меню -->*<div th:replace="~{/include/sidebar :: sidebar}"></div>  
  
<header class="header">  
 <div>Агентство недвижимости «TerraNova» — Недвижимость</div>  
 <div class="d-flex gap-2">  
 *<!-- Кнопка "Добавить" -->* <div sec:authorize="hasAnyRole('EMPLOYEE','ADMIN')" class="align-self-center">  
 <a href="/realty/add" class="btn btn-primary">Добавить</a>  
 </div>  
 *<!-- Добавить и удалить всех: только для EMPLOYEE и ADMIN -->* <div sec:authorize="hasAnyRole('EMPLOYEE','ADMIN')" class="align-self-center">  
 <form th:action="@{/realty/delete}" method="post"  
 onsubmit="return confirm('Удалить весь список недвижимости?');">  
 <input type="hidden" th:name="${\_csrf.parameterName}" th:value="${\_csrf.token}" />  
 <button type="submit" class="btn btn-danger">Удалить всех</button>  
 </form>  
 </div>  
  
 *<!-- Выпадающая кнопка сортировки -->* <div class="dropdown">  
 <button class="btn btn-outline-info dropdown-toggle" type="button" id="sortDropdown"  
 data-bs-toggle="dropdown" aria-expanded="false">  
 Сортировка  
 </button>  
 <ul class="dropdown-menu" aria-labelledby="sortDropdown">  
 <li>  
 <a class="dropdown-item" th:href="@{/realty(sortField='cost', sortDir='asc')}"  
 th:classappend="${sortField == 'cost' and sortDir == 'asc'} ? ' active' : ''">Стоимость ↑</a>  
 </li>  
 <li>  
 <a class="dropdown-item" th:href="@{/realty(sortField='cost', sortDir='desc')}"  
 th:classappend="${sortField == 'cost' and sortDir == 'desc'} ? ' active' : ''">Стоимость ↓</a>  
 </li>  
 <li>  
 <a class="dropdown-item" th:href="@{/realty(sortField='region', sortDir='asc')}"  
 th:classappend="${sortField == 'region' and sortDir == 'asc'} ? ' active' : ''">Регион ↑</a>  
 </li>  
 <li>  
 <a class="dropdown-item" th:href="@{/realty(sortField='region', sortDir='desc')}"  
 th:classappend="${sortField == 'region' and sortDir == 'desc'} ? ' active' : ''">Регион ↓</a>  
 </li>  
 <li>  
 <a class="dropdown-item" th:href="@{/realty(sortField='street', sortDir='asc')}"  
 th:classappend="${sortField == 'street' and sortDir == 'asc'} ? ' active' : ''">Улица ↑</a>  
 </li>  
 <li>  
 <a class="dropdown-item" th:href="@{/realty(sortField='street', sortDir='desc')}"  
 th:classappend="${sortField == 'street' and sortDir == 'desc'} ? ' active' : ''">Улица ↓</a>  
 </li>  
 <li><hr class="dropdown-divider"></li>  
 <li>  
 <a class="dropdown-item" th:href="@{/realty}"  
 th:classappend="${sortField == null} ? ' active' : ''">Без сортировки</a>  
 </li>  
 </ul>  
 </div>  
 </div>  
</header>  
  
<main class="content">  
  
 <div th:if="${errorMessage}" class="alert alert-danger" th:text="${errorMessage}"></div>  
 <div th:if="${warningMessage}" class="alert alert-warning" th:text="${warningMessage}"></div>  
  
 <table class="table table-bordered table-hover table-striped align-middle text-center">  
 <thead class="table-light">  
 <tr>  
 <th>Регион</th>  
 <th>Улица</th>  
 <th>Площадь</th>  
 <th>Стоимость</th>  
 <th sec:authorize="hasAnyRole('EMPLOYEE','ADMIN')">Действия</th>  
 </tr>  
 </thead>  
 <tbody>  
 <tr th:each="realty : ${realties}">  
 <td th:text="${realty.region}"></td>  
 <td th:text="${realty.street}"></td>  
 <td th:text="${realty.square + ' м²'}"></td>  
 <td th:text="${realty.formattedCost}"></td>  
  
 <td class="text-center" sec:authorize="hasAnyRole('EMPLOYEE','ADMIN')">  
 <a th:href="@{/realty/edit/{id}(id=${realty.id})}" class="btn btn-edit btn-sm me-1">Изменить</a>  
 <form th:action="@{/realty/delete/{id}(id=${realty.id})}" method="post"  
 th:onsubmit="return confirm('Удалить эту запись?');" style="display: inline-block;">  
 <input type="hidden" th:name="${\_csrf.parameterName}" th:value="${\_csrf.token}" />  
 <button type="submit" class="btn btn-delete btn-sm">Удалить</button>  
 </form>  
 </td>  
 </tr>  
 <tr th:if="${#lists.isEmpty(realties)}">  
 <td colspan="5" class="text-center text-muted">Недвижимость не найдена.</td>  
 </tr>  
 </tbody>  
 </table>  
  
</main>  
  
<script src="https://cdn.jsdelivr.net/npm/bootstrap@5.3.3/dist/js/bootstrap.bundle.min.js"  
 integrity="sha384-YvpcrYf0tY3lHB60NNkmXc5s9fDVZLESaAA55NDzOxhy9GkcIdslK1eN7N6jIeHz"  
 crossorigin="anonymous"></script>  
</body>  
</html>

<!DOCTYPE html>  
<html lang="ru" xmlns:th="http://www.thymeleaf.org">  
<head>  
 <meta charset="UTF-8">  
 <title>Агентство недвижимости «TerraNova» — Редактирование пользователя</title>  
 <meta name="viewport" content="width=device-width, initial-scale=1, shrink-to-fit=no">  
 <link href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.3/dist/css/bootstrap.min.css" rel="stylesheet"  
 integrity="sha384-QWTKZyjpPEjISv5WaRU9OFeRpok6YctnYmDr5pNlyT2bRjXh0JMhjY6hW+ALEwIH" crossorigin="anonymous">  
 <style>  
 body, html {  
 height: 100%;  
 margin: 0;  
 }  
 .sidebar {  
 width: 180px;  
 background-color: #f8f9fa;  
 border-right: 1px solid #ddd;  
 height: 100vh;  
 position: fixed;  
 padding-top: 1rem;  
 }  
 .sidebar .nav-link {  
 margin: 0.3rem 0;  
 }  
 .header {  
 height: 56px;  
 padding: 0 1rem;  
 background-color: #e9ecef;  
 border-bottom: 1px solid #ddd;  
 margin-left: 180px;  
 display: flex;  
 align-items: center;  
 gap: 0.5rem;  
 font-weight: 500;  
 font-size: 1rem;  
 }  
 .header .btn {  
 min-width: 100px;  
 }  
 .content {  
 margin-left: 180px;  
 margin-top: 56px;  
 padding: 1rem;  
 }  
 </style>  
</head>  
<body>  
  
*<!-- Боковое меню -->*<div th:replace="~{/include/sidebar :: sidebar}"></div>  
  
<header class="header d-flex align-items-center justify-content-between">  
 <div>Агентство недвижимости «TerraNova» — Редактирование пользователя</div>  
</header>  
  
<main class="content">  
 <div class="container mt-3">  
 <h2 class="mb-4">Редактирование пользователя</h2>  
  
 <form th:action="@{/user/update}" th:object="${user}" method="post">  
 <input type="hidden" th:field="\*{id}" />  
 <div>  
 <label>Роли:</label>  
 <div th:each="role : ${allRoles}">  
 <input type="checkbox" name="roles"  
 th:value="${role.name()}"  
 th:checked="${user.roles.contains(role)}" />  
 <span th:text="${role.name()}"></span>  
 </div>  
 </div>  
 <button type="submit" class="btn btn-primary">Сохранить</button>  
 </form>  
 </div>  
</main>  
  
<script src="https://cdn.jsdelivr.net/npm/bootstrap@5.3.3/dist/js/bootstrap.bundle.min.js"  
 integrity="sha384-YvpcrYf0tY3lHB60NNkmXc5s9fDVZLESaAA55NDzOxhy9GkcIdslK1eN7N6jIeHz"  
 crossorigin="anonymous"></script>  
</body>  
</html>

<!DOCTYPE html>  
<html lang="ru" xmlns:th="http://www.thymeleaf.org" xmlns:sec="http://www.w3.org/1999/xhtml">  
<head>  
 <meta charset="UTF-8">  
 <meta name="viewport" content="width=device-width, initial-scale=1, shrink-to-fit=no">  
 <title>Агентство недвижимости «TerraNova» — Список пользователей</title>  
 <link href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.3/dist/css/bootstrap.min.css" rel="stylesheet"  
 integrity="sha384-QWTKZyjpPEjISv5WaRU9OFeRpok6YctnYmDr5pNlyT2bRjXh0JMhjY6hW+ALEwIH" crossorigin="anonymous">  
 <style>  
 body, html {  
 height: 100%;  
 margin: 0;  
 }  
 .sidebar {  
 width: 180px;  
 background-color: #f8f9fa;  
 border-right: 1px solid #ddd;  
 height: 100vh;  
 position: fixed;  
 padding-top: 1rem;  
 }  
 .sidebar .nav-link {  
 margin: 0.3rem 0;  
 }  
 .header {  
 height: 56px;  
 padding: 0 1rem;  
 background-color: #e9ecef;  
 border-bottom: 1px solid #ddd;  
 margin-left: 180px;  
 display: flex;  
 align-items: center;  
 gap: 0.5rem;  
 font-weight: 500;  
 font-size: 1rem;  
 }  
 .header .btn {  
 min-width: 100px;  
 }  
 .content {  
 margin-left: 180px;  
 margin-top: 56px;  
 padding: 1rem;  
 }  
 .btn-edit {  
 background-color: #fff176;  
 border-color: #fff176;  
 color: #000;  
 }  
 .btn-edit:hover {  
 background-color: #f9f871;  
 border-color: #f9f871;  
 }  
 table thead th {  
 text-align: center;  
 vertical-align: middle;  
 }  
 table tbody td {  
 vertical-align: middle;  
 }  
 </style>  
</head>  
<body>  
  
*<!-- Боковое меню -->*<div th:replace="~{/include/sidebar :: sidebar}"></div>  
  
<header class="header d-flex align-items-center justify-content-between">  
 <div>Агентство недвижимости «TerraNova» — Список пользователей</div>  
 <div class="d-flex gap-2">  
 <div class="dropdown">  
 <button class="btn btn-outline-info dropdown-toggle" type="button" id="sortDropdown"  
 data-bs-toggle="dropdown" aria-expanded="false">  
 Сортировка  
 </button>  
 <ul class="dropdown-menu" aria-labelledby="sortDropdown">  
 <li>  
 <a class="dropdown-item" th:href="@{/user(sort='username\_asc')}"  
 th:classappend="${sort?.equals('username\_asc')} ? ' active' : ''">  
 Псевдоним ↑  
 </a>  
 </li>  
 <li>  
 <a class="dropdown-item" th:href="@{/user(sort='username\_desc')}"  
 th:classappend="${sort?.equals('username\_desc')} ? ' active' : ''">  
 Псевдоним ↓  
 </a>  
 </li>  
 <li>  
 <a class="dropdown-item" th:href="@{/user(sort='name\_asc')}"  
 th:classappend="${sort?.equals('name\_asc')} ? ' active' : ''">  
 Имя ↑  
 </a>  
 </li>  
 <li>  
 <a class="dropdown-item" th:href="@{/user(sort='name\_desc')}"  
 th:classappend="${sort?.equals('name\_desc')} ? ' active' : ''">  
 Имя ↓  
 </a>  
 </li>  
 <li><hr class="dropdown-divider"></li>  
 <li>  
 <a class="dropdown-item" th:href="@{/user}"  
 th:classappend="${sort == null} ? ' active' : ''">  
 Без сортировки  
 </a>  
 </li>  
 </ul>  
 </div>  
 </div>  
</header>  
  
<main class="content">  
 <div th:if="${ErrorMessage}" class="alert alert-warning alert-dismissible fade show" role="alert">  
 <span th:text="${ErrorMessage}"></span>  
 <button type="button" class="btn-close" data-bs-dismiss="alert" aria-label="Close"></button>  
 </div>  
  
 <table class="table table-bordered table-hover table-striped">  
 <thead class="table-light">  
 <tr>  
 <th>ID</th>  
 <th>Псевдоним</th>  
 <th>Роли</th>  
 <th>Действия</th>  
 </tr>  
 </thead>  
 <tbody>  
 <tr th:each="user : ${users}">  
 <td th:text="${user.id}"></td>  
 <td th:text="${user.username}"></td>  
 <td>  
 <span th:each="role : ${user.roles}" class="badge bg-secondary me-1" th:text="${role.name}"></span>  
 </td>  
 <td class="text-center">  
 <a th:href="@{'/user/edit/' + ${user.id}}" class="btn btn-edit btn-sm">Редактировать</a>  
 </td>  
 </tr>  
 <tr th:if="${#lists.isEmpty(users)}">  
 <td colspan="4" class="text-center text-muted">Пользователи отсутствуют.</td>  
 </tr>  
 </tbody>  
 </table>  
</main>  
  
<script src="https://cdn.jsdelivr.net/npm/bootstrap@5.3.3/dist/js/bootstrap.bundle.min.js"  
 integrity="sha384-YvpcrYf0tY3lHB60NNkmXc5s9fDVZLESaAA55NDzOxhy9GkcIdslK1eN7N6jIeHz"  
 crossorigin="anonymous"></script>  
</body>  
</html>