**Частное учреждение**

**профессиональная образовательная организация**

**ТЕХНИКУМ «БИЗНЕС И ПРАВО»**

**ОТЧЕТ О ПРОИЗВОДСТВЕННОЙ ПРАКТИКЕ ПО**

**ПМ 05 ПРОЕКТИРОВАНИЕ И РАЗРАБОТКА ИНФОРМАЦИОННЫХ СИСТЕМ**

*практика по профилю специальности*

* \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

*наименование организации*

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

в период с «09» июня 2025 г. по «05» июля 2025 г.

|  |  |
| --- | --- |
|  | обучающегося (йся)\_\_\_\_ курса очной формы обучения, группы \_\_\_\_\_\_\_\_\_\_\_\_  специальности 09.02.07 Информационные системы и программирование  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  *фамилия, имя, отчество*  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  Оценка \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  Руководитель практики  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  *фамилия, имя, отчество*  Дата\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ |

г.Белореченск

СОДЕРЖАНИЕ

|  |  |
| --- | --- |
| ВВЕДЕНИЕ | 3 |
| 1 ПРОЕКТИРОВАНИЕ ИНФОРМАЦИОННОЙ СИСТЕМЫ | 5 |
| 1.1 Анализ предметной области | 5 |
| 1.2 Техническое задание | 6 |
| 1.3 Эскизный проект | 25 |
| 2 РАЗРАБОТКА И ДОКУМЕНТИРОВАНИЕ ИНФОРМАЦИОННОЙ СИСТЕМЫ | 33 |
| 2.1 Тестирование клиентской части | 33 |
| 2.2 Руководство пользователя | 38 |
| 2.3 Описание страниц | 43 |
| ЗАКЛЮЧЕНИЕ | 53 |
| СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ | 54 |
| ПРИЛОЖЕНИЕ |  |

ВВЕДЕНИЕ

В современную эпоху цифровых технологий автоматизация процессов в различных сферах деятельности становится не просто преимуществом, а необходимостью для повышения эффективности, точности и конкурентоспособности. Здравоохранение не является исключением из этого правила. Частная клиника «DoctorPlus» стремится к тому, чтобы предоставить своим пациентам высококачественные медицинские услуги, обеспечить комфортное обслуживание и улучшить взаимодействие между персоналом и клиентами. Для достижения этих целей возникла необходимость разработки и внедрения информационной системы, способной автоматизировать ключевые бизнес-процессы клиники: учет пациентов, расписание врачей, проведение медицинских услуг, формирование отчетности и взаимодействие с сотрудниками.

Объектом исследования является частная клиника «DoctorPlus», её организационная структура, бизнес-процессы и информационные потоки.

Предметом исследования выступает разработка и внедрение информационной системы для частной клиники «DoctorPlus», предназначенной для автоматизации процессов учета пациентов, назначения приёмов, оказания медицинских услуг, управления персоналом и формирования отчётности.

Целью исследования является создание информационной системы, которая позволит повысить эффективность работы частной клиники «DoctorPlus» за счёт автоматизации ключевых бизнес-процессов, с учётом требований заказчика и особенностей функционирования медицинского учреждения.

Для достижения поставленной цели были определены следующие задачи исследования:

- Провести анализ организационной структуры, бизнес-процессов и информационных потоков частной клиники «DoctorPlus»;

- Выявить требования заказчика к функциональности и интерфейсу информационной системы;

- Разработать концептуальную модель информационной системы;

- Спроектировать архитектуру информационной системы;

- Разработать базу данных информационной системы;

- Создать модули системы: учет пациентов, расписание врача, медицинские услуги, учет сотрудников, отчеты;

- Выполнить тестирование и отладку информационной системы;

- Осуществить внедрение информационной системы в работу частной клиники «DoctorPlus»;

Подготовить техническую и пользовательскую документацию к информационной системе.

Для решения указанных задач были использованы следующие методы исследования:

- Анализ внутренних документов и регламентов клиники «DoctorPlus»;

- Интервьюирование сотрудников клиники для выявления текущих бизнес-процессов и проблемных зон;

­- Моделирование бизнес-процессов и информационных потоков с использованием специализированного программного обеспечения;

- Разработка концептуальной модели и архитектуры системы с применением UML-диаграмм;

- Проектирование и реализация базы данных с помощью SQL;

- Разработка программных модулей системы на языке Java;

- Тестирование и отладка системы с использованием инструментов автоматизированного тестирования;

- Внедрение системы в реальные условия клиники с последующей адаптацией под конкретные нужды;

Подготовка технической документации в соответствии с нормативными требованиями и запросами заказчика.

1 ПРОЕКТИРОВАНИЕ ИНФОРМАЦИОННОЙ СИСТЕМЫ

1.1 Анализ предметной области

Агентство недвижимости «DoctorPlus»

Частная клиника «DoctorPlus» предоставляет широкий спектр медицинских услуг населению: от профилактических осмотров до амбулаторного лечения и диагностики. Основными направлениями деятельности клиники являются приём пациентов, проведение диагностических исследований, оказание платных медицинских услуг и ведение учёта медицинской деятельности. Клиника работает как с физическими лицами - пациентами, так и с юридическими - организациями, заключающими договоры на обслуживание своих сотрудников.

Каждый пациент может обратиться за различными услугами, записаться на приём к врачу, пройти обследование и получить рекомендации. Врачи проводят первичный осмотр, ставят диагноз, назначают лечение и могут направить пациента на дополнительные исследования. Для оценки эффективности работы клиники, а также для соблюдения законодательных требований, клиника формирует отчетность для органов здравоохранения и налоговых инспекций.

Администратор системы активно взаимодействует с пациентами, регистрирует обращения, консультирует по вопросам записи, предоставляемых услуг и стоимости приёма. Также в его обязанности входит учет обращений пациентов - как устных, так и письменных, и контроль выполнения заявок.

Для повышения эффективности работы клиники и снижения трудоемкости рутинных операций разрабатывается информационная система, которая позволит автоматизировать процессы регистрации пациентов, назначения приёмов, ведения истории болезни, учёта оказанных услуг, управления расписанием врачей, а также формирования отчётности.

Виды запросов в информационной системе:

1. Показать список всех пациентов

2. Показать пациентов, записанных на текущий день

3. Показать перечень доступных услуг

4. Показать все типы медицинских услуг

5. Показать врача по пациенту

6. Показать все категории сотрудников (врачи, медсестры, администраторы)

7. Показать информацию о конкретном сотруднике

8. Показать список записей пациента

9. Показать сведения о пациентах

10. Показать всех сотрудников клиники

11. Показать список пациентов, записанных к определённому врачу

12. Показать количество свободных записей на текущий день

13. Показать пациентов с определённым диагнозом

14. Показать услуги определённой стоимости или категории

15. Показать историю обращений конкретного пациента

16. Реализовать добавление, редактирование, поиск, фильтрацию, удаление информации о пациентах

17. Реализовать добавление, редактирование, поиск, фильтрацию, удаление информации о медицинских услугах

18. Реализовать добавление, редактирование, поиск, фильтрацию, удаление информации о записях и приёмах

19. Реализовать добавление, редактирование, поиск, фильтрацию, удаление информации о диагнозах и лечении

20. Реализовать добавление, редактирование, поиск, фильтрацию, удаление информации о сотрудниках

1.2 Техническое задание

1 Общие сведенья

1.1 Полное наименование программы и ее условное обозначение

Полное наименование системы: Автоматизирования информационная система частной клинике «DoctorPlus»

Краткое наименование системы: АИС «DoctorPlus»

1.2 Шифр темы или шифр (номер) договора

Шифр темы: АИС-ГК-20

Номер договора: №2/34-56-78-002 от 9.06.2025.

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 частной клиники «DoctorPlus». Она позволит оптимизировать процессы учета пациентов, управления расписанием врачей, оказания медицинских услуг и взаимодействия с клиентами. Система будет предоставлять информацию о пациентах, их диагнозах, назначенных процедурах, истории обращений, а также данные о предоставляемых медицинских услугах и квалификации сотрудников.

Цели создания информационной системы:

Упрощение работы административного персонала и врачей за счет автоматизации рутинных операций

Повышение качества обслуживания пациентов через ускорение записи, выдачи результатов и ведения медицинской документации

Повышение общей эффективности функционирования клиники за счет системного подхода к управлению данными

Задачи информационной системы:

Ведение электронной базы данных пациентов с возможностью хранения истории обращений и назначений

Автоматизация процесса записи на приём к врачу и формирования расписания специалистов

Учет оказанных медицинских услуг и формирование финансовой отчетности

Контроль выполнения назначений, анализ повторных обращений и состояния здоровья пациентов

Анализ данных по обращениям, популярным услугам и загруженности врачей для планирования работы клиники

3 Характеристика объектов автоматизации

В ходе разработки информационной системы для частной клиники «DoctorPlus» будут автоматизированы ключевые процессы учета: пациентов, медицинских услуг, сотрудников, записей на прием и историй болезней.

Автоматизируемыми процессами станут следующие операции: регистрация пациентов, оформление записей на прием, ведение истории болезни, учет оказанных услуг, начисление заработной платы сотрудникам, формирование отчетности.

Система обеспечит возможность быстрого доступа к информации о пациентах, назначенных услугах, графиках работы врачей и статусе обращений, что позволит:

Упростить работу администраторов и врачей

Снизить вероятность ошибок при вводе и обработке данных

Повысить качество и скорость обслуживания пациентов

При внедрении системы будет достигнуто:

Упрощение процессов регистрации и записи пациентов

Повышение эффективности работы персонала за счет централизованного хранения данных

Увеличение удовлетворенности пациентов благодаря прозрачности и скорости обслуживания

Объекты автоматизации характеризуются:

Необходимостью доступа к базе данных пациентов, услуг, сотрудников и записей

Возможностью быстрого получения информации для принятия решений (например, загруженность врачей, наличие свободных мест)

Потребностью в постоянном доступе к системе через интернет или локальную сеть клиники

4 Требования к системе

4.1 Требования к структуре и функционированию системы

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

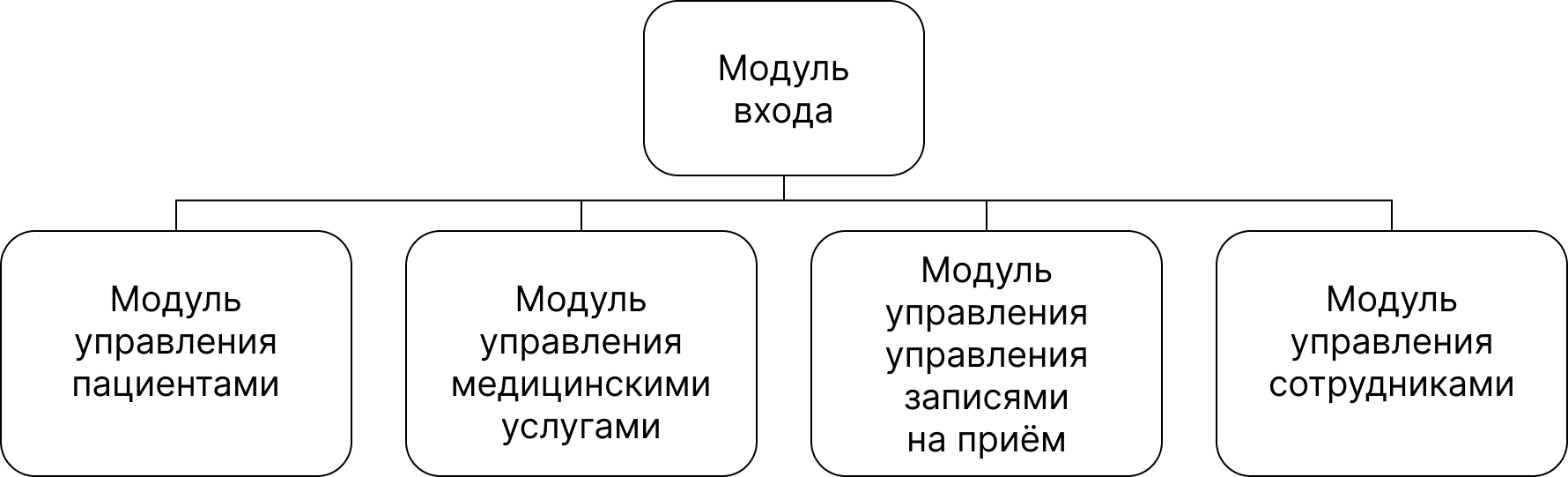


Рисунок 1 - Модули разрабатываемой системы

Серверная часть:

- Модуль авторизации

- Модуль управления пациентами

- Модуль управления медицинскими услугами

- Модуль управления записями на прием

- 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:

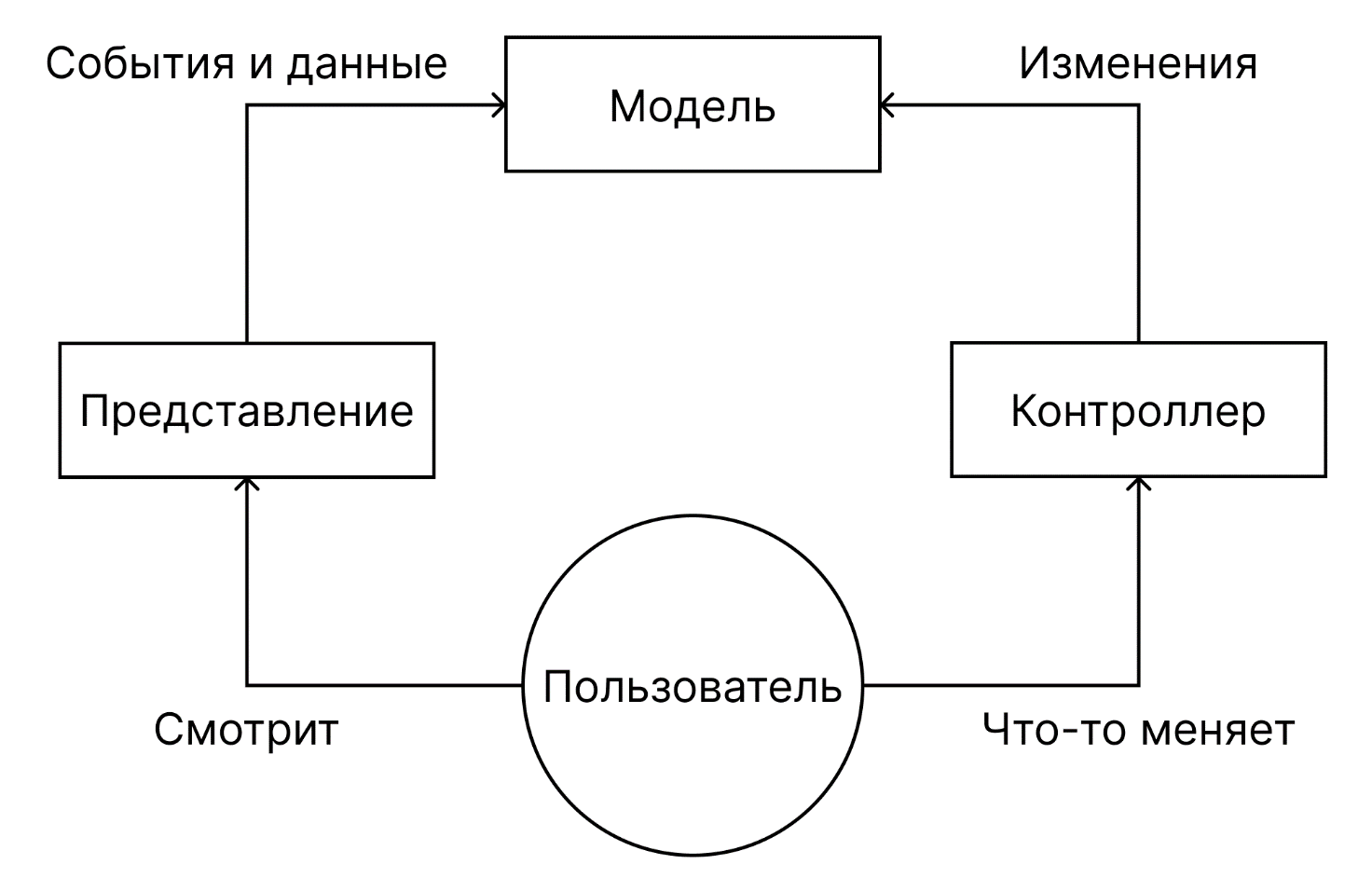


Рисунок 2 – Структура **Model-View-Controller** архитектуры

4.2 Требования к видам обеспечения

Для функционирования информационной системы частной клиники «DoctorPlus», работа клиента и сервера может быть организована следующим образом:

Клиентская часть:

На компьютерах сотрудников клиники требуется наличие сервера и клиентских компьютеров, на которых будет производиться 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:

Администраторы: управление пациентами, врачами, расписанием, финансовыми отчетами.

Врачи: просмотр истории болезни пациентов, назначение диагностики и лечения, оформление медицинских документов.

Пациенты: поиск врачей, запись на приём, просмотр результатов исследований и истории обращений.

Серверная часть:

Сервер базы данных будет хранить всю информацию о пациентах, врачах, услугах, записях на приём и финансовых операциях.

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 клиники. Приложение будет иметь графический интерфейс и функционал в зависимости от роли пользователя.

База данных будет содержать информацию о пациентах, врачах, услугах, записях на приём и финансовых операциях.

При анализе предметной области частной клиники «DoctorPlus» были выделены следующие сущности, на основе которых будет разрабатываться база данных системы:

- Пациент

- Врач

- Услуга

- Запись на приём

База данных информационной системы должна состоять из следующих связанных таблиц:

Patient - таблица, содержащая информацию о пациентах

Doctor - таблица, содержащая информацию о врачах

Serving - таблица, содержащая информацию о медицинских услугах

Appointment - таблица, содержащая информацию о записях на приём

На таблицах 1-4 предоставлены эскизы страниц сайта:

Таблица 1 - Таблица Patient

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Имя поля | Тип | Размер | Комментарий | Ограничение |
| id | INT | auto\_increment | Идентификатор пациента | Первичный ключ |
| full\_name | VARCHAR | 255 | Полное имя пациента | NOT NULL |
| birth | VARCHAR |  | Дата рождения | NOT NULL |
| phone | VARCHAR | 20 | Номер телефона | NOT NULL |
| email | VARCHAR | 255 | Электронная почта | NOT NULL |
| address | VARCHAR |  | Адрес проживания | NOT NULL |

Таблица 2 - Таблица Doctor

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Имя поля | Тип | Размер | Комментарий | Ограничение |
| id | INT | auto\_increment | Идентификатор врача | Первичный ключ |
| full\_name | VARCHAR | 255 | Полное имя врача | NOT NULL |
| speciality | VARCHAR | 255 | Специализация | NOT NULL |
| schedule | VARCHAR |  | Расписание работы | NOT NULL |

Таблица 3 - Таблица Serving

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Имя поля | Тип | Размер | Комментарий | Ограничение |
| id | INT | auto\_increment | Идентификатор услуги | Первичный ключ |
| name | VARCHAR | 255 | Название услуги | NOT NULL |
| description | TEXT |  | Описание услуги | NOT NULL |
| cost | DECIMAL | (10, 2) | Стоимость услуги | NOT NULL |

Таблица 4 - Таблица Appointment

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Имя поля | Тип | Размер | Комментарий | Ограничение |
| id | INT | auto\_increment | Идентификатор записи | Первичный ключ |
| patient\_id | INT |  | ID пациента | Внешний ключ (Patient.id) |
| doctor\_id | INT |  | ID врача | Внешний ключ (Doctor.id) |
| service\_id | INT |  | ID услуги | Внешний ключ (Service.id) |
| date | DATETIME |  | Дата и время записи | NOT NULL |
| status | VARCHAR | 50 | Статус записи (активна/отменена) | NOT NULL |

4.3 Требования к эргономике и технической эстетике

Общие требования к интерфейсу системы:

Фоновый цвет - белый, что обеспечивает чистоту и легкую восприимчивость информации.

При наведении указателя мыши на элементы кнопок, они должны визуально выделяться, например, изменением цвета фона или появлением тени.

Шрифты и размеры текста - четкие, легко читаемые (например, Roboto, Open Sans), с оптимальным размером для комфортного восприятия информации.

Адаптивный интерфейс, корректно отображающийся как на стационарных компьютерах, так и на мобильных устройствах.

Верхнее меню должно содержать следующие пункты:  
(Пациенты, Услуги, Запись на приём, Сотрудники, О системе, Выход) . Это обеспечивает удобный и интуитивно понятный доступ к основным функциям системы.

Все кнопки в интерфейсе должны иметь одинаковую высоту, чтобы сохранять единый стиль и порядок.

Элементы пользовательского интерфейса:

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

При фокусировке на строке таблицы она выделяется цветом, что помогает пользователю видеть активную запись.

Кнопки имеют закругленные углы, что придает им современный и привлекательный внешний вид, соответствующий дизайну медицинской сферы.

Экран авторизации:

Все элементы страницы должны быть отцентрированы по вертикали и горизонтали, чтобы обеспечить эстетичный и сбалансированный дизайн.

Заголовок страницы — «Авторизация в системе DoctorPlus».

Над полем ввода логина и пароля должен быть расположен заголовок или логотип клиники для усиления брендинга.

Элементы ввода текста (логин и пароль) и кнопка «Войти» должны иметь одинаковую ширину для соблюдения визуального порядка.

Кнопка «Войти»:

- Цвет - светло-серый.

- Цвет шрифта - черный.

- При наведении курсора на кнопку ее цвет становится немного светлее, создавая эффект наведения и улучшая взаимодействие.

4.4 Требования к содержанию и оформлению выводимых сообщений

Требования к интерфейсу и обработке ошибок

При вводе некорректных данных должно отображаться всплывающее уведомление с описанием ошибки. Уведомление автоматически исчезает, как только пользователь начинает заново вводить данные в соответствующее поле.

Если пользователь вводит неверные учетные данные (логин или пароль) и нажимает кнопку «Войти», система выводит окно с сообщением:  
«Неверный логин или пароль. Попробуйте заново»

Система должна отображать сообщения об ошибках при вводе данных, не соответствующих формату. Например:  
«Поле должно содержать только русские буквы и начинаться с заглавной»

Все всплывающие подсказки должны исчезать автоматически, как только пользователь начинает повторно вводить данные в соответствующее поле.

В случае, если обязательные поля формы не заполнены , система выводит предупреждение:  
«Пожалуйста, заполните все обязательные поля»

При попытке добавления, редактирования или удаления данных в случае возникновения ошибки, система выводит соответствующее уведомление, например:  
«Ошибка сохранения данных. Проверьте корректность введенной информации»

При проблемах с сервером или сетью система выводит сообщение:  
«Произошла ошибка соединения. Пожалуйста, проверьте интернет и попробуйте снова»

4.5 Требования к видам обеспечения

Для стабильного функционирования информационной системы частной клиники «DoctorPlus» необходимы сервер и клиентские устройства, на которых будет производиться запуск системы.

Требования к серверу базы данных:

- Операционная система: Windows 10 и выше / Linux (Ubuntu Server 20.04 и выше)

- Процессор: минимум 2 ядра (8 логических потоков), тактовая частота - 2 ГГц и выше

- Оперативная память: 8 ГБ и более

- Свободное дисковое пространство: не менее 2 ГБ

- Жесткий диск: SATA2, скорость вращения 7200 об/мин или SSD

- Сервер БД: PostgreSQL / MySQL / MariaDB

- Поддержка TCP/IP для клиент-серверного взаимодействия

Требования к клиентским устройствам:

- Операционная система: Windows 10 и выше, macOS 11 Big Sur и выше, Linux (Ubuntu 20.04 и выше)

- Браузеры: Google Chrome 131.0.6778.69 и выше, Microsoft Edge последней версии, Firefox 120+, Opera 115.0.5322.109 и выше

- Оперативная память: не менее 4 ГБ

- Разрешение экрана: рекомендуется Full HD (1920x1080) и выше для комфортной работы с интерфейсом

4.6 Эскизы основных окон системы

На рисунках 3-9 предоставлены эскизы страниц сайта:

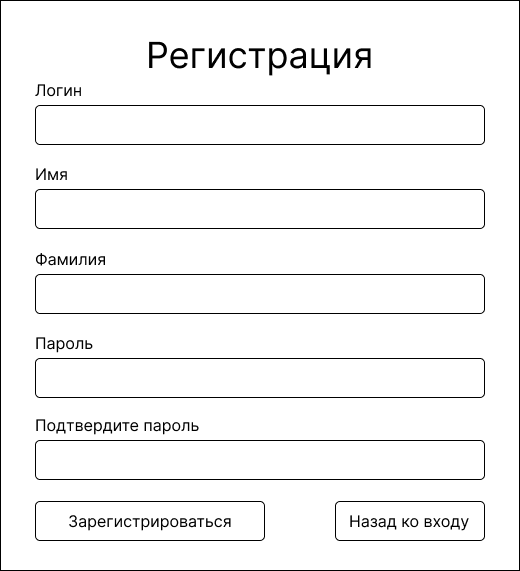


Рисунок 3 - Страница регистрации

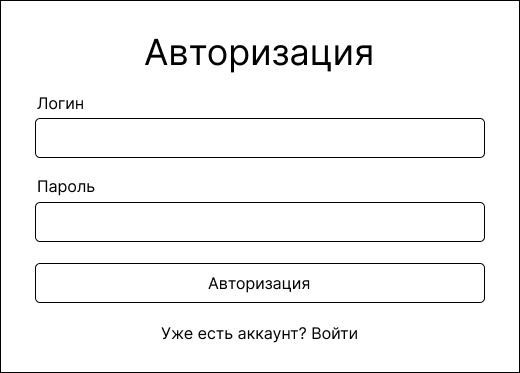


Рисунок 4 - Страница авторизации

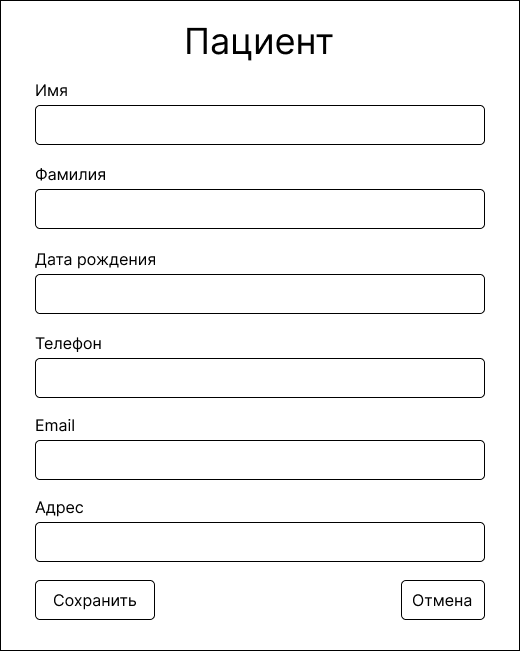


Рисунок 5 - Страница добавления пациента

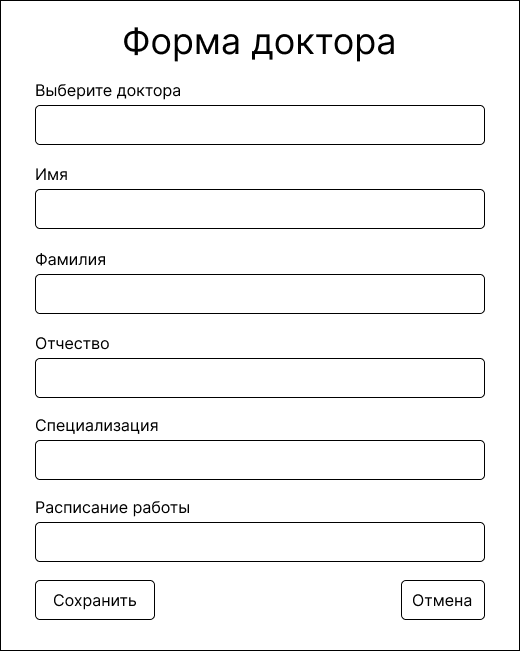


Рисунок 6 - Страница добавления доктора



Рисунок 7 - Страница добавления медицинской услуги

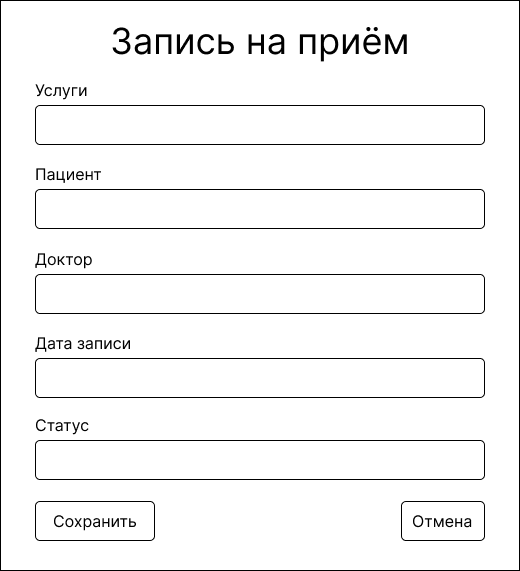


Рисунок 8 - Страница добавления записи на прием

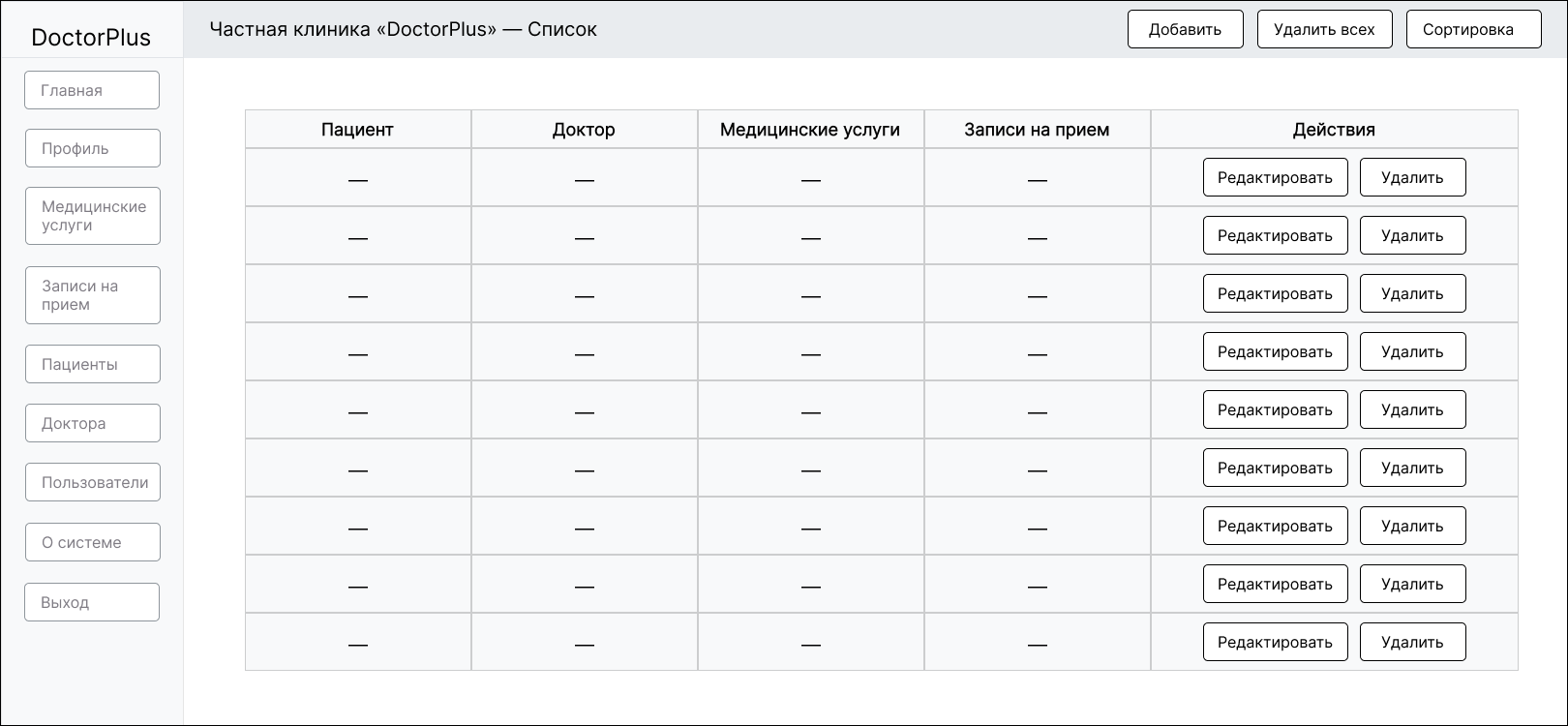


Рисунок 9 - Страница список

5. Состав и содержание работ по созданию системы

Комплекс работ по созданию АИС для частной клиники «DoctorPlus» должен осуществляться в несколько этапов:

1. Анализ требований:

- Сбор и анализ требований со стороны клиники, сотрудников и пациентов.

- Определение функциональных и нефункциональных требований к системе.

- Изучение текущих бизнес-процессов клиники: запись на приём, работа с пациентами, оказание услуг, формирование отчетности.

- Выявление проблемных зон в существующей организации работы.

2. Проектирование системы:

- Разработка концептуальной модели системы.

- Проектирование архитектуры системы (на основе MVC или другой подходящей парадигмы).

- Проектирование базы данных для хранения информации о пациентах, врачах, услугах, записях на приём и т.д.

- Проектирование пользовательского интерфейса и навигации для разных ролей (администратор, врач, пациент).

3. Разработка системы:

- Создание серверной части системы (Backend) — обработка запросов, логика работы с данными.

- Разработка клиентской части системы (Frontend) — реализация графического интерфейса.

- Реализация модулей: пациенты, услуги, запись на приём, сотрудники, история болезни.

- Интеграция модулей между собой и обеспечение корректного взаимодействия.

4. Тестирование системы:

- Модульное тестирование отдельных компонентов (например, формы добавления пациента, таблица записей).

- Интеграционное тестирование — проверка взаимодействия всех модулей между собой.

- Системное тестирование — проверка полного цикла использования системы.

- Тестирование безопасности, производительности и удобства интерфейса.

5. Развертывание и внедрение системы:

- Подготовка серверной и клиентской инфраструктуры.

- Установка и настройка системы на серверах клиники и на рабочих местах персонала.

- Перенос данных из старых систем (если применимо).

- Обучение персонала работе с новой системой.

6. Опытная эксплуатация и доработка системы:

- Проведение опытной эксплуатации в реальных условиях клиники.

- Сбор обратной связи от пользователей.

- Устранение выявленных недочетов и ошибок.

- Доработка системы согласно замечаниям и пожеланиям.

7. Ввод системы в промышленную эксплуатацию:

- Окончательный переход на использование новой системы.

- Отключение устаревших решений (при наличии).

- Финальная настройка и оптимизация работы системы.

8. Сопровождение и развитие системы:

- Оказание технической поддержки пользователям.

- Регулярные обновления системы в соответствии с изменяющимися требованиями законодательства и внутренними нуждами клиники.

- Добавление новых функций и улучшение пользовательского опыта по мере необходимости.

9. Приемка работ:

- Проведение предварительных испытаний системы на тестовой среде разработчика.

- Демонстрация системы заказчику.

- Подписание акта выполненных работ после успешного прохождения тестирования и опытной эксплуатации.

6. Порядок контроля и приемки системы

- Система передается в виде полностью функционирующего программного комплекса, установленного на вычислительной технике заказчика и исполнителя, в сроки, предусмотренные договором.

- Приемка системы проводится комиссией, в состав которой входят уполномоченные представители заказчика и исполнителя.

В рамках приемки:

- Проверяется соответствие системы заявленным требованиям.

- Проводится демонстрация всех ключевых функциональных возможностей.

- Проверяется корректность работы системы в различных сценариях использования.

- Полный комплект документации по информационной системе, разработанной исполнителем, передается заказчику:

- На бумажных носителях (в двух экземплярах).

- На USB-носителе (флеш-накопителе), один экземпляр которого после подписания акта приемки передается исполнителю.

- После завершения приемки оформляется акт сдачи-приемки выполненных работ, который подписывается обеими сторонами.

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 Автоматизированная информационная система «DoctorPlus» предназначена для комплексного информационно-управленческого обеспечения процессов в сфере медицинских услуг и управления частной клиникой.

Комплексное информационно-управленческое обеспечение процессов в частной клинике включает:

- Управление пациентами и их записью на приём

- Ведение истории болезней и результатов обследований

- Организация работы врачей и расписания приёмов

- Учет оказываемых медицинских услуг и финансовых операций

- Формирование отчетности по работе клиники

- Информационная система «DoctorPlus» разработана с целью повышения эффективности, качества обслуживания пациентов и оптимизации внутренних процессов клиники.

Система позволит централизованно хранить и обрабатывать информацию о пациентах, врачах, предоставляемых услугах и финансовых операциях. Для сотрудников клиники система обеспечит удобный доступ ко всей необходимой информации и инструментам управления. Основные цели внедрения системы:

- Повышение эффективности работы персонала клиники

- Улучшение качества обслуживания пациентов

- Автоматизация рутинных процессов (запись на приём, история болезни, отчетность)

- Повышение прозрачности и точности учета данных

- Достижение поставленных целей направлено на увеличение удовлетворенности пациентов, снижение административной нагрузки на персонал и повышение общей конкурентоспособности частной клиники за счет цифровизации ключевых бизнес-процессов.

Для достижения этих целей информационная система должна решать следующие основные задачи:

- Ведение электронной базы данных пациентов с возможностью хранения истории обращений и назначений

- Обеспечение записи на приём к врачу с возможностью выбора специалиста, даты и времени

- Учет предоставляемых медицинских услуг и формирование финансовой отчетности

- Ведение расписания врачей и контроль загруженности

- Формирование статистических отчетов по количеству пациентов, популярным услугам, доходам и т.д.

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

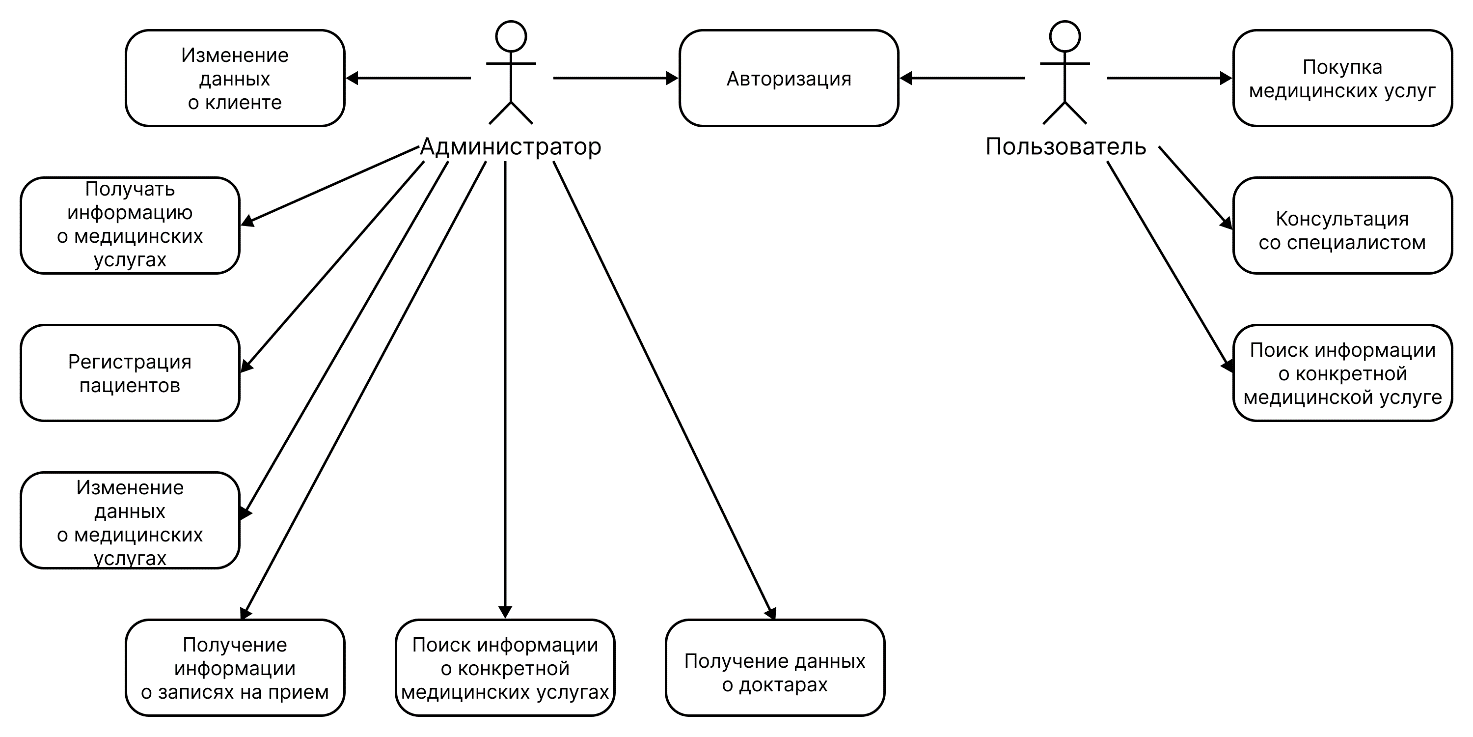


Рисунок 10 - Диаграмма варианта использования

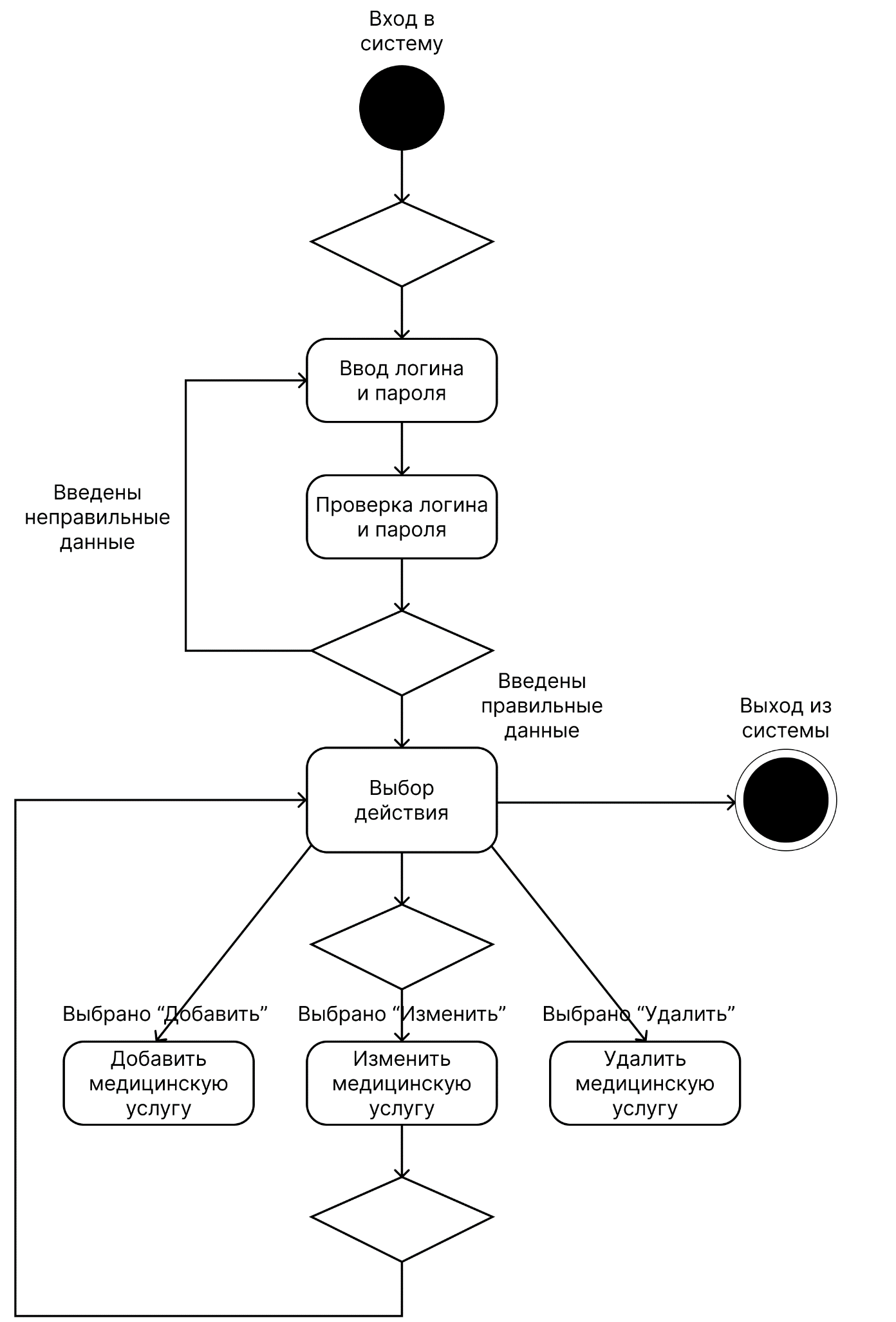


Рисунок 11 - Диаграмма состояния

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.

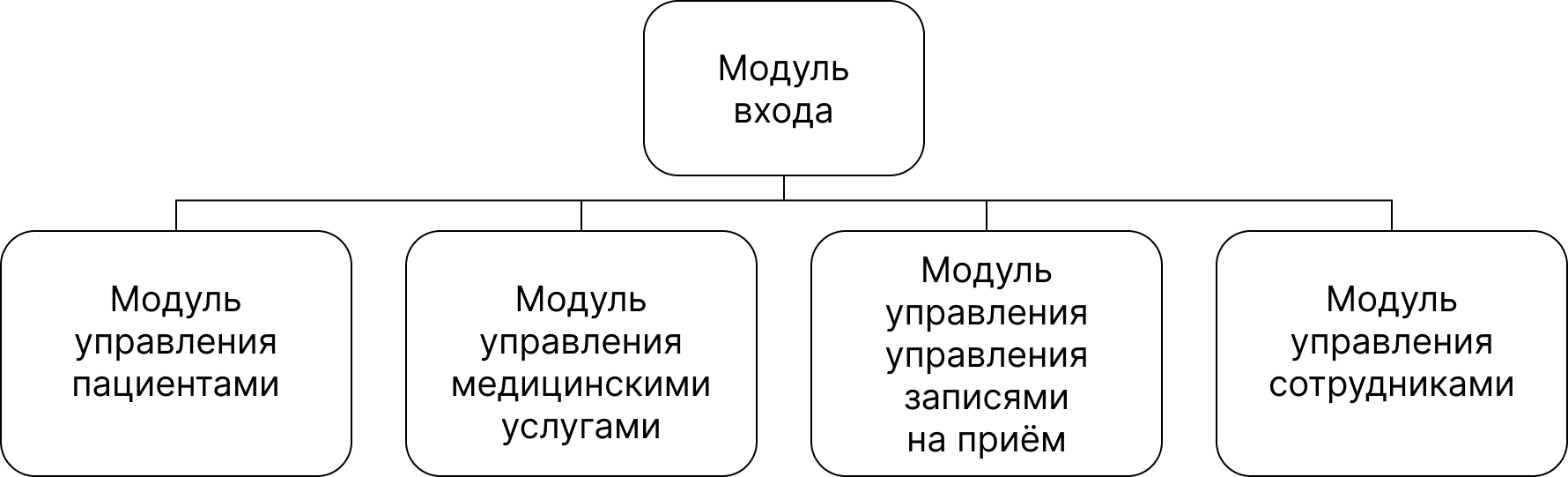


Рисунок 12 - Модули разрабатываемой системы

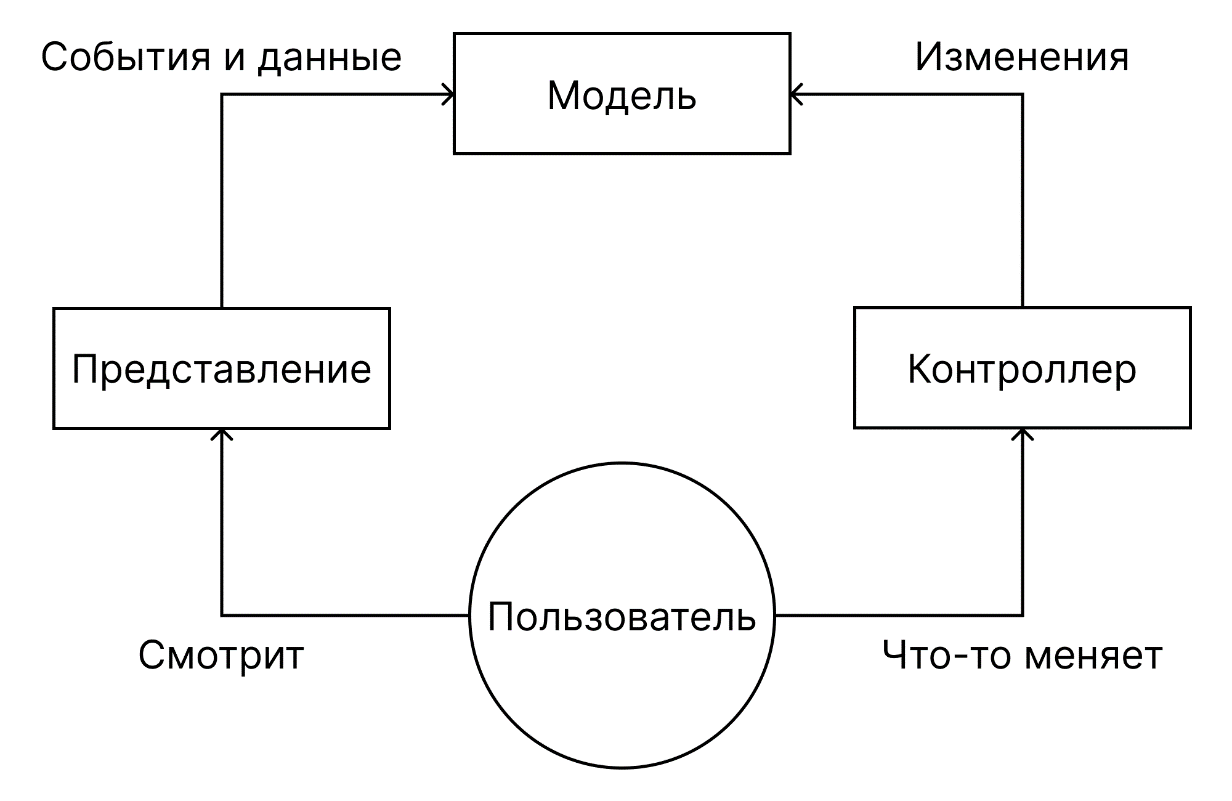


Рисунок 13 - Структура **Model-View-Controller** архитектуры

Серверная часть:

- Модуль авторизации

- Модуль управления медицинскими услугами

- Модуль управления пациентами

- Модуль управления записями на прием

- Модуль управления докторами

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

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:

Patient - таблица, содержащая информацию о пациентах

Doctor - таблица, содержащая информацию о врачах

Serving - таблица, содержащая информацию о медицинских услугах

Appointment - таблица, содержащая информацию о записях на приём

На таблицах 6-9 предоставлены эскизы страниц сайта:

Таблица 6 - Таблица Patient

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Имя поля | Тип | Размер | Комментарий | Ограничение |
| id | INT | auto\_increment | Идентификатор пациента | Первичный ключ |
| full\_name | VARCHAR | 255 | Полное имя пациента | NOT NULL |
| birth | VARCHAR |  | Дата рождения | NOT NULL |
| phone | VARCHAR | 20 | Номер телефона | NOT NULL |
| email | VARCHAR | 255 | Электронная почта | NOT NULL |
| address | VARCHAR |  | Адрес проживания | NOT NULL |

Таблица 7 - Таблица Doctor

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Имя поля | Тип | Размер | Комментарий | Ограничение |
| id | INT | auto\_increment | Идентификатор врача | Первичный ключ |
| full\_name | VARCHAR | 255 | Полное имя врача | NOT NULL |
| speciality | VARCHAR | 255 | Специализация | NOT NULL |
| schedule | VARCHAR |  | Расписание работы | NOT NULL |

Таблица 8 - Таблица Serving

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Имя поля | Тип | Размер | Комментарий | Ограничение |
| id | INT | auto\_increment | Идентификатор услуги | Первичный ключ |
| name | VARCHAR | 255 | Название услуги | NOT NULL |
| description | TEXT |  | Описание услуги | NOT NULL |
| cost | DECIMAL | (10, 2) | Стоимость услуги | NOT NULL |

Таблица 9 - Таблица Appointment

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Имя поля | Тип | Размер | Комментарий | Ограничение |
| id | INT | auto\_increment | Идентификатор записи | Первичный ключ |
| patient\_id | INT |  | ID пациента | Внешний ключ (Patient.id) |
| doctor\_id | INT |  | ID врача | Внешний ключ (Doctor.id) |
| service\_id | INT |  | ID услуги | Внешний ключ (Service.id) |
| date | DATETIME |  | Дата и время записи | NOT NULL |
| status | VARCHAR | 50 | Статус записи (активна/отменена) | 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 PatientController

1.1 Проверка добавления нового пациента:

Переходим на страницу /patient/add

Вводим данные пациента в соответствующие поля формы (ФИО, дата рождения, телефон, адрес и т.п.)

Нажимаем кнопку «Сохранить»

Проверяем, что новый пациент появился в списке на странице /patient

1.2 Проверка редактирования существующего пациента:

Переходим на страницу /patient

Нажимаем кнопку «Изменить» рядом с нужным пациентом

Изменяем данные в форме

Нажимаем кнопку «Сохранить»

Проверяем, что в списке пациентов отображаются обновлённые данные

1.3 Проверка удаления пациента:

Переходим на страницу /patient

Нажимаем кнопку «Удалить» рядом с нужным пациентом

Подтверждаем удаление (если реализовано)

Проверяем, что пациент исчез из списка

1.4 Проверка сортировки пациентов:

Переходим на страницу /patient?sortField=fullName&sortDir=asc

Проверяем, что список отсортирован по ФИО по возрастанию

Повторяем с sortDir=desc - проверяем сортировку по убыванию

2. Тестирование контроллера AppointmentController

2.1 Проверка добавления новой записи на приём:

Переходим на страницу /appointment/add

Заполняем форму: выбираем пациента, врача, услугу, указываем дату и время

Проверяем, что при выборе услуги автоматически проставляется стоимость

Проверяем, что при выборе пациента автоматически подставляется его номер

Нажимаем кнопку «Сохранить»

Проверяем, что новая запись отображается в списке /appointment

2.2 Проверка редактирования записи:

Переходим на страницу /appointment

Нажимаем кнопку «Изменить» рядом с нужной записью

Изменяем поля (например, выбираем другого врача или меняете дату)

Нажимаем «Сохранить»

Проверяем, что данные записи обновились в списке

2.3 Проверка удаления записи:

Переходим на страницу /appointment

Нажимаем кнопку «Удалить» рядом с нужной записью

Проверяем, что запись исчезла из списка

2.4 Проверка сортировки записей:

Переходим на страницу /appointment?sortField=date&sortDir=asc

Проверяем, что список отсортирован по дате по возрастанию

Меняем параметр sortDir=desc - убеждаемся, что сортировка обратная

3. Тестирование контроллера DoctorController

3.1 Проверка отображения списка врачей с сортировкой:

Открыть страницу /doctor без параметров сортировки.

Проверить, что отображается список всех врачей.

Открыть страницу /doctor?sortBy=lastName&sortDir=asc.

Проверить, что список врачей отсортирован по фамилии в порядке возрастания.

Открыть страницу /doctor?sortBy=lastName&sortDir=desc.

Проверить, что список врачей отсортирован по фамилии в порядке убывания.

3.2 Проверка добавления нового врача:

Перейти на страницу /doctor/add.

Заполнить поля формы для нового врача (Фамилия, Имя, Отчество, специальность, привязать к пользователю).

Нажать кнопку «Сохранить».

Проверить, что врач появился в списке /doctor.

3.3 Проверка валидации при добавлении врача:

Перейти на страницу /doctor/add.

Оставить обязательные поля пустыми.

Нажать «Сохранить».

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

3.4 Проверка редактирования врача:

На странице списка /doctor выбрать врача и перейти по ссылке редактирования /doctor/edit/{id}.

Изменить данные врача.

Нажать «Сохранить».

Проверить, что изменения отобразились на странице списка врачей.

3.5 Проверка удаления одного врача:

На странице списка /doctor выбрать врача.

Нажать кнопку «Удалить» рядом с ним.

Проверить, что врач удалён из списка.

3.6 Проверка массового удаления врачей:

Перейти на страницу /doctor/deleteAll.

Проверить, что список врачей полностью очищен.

4. Тестирование контроллера ServingController

4.1 Проверка отображения списка услуг с сортировкой:

Открыть страницу /serving без параметров сортировки.

Проверить, что отображается полный список услуг.

Открыть страницу /serving?sortField=cost&sortDir=asc.

Проверить, что список отсортирован по стоимости по возрастанию.

Открыть страницу /serving?sortField=name&sortDir=desc.

Проверить, что список отсортирован по названию услуг в порядке убывания.

Открыть страницу /serving?sortField=type&sortDir=asc.

Проверить сортировку по типу услуг в порядке возрастания.

4.2 Проверка добавления новой услуги:

Перейти на страницу /serving/add.

Заполнить форму, включая поля: название, тип, стоимость, описание.

Нажать кнопку «Сохранить».

Проверить, что новая услуга появилась в списке /serving.

4.3 Проверка редактирования услуги:

На странице списка выбрать услугу и перейти по ссылке редактирования /serving/edit/{id}.

Изменить поля услуги.

Нажать кнопку «Сохранить».

Проверить, что изменения корректно отображаются на странице списка.

4.4 Проверка удаления одной услуги:

На странице списка нажать кнопку удаления рядом с выбранной услугой.

Проверить, что услуга удалена из списка.

4.5 Проверка массового удаления услуг:

Перейти по ссылке /serving/delete.

Проверить, что список услуг полностью очищен.

4.6 Проверка обработки ошибки при попытке редактирования несуществующей услуги:

Открыть страницу /serving/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 /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.

Нажимаем «Зарегистрироваться».

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 записей на приём;

Список назначенных услуг (если есть).

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 (на пример: ФИО, номер телефона)

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 - центральный интерфейс управления клиникой. В нём расположены следующие элементы:

2.1 Таблицы с данными

Пациенты: список всех пациентов с контактными данными и историей записей.

Записи на приём: информация о назначенных и проведённых приёмах.

Врачи: список врачей с их специализациями и контактами.

Услуги: каталог предоставляемых услуг с ценами и типами.

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

2.2 Панель управления сущностями

Для каждой таблицы доступны 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 пациентами

3.1 Добавление нового пациента

Нажмите кнопку «Добавить пациента».

В открывшейся форме заполните обязательные поля:

ФИО

Дата рождения

Адрес

Контактный телефон

Дополнительная информация (если требуется)

Нажмите «Сохранить».

Если данные корректны, пациент появится в списке.

3.2 Изменение информации о пациенте

Выберите нужного пациента в таблице.

Нажмите «Изменить пациента».

Внесите изменения и нажмите «Сохранить».

Обновлённые данные отобразятся в таблице.

3.3 Удаление пациента

Выберите пациента в списке.

Нажмите «Удалить пациента».

Подтвердите удаление в появившемся диалоговом окне.

Запись будет удалена из базы.

4. Управление записями на приём

4.1 Добавление новой записи

Нажмите кнопку «Добавить запись».

Заполните поля формы:

Пациент (выберите из списка)

Врач (выберите специалиста)

Услуга (выберите из доступных)

Дата и время приёма

Нажмите «Сохранить». Запись появится в списке.

4.2 Изменение записи

Выберите запись в таблице.

Нажмите «Изменить запись».

Измените необходимые параметры и сохраните изменения кнопкой «Сохранить».

4.3 Удаление записи

Выберите запись в списке.

Нажмите «Удалить запись».

Подтвердите удаление.

Запись будет удалена из системы.

5. Управление врачами

5.1 Добавление врача

Нажмите «Добавить врача».

Заполните форму:

ФИО

Специальность

Контактный телефон

Описание (необязательно)

Нажмите «Сохранить».

5.2 Редактирование врача

Выберите врача в таблице.

Нажмите «Изменить врача».

Отредактируйте поля и сохраните.

5.3 Удаление врача

Выберите врача.

Нажмите «Удалить врача» и подтвердите действие.

6. Управление услугами

6.1 Добавление новой услуги

Нажмите «Добавить услугу».

Введите данные:

Название услуги

Тип (консультация, диагностика, лечение и т.п.)

Стоимость

Описание (по желанию)

Нажмите «Сохранить». Услуга появится в списке.

6.2 Редактирование услуги

Выберите услугу в таблице.

Нажмите «Изменить услугу».

Внесите изменения и сохраните.

6.3 Удаление услуги

Выберите услугу.

Нажмите «Удалить услугу». Подтвердите удаление.

7. Управление пользователями системы

7.1 Добавление пользователя

Нажмите «Добавить пользователя».

Заполните поля:

Логин

Пароль

Роль (администратор, врач, регистратор и др.)

Контактные данные

Нажмите «Сохранить».

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 ваших записей на приём

Назначенные вам услуги (если есть)

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

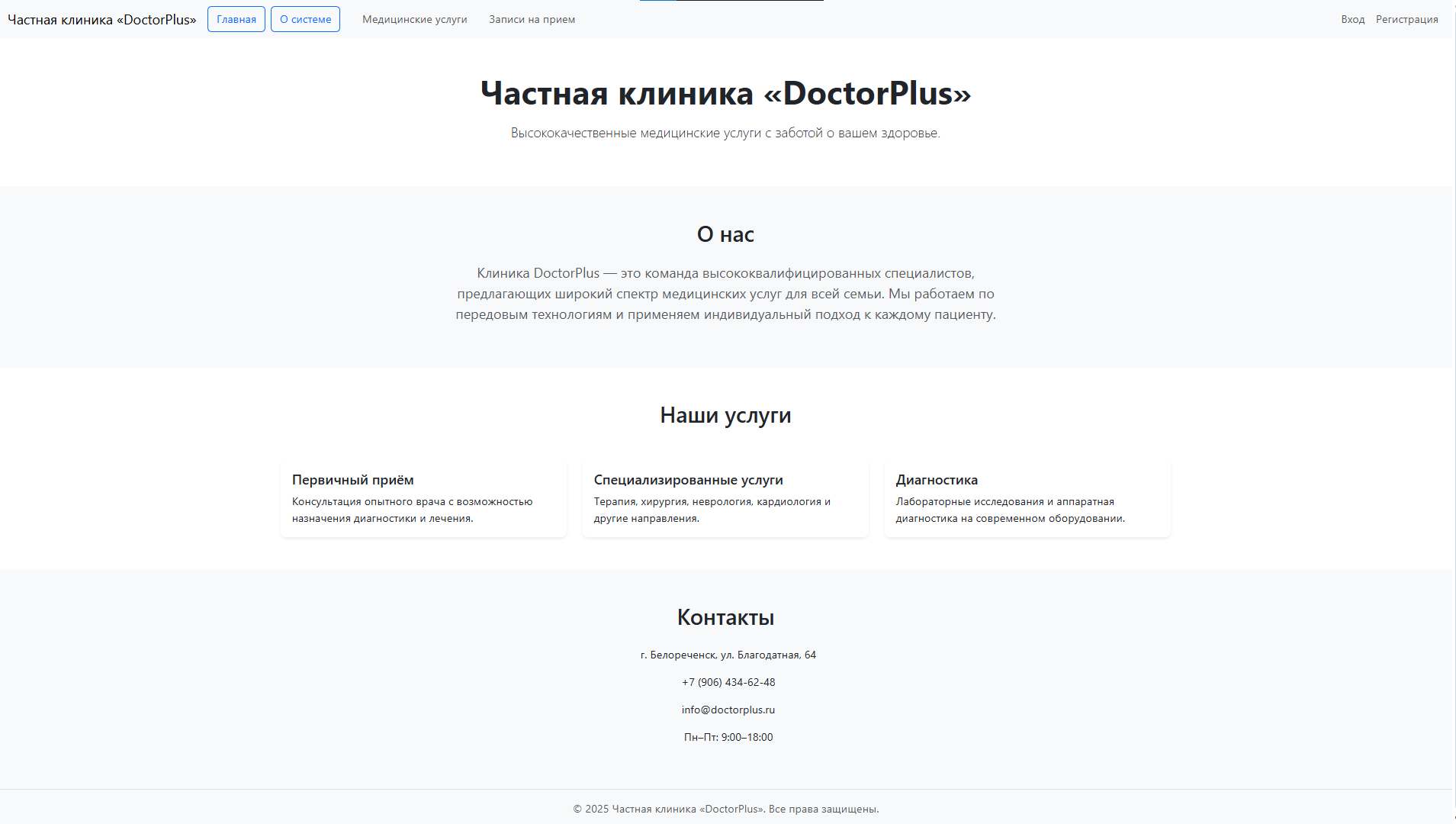


Рисунок 14 - Главная страница

На главной странице отображается основная информация о частной клиники «DoctorPlus». Пациентам представлено краткое описание клиники, перечень предоставляемых услуг (консультации).

Также на странице расположена кнопка для перехода к выбору услуг и кнопка к просмотру записей.

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

2. Страница регистрации

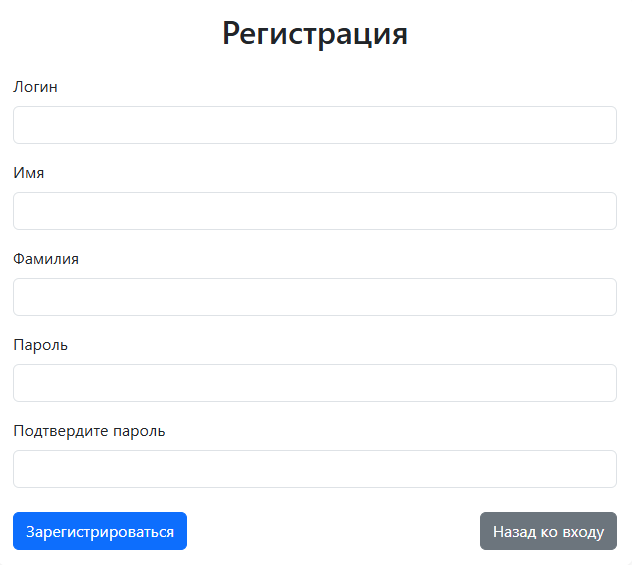


Рисунок 15 - Страница регистрации

Страница регистрации предназначена для создания новой учётной записи пользователя на платформе частной клиники «DoctorPlus». 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 «Зарегистрация» для отправки данных. При необходимости вернуться на страницу входа предусмотрена кнопка «Назад ко входу».

Адрес страницы регистрации: /registration.

3. Страница авторизации

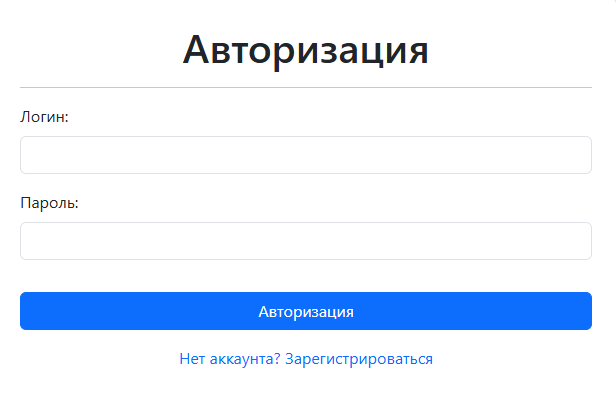


Рисунок 15 - Страница авторизации

Страница авторизации предназначена для входа пользователей в систему «DoctorPlus». Она включает поля для ввода логина и пароля, кнопку «Авторизация» для подтверждения данных. Интерфейс минималистичен, с четким расположением элементов и фирменным стилем компании, обеспечивая удобство и безопасность при входе.

4. Страница профиля

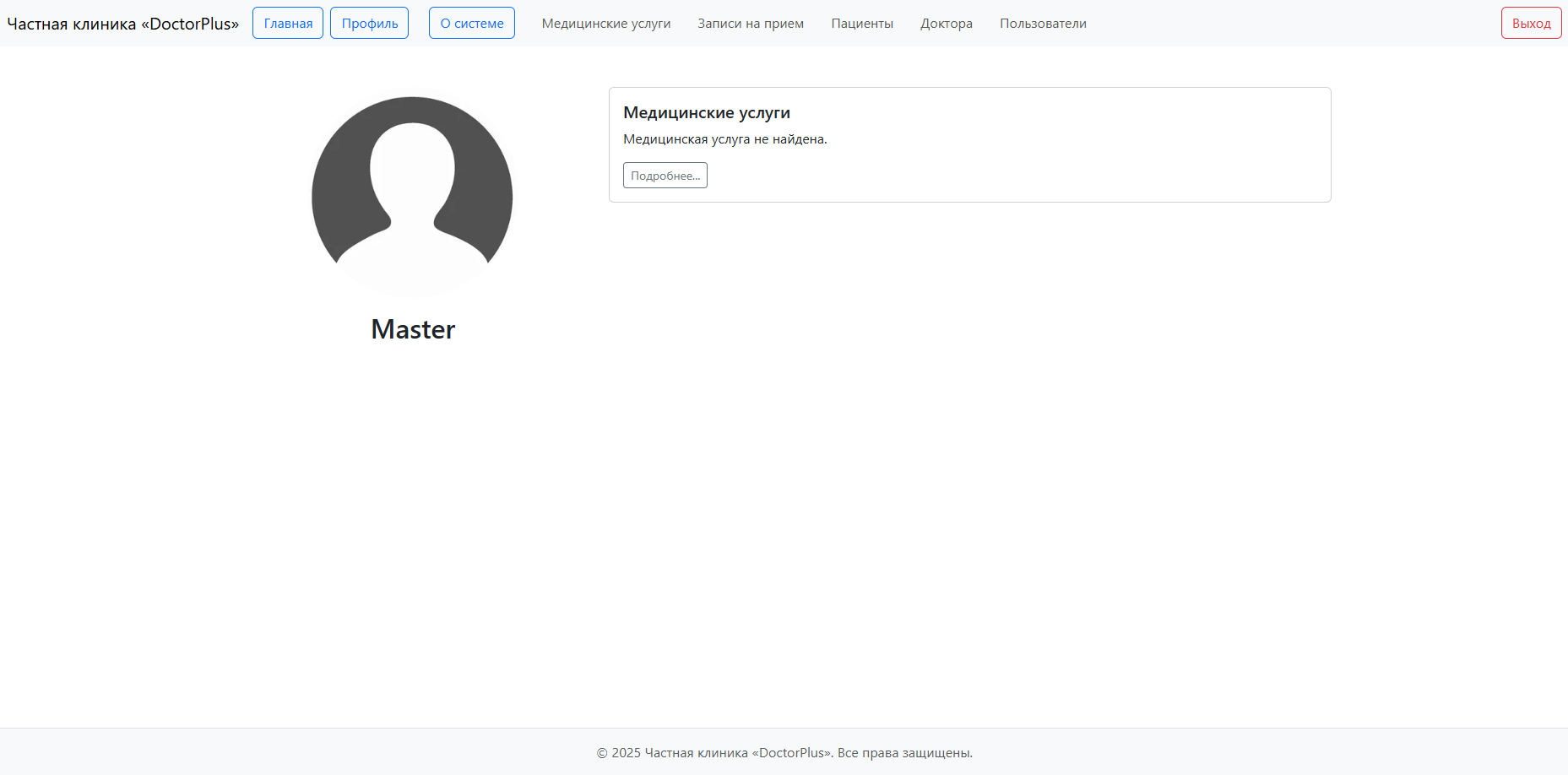


Рисунок 16 - Страница профиля

Страница профиля в системе «DoctorPlus» отображает основную информацию о пользователе. В центре представлена аватарка и имя пользователя. Справа находится раздел «Медицинские услуги», где отображаются недвижимости, с кнопкой «Подробнее» для перехода к дополнительной информации. В верхней части страницы расположено меню навигации с разделами «Профиль», «Медицинские услуги», «Записи на прием», «О системе», «Пациенты», «Доктора», «Пользователи», а также кнопка «Выход» для завершения сеанса. Дизайн страницы прост и функционален, обеспечивая удобный доступ к ключевым функциям.

5. Страница медицинских услуг

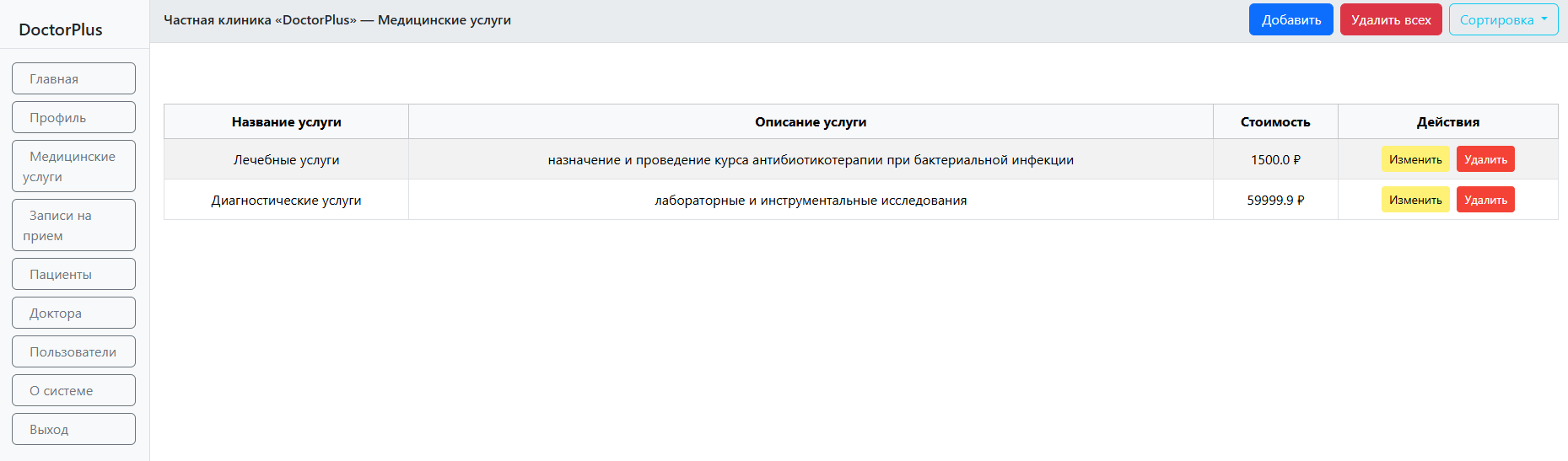


Рисунок 17 - Страница медицинских услуг

Страница недвижимости в системе «DoctorPlus» предназначена для управления объектами недвижимости. Она содержит таблицу с колонками: «Название услуги», «Описание услуги», «Стоимость» и «Действия». Для каждого объекта доступны действия: «Изменить» (желтая кнопка) и «Удалить» (красная кнопка). Слева расположено меню навигации с разделами, включая «Медицинские услуги», «Пациенты», «Доктора» 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 медицинской услуги

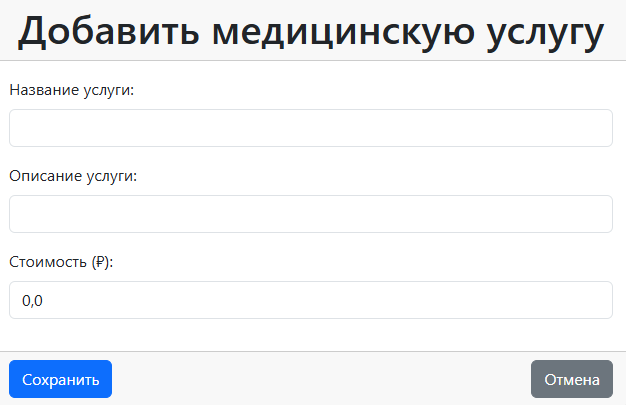


Рисунок 18 - Страница формы медицинской услуги

Страница формы медицинской услуги в системе «DoctorPlus» предназначена для добавления или редактирования данных об медицинских услугах. Она включает поля для указания названия услуги, описания услуги, стоимости. 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 записей на прием

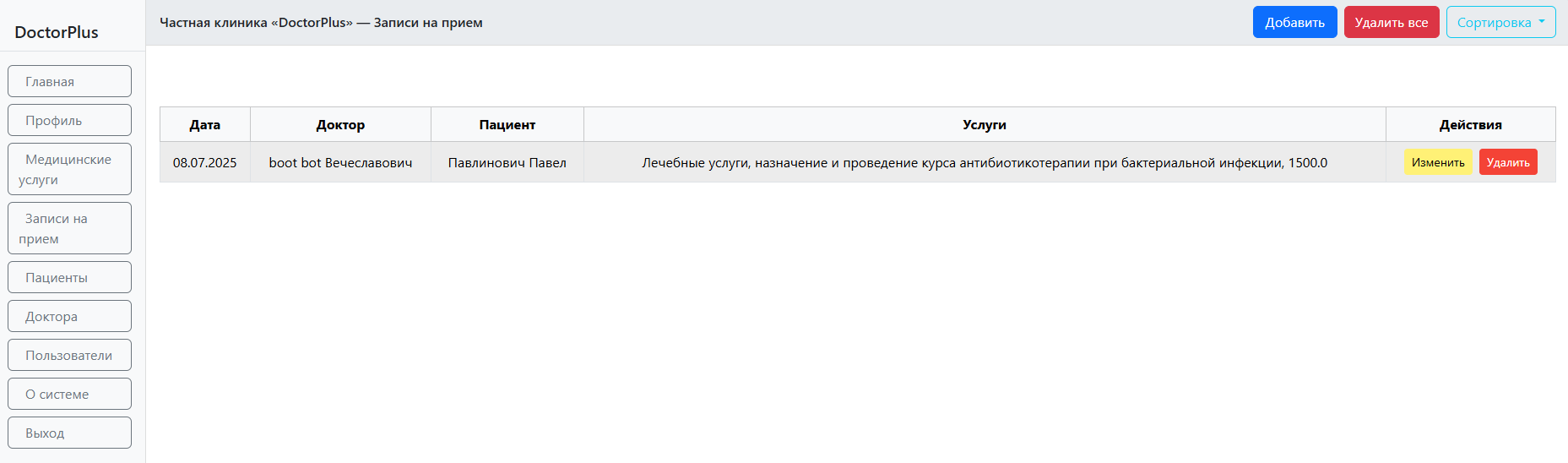


Рисунок 19 - Страница записей на прием

Страница записей на прием в системе «DoctorPlus» предназначена для управления и просмотра информации о совершенных сделках. Она включает таблицу с колонками: «Дата», «Доктор», «Пациент», «Услуги» 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 формы записей на прием

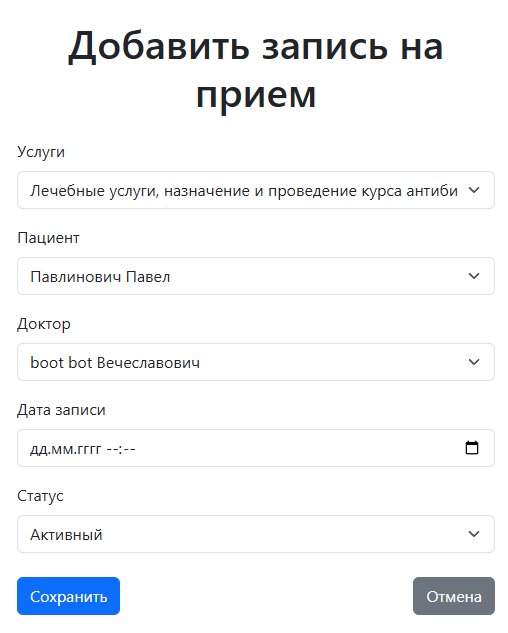


Рисунок 20 - Страница формы записей на прием

Страница формы записей на прием в системе «DoctorPlus» предназначена для добавления или редактирования информации о записях на прием. Она включает поля для выбора услуги, пациента, доктора, 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 пациентов

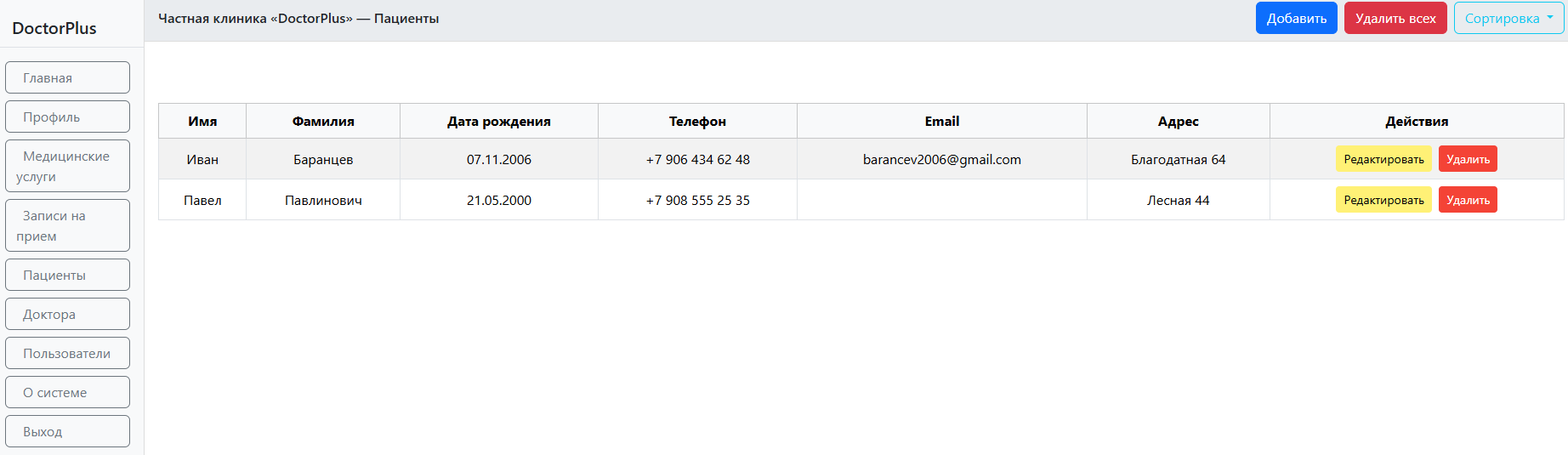


Рисунок 21 - Страница пациентов

Страница клиентов в системе «DoctorPlus» предназначена для управления и просмотра информации о пациентах. Она включает таблицу с колонками: «Имя», «Фамилия», «Дата рождения», «Телефон», «Email», «Адрес» и «Действия».

Для каждого клиента доступны действия: «Редактировать» (желтая кнопка) и «Удалить» (красная кнопка). Слева расположено меню навигации с разделами, включая «Пациенты», «Медицинские слуги», «Записи на прием» 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 пациентов

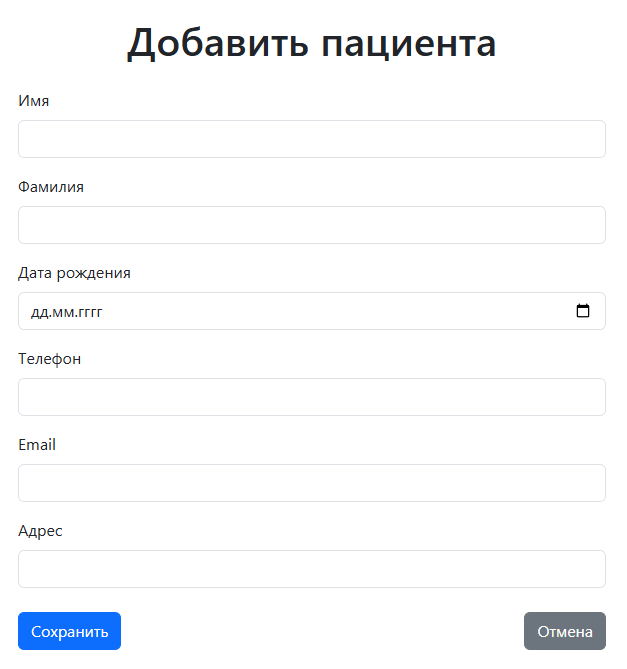


Рисунок 22 - Страница формы пациентов

Страница формы пациентов в системе «DoctorPlus» предназначена для добавления или редактирования информации о пациентах. Она включает поля для Имени, Фамилии, даты рождения (ДД.ММ.ГГГГ), Телефон, Email, Адрес, 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.

11. Страница докторов

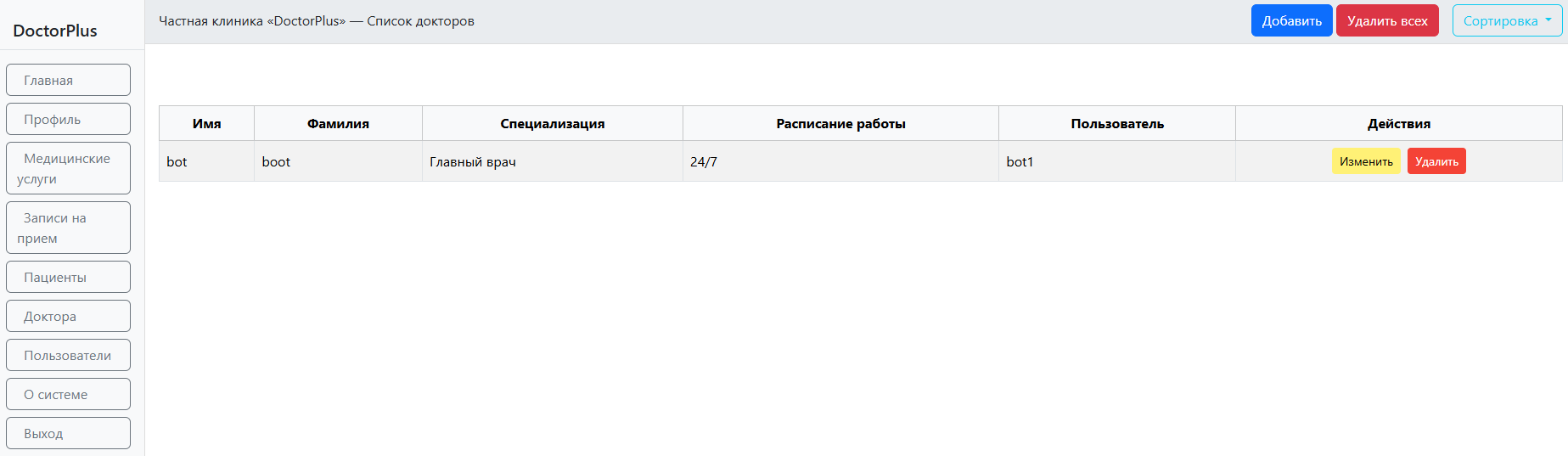
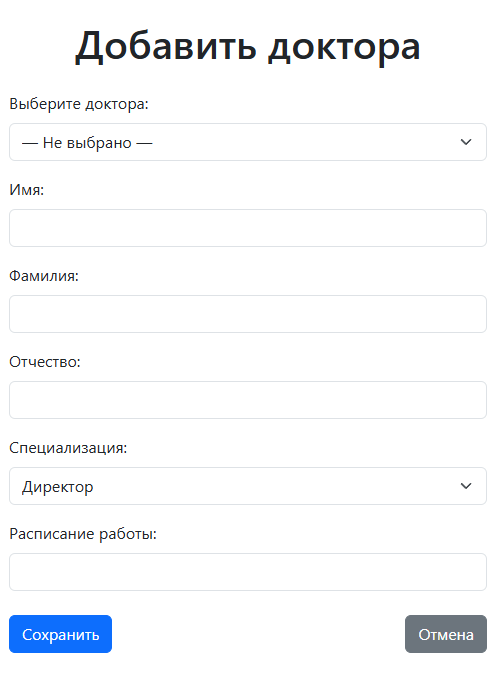


Рисунок 23 - Страница докторов

Страница докторов в системе «DoctorPlus» предназначена для управления и просмотра информации о докторах. Она включает таблицу с колонками: «Имя», «Фамилия», «Специализация», «Расписание работы», «Пользователь» и «Действия». Для каждого доктора доступны действия: «Изменить» (желтая кнопка) и «Удалить» (красная кнопка). Слева расположено меню навигации с разделами, включая «Доктора», «Пациенты», «Услуги» и другие. Вверху страницы есть кнопки «Добавить» и «Удалить все», а также иконка для дополнительных настроек. Интерфейс удобен для работы с данными докторами.

12. Страница формы докторов



Страница 24 - Страница формы докторов

Страница формы докторов в системе «DoctorPlus» предназначена для добавления или редактирования данных о докторах. Она включает поля для выбора пользователя, ввода имени, фамилии, отчества, специализации, расписания работы. Пользователь может сохранить изменения с помощью кнопки «Сохранить» или отменить действие кнопкой «Отмена». Также доступна кнопка «Отмена» для возврата к списку докторов. Интерфейс организован в виде таблицы с четким расположением, обеспечивая удобство ввода и проверки.

13. Страница пользователей

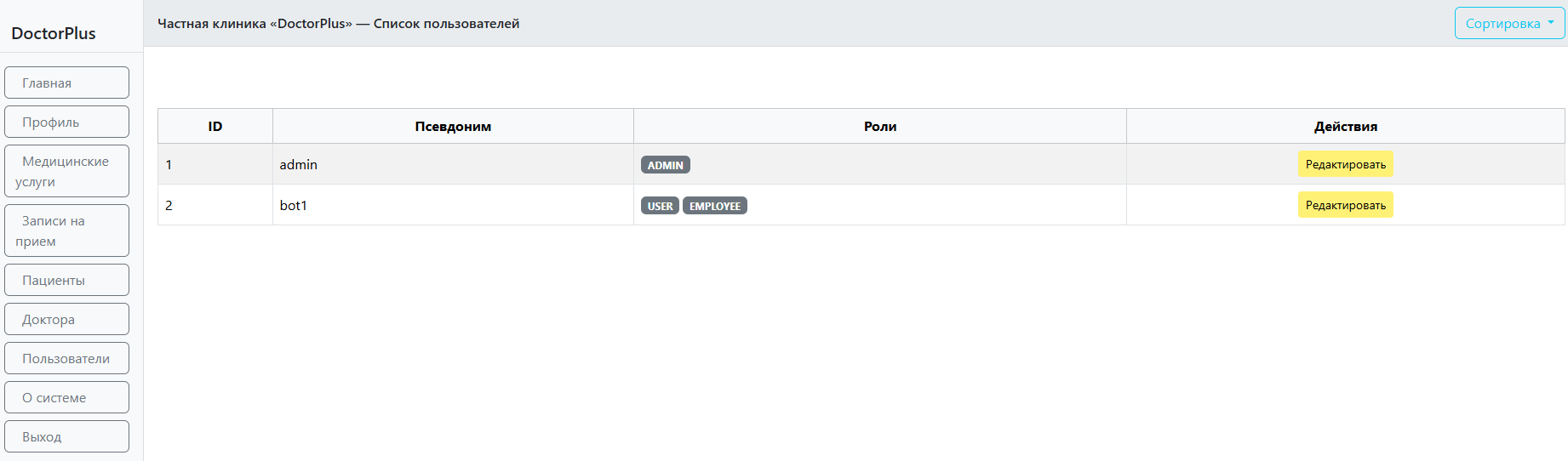
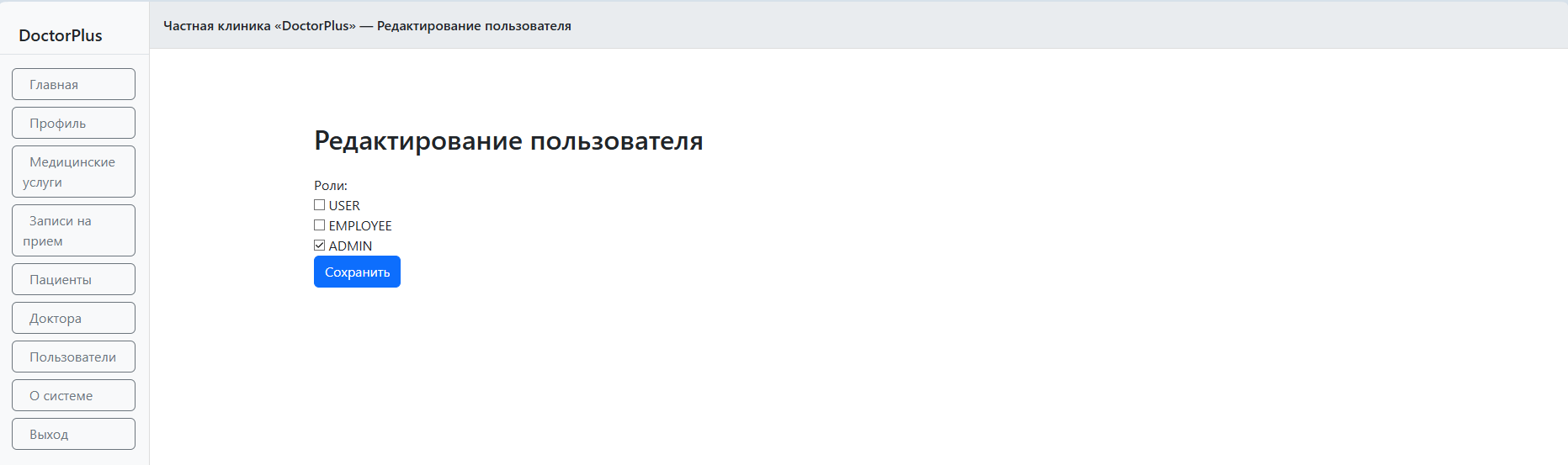


Рисунок 25 - Страница пользователей

Страница пользователей в системе «DoctorPlus» 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



Страница редактирования пользователей в системе «DoctorPlus» 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 разработана информационная система для частной клиники DoctorPlus, предназначенная для автоматизации ключевых процессов, связанных с управлением пациентами, врачами, услугами и записями на приём.

Была создана структурированная база данных, включающая информацию о пациентах, врачах, предоставляемых медицинских услугах, а также данные о записях и приёмах. Это позволяет персоналу клиники оперативно управлять данными и обеспечивать высокое качество обслуживания пациентов.

Система предоставляет возможность удобного ведения записей, назначения услуг, редактирования информации о пациентах и врачах, а также контроля за расписанием приёмов. Реализованный функционал способствует повышению прозрачности и эффективности внутренних процессов клиники.

Использование системы DoctorPlus позволяет сократить время на административные задачи, минимизировать риск ошибок при оформлении записей, а также повысить уровень сервиса для пациентов.

Для дальнейшего совершенствования рекомендуется регулярно обновлять систему, внедрять новые функции (например, интеграцию с платёжными системами или мобильным приложением), а также адаптировать её под рост клиники и изменяющиеся потребности пациентов.

Таким образом, разработанная информационная система DoctorPlus является надёжным инструментом для эффективного управления деятельностью частной клиники и повышения качества медицинского обслуживания.

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 com.example.DoctorPlus.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("/", "/serving/serving\_list", "/include/profile", "/include/profile/\*\*", "/appointment/appointment\_list")  
 .hasAnyRole("USER", "EMPLOYEE", "ADMIN")  
 .requestMatchers("/serving/serving\_form", "/appointment/appointment\_form", "/patient/patient\_list", "/appointment/appointment\_form")  
 .hasAnyRole("EMPLOYEE", "ADMIN")  
 .requestMatchers("/doctor/doctor\_form", "/doctor/doctor\_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 com.example.DoctorPlus.config;  
  
import org.springframework.beans.factory.annotation.Value;  
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;  
  
@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 com.example.DoctorPlus.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 com.example.DoctorPlus.controller;  
  
import com.example.DoctorPlus.model.Appointment;  
import com.example.DoctorPlus.model.Doctor;  
import com.example.DoctorPlus.model.Patient;  
import com.example.DoctorPlus.model.Serving;  
import com.example.DoctorPlus.service.AppointmentService;  
import com.example.DoctorPlus.service.DoctorService;  
import com.example.DoctorPlus.service.PatientService;  
import com.example.DoctorPlus.service.ServingService;  
import com.fasterxml.jackson.core.JsonProcessingException;  
import com.fasterxml.jackson.databind.ObjectMapper;  
import lombok.RequiredArgsConstructor;  
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("/appointment")  
@RequiredArgsConstructor  
public class AppointmentController {  
  
 private final AppointmentService appointmentService;  
 private final PatientService patientService;  
 private final ServingService servingService;  
 private final DoctorService doctorService;  
  
 @GetMapping  
 public String listAppointments(  
 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<Appointment> appointments = appointmentService.getAllAppointments();  
  
 model.addAttribute("appointments", appointments);  
 model.addAttribute("sortField", sortField);  
 model.addAttribute("sortDir", sortDir);  
 model.addAttribute("reverseSortDir", (sortDir != null && sortDir.equalsIgnoreCase("asc")) ? "desc" : "asc");  
  
 return "appointment/appointment\_list";  
 }  
  
 @PostMapping("/save")  
 public String saveAppointment(@ModelAttribute Appointment appointment) {  
 appointmentService.saveAppointment(appointment);  
 return "redirect:/appointment";  
 }  
  
 @GetMapping({"/add", "/edit/{id}"})  
 public String showAppointmentForm(@PathVariable(required = false) Long id, Model model) throws JsonProcessingException {  
 Appointment appointment = (id != null) ? appointmentService.findById(id).orElse(new Appointment()) : new Appointment();  
  
 List<Serving> servings = servingService.findAll();  
 List<Patient> patients = patientService.findAll();  
 List<Doctor> doctors = doctorService.findAll();  
  
 model.addAttribute("appointment", appointment);  
 model.addAttribute("servings", servings);  
 model.addAttribute("patients", patients);  
 model.addAttribute("doctors", doctors);  
  
 ObjectMapper objectMapper = new ObjectMapper();  
  
 *// JSON-данные по недвижимости* List<Map<String, Object>> servingJsonList = servings.stream().map(r -> {  
 Map<String, Object> map = new HashMap<>();  
 map.put("id", r.getId());  
 map.put("cost", r.getCost());  
 return map;  
 }).collect(Collectors.toList());  
  
 *// JSON-данные по клиентам* List<Map<String, Object>> patientJsonList = patients.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("servingJson", objectMapper.writeValueAsString(servingJsonList));  
 model.addAttribute("patientJson", objectMapper.writeValueAsString(patientJsonList));  
  
 return "appointment/appointment\_form";  
 }  
  
 @PostMapping("/delete/{id}")  
 public String deleteAppointment(@PathVariable Long id, RedirectAttributes redirectAttributes) {  
 try {  
 appointmentService.deleteAppointmentById(id);  
 redirectAttributes.addFlashAttribute("successMessage", "Записи о приеме успешно удалены");  
 } catch (Exception e) {  
 redirectAttributes.addFlashAttribute("errorMessage", "Ошибка при удалении записей о приеме");  
 }  
 return "redirect:/appointment";  
 }  
  
 @PostMapping("/deleteAll")  
 public String deleteAllAppointments(RedirectAttributes redirectAttributes) {  
 try {  
 appointmentService.deleteAllAppointments();  
 redirectAttributes.addFlashAttribute("successMessage", "Все записи о приеме успешно удалены");  
 } catch (Exception e) {  
 redirectAttributes.addFlashAttribute("errorMessage", "Ошибка при удалении всех записей о приеме");  
 }  
 return "redirect:/appointment";  
 }  
}

package com.example.DoctorPlus.controller;  
  
import com.example.DoctorPlus.model.Doctor;  
import com.example.DoctorPlus.service.DoctorService;  
import com.example.DoctorPlus.service.UserService;  
import jakarta.validation.Valid;  
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("/doctor")  
public class DoctorController {  
  
 private final DoctorService doctorService;  
 private final UserService userService;  
  
 public DoctorController(DoctorService doctorService, UserService userService) {  
 this.doctorService = doctorService;  
 this.userService = userService;  
 }  
  
 @GetMapping  
 public String listDoctors(@RequestParam(value = "sortBy", required = false) String sortBy,  
 @RequestParam(value = "sortDir", required = false) String sortDir,  
 Model model) {  
 List<Doctor> doctors;  
  
 if (sortBy == null || sortBy.isEmpty()) {  
 doctors = doctorService.getAllDoctorsWithUsers();  
 } else {  
 Sort.Direction direction = "desc".equalsIgnoreCase(sortDir) ? Sort.Direction.DESC : Sort.Direction.ASC;  
 doctors = doctorService.getAllDoctorsSorted(Sort.by(direction, sortBy));  
 }  
  
 model.addAttribute("doctors", doctors);  
 model.addAttribute("sortBy", sortBy);  
 model.addAttribute("sortDir", sortDir);  
 return "doctor/doctor\_list";  
 }  
  
 @GetMapping("/add")  
 public String showAddForm(Model model) {  
 model.addAttribute("doctor", new Doctor());  
 model.addAttribute("users", userService.findAll());  
 return "doctor/doctor\_form";  
 }  
  
 @PostMapping("/save")  
 public String saveDoctor(@Valid @ModelAttribute("doctor") Doctor doctor,  
 BindingResult result,  
 Model model) {  
 if (result.hasErrors()) {  
 model.addAttribute("users", userService.findAll());  
 return "doctor/doctor\_form";  
 }  
  
 try {  
 doctorService.saveDoctor(doctor);  
 return "redirect:/doctor";  
 } catch (Exception e) {  
 model.addAttribute("validationErrors", List.of("Ошибка при сохранении доктора: " + e.getMessage()));  
 model.addAttribute("users", userService.findAll());  
 return "doctor/doctor\_form";  
 }  
 }  
  
 @GetMapping("/edit/{id}")  
 public String showEditForm(@PathVariable Long id, Model model) {  
 Doctor doctor = doctorService.getDoctorById(id);  
 if (doctor == null) {  
 return "redirect:/doctor";  
 }  
 model.addAttribute("doctor", doctor);  
 model.addAttribute("users", userService.findAll());  
 return "doctor/doctor\_form";  
 }  
  
 @GetMapping("/delete/{id}")  
 public String deleteDoctor(@PathVariable Long id, RedirectAttributes redirectAttributes) {  
 try {  
 doctorService.deleteDoctor(id);  
 } catch (IllegalStateException e) {  
 redirectAttributes.addAttribute("error", true);  
 }  
 return "redirect:/doctor";  
 }  
  
 @GetMapping("/deleteAll")  
 public String deleteAllDoctors(RedirectAttributes redirectAttributes) {  
 List<Doctor> notDeleted = doctorService.deleteAllExceptWithAppointments();  
  
 if (!notDeleted.isEmpty()) {  
 redirectAttributes.addAttribute("partialError", notDeleted.size());  
 }  
  
 return "redirect:/doctor";  
 }  
  
 @GetMapping("/patronymic/{userId}")  
 @ResponseBody  
 public ResponseEntity<Map<String, String>> getPatronymicByUserId(@PathVariable Long userId) {  
 Doctor doctor = doctorService.getDoctorByUserId(userId);  
 if (doctor != null && doctor.getPatronymic() != null) {  
 return ResponseEntity.ok(Map.of("patronymic", doctor.getPatronymic()));  
 } else {  
 return ResponseEntity.ok(Map.of("patronymic", ""));  
 }  
 }  
}

package com.example.DoctorPlus.controller;  
  
import com.example.DoctorPlus.model.Patient;  
import com.example.DoctorPlus.service.PatientService;  
import jakarta.validation.Valid;  
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("/patient")  
public class PatientController {  
  
 private final PatientService patientService;  
  
 public PatientController(PatientService patientService) {  
 this.patientService = patientService;  
 }  
  
 @GetMapping  
 public String listPatients(Model model,  
 @RequestParam(required = false) String sortField,  
 @RequestParam(required = false) String sortDir) {  
  
 List<Patient> patients;  
 String direction = (sortDir != null && sortDir.equalsIgnoreCase("desc")) ? "desc" : "asc";  
  
 if (sortField == null || sortField.isBlank()) {  
 patients = patientService.findAll();  
 sortField = null;  
 direction = null;  
 } else {  
 patients = patientService.findAllSortedByField(sortField.toLowerCase(), direction);  
 }  
  
 model.addAttribute("patients", patients);  
 model.addAttribute("sortField", sortField);  
 model.addAttribute("sortDir", direction);  
  
 return "patient/patient\_list";  
 }  
  
 @GetMapping("/add")  
 public String addPatientForm(Model model) {  
 model.addAttribute("patient", new Patient());  
 return "patient/patient\_form";  
 }  
  
 @GetMapping("/edit/{id}")  
 public String editPatientForm(@PathVariable Long id, Model model) {  
 Optional<Patient> patientOpt = patientService.findById(id);  
 if (patientOpt.isEmpty()) {  
 return "redirect:/patient";  
 }  
 model.addAttribute("patient", patientOpt.get());  
 return "patient/patient\_form";  
 }  
  
 @PostMapping("/save")  
 public String savePatient(@Valid @ModelAttribute("patient") Patient patient,  
 BindingResult bindingResult) {  
 if (bindingResult.hasErrors()) {  
 return "patient/patient\_form";  
 }  
 patientService.save(patient);  
 return "redirect:/patient";  
 }  
  
 @PostMapping("/delete/{id}")  
 public String deletePatient(@PathVariable Long id) {  
 boolean deleted = patientService.deleteByIdIfPossible(id);  
 return deleted ? "redirect:/patient" : "redirect:/patient?error=true";  
 }  
  
 @PostMapping("/deleteAll")  
 public String deleteAllPatients() {  
 int notDeletedCount = patientService.deleteAllExceptWithAppointments();  
 if (notDeletedCount > 0) {  
 return "redirect:/patient?partialError=" + notDeletedCount;  
 }  
 return "redirect:/patient";  
 }  
}

package com.example.DoctorPlus.controller;  
  
import com.example.DoctorPlus.model.Appointment;  
import com.example.DoctorPlus.model.Serving;  
import com.example.DoctorPlus.model.User;  
import com.example.DoctorPlus.service.AppointmentService;  
import com.example.DoctorPlus.service.ServingService;  
import com.example.DoctorPlus.service.UserService;  
import jakarta.servlet.ServletContext;  
import lombok.Setter;  
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;  
  
 @Autowired  
 private final UserService userService;  
 private final ServingService servingService;  
 private final AppointmentService appointmentService;  
 @Autowired  
 private ServletContext servletContext;  
  
 *// Значение можно получить из application.properties (fallback — "uploads")* @Setter  
 @Value("${upload.path:uploads}")  
 private String uploadPath;  
  
 public ProfileController(UserService userService, ServingService servingService, AppointmentService appointmentService) {  
 this.userService = userService;  
 this.servingService = servingService;  
 this.appointmentService = appointmentService;  
 }  
  
 @GetMapping("/profile")  
 public String userProfile(Principal principal, Model model) {  
 User user = userService.findByUsername(principal.getName())  
 .orElseThrow(() -> new UsernameNotFoundException("Пользователь не найден"));  
  
 */\*List<Serving> userServings = servingService.findAllByOwner(user);\*/* List<Appointment> userAppointments = appointmentService.findAllByUser(user);  
  
 model.addAttribute("user", user);  
 */\*model.addAttribute("servingList", userServings);\*/* model.addAttribute("appointmentList", userAppointments);  
  
 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;  
 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);  
 userService.save(user);  
 } else {  
 throw new IOException("Пользователь не найден: " + principal.getName());  
 }  
 }  
 return "redirect:/profile";  
 }  
}

package com.example.DoctorPlus.controller;  
  
import com.example.DoctorPlus.dto.UserDTO;  
import com.example.DoctorPlus.service.UserService;  
import jakarta.validation.Valid;  
import lombok.RequiredArgsConstructor;  
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 com.example.DoctorPlus.controller;  
  
import com.example.DoctorPlus.model.Serving;  
import com.example.DoctorPlus.service.AppointmentService;  
import com.example.DoctorPlus.service.PatientService;  
import com.example.DoctorPlus.service.ServingService;  
import com.example.DoctorPlus.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("/serving")  
public class ServingController {  
  
 private final ServingService servingService;  
 private final UserService userService;  
 private final AppointmentService appointmentService;  
 private final PatientService patientService;  
  
 public ServingController(ServingService servingService, UserService userService, AppointmentService appointmentService, PatientService patientService) {  
 this.servingService = servingService;  
 this.userService = userService;  
 this.appointmentService = appointmentService;  
 this.patientService = patientService;  
 }  
  
 @GetMapping  
 public String listServing(  
 @RequestParam(required = false) String sortField,  
 @RequestParam(required = false) String sortDir,  
 Model model) {  
  
 List<Serving> servings = servingService.findAll();  
  
 if (sortField != null && sortDir != null) {  
 Comparator<Serving> comparator = null;  
  
 switch (sortField) {  
 case "cost":  
 comparator = Comparator.comparing(Serving::getCost);  
 break;  
 case "name":  
 comparator = Comparator.comparing(Serving::getName, String.CASE\_INSENSITIVE\_ORDER);  
 break;  
 case "description":  
 comparator = Comparator.comparing(Serving::getDescription, String.CASE\_INSENSITIVE\_ORDER);  
 break;  
 }  
  
 if (comparator != null) {  
 if ("desc".equalsIgnoreCase(sortDir)) {  
 comparator = comparator.reversed();  
 }  
 servings.sort(comparator);  
 }  
 }  
  
 model.addAttribute("servings", servings);  
 model.addAttribute("sortField", sortField);  
 model.addAttribute("sortDir", sortDir);  
  
 return "serving/serving\_list";  
 }  
  
 @GetMapping("/add")  
 public String showAddForm(Model model) {  
 model.addAttribute("serving", new Serving());  
 model.addAttribute("users", userService.findAll());  
 model.addAttribute("appointment", appointmentService.findAll());  
 model.addAttribute("patient", patientService.findAll());  
 return "serving/serving\_form";  
 }  
  
 @PostMapping("/save")  
 public String saveServing(@ModelAttribute Serving serving) {  
 servingService.saveServing(serving);  
 return "redirect:/serving";  
 }  
  
 @PostMapping("/delete/{id}")  
 public String deleteServing(@PathVariable Long id, RedirectAttributes redirectAttributes) {  
 if (servingService.existsByServingId(id)) {  
 redirectAttributes.addFlashAttribute("errorMessage", "Невозможно удалить: услугу которая участвует в записях на прием.");  
 } else {  
 servingService.deleteById(id);  
 redirectAttributes.addFlashAttribute("successMessage", "Услуга успешно удалена.");  
 }  
 return "redirect:/serving";  
 }  
  
 @GetMapping("/edit/{id}")  
 public String showEditForm(@PathVariable Long id, Model model) {  
 Serving serving = servingService.getServingById(id);  
 if (serving == null) {  
 throw new NoSuchElementException("Услуга с id " + id + " не найдена");  
 }  
 model.addAttribute("serving", serving);  
 model.addAttribute("users", userService.findAll());  
 model.addAttribute("appointment", appointmentService.findAll());  
 model.addAttribute("patient", patientService.findAll());  
 return "serving/serving\_form";  
 }  
  
 @PostMapping("/delete")  
 public String deleteAllExceptWithAppointments(RedirectAttributes redirectAttributes) {  
 List<Serving> deleted = servingService.deleteAllExceptWithAppointments();  
 if (deleted.isEmpty()) {  
 redirectAttributes.addFlashAttribute("warningMessage", "Удаление невозможно: все объекты участвуют в сделках.");  
 } else {  
 int deletedCount = deleted.size();  
 redirectAttributes.addFlashAttribute("warningMessage",  
 "Были удалены все услуги, которые не находятся в записях на прием. Количество удалённых: " + deletedCount);  
 }  
 return "redirect:/serving";  
 }  
}

package com.example.DoctorPlus.controller;  
  
import com.example.DoctorPlus.dto.UserDTO;  
import com.example.DoctorPlus.impl.ServingServiceImpl;  
import com.example.DoctorPlus.model.Role;  
import com.example.DoctorPlus.model.Serving;  
import com.example.DoctorPlus.model.User;  
import com.example.DoctorPlus.service.UserService;  
import jakarta.validation.Valid;  
import lombok.RequiredArgsConstructor;  
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 ServingServiceImpl servingService;  
  
 @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("/serving")  
 public String userServing(Model model, @AuthenticationPrincipal UserDetails userDetails) {  
 User currentUser = (User) userDetails;  
 return "profile/serving\_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 com.example.DoctorPlus.dto;  
  
import jakarta.validation.constraints.NotEmpty;  
import jakarta.validation.constraints.Size;  
import lombok.\*;  
  
import java.time.LocalDate;  
  
@NoArgsConstructor  
@AllArgsConstructor  
@Getter  
@Setter  
@Data  
public class AppointmentDTO {  
 @NotEmpty(message = "Дата не может быть пустым")  
 private LocalDate date;  
 @NotEmpty(message = "Статус сделки не может быть пустым")  
 private String status;  
}

package com.example.DoctorPlus.dto;  
  
import jakarta.validation.constraints.NotEmpty;  
import jakarta.validation.constraints.Size;  
import lombok.\*;  
  
@Data  
@NoArgsConstructor  
@AllArgsConstructor  
@Getter  
@Setter  
public class DoctorDTO {  
 @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 speciality;  
 @NotEmpty(message = "Специализация не может быть пустым")  
 private String schedule;  
}

package com.example.DoctorPlus.dto;  
  
import jakarta.validation.constraints.NotEmpty;  
import jakarta.validation.constraints.Size;  
import lombok.\*;  
  
@NoArgsConstructor  
@AllArgsConstructor  
@Getter  
@Setter  
@Data  
public class PatientDTO {  
 @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;  
 *// company - может быть пустым* private String email;  
 *// license - может быть пустым* private String address;  
}

package com.example.DoctorPlus.dto;  
  
import jakarta.validation.constraints.NotEmpty;  
import jakarta.validation.constraints.Size;  
import lombok.\*;  
  
@Data  
@NoArgsConstructor  
@AllArgsConstructor  
@Getter  
@Setter  
public class ServingDTO {  
 @NotEmpty(message = "Регион не может быть пустым")  
 @Size(min = 3, max = 50, message = "От 3 до 20 символов")  
 private String name;  
 @NotEmpty(message = "Местонахождение не может быть пустым")  
 @Size(min = 3, max = 50, message = "От 3 до 20 символов")  
 private String description;  
 @NotEmpty(message = "cost комнат не может быть пустым")  
 private double cost;  
}

package com.example.DoctorPlus.dto;  
  
import com.example.DoctorPlus.model.Role;  
import com.example.DoctorPlus.model.User;  
import jakarta.validation.constraints.NotBlank;  
import jakarta.validation.constraints.Size;  
import lombok.AllArgsConstructor;  
import lombok.Getter;  
import lombok.NoArgsConstructor;  
import lombok.Setter;  
  
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<>();  
  
 public <R> UserDTO(Long id, String username, String name, String surname, R collect) {}  
  
 *// 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 com.example.DoctorPlus.impl;  
  
import com.example.DoctorPlus.model.\*;  
import com.example.DoctorPlus.repo.AppointmentRepo;  
import com.example.DoctorPlus.repo.DoctorRepo;  
import com.example.DoctorPlus.repo.PatientRepo;  
import com.example.DoctorPlus.repo.ServingRepo;  
import com.example.DoctorPlus.service.AppointmentService;  
import lombok.RequiredArgsConstructor;  
import org.springframework.stereotype.Service;  
import java.util.List;  
import java.util.Optional;  
  
@Service  
@RequiredArgsConstructor  
public class AppointmentServiceImpl implements AppointmentService {  
  
 private final AppointmentRepo appointmentRepository;  
 private final ServingRepo servingRepository;  
 private final PatientRepo patientRepository;  
 private final DoctorRepo doctorRepository;  
  
 @Override  
 public boolean existsById(Long id) {  
 return appointmentRepository.existsById(id);  
 }  
  
 @Override  
 public List<Appointment> getAllAppointments() {  
 return appointmentRepository.findAll();  
 }  
  
 @Override  
 public Appointment saveAppointment(Appointment appointment) {  
 if (appointment.getServing() != null && appointment.getServing().getId() != null) {  
 Serving serving = servingRepository.findById(appointment.getServing().getId()).orElse(null);  
 appointment.setServing(serving);  
 }  
  
 if (appointment.getPatient() != null && appointment.getPatient().getId() != null) {  
 Patient patient = patientRepository.findById(appointment.getPatient().getId()).orElse(null);  
 appointment.setPatient(patient);  
 }  
  
 if (appointment.getDoctor() != null && appointment.getDoctor().getId() != null) {  
 Doctor doctor = doctorRepository.findById(appointment.getDoctor().getId()).orElse(null);  
 appointment.setDoctor(doctor);  
 }  
  
 appointmentRepository.save(appointment);  
 return appointment;  
 }  
  
 @Override  
 public void deleteAppointment(Long id) {  
 if (appointmentRepository.existsById(id)) {  
 appointmentRepository.deleteById(id);  
 }  
 }  
  
 @Override  
 public List<Appointment> findAll() {  
 return appointmentRepository.findAll();  
 }  
  
 @Override  
 public List<Appointment> findAllByUser(User user) {  
 return appointmentRepository.findAllByUser(user);  
 }  
  
 @Override  
 public Optional<Appointment> findById(Long id) {  
 return appointmentRepository.findById(id);  
 }  
  
 @Override  
 public void deleteAllAppointments() {  
 appointmentRepository.deleteAll();  
 }  
  
 @Override  
 public boolean existsByServingId(Long servingId) {  
 return appointmentRepository.existsByServingId(servingId);  
 }  
  
 @Override  
 public void deleteAppointmentById(Long id) {  
 appointmentRepository.deleteById(id);  
 }  
  
 @Override  
 public void deleteAppointmentsByIds(List<Long> ids) {  
 List<Appointment> dealsToDelete = appointmentRepository.findAllById(ids);  
 appointmentRepository.deleteAll(dealsToDelete);  
 }  
}

package com.example.DoctorPlus.impl;  
  
import com.example.DoctorPlus.model.Doctor;  
import com.example.DoctorPlus.model.User;  
import com.example.DoctorPlus.repo.AppointmentRepo;  
import com.example.DoctorPlus.repo.DoctorRepo;  
import com.example.DoctorPlus.service.DoctorService;  
import lombok.RequiredArgsConstructor;  
import org.springframework.data.domain.Sort;  
import org.springframework.stereotype.Service;  
import java.util.ArrayList;  
import java.util.Arrays;  
import java.util.List;  
import java.util.Optional;  
  
@Service  
@RequiredArgsConstructor  
public class DoctorServiceImpl implements DoctorService {  
  
 private final DoctorRepo doctorRepo;  
 private final AppointmentRepo dealRepo;  
  
 @Override  
 public Optional<Doctor> getDoctorByIdOptional(Long id) {  
 return doctorRepo.findById(id); *// Возвращаем Optional* }  
  
 @Override  
 public List<Doctor> getAllDoctorSortedBy(String sortBy) {  
 List<String> allowedFields = Arrays.asList("surname", "name", "speciality", "schedule");  
  
 if (!allowedFields.contains(sortBy)) {  
 sortBy = "surname";  
 }  
  
 return doctorRepo.findAll(Sort.by(Sort.Direction.ASC, sortBy));  
 }  
  
 @Override  
 public List<Doctor> getAllDoctorsSortedBy(String field) {  
 return doctorRepo.findAll(Sort.by(Sort.Direction.ASC, field));  
 }  
  
 @Override  
 public Doctor getDoctorById(Long id) {  
 return doctorRepo.findById(id).orElse(null);  
 }  
  
 @Override  
 public Doctor saveDoctor(Doctor doctor) {  
 doctorRepo.save(doctor);  
 return doctor;  
 }  
  
 @Override  
 public void deleteAllDoctors() {  
 doctorRepo.deleteAll();  
 }  
  
 @Override  
 public List<Doctor> getAllDoctorsSorted(Sort sort) {  
 return doctorRepo.findAll(sort);  
 }  
  
 @Override  
 public List<Doctor> getAllDoctors() {  
 return doctorRepo.findAll();  
 }  
  
 @Override  
 public Doctor findByUser(User user) {  
 return doctorRepo.findByUser(user)  
 .orElseThrow(() -> new IllegalArgumentException("Доктор не найден"));  
 }  
  
 @Override  
 public List<Doctor> findAll() {  
 return doctorRepo.findAll();  
 }  
  
 @Override  
 public Doctor getDoctorByUserId(Long userId) {  
 return doctorRepo.findByUserId(userId);  
 }  
  
 @Override  
 public List<Doctor> getAllDoctorsWithUsers() {  
 return doctorRepo.findAllWithUser();  
 }  
  
 @Override  
 public void deleteDoctor(Long id) {  
 Doctor doctor = doctorRepo.findById(id).orElse(null);  
 if (doctor == null) return;  
  
 if (doctorRepo.existsById(id)) {  
 throw new IllegalStateException("Doctor is involved in a appointment.");  
 }  
  
 doctorRepo.delete(doctor);  
 }  
  
 @Override  
 public List<Doctor> deleteAllExceptWithAppointments() {  
 List<Doctor> allDoctors = doctorRepo.findAll();  
 List<Doctor> undeleted = new ArrayList<>();  
  
 for (Doctor doctor : allDoctors) {  
 if (dealRepo.existsByDoctor(doctor)) {  
 undeleted.add(doctor);  
 } else {  
 doctorRepo.delete(doctor);  
 }  
 }  
 return undeleted;  
 }  
}

package com.example.DoctorPlus.impl;  
  
import com.example.DoctorPlus.model.Patient;  
import com.example.DoctorPlus.repo.AppointmentRepo;  
import com.example.DoctorPlus.repo.PatientRepo;  
import com.example.DoctorPlus.service.PatientService;  
import lombok.RequiredArgsConstructor;  
import org.springframework.data.domain.Sort;  
import org.springframework.stereotype.Service;  
  
import java.util.List;  
import java.util.Optional;  
  
@Service  
@RequiredArgsConstructor  
public class PatientServiceImpl implements PatientService {  
  
 private final PatientRepo patientRepository;  
 private final AppointmentRepo appointmentRepository;  
 private static final List<String> ALLOWED\_FIELDS = List.of("name", "surname", "birth", "phone", "email", "address");  
  
 @Override  
 public List<Patient> findAll() {  
 return patientRepository.findAll();  
 }  
  
 @Override  
 public List<Patient> 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 "birth" -> "birth";  
 case "phone" -> "phone";  
 case "email" -> "email";  
 case "address" -> "address";  
 default -> fieldName;  
 };  
  
 return patientRepository.findAll(Sort.by(direction, actualFieldName));  
 }  
  
 @Override  
 public Optional<Patient> findById(Long id) {  
 return patientRepository.findById(id);  
 }  
  
 @Override  
 public Patient save(Patient patient) {  
 patientRepository.save(patient);  
 return patient;  
 }  
  
 @Override  
 public void deleteById(Long id) {  
 patientRepository.deleteById(id);  
 }  
  
 @Override  
 public void deleteAll() {  
 patientRepository.deleteAll();  
 }  
  
 @Override  
 public boolean deleteByIdIfPossible(Long id) {  
 Optional<Patient> optionalPatient = patientRepository.findById(id);  
 if (optionalPatient.isPresent()) {  
 Patient patient = optionalPatient.get();  
 if (!appointmentRepository.existsByPatient(patient)) {  
 patientRepository.delete(patient);  
 return true;  
 }  
 }  
 return false;  
 }  
  
 @Override  
 public int deleteAllExceptWithAppointments() {  
 List<Patient> allPatients = patientRepository.findAll();  
 int notDeletedCount = 0;  
 for (Patient patient : allPatients) {  
 if (!appointmentRepository.existsByPatient(patient)) {  
 patientRepository.delete(patient);  
 } else {  
 notDeletedCount++;  
 }  
 }  
 return notDeletedCount;  
 }  
}

package com.example.DoctorPlus.impl;  
  
import com.example.DoctorPlus.model.Doctor;  
import com.example.DoctorPlus.model.Serving;  
import com.example.DoctorPlus.model.User;  
import com.example.DoctorPlus.repo.AppointmentRepo;  
import com.example.DoctorPlus.repo.DoctorRepo;  
import com.example.DoctorPlus.repo.ServingRepo;  
import com.example.DoctorPlus.service.ServingService;  
import lombok.RequiredArgsConstructor;  
import org.springframework.stereotype.Service;  
  
import java.util.List;  
import java.util.stream.Collectors;  
  
@Service  
@RequiredArgsConstructor  
public class ServingServiceImpl implements ServingService {  
  
 private final ServingRepo servingRepo;  
 private final DoctorRepo doctorRepo;  
 private final AppointmentRepo appointmentRepo;  
  
 @Override  
 public boolean existsByServingId(Long id) {  
 return servingRepo.existsById(id);  
 }  
  
 @Override  
 public Serving getServingById(Long id) {  
 return servingRepo.findById(id).orElse(null);  
 }  
  
 @Override  
 public Serving saveServing(Serving serving) {  
 servingRepo.save(serving);  
 return serving;  
 }  
  
 @Override  
 public void deleteServing(Long id) {  
 servingRepo.deleteById(id);  
 }  
  
 @Override  
 public List<Serving> findAll() {  
 return servingRepo.findAll();  
 }  
  
 @Override  
 public void deleteAll() {  
 servingRepo.deleteAll();  
 }  
  
 @Override  
 public Doctor findByUser(User user) {  
 return doctorRepo.findByUser(user)  
 .orElseThrow(() -> new IllegalArgumentException("Доктор не найден"));  
 }  
  
 @Override  
 public void deleteById(Long id) {  
 servingRepo.deleteById(id);  
 }  
  
 @Override  
 public List<Serving> findAllByPatientUser(User user) {  
 return servingRepo.findAllByPatientUser(user);  
 }  
  
 @Override  
 public int deleteAllExceptInAppointments() {  
 List<Long> servingIdsInAppointments = appointmentRepo.findAll()  
 .stream()  
 .map(deal -> deal.getServing().getId())  
 .distinct()  
 .toList();  
  
 if (servingIdsInAppointments.isEmpty()) {  
 int count = (int) servingRepo.count();  
 servingRepo.deleteAll();  
 return count;  
 }  
  
 return servingRepo.deleteByIdNotIn(servingIdsInAppointments);  
 }  
  
 @Override  
 public int deleteUnlinkedServings() {  
 List<Serving> all = servingRepo.findAll();  
 int deleted = 0;  
 for (Serving serving : all) {  
 if (appointmentRepo.findByServing(serving).isEmpty()) {  
 servingRepo.delete(serving);  
 deleted++;  
 }  
 }  
 return deleted;  
 }  
  
 @Override  
 public int countLinkedServings() {  
 return (int) servingRepo.findAll().stream()  
 .filter(serving -> !appointmentRepo.findByServing(serving).isEmpty()).count();  
 }  
  
 @Override  
 public List<Serving> deleteAllExceptWithAppointments() {  
 List<Serving> all = servingRepo.findAll();  
 List<Serving> toDelete = all.stream()  
 .filter(serving -> appointmentRepo.findByServing(serving).isEmpty())  
 .collect(Collectors.toList());  
 servingRepo.deleteAll(toDelete);  
 return toDelete;  
 }  
}

package com.example.DoctorPlus.impl;  
  
import com.example.DoctorPlus.model.User;  
import com.example.DoctorPlus.service.UserService;  
import lombok.RequiredArgsConstructor;  
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 com.example.DoctorPlus.impl;  
  
import com.example.DoctorPlus.dto.UserDTO;  
import com.example.DoctorPlus.model.Role;  
import com.example.DoctorPlus.model.User;  
import com.example.DoctorPlus.repo.UserRepo;  
import com.example.DoctorPlus.service.UserService;  
import lombok.RequiredArgsConstructor;  
import org.springframework.data.domain.Sort;  
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 update(UserDTO userDTO) {  
 Optional<User> optionalUser = userRepo.findById(userDTO.getId());  
  
 if (optionalUser.isEmpty()) {  
 throw new IllegalArgumentException("Пользователь с ID " + userDTO.getId() + " не найден");  
 }  
  
 User user = optionalUser.get();  
  
 *// Обновляем поля* user.setUsername(userDTO.getUsername());  
 user.setName(userDTO.getName());  
 user.setSurname(userDTO.getSurname());  
  
 *// Обновляем роли* Set<Role> roles = new HashSet<>();  
 for (Role role : userDTO.getRoles()) {  
 roles.add(Role.valueOf(role.toString()));  
 }  
 user.setRoles(roles);  
  
 return userRepo.save(user);  
 }  
  
 @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 com.example.DoctorPlus.model;  
  
import jakarta.persistence.\*;  
import lombok.AllArgsConstructor;  
import lombok.Getter;  
import lombok.NoArgsConstructor;  
import lombok.Setter;  
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 = "appointment")  
public class Appointment {  
  
 @Id  
 @GeneratedValue(strategy = GenerationType.IDENTITY)  
 private Long id;  
  
 @ManyToOne(fetch = FetchType.LAZY)  
 @JoinColumn(name = "serving\_id")  
 private Serving serving;  
  
 @ManyToOne(fetch = FetchType.LAZY)  
 @JoinColumn(name = "patient\_id")  
 private Patient patient;  
  
 @ManyToOne(fetch = FetchType.LAZY)  
 @JoinColumn(name = "doctor\_id")  
 private Doctor doctor;  
  
 @DateTimeFormat(iso = DateTimeFormat.ISO.DATE\_TIME)  
 private LocalDateTime date;  
  
 private String status;  
  
 @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 "—";  
 }  
}

package com.example.DoctorPlus.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 = "doctor")  
public class Doctor {  
  
 @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 speciality;  
  
 private String schedule;  
  
 */\*private String phone;\*/* @OneToMany(mappedBy = "doctor", fetch = FetchType.LAZY)  
 private List<Appointment> appointments = 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 com.example.DoctorPlus.model;  
  
import jakarta.persistence.\*;  
import lombok.AllArgsConstructor;  
import lombok.Getter;  
import lombok.NoArgsConstructor;  
import lombok.Setter;  
  
@Getter  
@Setter  
@NoArgsConstructor  
@AllArgsConstructor  
@Entity  
@Table(name = "patient")  
public class Patient {  
  
 @Id  
 @GeneratedValue(strategy = GenerationType.IDENTITY)  
 private Long id;  
  
 private String name;  
 private String surname;  
 private String patronymic;  
  
 *// Вспомогательный метод для отображения полного имени клиента* public String getFullName() {  
 return surname + " " + name + (patronymic != null ? " " + patronymic : "");  
 }  
   
 private String birth;  
 private String phone;  
 */\*private String ownerType;\*/* private String email;  
 private String address;  
  
 *// Связь OneToOne с User — владелец клиента* @OneToOne(fetch = FetchType.LAZY, optional = true) *// Устанавливаем optional = true* @JoinColumn(name = "user\_id", unique = true, nullable = true) *// Устанавливаем nullable = true* private User user;  
}

package com.example.DoctorPlus.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 com.example.DoctorPlus.model;  
  
import jakarta.persistence.\*;  
import lombok.AllArgsConstructor;  
import lombok.Getter;  
import lombok.NoArgsConstructor;  
import lombok.Setter;  
  
import java.time.LocalDateTime;  
  
@Getter  
@Setter  
@NoArgsConstructor  
@AllArgsConstructor  
@Entity  
@Table(name = "serving")  
public class Serving {  
  
 @Id  
 @GeneratedValue(strategy = GenerationType.IDENTITY)  
 private Long id;  
  
 private String name;  
 private String description;  
 private double cost;  
  
 @Column(name = "created\_at", updatable = false)  
 private LocalDateTime createdAt;  
  
 @PrePersist  
 public void prePersist() {  
 this.createdAt = LocalDateTime.now();  
 }  
  
 @ManyToOne(fetch = FetchType.LAZY)  
 @JoinColumn(name = "appointment\_id")  
 private Appointment appointment;  
  
 @ManyToOne(fetch = FetchType.LAZY)  
 @JoinColumn(name = "patient\_id")  
 private Patient patient;  
   
 @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 com.example.DoctorPlus.model;  
  
import jakarta.persistence.\*;  
import lombok.AllArgsConstructor;  
import lombok.Getter;  
import lombok.NoArgsConstructor;  
import lombok.Setter;  
  
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 Patient patient;  
  
 public String getFullName() {  
 return name + " " + surname;  
 }  
}

package com.example.DoctorPlus.repo;  
  
import com.example.DoctorPlus.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.List;  
  
public interface AppointmentRepo extends JpaRepository<Appointment, Long> {  
  
 List<Appointment> findAll(Sort sort);  
 @Query("SELECT d FROM Appointment d WHERE d.patient.user = :user")  
 List<Appointment> findAllByUser(@Param("user") User user);  
 boolean existsByPatient(Patient patient);  
 boolean existsByDoctor(Doctor doctor);  
 boolean existsByServingId(Long servingId);  
 List<Appointment> findByServing(Serving serving);  
}

package com.example.DoctorPlus.repo;  
  
import com.example.DoctorPlus.model.Doctor;  
import com.example.DoctorPlus.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 DoctorRepo extends JpaRepository<Doctor, Long> {  
 Optional<Doctor> findByUser(User user);  
 List<Doctor> findAll(Sort sort);  
 Doctor findByUserId(Long userId);  
 @Query("SELECT e FROM Doctor e LEFT JOIN FETCH e.user")  
 List<Doctor> findAllWithUser();  
 boolean existsById(Long id);  
}

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

package com.example.DoctorPlus.repo;  
  
import com.example.DoctorPlus.model.Patient;  
import com.example.DoctorPlus.model.Serving;  
import com.example.DoctorPlus.model.User;  
import jakarta.transaction.Transactional;  
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;  
  
@Repository  
public interface ServingRepo extends JpaRepository<Serving, Long> {  
 List<Serving> findByPatient(Patient patient);  
 @Query("SELECT s FROM Serving s WHERE s.patient.user = :user")  
 List<Serving> findAllByPatientUser(@Param("user") User user);  
 *// Удалить все объекты, ID которых не входят в переданный список* @Modifying  
 @Transactional  
 @Query("DELETE FROM Serving r WHERE r.id NOT IN :ids")  
 int deleteByIdNotIn(@Param("ids") List<Long> ids);  
}

package com.example.DoctorPlus.repo;  
  
import com.example.DoctorPlus.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 com.example.DoctorPlus.RestController;  
  
import org.springframework.web.bind.annotation.GetMapping;  
import org.springframework.web.bind.annotation.RestController;  
  
import java.util.Map;  
  
@RestController  
public class AboutApiController {  
  
 @GetMapping("/api/about")  
 public Map<String, Object> aboutApi() {  
 return Map.of(  
 "application", "DoctorPlus",  
 "version", "1.0",  
 "description", "Healthcare management system for clinics and doctors",  
 "author", "Your Name"  
 );  
 }  
}

package com.example.DoctorPlus.RestController;  
  
import com.example.DoctorPlus.model.Appointment;  
import com.example.DoctorPlus.service.AppointmentService;  
import org.springframework.http.HttpStatus;  
import org.springframework.http.ResponseEntity;  
import org.springframework.web.bind.annotation.\*;  
  
import java.util.List;  
  
@RestController  
@RequestMapping("/api/appointments")  
public class AppointmentRestController {  
  
 private final AppointmentService appointmentService;  
  
 public AppointmentRestController(AppointmentService appointmentService) {  
 this.appointmentService = appointmentService;  
 }  
  
 *// Получить все записи* @GetMapping  
 public ResponseEntity<List<Appointment>> getAllAppointments() {  
 List<Appointment> appointments = appointmentService.getAllAppointments();  
 return ResponseEntity.ok(appointments);  
 }  
  
 *// Получить запись по ID* @GetMapping("/{id}")  
 public ResponseEntity<Appointment> getAppointmentById(@PathVariable Long id) {  
 return appointmentService.findById(id)  
 .map(ResponseEntity::ok)  
 .orElseGet(() -> ResponseEntity.status(HttpStatus.NOT\_FOUND).build());  
 }  
  
 *// Создать новую запись* @PostMapping  
 public ResponseEntity<Appointment> createAppointment(@RequestBody Appointment appointment) {  
 Appointment saved = appointmentService.saveAppointment(appointment);  
 return ResponseEntity.status(HttpStatus.CREATED).body(saved);  
 }  
  
 *// Обновить существующую запись* @PutMapping("/{id}")  
 public ResponseEntity<Appointment> updateAppointment(  
 @PathVariable Long id,  
 @RequestBody Appointment appointmentDetails) {  
  
 return appointmentService.findById(id)  
 .map(appointment -> {  
 appointment.setDoctor(appointmentDetails.getDoctor());  
 appointment.setPatient(appointmentDetails.getPatient());  
 appointment.setServing(appointmentDetails.getServing());  
 appointment.setDate(appointmentDetails.getDate());  
 Appointment updated = appointmentService.saveAppointment(appointment);  
 return ResponseEntity.ok(updated);  
 })  
 .orElseGet(() -> ResponseEntity.status(HttpStatus.NOT\_FOUND).build());  
 }  
  
 *// Удалить запись по ID* @DeleteMapping("/{id}")  
 public ResponseEntity<Void> deleteAppointment(@PathVariable Long id) {  
 if (!appointmentService.existsById(id)) {  
 return ResponseEntity.status(HttpStatus.NOT\_FOUND).build();  
 }  
 appointmentService.deleteAppointmentById(id);  
 return ResponseEntity.noContent().build();  
 }  
  
 *// Удалить все записи* @DeleteMapping  
 public ResponseEntity<Void> deleteAllAppointments() {  
 appointmentService.deleteAllAppointments();  
 return ResponseEntity.noContent().build();  
 }  
}

package com.example.DoctorPlus.RestController;  
  
import com.example.DoctorPlus.model.Doctor;  
import com.example.DoctorPlus.service.DoctorService;  
import org.springframework.http.HttpStatus;  
import org.springframework.http.ResponseEntity;  
import org.springframework.web.bind.annotation.\*;  
  
import java.util.List;  
import java.util.Map;  
  
@RestController  
@RequestMapping("/api/doctors")  
public class DoctorRestController {  
  
 private final DoctorService doctorService;  
  
 public DoctorRestController(DoctorService doctorService) {  
 this.doctorService = doctorService;  
 }  
  
 *// Получить всех врачей* @GetMapping  
 public ResponseEntity<List<Doctor>> getAllDoctors() {  
 List<Doctor> doctors = doctorService.getAllDoctorsWithUsers();  
 return ResponseEntity.*ok*(doctors);  
 }  
  
 *// Получить врача по ID* @GetMapping("/{id}")  
 public ResponseEntity<Doctor> getDoctorById(@PathVariable Long id) {  
 return doctorService.getDoctorByIdOptional(id)  
 .map(ResponseEntity::*ok*)  
 .orElseGet(() -> ResponseEntity.*status*(HttpStatus.*NOT\_FOUND*).build());  
 }  
  
 *// Создать нового врача* @PostMapping  
 public ResponseEntity<Doctor> createDoctor(@RequestBody Doctor doctor) {  
 Doctor savedDoctor = doctorService.saveDoctor(doctor);  
 return ResponseEntity.*status*(HttpStatus.*CREATED*).body(savedDoctor);  
 }  
  
 *// Обновить существующего врача* @PutMapping("/{id}")  
 public ResponseEntity<Doctor> updateDoctor(  
 @PathVariable Long id,  
 @RequestBody Doctor doctorDetails) {  
  
 return doctorService.getDoctorByIdOptional(id)  
 .map(doctor -> {  
 doctor.setName(doctorDetails.getName());  
 doctor.setSurname(doctorDetails.getSurname());  
 doctor.setPatronymic(doctorDetails.getPatronymic());  
 doctor.setUser(doctorDetails.getUser());  
 Doctor updated = doctorService.saveDoctor(doctor);  
 return ResponseEntity.*ok*(updated);  
 })  
 .orElseGet(() -> ResponseEntity.*status*(HttpStatus.*NOT\_FOUND*).build());  
 }  
  
 *// Удалить врача по ID* @DeleteMapping("/{id}")  
 public ResponseEntity<Void> deleteDoctor(@PathVariable Long id) {  
 try {  
 doctorService.deleteDoctor(id);  
 return ResponseEntity.*noContent*().build();  
 } catch (Exception e) {  
 return ResponseEntity.*status*(HttpStatus.*NOT\_FOUND*).build();  
 }  
 }  
  
 *// Удалить всех врачей* @DeleteMapping  
 public ResponseEntity<Void> deleteAllDoctors() {  
 List<Doctor> notDeleted = doctorService.deleteAllExceptWithAppointments();  
 if (notDeleted.isEmpty()) {  
 return ResponseEntity.*noContent*().build();  
 } else {  
 return ResponseEntity.*status*(HttpStatus.*CONFLICT*)  
 .header("X-Pending-Deletion", String.*valueOf*(notDeleted.size()))  
 .build();  
 }  
 }  
  
 *// Получить отчество по userId* @GetMapping("/patronymic/{userId}")  
 public ResponseEntity<Map<String, String>> getPatronymicByUserId(@PathVariable Long userId) {  
 Doctor doctor = doctorService.getDoctorByUserId(userId);  
 if (doctor != null && doctor.getPatronymic() != null) {  
 return ResponseEntity.*ok*(Map.*of*("patronymic", doctor.getPatronymic()));  
 } else {  
 return ResponseEntity.*ok*(Map.*of*("patronymic", ""));  
 }  
 }  
}

package com.example.DoctorPlus.RestController;  
  
import com.example.DoctorPlus.model.Patient;  
import com.example.DoctorPlus.service.PatientService;  
import org.springframework.http.HttpStatus;  
import org.springframework.http.ResponseEntity;  
import org.springframework.web.bind.annotation.\*;  
import java.util.List;  
import java.util.Optional;  
  
@RestController  
@RequestMapping("/api/patients")  
public class PatientRestController {  
  
 private final PatientService patientService;  
  
 public PatientRestController(PatientService patientService) {  
 this.patientService = patientService;  
 }  
  
 *// Получить всех пациентов* @GetMapping  
 public ResponseEntity<List<Patient>> getAllPatients() {  
 List<Patient> patients = patientService.findAll();  
 return ResponseEntity.*ok*(patients);  
 }  
  
 *// Получить пациента по ID* @GetMapping("/{id}")  
 public ResponseEntity<Patient> getPatientById(@PathVariable Long id) {  
 Optional<Patient> patientOpt = patientService.findById(id);  
 return patientOpt.map(ResponseEntity::*ok*).orElseGet(() -> ResponseEntity.*status*(HttpStatus.*NOT\_FOUND*).build());  
 }  
  
 *// Создать нового пациента* @PostMapping  
 public ResponseEntity<Patient> createPatient(@RequestBody Patient patient) {  
 Patient savedPatient = patientService.save(patient);  
 return ResponseEntity.*status*(HttpStatus.*CREATED*).body(savedPatient);  
 }  
  
 *// Обновить существующего пациента* @PutMapping("/{id}")  
 public ResponseEntity<Patient> updatePatient(  
 @PathVariable Long id,  
 @RequestBody Patient patientDetails) {  
  
 Optional<Patient> patientOpt = patientService.findById(id);  
  
 if (patientOpt.isEmpty()) {  
 return ResponseEntity.*status*(HttpStatus.*NOT\_FOUND*).build();  
 }  
  
 Patient patient = patientOpt.get();  
 patient.setName(patientDetails.getName());  
 patient.setSurname(patientDetails.getSurname());  
 patient.setPhone(patientDetails.getPhone());  
 patient.setEmail(patientDetails.getEmail());  
 patient.setBirth(patientDetails.getBirth());  
  
 Patient updatedPatient = patientService.save(patient);  
 return ResponseEntity.*ok*(updatedPatient);  
 }  
  
 *// Удалить пациента по ID* @DeleteMapping("/{id}")  
 public ResponseEntity<Void> deletePatient(@PathVariable Long id) {  
 boolean deleted = patientService.deleteByIdIfPossible(id);  
 if (!deleted) {  
 return ResponseEntity.*status*(HttpStatus.*CONFLICT*).build(); *// есть связанные записи* }  
 return ResponseEntity.*noContent*().build();  
 }  
  
 *// Удалить всех пациентов* @DeleteMapping  
 public ResponseEntity<Void> deleteAllPatients() {  
 int notDeletedCount = patientService.deleteAllExceptWithAppointments();  
  
 if (notDeletedCount > 0) {  
 return ResponseEntity.*status*(HttpStatus.*CONFLICT*)  
 .header("X-Not-Deleted", String.*valueOf*(notDeletedCount))  
 .build();  
 }  
  
 return ResponseEntity.*noContent*().build();  
 }  
}

package com.example.DoctorPlus.RestController;  
  
import com.example.DoctorPlus.model.Appointment;  
import com.example.DoctorPlus.model.Serving;  
import com.example.DoctorPlus.model.User;  
import com.example.DoctorPlus.service.AppointmentService;  
import com.example.DoctorPlus.service.ServingService;  
import com.example.DoctorPlus.service.UserService;  
import jakarta.servlet.ServletContext;  
import lombok.RequiredArgsConstructor;  
import org.springframework.beans.factory.annotation.Value;  
import org.springframework.core.io.Resource;  
import org.springframework.core.io.UrlResource;  
import org.springframework.http.HttpHeaders;  
import org.springframework.http.MediaType;  
import org.springframework.http.ResponseEntity;  
import org.springframework.web.bind.annotation.\*;  
import org.springframework.web.multipart.MultipartFile;  
import java.io.File;  
import java.io.IOException;  
import java.net.MalformedURLException;  
import java.security.Principal;  
import java.util.List;  
import java.util.Optional;  
  
@RestController  
@RequestMapping("/api/profile")  
@RequiredArgsConstructor  
public class ProfileRestController {  
  
 private final UserService userService;  
 private final ServingService servingService;  
 private final AppointmentService appointmentService;  
 private final ServletContext servletContext;  
  
 @Value("${upload.path:uploads}")  
 private String uploadPath;  
  
 *// Получить данные текущего пользователя* @GetMapping  
 public ResponseEntity<User> getCurrentUserProfile(Principal principal) {  
 Optional<User> userOpt = userService.findByUsername(principal.getName());  
 return userOpt.map(ResponseEntity::ok).orElseGet(() -> ResponseEntity.status(404).build());  
 }  
  
 *// Получить все записи текущего пользователя* @GetMapping("/appointments")  
 public ResponseEntity<List<Appointment>> getUserAppointments(Principal principal) {  
 Optional<User> userOpt = userService.findByUsername(principal.getName());  
 if (userOpt.isEmpty()) {  
 return ResponseEntity.status(404).build();  
 }  
 List<Appointment> appointments = appointmentService.findAllByUser(userOpt.get());  
 return ResponseEntity.ok(appointments);  
 }  
  
 *// Загрузить аватар* @PostMapping(path = "/avatar", consumes = MediaType.MULTIPART\_FORM\_DATA\_VALUE)  
 public ResponseEntity<String> uploadAvatar(@RequestParam("avatar") MultipartFile avatar, Principal principal) throws IOException {  
 if (avatar == null || avatar.isEmpty()) {  
 return ResponseEntity.badRequest().body("Файл не выбран");  
 }  
  
 String realPath = servletContext.getRealPath(uploadPath);  
 if (realPath == null) {  
 return ResponseEntity.status(500).body("Не удалось определить путь для загрузки");  
 }  
  
 File uploadDir = new File(realPath);  
 if (!uploadDir.exists() && !uploadDir.mkdirs()) {  
 return ResponseEntity.status(500).body("Не удалось создать директорию");  
 }  
  
 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(uploadPath + "/" + filename);  
 userService.save(user);  
 return ResponseEntity.ok("Аватар успешно загружен");  
 } else {  
 return ResponseEntity.status(404).body("Пользователь не найден");  
 }  
 }  
  
 *// Получить аватар по имени файла* @GetMapping("/avatar/{filename:.+}")  
 public ResponseEntity<Resource> getAvatar(@PathVariable String filename) throws MalformedURLException {  
 String realPath = servletContext.getRealPath(uploadPath);  
 File file = new File(realPath, filename);  
 Resource resource = new UrlResource(file.toURI());  
  
 if (resource.exists() && resource.isReadable()) {  
 return ResponseEntity.ok()  
 .header(HttpHeaders.CONTENT\_TYPE, MediaType.IMAGE\_JPEG\_VALUE)  
 .body(resource);  
 } else {  
 return ResponseEntity.status(404).build();  
 }  
 }  
}

package com.example.DoctorPlus.RestController;  
  
import com.example.DoctorPlus.dto.UserDTO;  
import com.example.DoctorPlus.service.UserService;  
import jakarta.validation.Valid;  
import lombok.RequiredArgsConstructor;  
import org.springframework.http.HttpStatus;  
import org.springframework.http.ResponseEntity;  
import org.springframework.web.bind.annotation.\*;  
  
import java.util.HashMap;  
import java.util.Map;  
  
@RestController  
@RequiredArgsConstructor  
@RequestMapping("/api/auth")  
public class RegistrationRestController {  
  
 private final UserService userService;  
  
 @PostMapping("/register")  
 public ResponseEntity<?> registerUser(@Valid @RequestBody UserDTO userDTO) {  
 Map<String, String> response = new HashMap<>();  
  
 if (!userService.isUsernameAvailable(userDTO.getUsername())) {  
 response.put("usernameError", "Логин уже занят");  
 return ResponseEntity.status(HttpStatus.CONFLICT).body(response);  
 }  
  
 if (!userDTO.getPassword().equals(userDTO.getConfirmPassword())) {  
 response.put("passwordMismatch", "Пароли не совпадают");  
 return ResponseEntity.badRequest().body(response);  
 }  
  
 userService.save(userDTO);  
 response.put("message", "Регистрация успешна");  
 return ResponseEntity.ok(response);  
 }  
}

package com.example.DoctorPlus.RestController;  
  
import com.example.DoctorPlus.model.Serving;  
import com.example.DoctorPlus.service.ServingService;  
import org.springframework.http.HttpStatus;  
import org.springframework.http.ResponseEntity;  
import org.springframework.web.bind.annotation.\*;  
  
import java.util.List;  
  
@RestController  
@RequestMapping("/api/servings")  
public class ServingRestController {  
  
 private final ServingService servingService;  
  
 public ServingRestController(ServingService servingService) {  
 this.servingService = servingService;  
 }  
  
 *// Получить все услуги* @GetMapping  
 public ResponseEntity<List<Serving>> getAllServings() {  
 List<Serving> servings = servingService.findAll();  
 return ResponseEntity.ok(servings);  
 }  
  
 *// Получить услугу по ID* @GetMapping("/{id}")  
 public ResponseEntity<Serving> getServingById(@PathVariable Long id) {  
 Serving serving = servingService.getServingById(id);  
 return ResponseEntity.ok(serving);  
 }  
  
 *// Создать новую услугу* @PostMapping  
 public ResponseEntity<Serving> createServing(@RequestBody Serving serving) {  
 Serving savedServing = servingService.saveServing(serving);  
 return ResponseEntity.status(HttpStatus.CREATED).body(savedServing);  
 }  
  
 *// Обновить существующую услугу* @PutMapping("/{id}")  
 public ResponseEntity<Serving> updateServing(  
 @PathVariable Long id,  
 @RequestBody Serving servingDetails) {  
  
 Serving serving = servingService.getServingById(id);  
 serving.setName(servingDetails.getName());  
 serving.setDescription(servingDetails.getDescription());  
 serving.setCost(servingDetails.getCost());  
  
 Serving updatedServing = servingService.saveServing(serving);  
 return ResponseEntity.ok(updatedServing);  
 }  
  
 *// Удалить услугу по ID* @DeleteMapping("/{id}")  
 public ResponseEntity<Void> deleteServing(@PathVariable Long id) {  
 if (servingService.existsByServingId(id)) {  
 return ResponseEntity.status(HttpStatus.CONFLICT)  
 .header("X-Error", "Невозможно удалить: услуга участвует в записях на прием.")  
 .build();  
 }  
 servingService.deleteById(id);  
 return ResponseEntity.noContent().build();  
 }  
  
 *// Удалить все услуги, кроме тех, которые используются* @DeleteMapping  
 public ResponseEntity<Void> deleteAllServings() {  
 List<Serving> deleted = servingService.deleteAllExceptWithAppointments();  
  
 if (deleted.isEmpty()) {  
 return ResponseEntity.status(HttpStatus.CONFLICT)  
 .header("X-Warning", "Все услуги участвуют в записях на прием — ничего не удалено.")  
 .build();  
 }  
  
 return ResponseEntity.status(HttpStatus.OK)  
 .header("X-Deleted-Count", String.valueOf(deleted.size()))  
 .build();  
 }  
}

package com.example.DoctorPlus.RestController;  
  
import com.example.DoctorPlus.dto.UserDTO;  
import com.example.DoctorPlus.model.Role;  
import com.example.DoctorPlus.model.User;  
import com.example.DoctorPlus.service.UserService;  
import lombok.RequiredArgsConstructor;  
import org.springframework.http.HttpStatus;  
import org.springframework.http.ResponseEntity;  
import org.springframework.web.bind.annotation.\*;  
  
import java.util.List;  
import java.util.Set;  
import java.util.stream.Collectors;  
  
@RestController  
@RequiredArgsConstructor  
@RequestMapping("/api/users")  
public class UserRestController {  
  
 private final UserService userService;  
  
 *// Получить всех пользователей* @GetMapping  
 public ResponseEntity<List<UserDTO>> getAllUsers() {  
 List<User> users = userService.findAll();  
 List<UserDTO> userDTOs = users.stream()  
 .map(this::convertToDTO)  
 .collect(Collectors.*toList*());  
 return ResponseEntity.*ok*(userDTOs);  
 }  
  
 *// Получить пользователя по ID* @GetMapping("/{id}")  
 public ResponseEntity<UserDTO> getUserById(@PathVariable Long id) {  
 User user = userService.findById(id);  
 return ResponseEntity.*ok*(convertToDTO(user));  
 }  
  
 *// Создать нового пользователя* @PostMapping  
 public ResponseEntity<UserDTO> createUser(@RequestBody UserDTO userDTO) {  
 User savedUser = userService.save(userDTO);  
 return ResponseEntity.*status*(HttpStatus.*CREATED*).body(convertToDTO(savedUser));  
 }  
  
 *// Обновить пользователя* @PutMapping("/{id}")  
 public ResponseEntity<UserDTO> updateUser(@PathVariable Long id, @RequestBody UserDTO userDTO) {  
 userDTO.setId(id);  
 User updatedUser = userService.update(userDTO);  
 return ResponseEntity.*ok*(convertToDTO(updatedUser));  
 }  
  
 *// Удалить пользователя* @DeleteMapping("/{id}")  
 public ResponseEntity<Void> deleteUser(@PathVariable Long id) {  
 userService.deleteUser(id);  
 return ResponseEntity.*noContent*().build();  
 }  
  
 *// Сортировка пользователей* @GetMapping("/sorted")  
 public ResponseEntity<List<UserDTO>> getUsersSorted(@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();  
 };  
  
 List<UserDTO> userDTOs = users.stream()  
 .map(this::convertToDTO)  
 .collect(Collectors.*toList*());  
  
 return ResponseEntity.*ok*(userDTOs);  
 }  
  
 *// Получить все доступные роли* @GetMapping("/roles")  
 public ResponseEntity<Set<String>> getAllRoles() {  
 Set<String> roleNames = Set.*of*(Role.*values*())  
 .stream()  
 .map(Enum::name)  
 .collect(Collectors.*toSet*());  
 return ResponseEntity.*ok*(roleNames);  
 }  
  
 *// Приватный метод: конвертация User → UserDTO* private UserDTO convertToDTO(User user) {  
 return new UserDTO(  
 user.getId(),  
 user.getUsername(),  
 user.getName(),  
 user.getSurname(),  
 user.getRoles()  
 .stream()  
 .map(r -> r.name().toUpperCase())  
 .collect(Collectors.*toSet*())  
 );  
 }  
}

package com.example.DoctorPlus.service;  
  
import com.example.DoctorPlus.model.Appointment;  
import com.example.DoctorPlus.model.User;  
  
import java.util.List;  
import java.util.Optional;  
  
public interface AppointmentService {  
 List<Appointment> getAllAppointments();  
 Appointment saveAppointment(Appointment appointment);  
 void deleteAppointment(Long id);  
 List<Appointment> findAll();  
 List<Appointment> findAllByUser(User user);  
 Optional<Appointment> findById(Long id);  
 void deleteAllAppointments();  
 boolean existsByServingId(Long id);  
 void deleteAppointmentById(Long id);  
 void deleteAppointmentsByIds(List<Long> ids);  
  
 boolean existsById(Long id);  
}

package com.example.DoctorPlus.service;  
  
import com.example.DoctorPlus.model.Doctor;  
import com.example.DoctorPlus.model.User;  
import org.springframework.data.domain.Sort;  
  
import java.util.List;  
import java.util.Optional;  
  
public interface DoctorService {  
 List<Doctor> getAllDoctorsSortedBy(String sortBy);  
 List<Doctor> getAllDoctorSortedBy(String sortBy);  
 Doctor getDoctorById(Long id);  
 Doctor saveDoctor(Doctor doctor);  
 void deleteDoctor(Long id);  
 void deleteAllDoctors();  
 List<Doctor> getAllDoctors();  
 List<Doctor> getAllDoctorsSorted(Sort sort);  
 Doctor findByUser(User user);  
 List<Doctor> findAll();  
 Doctor getDoctorByUserId(Long userId);  
 List<Doctor> getAllDoctorsWithUsers();  
 List<Doctor> deleteAllExceptWithAppointments();  
  
 Optional<Doctor> getDoctorByIdOptional(Long id);  
}

package com.example.DoctorPlus.service;  
  
import com.example.DoctorPlus.model.Patient;  
  
import java.util.List;  
import java.util.Optional;  
  
public interface PatientService {  
 List<Patient> findAll();  
 List<Patient> findAllSortedByField(String fieldName, String sortDirection);  
 Optional<Patient> findById(Long id);  
 Patient save(Patient patient);  
 void deleteById(Long id);  
 void deleteAll();  
 boolean deleteByIdIfPossible(Long id);  
 int deleteAllExceptWithAppointments();  
}

package com.example.DoctorPlus.service;  
  
import com.example.DoctorPlus.model.Doctor;  
import com.example.DoctorPlus.model.Serving;  
import com.example.DoctorPlus.model.User;  
  
import java.util.List;  
  
public interface ServingService {  
 Serving getServingById(Long id);  
 Serving saveServing(Serving realty);  
 void deleteServing(Long id);  
 List<Serving> findAll();  
 void deleteAll();  
 Doctor findByUser(User user);  
 */\*List<Serving> findAllByOwner(User owner);\*/* List<Serving> findAllByPatientUser(User user);  
 void deleteById(Long id);  
 int deleteAllExceptInAppointments();  
 int deleteUnlinkedServings();  
 int countLinkedServings();  
 List<Serving> deleteAllExceptWithAppointments();  
 boolean existsByServingId(Long id);  
}

package com.example.DoctorPlus.service;  
  
import com.example.DoctorPlus.dto.UserDTO;  
import com.example.DoctorPlus.model.Role;  
import com.example.DoctorPlus.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);  
  
 User update(UserDTO userDTO);  
}

package com.example.DoctorPlus;  
  
import org.springframework.boot.SpringApplication;  
import org.springframework.boot.autoconfigure.SpringBootApplication;  
  
@SpringBootApplication  
public class DoctorPlusApplication {  
 public static void main(String[] args) {  
 SpringApplication.run(DoctorPlusApplication.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>Частная клиника «DoctorPlus» — Форма для записей на прием</title>  
 <title th:text="${appointment.id == null} ? 'Создание записи на прием' : 'Редактирование записи на прием'">Запись на прием</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">  
  
<main class="flex-grow-1 d-flex align-items-center min-vh-100">  
 <div class="container">  
 <div class="row justify-content-center">  
 <div class="col-md-6 col-lg-5">  
 <div class="card shadow-sm border-0 rounded-3">  
 <div class="card-body p-4">  
 <h1 class="text-center mb-4" th:text="${appointment.id == null} ? 'Добавить запись на прием' : 'Редактировать запись на прием'">  
 Запись на приём  
 </h1>  
  
 <form th:action="@{/appointment/save}" th:object="${appointment}" method="post" novalidate>  
 <input type="hidden" th:field="\*{id}" />  
  
 *<!-- Услуги -->* <div class="mb-3">  
 <label for="servingSelect" class="form-label">Услуги</label>  
 <select th:field="\*{serving.id}" class="form-select w-100" id="servingSelect" onchange="updateServingFields()" required>  
 <option value="" disabled th:selected="\*{serving == null}">-- Выберите Услугу --</option>  
 <option th:each="serving : ${servings}"  
 th:value="${serving.id}"  
 th:text="${serving.name + ', ' + serving.description + ', ' + serving.cost}">  
 </option>  
 </select>  
 <div class="invalid-feedback">Пожалуйста, выберите услугу.</div>  
 </div>  
  
 *<!-- Пациент -->* <div class="mb-3">  
 <label for="patientSelect" class="form-label">Пациент</label>  
 <select th:field="\*{patient.id}" class="form-select w-100" id="patientSelect" onchange="updatePatientFields()" required>  
 <option value="" disabled th:selected="\*{patient == null}">-- Выберите пациента --</option>  
 <option th:each="patient : ${patients}"  
 th:value="${patient.id}"  
 th:text="${patient.fullName}">  
 </option>  
 </select>  
 <div class="invalid-feedback">Пожалуйста, выберите пациента.</div>  
 </div>  
  
 *<!-- Доктор -->* <div class="mb-3">  
 <label for="doctorSelect" class="form-label">Доктор</label>  
 <select th:field="\*{doctor.id}" class="form-select w-100" id="doctorSelect" required>  
 <option value="" disabled th:selected="\*{doctor == null}">-- Выберите доктора --</option>  
 <option th:each="doctor : ${doctors}"  
 th:value="${doctor.id}"  
 th:text="${doctor.fullName}">  
 </option>  
 </select>  
 <div class="invalid-feedback">Пожалуйста, выберите доктора.</div>  
 </div>  
  
 *<!-- Дата -->* <div class="mb-3">  
 <label for="dateField" class="form-label">Дата записи</label>  
 <input type="datetime-local" th:field="\*{date}" id="dateField" class="form-control w-100" required />  
 <div class="invalid-feedback">Пожалуйста, укажите дату записи.</div>  
 </div>  
  
 *<!-- Статус -->* <div class="mb-3">  
 <label for="statusField" class="form-label">Статус</label>  
 <select th:field="\*{status}" id="statusField" class="form-select w-100" required>  
 <option value="" th:selected="${appointment.status == null}" disabled>Выберите статус</option>  
 <option value="Активный" th:selected="${appointment.status == 'Активный'}">Активный</option>  
 <option value="Неактивный" th:selected="${appointment.status == 'Неактивный'}">Неактивный</option>  
 </select>  
 <div class="invalid-feedback">Выберите пожалуйста статус.</div>  
 </div>  
  
 <div class="d-grid gap-2 d-md-flex justify-content-between mt-4">  
 <button type="submit" class="btn btn-primary">Сохранить</button>  
 <a th:href="@{/appointment}" class="btn btn-secondary">Отмена</a>  
 </div>  
 </form>  
 </div>  
 </div>  
 </div>  
 </div>  
 </div>  
</main>  
  
*<!-- JSON данные -->*<div id="servingJson" th:attr="data-json=${servingJson}" hidden></div>  
<div id="patientJson" th:attr="data-json=${patientJson}" hidden></div>  
  
<script>  
 function updateServingFields() {  
 const servingElement = document.getElementById("servingJson");  
 const data = servingElement ? servingElement.dataset.json : "[]";  
 const servingData = JSON.parse(data || "[]");  
 const selectedId = document.getElementById("servingSelect").value;  
 const serving = servingData.find(r => r.id == selectedId);  
 }  
  
 function updatePatientFields() {  
 const patientElement = document.getElementById("patientJson");  
 const data = patientElement ? patientElement.dataset.json : "[]";  
 const patientData = JSON.parse(data || "[]");  
 const selectedId = document.getElementById("patientSelect").value;  
 const patient = patientData.find(p => p.id == selectedId);  
 }  
  
 document.addEventListener("DOMContentLoaded", function () {  
 updateServingFields();  
 updatePatientFields();  
  
 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>Частная клиника «DoctorPlus» — Записи на прием</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>Частная клиника «DoctorPlus» — Записи на прием</div>  
 <div class="d-flex gap-2">  
 *<!-- Кнопка "Добавить" -->* <div sec:authorize="hasAnyRole('EMPLOYEE','ADMIN')" class="align-self-center">  
 <a href="/appointment/add" class="btn btn-primary">Добавить</a>  
 </div>  
 *<!-- Кнопка "Удалить все" -->* <div sec:authorize="hasAnyRole('EMPLOYEE','ADMIN')" class="align-self-center">  
 <form th:action="@{/appointment/deleteAll}" method="post"  
 onsubmit="return confirm('Удалить все записи на прием?');" 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="@{/appointment(sortField='date', sortDir='asc')}"  
 th:classappend="${(sortField == 'date' and sortDir == 'asc') ? ' active' : ''}">Дата ↑</a>  
 </li>  
 <li>  
 <a class="dropdown-item" th:href="@{/appointment(sortField='date', sortDir='desc')}"  
 th:classappend="${(sortField == 'date' and sortDir == 'desc') ? ' active' : ''}">Дата ↓</a>  
 </li>  
 <li><hr class="dropdown-divider"></li>  
 <li>  
 <a class="dropdown-item" th:href="@{/appointment}"  
 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>Доктор</th>  
 <th>Пациент</th>  
 <th>Услуги</th>  
 <th sec:authorize="hasAnyRole('EMPLOYEE','ADMIN')">Действия</th>  
 </tr>  
 </thead>  
 <tbody>  
 <tr th:each="appointment : ${appointments}">  
 <td th:text="${#temporals.format(appointment.date, 'dd.MM.yyyy')}">—</td>  
 <td th:text="${appointment?.doctor?.fullName}">—</td>  
 <td th:text="${appointment?.patient?.fullName}">—</td>  
 <td th:text="${appointment?.serving?.name + ', ' + appointment?.serving?.description + ', ' + appointment?.serving?.cost}">—</td>  
 <td sec:authorize="hasAnyRole('EMPLOYEE','ADMIN')">  
 <a th:href="@{/appointment/edit/{id}(id=${appointment.id})}" class="btn btn-edit btn-sm me-1">Изменить</a>  
 <form th:action="@{/appointment/delete/{id}(id=${appointment.id})}" method="post"  
 onsubmit="return confirm('Удалить эту запись на прием?');" style="display:inline-block; margin: 0;">  
 <input type="hidden" th:name="${\_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(appointments)}">  
 <td colspan="6" class="text-center text-muted">Записей на прием не найдено.</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>Частная клиника «DoctorPlus» — Форма докторов</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">  
  
<main class="flex-grow-1 d-flex align-items-center">  
 <div class="container">  
 <div class="row justify-content-center">  
 <div class="col-md-6 col-lg-5">  
 <div class="card shadow-sm border-0 rounded-3 p-4">  
 <h1 class="text-center mb-4" th:text="${doctor.id == null ? 'Добавить доктора' : 'Редактировать доктора'}">  
 Форма доктора  
 </h1>  
  
 <form th:action="@{/doctor/save}" th:object="${doctor}" method="post" novalidate>  
 <input type="hidden" th:field="\*{id}" />  
  
 *<!-- Выбор пользователя -->* <div class="mb-3">  
 <label for="userSelect" class="form-label">Выберите доктора:</label>  
 <select class="form-select w-100" id="userSelect" th:field="\*{user.id}">  
 <option value="">— Не выбрано —</option>  
 <option th:each="user : ${users}"  
 th:value="${user.id}"  
 th:selected="${doctor.user != null and user.id == doctor.user.id}"  
 th:data-name="${user.name}"  
 th:data-surname="${user.surname}"  
 th:text="|${user.username} (${user.name} ${user.surname})|">  
 </option>  
 </select>  
 </div>  
  
 *<!-- Имя -->* <div class="mb-3">  
 <label for="name" class="form-label">Имя:</label>  
 <input type="text" class="form-control w-100" id="name" th:field="\*{name}" required />  
 </div>  
  
 *<!-- Фамилия -->* <div class="mb-3">  
 <label for="surname" class="form-label">Фамилия:</label>  
 <input type="text" class="form-control w-100" id="surname" th:field="\*{surname}" required />  
 </div>  
  
 *<!-- Отчество -->* <div class="mb-3">  
 <label for="patronymic" class="form-label">Отчество:</label>  
 <input type="text" class="form-control w-100" id="patronymic" th:field="\*{patronymic}" />  
 </div>  
  
 *<!-- Специализация -->* <div class="mb-3">  
 <label for="speciality" class="form-label">Специализация:</label>  
 <select th:field="\*{speciality}" id="speciality" class="form-select w-100" required>  
 <option value="Директор">Директор</option>  
 <option value="Главный врач">Главный врач</option>  
 <option value="Исполнительный директор">Исполнительный директор</option>  
 <option value="Терапевт">Терапевт</option>  
 <option value="Гинеколог">Гинеколог</option>  
 <option value="Оториноларинголог ">Оториноларинголог</option>  
 <option value="Психотерапевт">Психотерапевт</option>  
 <option value="Дерматолог">Дерматолог</option>  
 <option value="Офтальмолог">Офтальмолог</option>  
 <option value="Кардиолог">Кардиолог</option>  
 <option value="Эндокринолог">Эндокринолог</option>  
 <option value="Уролог">Уролог</option>  
 </select>  
 <div class="invalid-feedback">Выберите пожалуйста специализацию.</div>  
 </div>  
  
 *<!-- Расписание -->* <div class="mb-3">  
 <label for="schedule" class="form-label">Расписание работы:</label>  
 <input type="text" class="form-control w-100" id="schedule" th:field="\*{schedule}" required />  
 </div>  
  
 *<!-- Кнопки -->* <div class="d-grid gap-2 d-md-flex justify-content-between mt-4">  
 <button type="submit" class="btn btn-primary">Сохранить</button>  
 <a th:href="@{/doctor}" class="btn btn-secondary">Отмена</a>  
 </div>  
 </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>  
 </div>  
 </div>  
 </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>  
  
</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>Частная клиника «DoctorPlus» — доктора</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>Частная клиника «DoctorPlus» — Список докторов</div>  
 <div class="d-flex gap-2">  
 <div sec:authorize="hasAnyRole('EMPLOYEE', 'ADMIN')">  
 <a th:href="@{/doctor/add}" class="btn btn-primary">Добавить</a>  
 <a th:href="@{/doctor/deleteAll}" class="btn btn-danger"  
 onclick="return confirm('Вы уверены, что хотите удалить всех докторов?');">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="@{/doctor(sortBy='surname', sortDir='asc')}"  
 th:classappend="${sortBy == 'surname' and sortDir == 'asc'} ? ' active' : ''">Фамилия ↑</a></li>  
 <li><a class="dropdown-item" th:href="@{/doctor(sortBy='surname', sortDir='desc')}"  
 th:classappend="${sortBy == 'surname' and sortDir == 'desc'} ? ' active' : ''">Фамилия ↓</a></li>  
 <li><a class="dropdown-item" th:href="@{/doctor(sortBy='name', sortDir='asc')}"  
 th:classappend="${sortBy == 'name' and sortDir == 'asc'} ? ' active' : ''">Имя ↑</a></li>  
 <li><a class="dropdown-item" th:href="@{/doctor(sortBy='name', sortDir='desc')}"  
 th:classappend="${sortBy == 'name' and sortDir == 'desc'} ? ' active' : ''">Имя ↓</a></li>  
 <li><a class="dropdown-item" th:href="@{/doctor(sortBy='speciality', sortDir='asc')}"  
 th:classappend="${sortBy == 'position' and sortDir == 'asc'} ? ' active' : ''">Специализация ↑</a></li>  
 <li><a class="dropdown-item" th:href="@{/doctor(sortBy='speciality', sortDir='desc')}"  
 th:classappend="${sortBy == 'position' and sortDir == 'desc'} ? ' active' : ''">Специализация ↓</a></li>  
 <li><hr class="dropdown-divider"></li>  
 <li><a class="dropdown-item" th:href="@{/doctor}"  
 th:classappend="${sortBy == null} ? ' active' : ''">Без сортировки</a></li>  
 </ul>  
 </div>  
 </div>  
</header>  
  
*<!-- Контент -->*<main class="content">  
 *<!-- Сообщение при ошибке удаления одного доктора -->* <div th:if="${param.error}" class="alert alert-danger text-center" role="alert">  
 Невозможно удалить доктора — он участвует в записи на прием.  
 </div>  
  
 *<!-- Сообщение при частичном удалении -->* <div th:if="${param.partialError}" class="alert alert-warning text-center" role="alert">  
 Удалены все доктора, не участвующие в записях на прием. <br/>  
 <span th:text="'Невозможно удалить ' + ${param.partialError} + ' доктор(ов), так как они участвуют в записях на прием.'"></span>  
 </div>  
  
 <table class="table table-bordered table-hover table-striped">  
 <thead class="table-light">  
 <tr>  
 <th>Имя</th>  
 <th>Фамилия</th>  
 <th>Специализация</th>  
 <th>Расписание работы</th>  
 <th>Пользователь</th>  
 <th sec:authorize="hasAnyRole('EMPLOYEE', 'ADMIN')">Действия</th>  
 </tr>  
 </thead>  
  
 <tbody>  
 <tr th:each="doctor : ${doctors}">  
 <td th:text="${doctor.name}">Имя</td>  
 <td th:text="${doctor.surname}">Фамилия</td>  
 <td th:text="${doctor.speciality}">Специализация</td>  
 <td th:text="${doctor.schedule}">Расписание работы</td>  
 <td th:text="${doctor.user != null ? doctor.user.username : '-'}">Пользователь</td>  
 <td class="text-center" sec:authorize="hasAnyRole('EMPLOYEE', 'ADMIN')">  
 <a th:href="@{/doctor/edit/{id}(id=${doctor.id})}" class="btn btn-edit btn-sm me-1">Изменить</a>  
 <a th:href="@{/doctor/delete/{id}(id=${doctor.id})}"  
 class="btn btn-delete btn-sm"  
 onclick="return confirm('Удалить доктора?');">Удалить</a>  
 </td>  
 </tr>  
 <tr th:if="${#lists.isEmpty(doctors)}">  
 <td colspan="6" class="text-center text-muted">Доктора 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>Частная клиника «DoctorPlus» — 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 DoctorPlus</h1>  
  
 <p>Добро пожаловать в информационную систему частной клиники <strong>«DoctorPlus»</strong>. Наша платформа предназначена для эффективного управления медицинскими услугами, записей на прием и пациентами.</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 медицинских услуг с возможностью добавления, редактирования и удаления.</li>  
 <li class="list-group-item">Ведение и контроль записей на прием с медицинскими услугами.</li>  
 <li class="list-group-item">Работа с пациентами и докторами клиники.</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@doctorplus.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 Частная клиника «DoctorPlus». 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="/">Частная клиника «DoctorPlus»</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>  
 <a class="btn btn-outline-primary me-4" th:href="@{/about}" role="button">О системе</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="@{/serving}">Медицинские услуги</a>  
 </li>  
 <li class="nav-item me-3">  
 <a class="nav-link" th:href="@{/appointment}">Записи на прием</a>  
 </li>  
 <li class="nav-item me-3" sec:authorize="hasAnyRole('EMPLOYEE', 'ADMIN')">  
 <a class="nav-link" th:href="@{/patient}">Пациенты</a>  
 </li>  
 <li class="nav-item me-3" sec:authorize="hasAnyRole( 'ADMIN')">  
 <a class="nav-link" th:href="@{/doctor}">Доктора</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>Частная клиника «DoctorPlus» — Главная</title>  
  
 *<!-- Подключение Bootstrap CSS -->* <link href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.3/dist/css/bootstrap.min.css" rel="stylesheet"  
 integrity="sha384-QWTKZyjpPEjISv5WaRU9OFeRpok6YctnYmDr5pNlyT2bRjXh0JMhjY6hW+ALEwIH" crossorigin="anonymous"/>  
  
 *<!-- Подключение Bootstrap Icons -->* <link href="https://cdn.jsdelivr.net/npm/bootstrap-icons @1.11.3/font/bootstrap-icons.min.css" rel="stylesheet">  
</head>  
<body class="d-flex flex-column min-vh-100 bg-light">  
  
<div th:replace="~{/include/header}"></div>  
  
<main class="flex-grow-1">  
 <section class="py-5 text-center bg-white">  
 <div class="container">  
 <i class="bi bi-heart-pulse display-4 text-primary mb-3"></i>  
 <h1 class="display-5 fw-bold">Частная клиника «DoctorPlus»</h1>  
 <p class="lead mt-3">Высококачественные медицинские услуги с заботой о вашем здоровье.</p>  
 </div>  
 </section>  
  
 <section class="container py-5">  
 <div class="row justify-content-center">  
 <div class="col-md-8 text-center">  
 <h2 class="mb-4">О нас</h2>  
 <p class="fs-5 text-muted">  
 Клиника DoctorPlus — это команда высококвалифицированных специалистов,  
 предлагающих широкий спектр медицинских услуг для всей семьи.  
 Мы работаем по передовым технологиям и применяем индивидуальный подход к каждому пациенту.  
 </p>  
 </div>  
 </div>  
 </section>  
  
 <section class="bg-white py-5">  
 <div class="container">  
 <h2 class="text-center mb-5">Наши услуги</h2>  
 <div class="row g-4">  
 <div class="col-md-4">  
 <div class="card h-100 shadow-sm border-0 rounded-3">  
 <div class="card-body">  
 <i class="bi bi-person-vcard fs-1 text-primary mb-3"></i>  
 <h5 class="card-title">Первичный приём</h5>  
 <p class="card-text">Консультация опытного врача с возможностью назначения диагностики и лечения.</p>  
 </div>  
 </div>  
 </div>  
 <div class="col-md-4">  
 <div class="card h-100 shadow-sm border-0 rounded-3">  
 <div class="card-body">  
 <i class="bi bi-stethoscope fs-1 text-primary mb-3"></i>  
 <h5 class="card-title">Специализированные услуги</h5>  
 <p class="card-text">Терапия, хирургия, неврология, кардиология и другие направления.</p>  
 </div>  
 </div>  
 </div>  
 <div class="col-md-4">  
 <div class="card h-100 shadow-sm border-0 rounded-3">  
 <div class="card-body">  
 <i class="bi bi-bar-chart fs-1 text-primary mb-3"></i>  
 <h5 class="card-title">Диагностика</h5>  
 <p class="card-text">Лабораторные исследования и аппаратная диагностика на современном оборудовании.</p>  
 </div>  
 </div>  
 </div>  
 </div>  
 </div>  
 </section>  
  
 <section class="container py-5">  
 <div class="row justify-content-center">  
 <div class="col-md-8 text-center">  
 <h2 class="mb-4">Контакты</h2>  
 <p><i class="bi bi-geo-alt-fill text-primary me-2"></i>г. Белореченск, ул. Благодатная, 64</p>  
 <p><i class="bi bi-telephone-fill text-primary me-2"></i>+7 (906) 434-62-48</p>  
 <p><i class="bi bi-envelope-fill text-primary me-2"></i>info@doctorplus.ru</p>  
 <p><i class="bi bi-clock-fill text-primary me-2"></i>Пн–Пт: 9:00–18:00</p>  
 </div>  
 </div>  
 </section>  
</main>  
  
<div th:replace="~{/include/footer}"></div>  
  
*<!-- Подключение Bootstrap JS -->*<script src="/webjars/bootstrap/5.3.3/js/bootstrap.bundle.min.js"></script>  
  
</body>  
</html>

<!DOCTYPE html>  
<html lang="ru" xmlns:th="http://www.thymeleaf.org/">  
<head>  
 <meta charset="UTF-8">  
 <title>Частная клиника «DoctorPlus» — 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">  
  
<main class="flex-grow-1 d-flex justify-content-center align-items-center">  
 <div class="container col-4 border rounded shadow-sm bg-white p-4">  
 <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 w-100 mt-3" type="submit">Авторизация</button>  
 <p class="mt-3 text-center">  
 <a th:href="@{/include/registration}" class="text-decoration-none">Нет аккаунта? Зарегистрироваться</a>  
 </p>  
 </form>  
 </div>  
</main>  
  
<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 -->* <div class="col-md-8">  
  
 *<!-- Медицинские услуги -->* <div class="card mb-4">  
 <div class="card-body">  
 <h5 class="card-title">Медицинские услуги</h5>  
 <ul th:if="${servingList != null and !#lists.isEmpty(servingList)}" class="list-group">  
 <li th:each="serving : ${servingList}" class="list-group-item">  
 <span th:text="${serving.name + ', ' + serving.description + ', ' + serving.cost}"></span><br/>  
 <span>Добавлено: <span  
 th:text="${#temporals.format(serving.createdAt, 'dd.MM.yyyy')}"></span></span>  
 </li>  
 </ul>  
 <div th:if="${servingList == null or #lists.isEmpty(servingList)}">  
 <p>Медицинская услуга не найдена.</p>  
 </div>  
 <div class="mt-3">  
 <a href="/serving" class="btn btn-outline-secondary btn-sm">Подробнее...</a>  
 </div>  
 </div>  
 </div>  
  
 </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>Частная клиника «DoctorPlus» — 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">  
 <style>main .card {  
 border-radius: 10px;  
 border: none;  
 }</style>  
</head>  
<body class="d-flex flex-column min-vh-100">  
  
<main class="flex-grow-1 d-flex align-items-center">  
 <div class="container mt-5">  
 <div class="row justify-content-center">  
 <div class="col-md-6">  
 <div class="card shadow-sm">  
 <div class="card-body">  
 <h2 class="card-title text-center mb-4">Регистрация</h2>  
 <form th:action="@{/include/registration/save}" method="post" th:object="${userDTO}" class="needs-validation" novalidate>  
 <input type="hidden" th:name="${\_csrf.parameterName}" th:value="${\_csrf.token}"/>  
  
 <div class="mb-3">  
 <label for="username" class="form-label">Логин</label>  
 <input type="text" class="form-control" th:field="\*{username}" id="username" required>  
 <div class="text-danger" th:if="${#fields.hasErrors('username')}" th:errors="\*{username}"></div>  
 </div>  
  
 <div class="mb-3">  
 <label for="name" class="form-label">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="mb-3">  
 <label for="surname" class="form-label">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="mb-3">  
 <label for="password" class="form-label">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="mb-3">  
 <label for="confirmPassword" class="form-label">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>  
  
 <div class="d-grid gap-2 d-md-flex justify-content-between mt-4">  
 <button type="submit" class="btn btn-primary">Зарегистрироваться</button>  
 <a th:href="@{/include/login}" class="btn btn-secondary">Назад ко входу</a>  
 </div>  
 </form>  
 </div>  
 </div>  
 </div>  
 </div>  
 </div>  
</main>  
  
<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">DoctorPlus</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="@{/serving}">  
 <i class="bi bi-geo-alt me-2"></i>Медицинские услуги  
 </a>  
 <a class="btn btn-outline-secondary mb-2 text-start" th:href="@{/appointment}">  
 <i class="bi bi-file-earmark-text me-2"></i>Записи на прием  
 </a>  
  
 <a class="btn btn-outline-secondary mb-2 text-start" th:href="@{/patient}"  
 sec:authorize="hasAnyRole('EMPLOYEE', 'ADMIN')">  
 <i class="bi bi-people me-2"></i>Пациенты  
 </a>  
  
 <a class="btn btn-outline-secondary mb-2 text-start" th:href="@{/doctor}"  
 sec:authorize="hasRole('ADMIN')">  
 <i class="bi bi-person-workspace me-2"></i>Доктора  
 </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>Частная клиника «DoctorPlus» — Форма для пациентов</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">  
  
<main class="flex-grow-1 d-flex align-items-center">  
 <div class="container py-5">  
 <div class="row justify-content-center">  
 <div class="col-md-6">  
 <div class="card shadow-sm border-0 rounded-3 p-4">  
 <h1 class="text-center mb-4" th:text="${#strings.isEmpty(patient.id)} ? 'Добавить пациента' : 'Редактировать пациента'">  
 Пациент  
 </h1>  
  
 <form th:action="@{/patient/save}" th:object="${patient}" method="post" novalidate>  
 <input type="hidden" th:field="\*{id}" />  
  
 *<!-- Имя -->* <div class="mb-3">  
 <label for="name" class="form-label">Имя</label>  
 <input type="text" class="form-control w-100" id="name" th:field="\*{name}" required />  
 </div>  
  
 *<!-- Фамилия -->* <div class="mb-3">  
 <label for="surname" class="form-label">Фамилия</label>  
 <input type="text" class="form-control w-100" id="surname" th:field="\*{surname}" required />  
 </div>  
  
 *<!-- Дата рождения -->* <div class="mb-3">  
 <label for="birth" class="form-label">Дата рождения</label>  
 <input type="date" class="form-control w-100" id="birth" th:field="\*{birth}" required />  
 </div>  
  
 *<!-- Телефон -->* <div class="mb-3">  
 <label for="phone" class="form-label">Телефон</label>  
 <input type="tel" class="form-control w-100" id="phone" th:field="\*{phone}" required  
 pattern="^\+?[0-9\s\-]{7,15}$" title="Введите корректный номер телефона" />  
 </div>  
  
 *<!-- Email -->* <div class="mb-3">  
 <label for="email" class="form-label">Email</label>  
 <input type="email" class="form-control w-100" id="email" th:field="\*{email}" required />  
 </div>  
  
 *<!-- Адрес -->* <div class="mb-3">  
 <label for="address" class="form-label">Адрес</label>  
 <input type="text" class="form-control w-100" id="address" th:field="\*{address}" required />  
 </div>  
  
 *<!-- Кнопки -->* <div class="d-grid gap-2 d-md-flex justify-content-between mt-4">  
 <button type="submit" class="btn btn-primary">Сохранить</button>  
 <a th:href="@{/patient}" class="btn btn-secondary">Отмена</a>  
 </div>  
 </form>  
 </div>  
 </div>  
 </div>  
 </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>Частная клиника «DoctorPlus» — Пациенты</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>Частная клиника «DoctorPlus» — Пациенты</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="@{/patient/add}" class="btn btn-primary">Добавить</a>  
 <form th:action="@{/patient/deleteAll}" method="post" style="display: inline;"  
 sec:authorize="hasAnyRole('EMPLOYEE','ADMIN')"  
 onsubmit="return confirm('Удалить всех пациентов?');">  
 <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="@{/patient(sortField='surname', sortDir='asc')}"  
 th:classappend="${sortField == 'surname' and sortDir == 'asc'} ? ' active' : ''">Фамилия ↑</a></li>  
 <li><a class="dropdown-item" th:href="@{/patient(sortField='surname', sortDir='desc')}"  
 th:classappend="${sortField == 'surname' and sortDir == 'desc'} ? ' active' : ''">Фамилия ↓</a></li>  
 <li><a class="dropdown-item" th:href="@{/patient(sortField='name', sortDir='asc')}"  
 th:classappend="${sortField == 'name' and sortDir == 'asc'} ? ' active' : ''">Имя ↑</a></li>  
 <li><a class="dropdown-item" th:href="@{/patient(sortField='name', sortDir='desc')}"  
 th:classappend="${sortField == 'name' and sortDir == 'desc'} ? ' active' : ''">Имя ↓</a></li>  
 <li><a class="dropdown-item" th:href="@{/patient}"  
 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 пациента — он участвует в записи на прием.  
 </div>  
  
 <div th:if="${param.partialError}" class="alert alert-warning text-center" role="alert">  
 Удалены все пациенты, не участвующие в записях на прием. <br />  
 <span th:text="'Невозможно удалить ' + ${param.partialError} + ' пациент(ов), так как они участвуют в записях на прием.'"></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>Дата рождения</th>  
 <th>Телефон</th>  
 <th>Email</th>  
 <th>Адрес</th>  
 <th sec:authorize="hasAnyRole('EMPLOYEE','ADMIN')">Действия</th>  
 </tr>  
 </thead>  
 <tbody>  
 <tr th:each="patient : ${patients}">  
 <td th:text="${patient.name}"></td>  
 <td th:text="${patient.surname}"></td>  
 <td th:text="${patient.birth}"></td>  
 <td th:text="${patient.phone}"></td>  
 <td th:text="${patient.email}"></td>  
 <td th:text="${patient.address}"></td>  
 <td sec:authorize="hasAnyRole('EMPLOYEE','ADMIN')">  
 <a th:href="@{/patient/edit/{id}(id=${patient.id})}" class="btn btn-sm btn-edit me-1">Редактировать</a>  
  
 <form th:id="'deleteForm\_' + ${patient.id}"  
 th:action="@{/patient/delete/{id}(id=${patient.id})}"  
 method="post" style="display: inline-block;"  
 onsubmit="return confirm('Удалить пациента?');">  
 <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\_' + ${patient.id}" class="delete-confirmation" style="display:none; margin-top: 0.3rem;">  
 <span>Удалить пациента?</span>  
 <button type="button" class="btn btn-sm btn-danger ms-2" th:attr="onclick=|submitDeleteForm(${patient.id})|">Подтвердить</button>  
 <button type="button" class="btn btn-sm btn-secondary ms-1" th:attr="onclick=|hideDeleteConfirm(${patient.id})|">Отмена</button>  
 </div>  
 </td>  
 </tr>  
 <tr th:if="${#lists.isEmpty(patients)}">  
 <td colspan="6" class="text-center text-muted">Пациенты 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>Частная клиника «DoctorPlus» — Форма для медицинских услуг</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">  
  
<main class="flex-grow-1 d-flex align-items-center">  
 <div class="container py-5">  
 <div class="row justify-content-center">  
 <div class="col-md-8 col-lg-6">  
 <div class="card shadow-sm border-0 rounded-3">  
 <div class="card-header text-center">  
 <h1 class="mb-0" th:text="${#strings.isEmpty(serving.id)} ? 'Добавить медицинскую услугу' : 'Редактировать медицинскую услугу'">  
 Медицинская услуга  
 </h1>  
 </div>  
 <form th:action="@{/serving/save}" th:object="${serving}" method="post" novalidate>  
 <div class="card-body">  
  
 <input type="hidden" th:field="\*{id}"/>  
  
 *<!-- Название услуги -->* <div class="mb-3">  
 <label for="name" class="form-label">Название услуги:</label>  
 <input type="text" class="form-control w-100" id="name" th:field="\*{name}" required/>  
 </div>  
  
 *<!-- Описание услуги -->* <div class="mb-3">  
 <label for="description" class="form-label">Описание услуги:</label>  
 <input type="text" class="form-control w-100" id="description" th:field="\*{description}" required/>  
 </div>  
  
 *<!-- Стоимость -->* <div class="mb-3">  
 <label for="cost" class="form-label">Стоимость (₽):</label>  
 <input type="number" step="0.01" class="form-control w-100" id="cost" th:field="\*{cost}" required min="0"/>  
 </div>  
  
 </div>  
 <div class="card-footer d-flex justify-content-between">  
 <button type="submit" class="btn btn-primary">Сохранить</button>  
 <a th:href="@{/serving}" class="btn btn-secondary">Отмена</a>  
 </div>  
 </form>  
 </div>  
 </div>  
 </div>  
 </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>Частная клиника «DoctorPlus» — Медицинские услуги</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>Частная клиника «DoctorPlus» — Медицинские услуги</div>  
 <div class="d-flex gap-2">  
 *<!-- Кнопка "Добавить" -->* <div sec:authorize="hasAnyRole('EMPLOYEE','ADMIN')" class="align-self-center">  
 <a href="/serving/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="@{/serving/delete}" method="post"  
 onsubmit="return confirm('Удалить весь список медицинских услуг?');">  
 <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="@{/serving(sortField='cost', sortDir='asc')}"  
 th:classappend="${sortField == 'cost' and sortDir == 'asc'} ? ' active' : ''">Стоимость ↑</a>  
 </li>  
 <li>  
 <a class="dropdown-item" th:href="@{/serving(sortField='cost', sortDir='desc')}"  
 th:classappend="${sortField == 'cost' and sortDir == 'desc'} ? ' active' : ''">Стоимость ↓</a>  
 </li>  
 <li>  
 <a class="dropdown-item" th:href="@{/serving(sortField='name', sortDir='asc')}"  
 th:classappend="${sortField == 'region' and sortDir == 'asc'} ? ' active' : ''">Название ↑</a>  
 </li>  
 <li>  
 <a class="dropdown-item" th:href="@{/serving(sortField='region', sortDir='desc')}"  
 th:classappend="${sortField == 'region' and sortDir == 'desc'} ? ' active' : ''">Название ↓</a>  
 </li>  
 <li><hr class="dropdown-divider"></li>  
 <li>  
 <a class="dropdown-item" th:href="@{/serving}"  
 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>Название услуги</th>  
 <th>Описание услуги</th>  
 <th>Стоимость</th>  
 <th sec:authorize="hasAnyRole('EMPLOYEE','ADMIN')">Действия</th>  
 </tr>  
 </thead>  
 <tbody>  
 <tr th:each="serving : ${servings}">  
 <td th:text="${serving.name}"></td>  
 <td th:text="${serving.description}"></td>  
 <td th:text="${serving.formattedCost}"></td>  
  
 <td class="text-center" sec:authorize="hasAnyRole('EMPLOYEE','ADMIN')">  
 <a th:href="@{/serving/edit/{id}(id=${serving.id})}" class="btn btn-edit btn-sm me-1">Изменить</a>  
 <form th:action="@{/serving/delete/{id}(id=${serving.id})}" method="post"  
 th:onsubmit="return confirm('Удалить эту услугу?');" 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(servings)}">  
 <td colspan="5" class="text-center text-muted">Услуга 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>Частная клиника «DoctorPlus» — 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>Частная клиника «DoctorPlus» — 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>Частная клиника «DoctorPlus» — 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>Частная клиника «DoctorPlus» — 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>

spring.application.name=doctor\_plus  
  
spring.datasource.url=jdbc:mysql://localhost:3306/doctor\_plus  
spring.datasource.username=root  
spring.datasource.password=  
  
spring.jpa.hibernate.ddl-auto=update  
spring.jpa.show-sql=true  
  
spring.thymeleaf.cache=false

*<?*xml version="1.0" encoding="UTF-8"*?>*<project xmlns="http://maven.apache.org/POM/4.0.0"  
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"  
 xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 https://maven.apache.org/xsd/maven-4.0.0.xsd ">  
  
 <modelVersion>4.0.0</modelVersion>  
  
 <parent>  
 <groupId>org.springframework.boot</groupId>  
 <artifactId>spring-boot-starter-parent</artifactId>  
 <version>3.5.3</version>  
 <relativePath/> *<!-- lookup parent from repository -->* </parent>  
  
 <groupId>com.example</groupId>  
 <artifactId>DoctorPlus</artifactId>  
 <version>0.0.1-SNAPSHOT</version>  
 <name>${project.artifactId}</name>  
 <description>Demo project for Spring Boot</description>  
 <packaging>jar</packaging>  
  
 <url>https://doctorplus.app </url>  
  
 <licenses>  
 <license>  
 <name>Apache License, Version 2.0</name>  
 <url>https://www.apache.org/licenses/LICENSE-2.0.txt </url>  
 <distribution>manual</distribution>  
 </license>  
 </licenses>  
  
 <developers>  
 <developer>  
 <name>Developer Name</name>  
 <email>dev@example.com</email>  
 </developer>  
 </developers>  
  
 <scm>  
 <connection>scm:git:https://github.com/d1master1/DoctorPlus.git </connection>  
 <developerConnection>scm:git:https://github.com/d1master1/DoctorPlus.git </developerConnection>  
 <url>https://github.com/d1master1/DoctorPlus </url>  
 <tag>HEAD</tag>  
 </scm>  
  
 <properties>  
 <java.version>21</java.version>  
 <spring-boot.version>3.5.3</spring-boot.version>  
 <thymeleaf.extras.springsecurity.version>3.1.3.RELEASE</thymeleaf.extras.springsecurity.version>  
 </properties>  
  
 <dependencies>  
  
 *<!-- OpenAPI / Swagger -->* <dependency>  
 <groupId>org.springdoc</groupId>  
 <artifactId>springdoc-openapi-starter-webmvc-ui</artifactId>  
 <version>2.4.0</version>  
 </dependency>  
  
 *<!-- Тестирование -->* <dependency>  
 <groupId>org.mockito</groupId>  
 <artifactId>mockito-inline</artifactId>  
 <version>5.2.0</version>  
 <scope>test</scope>  
 </dependency>  
 <dependency>  
 <groupId>org.junit.jupiter</groupId>  
 <artifactId>junit-jupiter</artifactId>  
 <version>5.9.2</version>  
 <scope>test</scope>  
 </dependency>  
 <dependency>  
 <groupId>org.springframework.security</groupId>  
 <artifactId>spring-security-test</artifactId>  
 <scope>test</scope>  
 </dependency>  
 <dependency>  
 <groupId>org.springframework.boot</groupId>  
 <artifactId>spring-boot-starter-test</artifactId>  
 <scope>test</scope>  
 </dependency>  
  
 *<!-- Jackson JSON -->* <dependency>  
 <groupId>com.fasterxml.jackson.core</groupId>  
 <artifactId>jackson-databind</artifactId>  
 <version>2.15.2</version>  
 </dependency>  
  
 *<!-- Thymeleaf + Spring Security -->* <dependency>  
 <groupId>org.thymeleaf.extras</groupId>  
 <artifactId>thymeleaf-extras-springsecurity6</artifactId>  
 <version>3.1.3.RELEASE</version>  
 </dependency>  
  
 *<!-- Lombok -->* <dependency>  
 <groupId>org.projectlombok</groupId>  
 <artifactId>lombok</artifactId>  
 <version>1.18.36</version>  
 <scope>provided</scope>  
 </dependency>  
  
 *<!-- Logging -->* <dependency>  
 <groupId>org.slf4j</groupId>  
 <artifactId>slf4j-api</artifactId>  
 <version>2.0.17</version>  
 </dependency>  
 <dependency>  
 <groupId>ch.qos.logback</groupId>  
 <artifactId>logback-classic</artifactId>  
 <version>1.5.18</version>  
 </dependency>  
  
 *<!-- Spring Boot Starters -->* <dependency>  
 <groupId>org.springframework.boot</groupId>  
 <artifactId>spring-boot-starter-actuator</artifactId>  
 </dependency>  
 <dependency>  
 <groupId>org.springframework.boot</groupId>  
 <artifactId>spring-boot-starter-jdbc</artifactId>  
 </dependency>  
 <dependency>  
 <groupId>org.springframework.boot</groupId>  
 <artifactId>spring-boot-starter-security</artifactId>  
 </dependency>  
 <dependency>  
 <groupId>org.springframework.boot</groupId>  
 <artifactId>spring-boot-starter-thymeleaf</artifactId>  
 </dependency>  
 <dependency>  
 <groupId>org.springframework.boot</groupId>  
 <artifactId>spring-boot-starter-web</artifactId>  
 </dependency>  
 <dependency>  
 <groupId>org.springframework.boot</groupId>  
 <artifactId>spring-boot-starter-data-jpa</artifactId>  
 </dependency>  
 <dependency>  
 <groupId>org.springframework.boot</groupId>  
 <artifactId>spring-boot-starter-validation</artifactId>  
 </dependency>  
  
 *<!-- Devtools -->* <dependency>  
 <groupId>org.springframework.boot</groupId>  
 <artifactId>spring-boot-devtools</artifactId>  
 <scope>runtime</scope>  
 <optional>true</optional>  
 </dependency>  
  
 *<!-- MySQL Connector -->* <dependency>  
 <groupId>com.mysql</groupId>  
 <artifactId>mysql-connector-j</artifactId>  
 <version>9.0.0</version>  
 <scope>runtime</scope>  
 </dependency>  
 </dependencies>  
  
 <build>  
 <plugins>  
 *<!-- CycloneDX SBOM -->* <plugin>  
 <groupId>org.cyclonedx</groupId>  
 <artifactId>cyclonedx-maven-plugin</artifactId>  
 <version>2.7.3</version>  
 </plugin>  
  
 *<!-- Spring Boot Maven Plugin -->* <plugin>  
 <groupId>org.springframework.boot</groupId>  
 <artifactId>spring-boot-maven-plugin</artifactId>  
 </plugin>  
  
 *<!-- Compiler Plugin -->* <plugin>  
 <groupId>org.apache.maven.plugins</groupId>  
 <artifactId>maven-compiler-plugin</artifactId>  
 <version>3.11.0</version>  
 <configuration>  
 <source>${java.version}</source>  
 <target>${java.version}</target>  
 </configuration>  
 </plugin>  
  
 *<!-- Surefire Plugin -->* <plugin>  
 <groupId>org.apache.maven.plugins</groupId>  
 <artifactId>maven-surefire-plugin</artifactId>  
 <version>3.0.0-M9</version>  
 </plugin>  
 </plugins>  
 </build>  
</project>