**Министерство науки и высшего образования Российской Федерации**

**ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ**

**«НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО»**

**(Университет ИТМО)**

Факультет  **Инфокоммуникационные технологии**

Направление подготовки **45.03.04 Интеллектуальные системы в гуманитарной сфере**

Образовательная программа **Языковые модели и искусственный интеллект**

**К У Р С О В О Й П Р О Е К Т**

**Тема:** «Сервис для управления медицинскими данными на основе блокчейн»

**Обучающийся:** Куделенская Эмилия Сергеевна, K3162

Санкт-Петербург 2024

# **СОДЕРЖАНИЕ**

[**СОДЕРЖАНИЕ 2**](#_heading=h.gjdgxs)

[**ВВЕДЕНИЕ 2**](#_heading=h.30j0zll)

[1.1 Актуальность рассматриваемой темы 2](#_heading=h.1fob9te)

[1.2 Цель проекта 3](#_heading=h.3znysh7)

[1.3 Задачи проекта 3](#_heading=h.2et92p0)

[**РАБОТА НАД ПРОЕКТОМ 4**](#_heading=h.tyjcwt)

[2.1 Суть проекта 4](#_heading=h.3dy6vkm)

[2.2 Задачи команды 5](#_heading=h.1t3h5sf)

[2.3 Мои задачи 7](#_heading=h.4d34og8)

[**КОММУНИКАЦИЯ В КОМАНДЕ 14**](#_heading=h.2s8eyo1)

[**ЗАКЛЮЧЕНИЕ 14**](#_heading=h.17dp8vu)

[**СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ 15**](#_heading=h.3rdcrjn)

[**ПРИЛОЖЕНИЕ. ТЕХНИЧЕСКОЕ ЗАДАНИЕ 17**](#_heading=h.lnxbz9)

# **ВВЕДЕНИЕ**

## **1.1 Актуальность рассматриваемой темы**

В настоящее время в сфере хранения медицинских данных существует ряд проблем:

– проблема разрозненного хранения данных. В большинстве медицинских учреждений данные пациентов хранятся в локальных базах данных. Каждый пациент имеет отдельные карточки в разных учреждениях, что создаёт серьёзные сложности при попытке собрать полную медицинскую историю. Эта разрозненность препятствует быстрой и точной диагностике, а также снижает качество медицинского обслуживания.

– проблема отсутствия прозрачности доступа к данным.В текущей системе пациенты не имеют контроля над тем, кто и когда просматривает их медицинские данные. Часто такие доступы происходят без ведома пациента, что ставит под угрозу его право на приватность и защиту данных. К тому же пациенты не имеют уверенности в подлинности внесенных записей, так как нет прозрачного механизма отслеживания изменений.

– проблема централизованного хранения с риском утечек данных.В большинстве медицинских учреждений данные хранятся на централизованных серверах, которые часто подвергаются риску взломов и утечек. Учитывая, что многие учреждения сталкиваются с ограничениями по бюджету, уделять внимания мерам информационной безопасности оказывается недостаточным. Это создает условия для возможных утечек конфиденциальной информации, о которых пациент зачастую даже не будет знать.

Разработка безопасной и прозрачной системы хранения медицинских данных на основе блокчейна решает указанные проблемы, улучшая качество медицинского обслуживания.

Результаты проекта предназначены для медицинских учреждений, врачей и пациентов. Благодаря данной системе пациенты смогут получить контроль над своими личными данными, а врачи - доступ к актуальной медицинской информации, что поможет повысить качество обслуживания.

## **1.2 Цель проекта**

Целью данного проекта является разработка серверной части прототипа системы хранения медицинских данных на основе блокчейна, направленной на улучшение качества обслуживания пациентов, а также обеспечение безопасности их личных данных.

## **1.3 Задачи проекта**

Проект решает ряд ключевых задач:

– обеспечение безопасного и децентрализованного хранения данных: Разработка системы с использованием блокчейн-технологий позволит обеспечить более безопасное хранение личных данных пользователей.

– прозрачность операций, связанных с изменением данных: Система будет отслеживать все произведенные изменения и отображать, кто и когда их внес.

– контроль доступа к данным: Для получения доступа к данным пациента врачу будет необходимо запросить его. После получения запроса, пользователь сможет разрешить или запретить врачу доступ к своим данным.

– актуальность и облегчение поиска медицинских данных: Благодаря системе, которая позволит хранить в себе все необходимые для врача медицинские данные о пациенте, повысится качество обслуживания посетителей медицинских учреждений.

Все перечисленные задачи направлены на создание безопасной системы, которая позволит пользователям самостоятельно распоряжаться доступом к своим личным данным, а врачи смогут быстро получать доступ ко всей необходимой для них информации.

# **РАБОТА НАД ПРОЕКТОМ**

## **2.1 Суть проекта**

Система управления медицинскими данными представляет собой сервис для специальных учреждений. Разработанный веб-сайт направлен на улучшение качества обслуживания пациентов, безопасности хранения медицинских данных, а также облегчение поиска необходимой для врачей информации.

Проект был разработан с использованием современных технологий, обеспечивающих кроссплатформенность и высокую производительность. Более подробное описание используемых технологий:

– сервис Figma использовался для разработки дизайна веб-сайта. Это отличный инструмент для командной работы, а также довольно простой в использовании.

– react.js использовался на клиентской части сервиса для обеспечения кроссплатформенности, а также для облегчения работы в команде. Благодаря архитектуре, он позволяет разрабатывать свою часть системы, не думая о том, что можно повредить чужую.

– для создания серверной части сервиса использовался Python, в частности фреймворк Django. Python является довольно удобным и простым инструментом для прототипирования.

– для работы с базой данных использовался PostgreSQL. Это СУБД с открытым исходным кодом и поддержкой множества типов данных, которая отлично подходит для прототипирования.

Таким образом, комбинация React.js, Figma, Python, PostgreSQL обеспечила разработку эффективного и кроссплатформенного прототипа системы управления медицинскими данными с высокой производительностью и удобным пользовательским интерфейсом.

## **2.2 Задачи команды**

В начале работы над проектом командой был составлен план разработки:

– создание и редактирование технического задания

– распределение обязанностей и установка сроков выполнения задач

– выполнение индивидуальных задач

– подготовка к защите проекта

– защита проекта - командная защита с презентацией

– написание индивидуального отчета

Непосредственно на бэкенде, где я работала вместе с Артемом Трикулой, задачи были следующие:

– спроектировать архитектуру базы данных: создать ER-диаграммы, чтобы отобразить структуру данных, основные сущности (таблицы) и связи между ними.

– разработать схемы API-запросов: с помощью Swagger создать прототип запросов. Прототип запросов на Swagger поможет быстрее наладить интеграцию фронтенда с бэкендом. Для доступа к документации API по адресу {serverAddress}/docs настроить URL и конфигурации Swagger в проекте Django.

– настроить контейнеризацию и сборку приложения.

Основные папки:

1. manage.py
2. medchain (основная папка проекта)
3. medchainapi (папка приложения)

Вспомогательные файлы и папки:

1. .gitignore
2. .dockerignore
3. docker-compose.yml
4. dockerfile
5. requirements.txt
6. db.sqlite3
7. migrations/

– развертка на сервере: настроить виртуальную машину или сервер и развернуть приложение. Настроить веб-сервер (например, Nginx), подключить его к Django через Gunicorn или другой WSGI-сервер. На этом этапе проверить работу миграций, подключить базу данных, кэш и фоновые задачи, если они требуются.

Остальным командам в нашем проекте необходимо было выполнить такие задачи, как создание макетов страниц в figma и адаптивов под мобильные устройства в подобранной заранее цветовой палитре и стилистике (команда дизайнеров), сверстать страницы по макетам с учетом адаптива под мобильные устройства и настройка интеграции с бэкендом (команда фронтендеров), написание смарт контрактов и создание примеров интеграции фронтенда и блокчейна (команда блокчейна). С этими задачами наши коллеги так же успешно справились, что позволило нам прийти к созданию полноценного прототипа готового продукта.

## **2.3 Мои задачи**

1. Проектирование архитектуры базы данных

Первым этапом в разработке бэкенда было создание архитектуры базы данных. Для этого была создана ER-диаграмма в Ченовской нотации (см. Рисунок 1), в чем мне помогли знания с университетской дисциплины “Базы данных”, а так же статья на Яндекс Практикуме[1]. Моя диаграмма отразила основные сущности и связи между ними, что впоследствии помогло нам определить структуру базы данных проекта PostgreSQL.

Сущности в рамках проекта включают:

– пациенты.

– врачи.

– запросы доступа: сущность, отражающая запросы врачей на доступ к данным конкретных пациентов.

– журнал действий.

– медицинские учреждения.

Также на диаграмме отражены атрибуты сущностей, ставшие полями в базе данных (см. Рисунок 2). На основе приведенных сущностей и атрибутов была проработана логика сервера, созданы соответствующие классы, модели и сериализаторы.

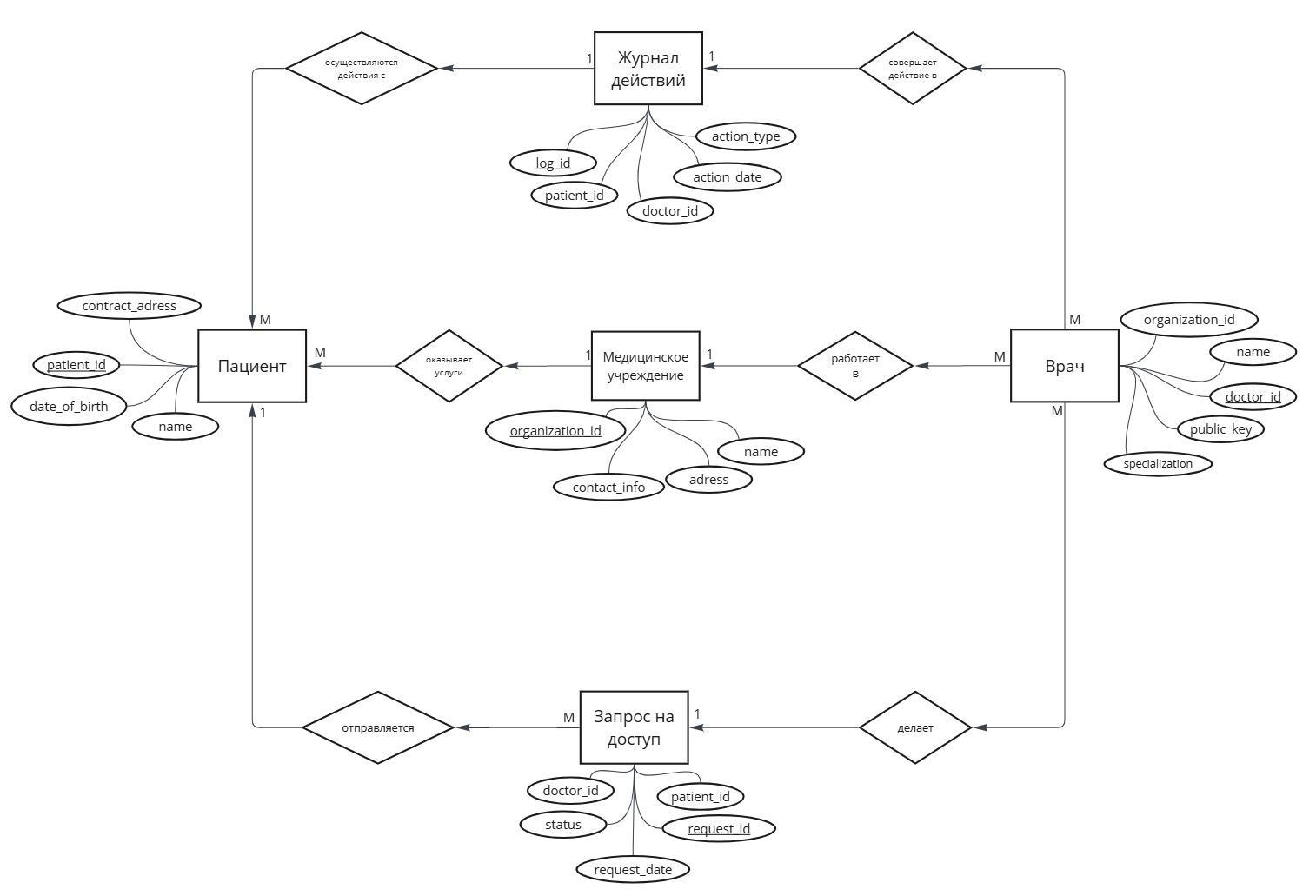


Рисунок 1 – ER-диаграмма

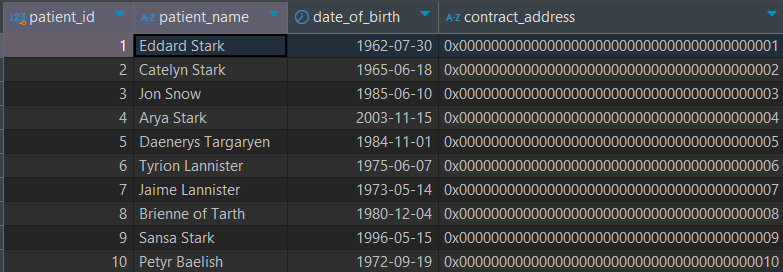


Рисунок 2 – Пример таблицы из базы данных PostgreSQL

2. Разработка схем API-запросов

Следующим шагом стало определение схем API-запросов для взаимодействия между фронтендом и бэкендом. С помощью Swagger (см. Рисунок 3) был создан прототип запросов, чтобы облегчить понимание его структуры и функционала для других разработчиков. С этим мне помогла статья на habr.com[2]. Документация API была настроена по адресу {serverAddress}/docs, что упростило интеграцию с фронтендом. Основные запросы включают:

Запросы для пациентов:

– получение списка запросов доступа.

– подтверждение или отклонение доступа.

– получение списка врачей с доступом.

– добавление пациента.

Запросы для врачей:

– получение данных пациента.

– запрос доступа к данным пациента.

– поиск пациентов.

– запрос "Мои пациенты".

Запросы для больниц:

– запрос на добавление больницы

– запрос на добавление нового врача

– запрос на получение информации о врачах учреждения

Тем не менее, последняя категория запросов не была реализована из-за нехватки времени. Однако основной функционал проекта не пострадал, остальные запросы работают успешно, а запросы для больниц можно в случае необходимости создать по аналогии в короткие сроки.

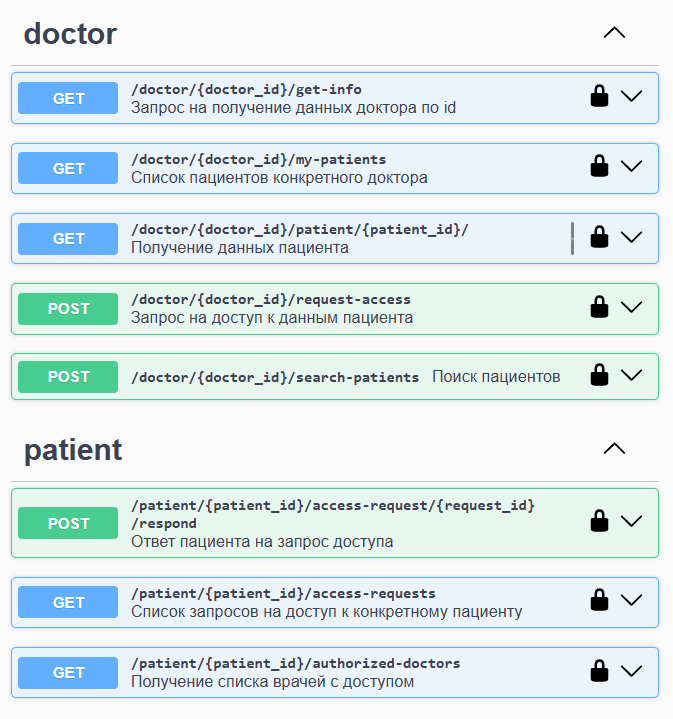


Рисунок 3 – Документация Swagger

3. В рамках проектирования бэкенда для сервиса управления медицинскими данными был разработан набор RESTful API-эндпоинтов, позволяющих пациентам и врачам эффективно взаимодействовать с системой. Так как ранее у меня не было опыта работы с бэкендом и фреймворком Django на Python, мне пришлось осваивать материал с нуля, в чем мне помогли видеокурсы на YouTube[3] - [6], Stepik[7; 8], а также документация Django Framework[9]. Я прописала запросы пациентов, перечисленные в п.2., а мой коллега Артем – запросы докторов. Были созданы соответствующие модели, представления и сериализаторы, а также добавлены маршруты эндпоинтов (см. Рисунок 4).



Рисунок 4. Пример запроса

**Создание сериализаторов**

На основе разработанных моделей (см. Рисунок 5), зарегистрированных в файле admin.py (см. Рисунок 6), были созданы сериализаторы (см. Рисунок 7). Сериализатор — это компонент Django REST Framework, который отвечает за преобразование данных между форматами, пригодными для хранения в моделях (например, объекты Python) и форматами, используемыми в API (например, JSON). Сериализаторы также выполняют валидацию данных, поступающих от клиента. Реализация включала следующие функции:

– сериализация данных: преобразование объектов моделей в формат, удобный для передачи клиенту (например, JSON).

– десериализация данных: преобразование данных, полученных от клиента, в объекты моделей с предварительной проверкой их корректности.

– валидация данных: проверка входящих данных на соответствие заданным правилам, таким как длина строки, уникальность значений или их формат.

– гибкость управления данными: возможность выборочного отображения или скрытия полей в зависимости от потребностей клиента.

Эти подходы позволили упростить обмен данными между сервером и клиентами, а также обеспечить надежность и структурированность взаимодействия.

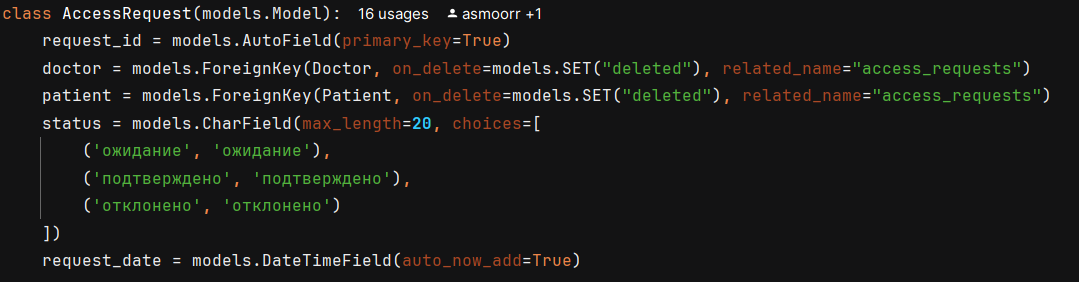


Рисунок 5 – Пример модели

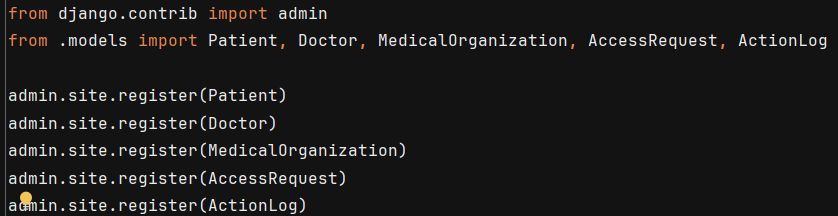


Рисунок 6 – зарегистрированные модели в файле admin.py

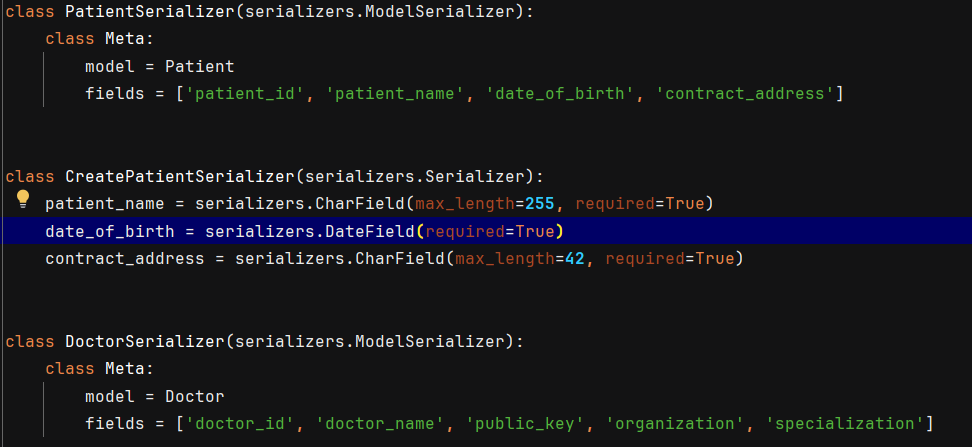


Рисунок 7 – Примеры сериализаторов

**Создание эндпоинтов**

Для обеспечения взаимодействия между врачами, пациентами и системой были разработаны эндпоинты API (см. Рисунок 8). В рамках этой работы выполнено следующее:

– разработка API view: созданы представления (views) для обработки запросов, связанных с данными пациентов, запросами на доступ, подтверждением и управлением доступом.

– взаимодействие с моделями: реализована логика для получения, обновления и сохранения данных в моделях.

– дополнительная валидация данных: добавлены проверки корректности данных перед их обработкой.

– обработка ошибок: реализована обработка ошибок, связанных с пользовательским вводом, для обеспечения стабильной работы системы и удобства взаимодействия с API.

Эти шаги позволили создать удобное и надежное API для взаимодействия между участниками системы.

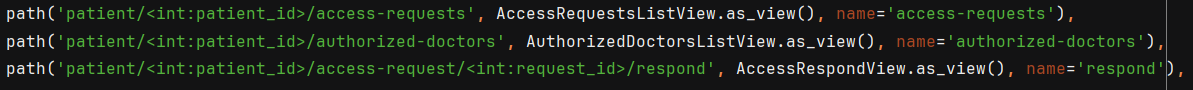


Рисунок 8 – Эндпоинты для запросов пациентов

# **КОММУНИКАЦИЯ В КОМАНДЕ**

Процесс работы был организован по составленному в начале выполнения проекта плану, что включало регулярные совещания, где мы представляли наши результаты, делились успехами и неудачами и помогали друг другу справиться с трудностями.

В серверной части вебсайта на бэкенде со мной работал Трикула Артем, с которым мы быстро нашли общий язык и часто коммуницировали в ходе работы, помогая друг другу справиться с поставленными перед нами задачами.

Помимо бэкенда в серверной части была команда блокчейна, которая действовала автономно, ведь в цели данного учебного проекта по созданию прототипа сервиса не входила интеграция бэкенда с блокчейном. В клиентской же части были команды фронтендеров и дизайнеров, с которыми также было не очень много взаимодействия. Лишь фронтендеры иногда писали нам на бэкенд по поводу некоторых запросов, мы получали от них обратную связь и могли исправить наши ошибки, поэтому можно считать, что наше с ними взаимодействие было продуктивным и полезным.

Наш руководитель Лаврова Анастасия внесла большой вклад в наш труд, оперативно отвечала на вопросы, хвалила за достижения и помогала решить проблемы, ее указания были точными, а подсказки понятными, поэтому, по моему мнению, она заслуживает наивысшей оценки.

# **ЗАКЛЮЧЕНИЕ**

В рамках выполнения данной проектной работы были выполнены практически все поставленные изначально задачи, за исключением ранее упомянутых запросов для больниц, которые не были реализованы по причине нехватки времени, и достигнута поставленная нами цель. Были созданы все основные части нашего проекта: клиентская часть сайта, блокчейн, а также серверная часть проекта, включающая в себя бэкенд и базы данных.

Я считаю, что внесла ощутимый вклад в достижение этого результата, так как я старалась выполнять все поставленные передо мной задачи, а также активно участвовала в подготовке к защите проекта.

Стоит отметить возникшие сложности при разработке запросов, для решения которых пришлось изучать разнообразные статьи и видео-материалы на эту тему. Думаю, что этот учебный проект сильно мне поможет в дальнейшей учебной и профессиональной деятельности, ведь, несмотря на трудности, мне удалось познакомиться со сферой сервера и бэкенда, а также я приобрела полезный опыт.

# **СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ**

1. Статья на Яндекс Практикуме по базам данных <https://practicum.yandex.ru/blog/chto-takoe-er-diagramma/>
2. Статья на habr.com про Swagger <https://habr.com/ru/articles/733868/>
3. Видео на YouTube о Rest API, архитектуре клиент-сервер <https://youtu.be/XaTwnKLQi4A?si=3QEJxT-0kxFMXe3l>
4. Видеокурс на YouTube о Django REST Framework <https://www.youtube.com/playlist?list=PLA0M1Bcd0w8xZA3Kl1fYmOH_MfLpiYMRs>
5. Видеокурс на YouTube о создании сайта на Django <https://www.youtube.com/playlist?list=PLDyJYA6aTY1nZ9fSGcsK4wqeu-xaJksQQ>
6. Видео на YouTube о создании сайта на Python с базой данных с нуля <https://youtu.be/c0YzMwCEje0?si=kcNv5u5CE71sdaA4>
7. Курс на Stepik о разработке бэкенд приложений на Django <https://stepik.org/course/155057/syllabus>
8. Курс на Stepik о создании сайта на Django Rest Framework <https://stepik.org/course/82067/promo?search=6275140107>
9. Документация фреймворка Джанго <https://docs.djangoproject.com/en/5.1/>

# 

# **ПРИЛОЖЕНИЕ. ТЕХНИЧЕСКОЕ ЗАДАНИЕ**

Техническое задание

Сервис для управления медицинскими данными на основе блокчейн

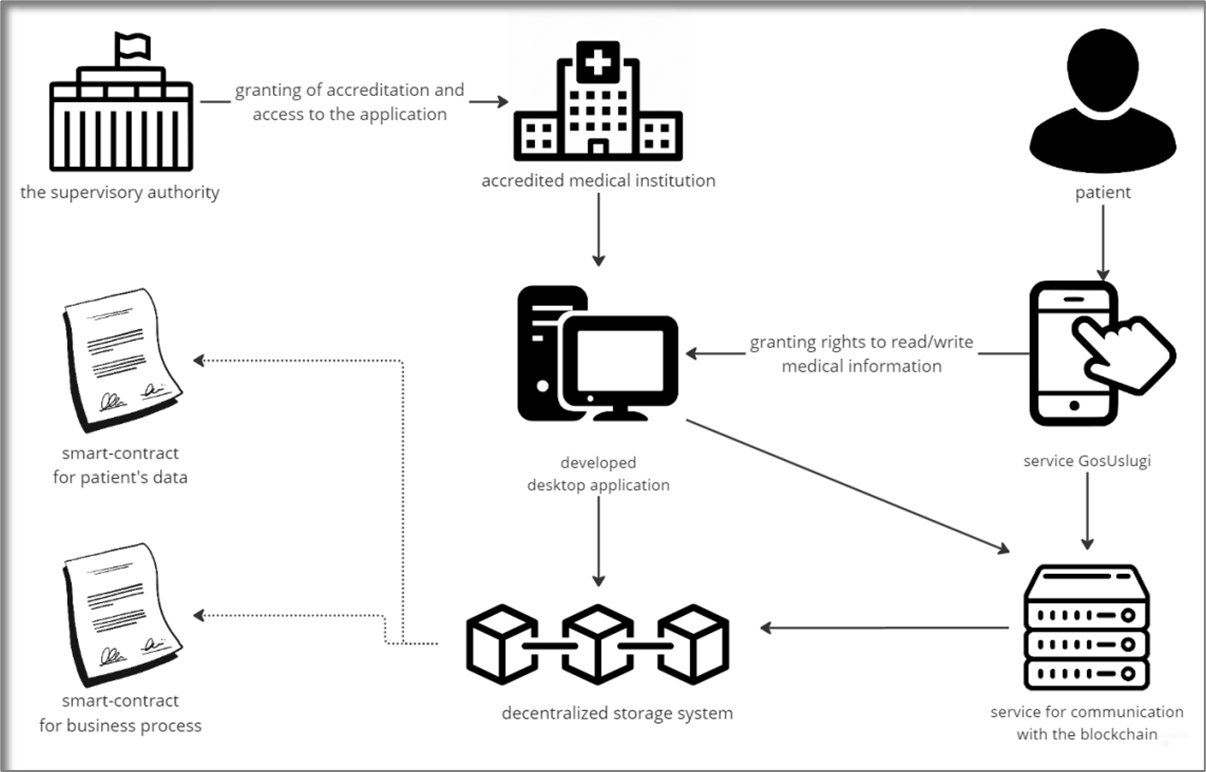
Клиентская часть: Алмазова Л.

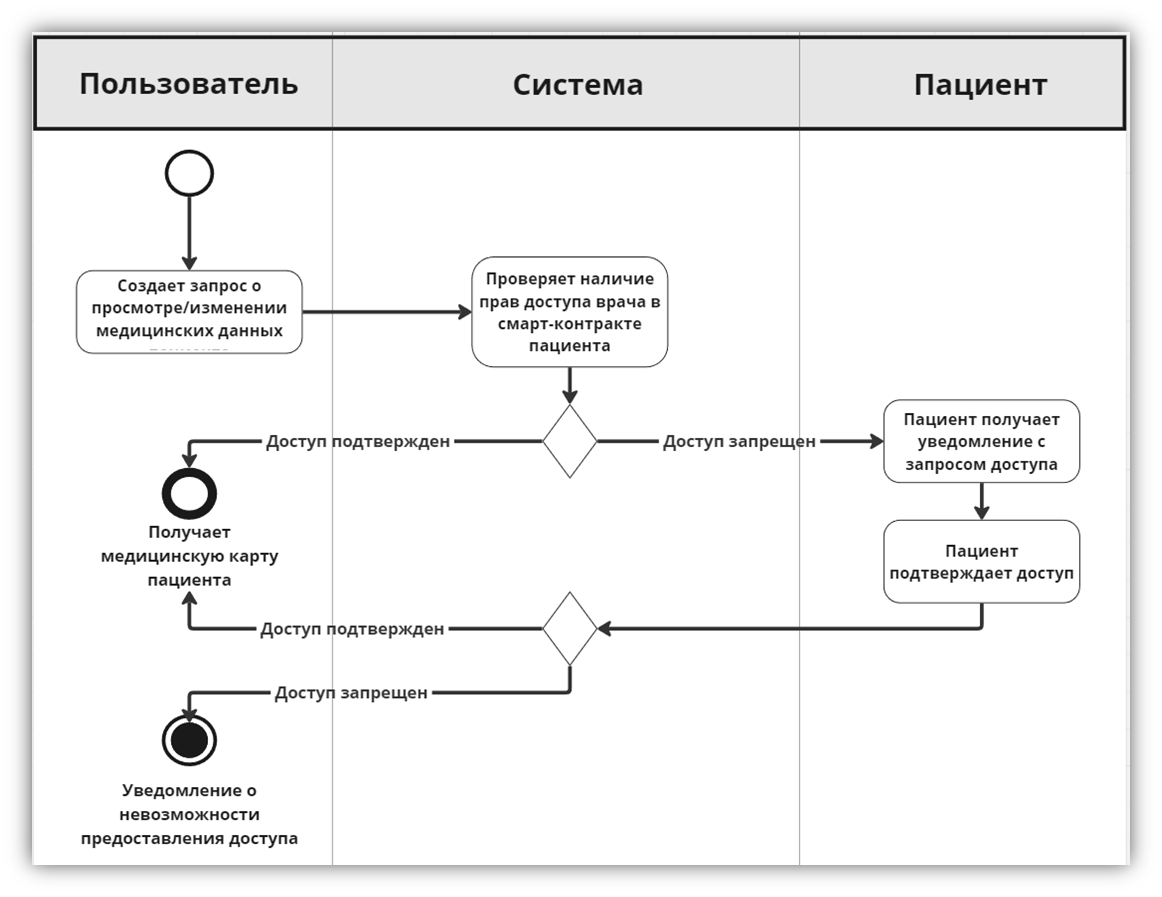
Серверная часть: Лаврова А.К.

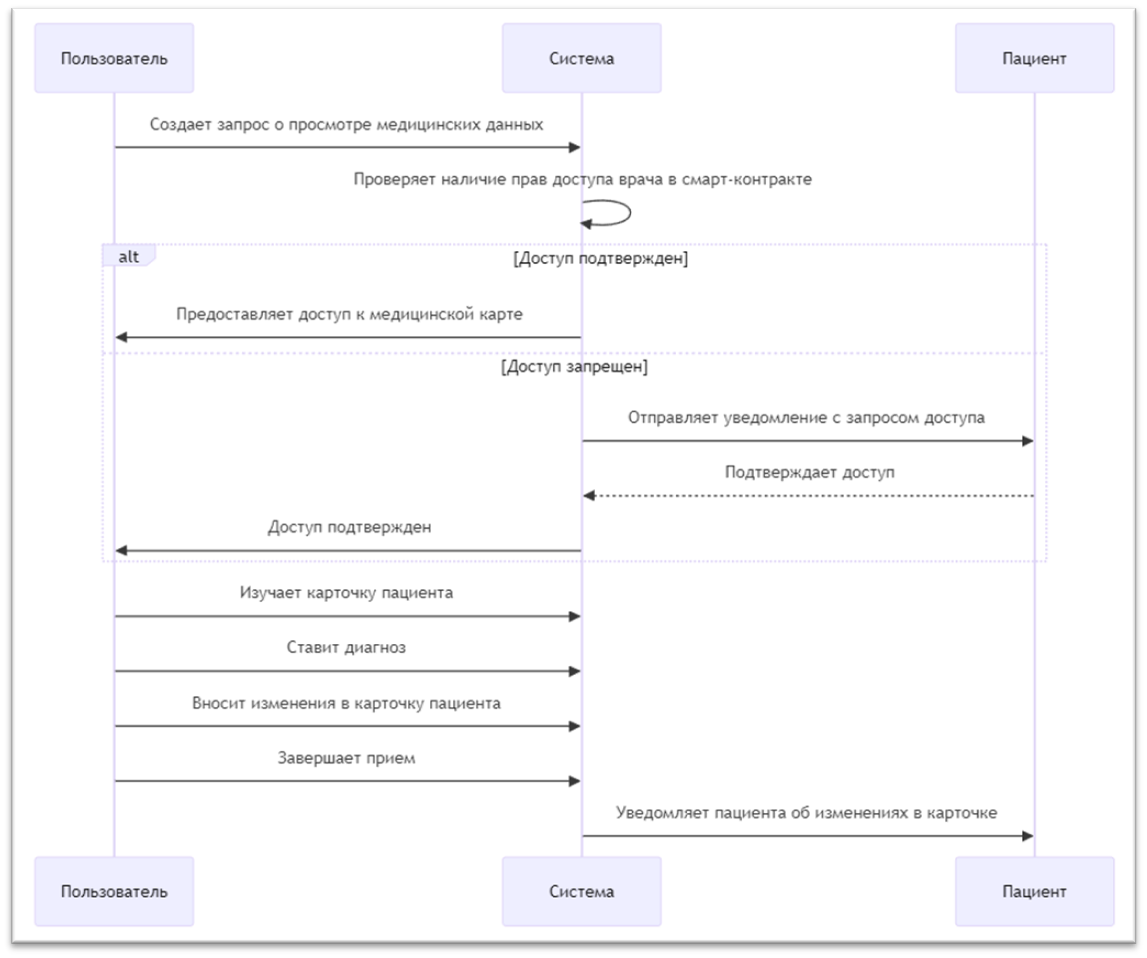
# 1. Общее описание проекта

В настоящее время в сфере хранения медицинских данных существует ряд проблем:

1. **Разрозненное хранение данных**В большинстве медицинских учреждений данные пациентов хранятся в локальных базах данных. Каждый пациент имеет отдельные карточки в разных учреждениях, что создаёт серьёзные сложности при попытке собрать полную медицинскую историю. Эта разрозненность препятствует быстрой и точной диагностике, а также снижает качество медицинского обслуживания.
2. **Отсутствие прозрачности доступа к данным**В текущей системе пациенты не имеют контроля над тем, кто и когда просматривает их медицинские данные. Часто такие доступы происходят без ведома пациента, что ставит под угрозу его право на приватность и защиту данных. К тому же пациенты не имеют уверенности в подлинности внесённых записей, так как нет прозрачного механизма отслеживания изменений.
3. **Централизованное хранение с риском утечек данных**В большинстве медучреждений данные хранятся на централизованных серверах, которые часто подвергаются риску взломов и утечек. Учитывая, что многие учреждения сталкиваются с ограничениями по бюджету, уделять внимания мерам информационной безопасности оказывается недостаточным. Это создаёт условия для возможных утечек конфиденциальной информации, о которых пациент зачастую даже не будет знать.

Общая идея решения выглядит следующим образом:





# Требования

## План MVP

Реализовать следующие сценарии:

1. Пациент заходит на свою страницу (временная замена госуслугам), видит запрос на доступ со стороны врача А. Пользователь подтверждает/отклоняет этот запрос.
2. Пациент заходит на свою страницу и видит перечень врачей, которым выдан доступ к его странице
3. Врач ищет пациента в базе и запрашивает доступ на изменение его данных.
4. Врач ознакамливается с карточкой пациента.
5. Врач вносит запись в карточку пациента.

## Требования к стеку

* Дизайн: Figma
* Front-end: React.JS
* Back-end: Python (Django)
* Блокчейн: Solidity (Hardhat, Ethers.js)

## Полезные ссылки

* Репозиторий GitHub: <https://github.com/EgoInc/MedChainMVP>
* *скоро здесь появится что-то еще*

# 

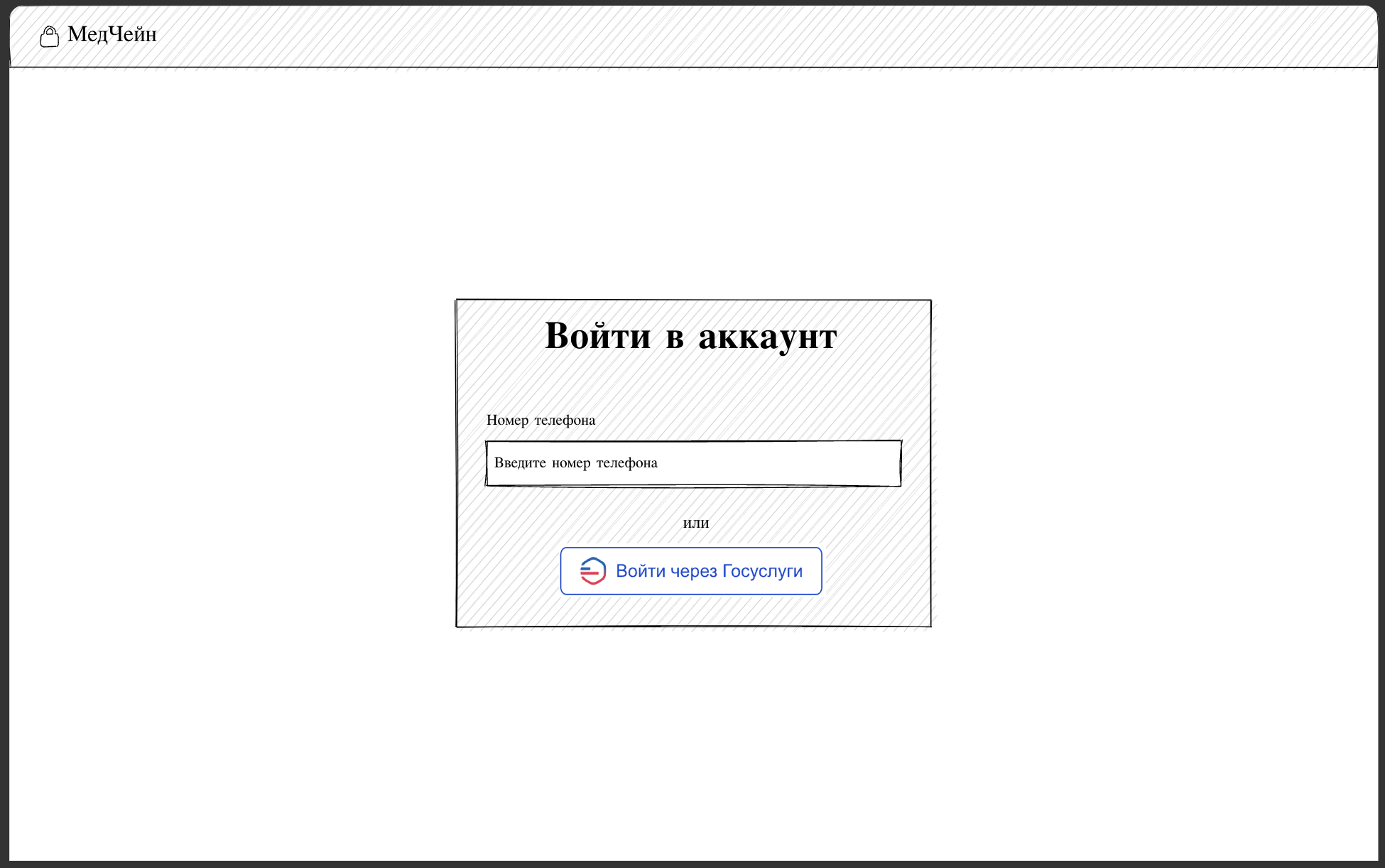
# Задачи дизайн

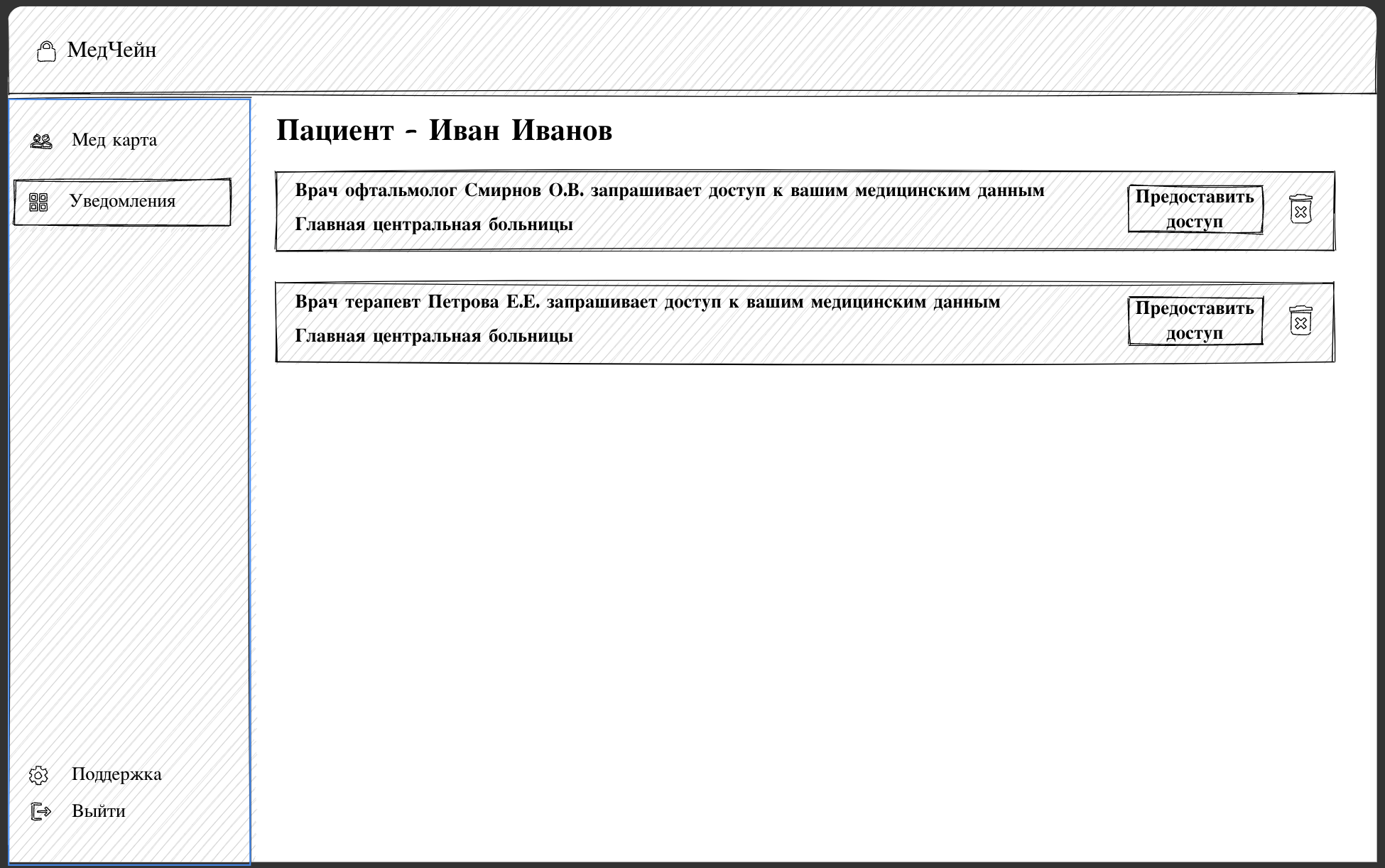
* Проработать цветовую палитру и стилистику
* Создать макеты страниц в figma
* Создать макеты адаптива под мобильные устройства

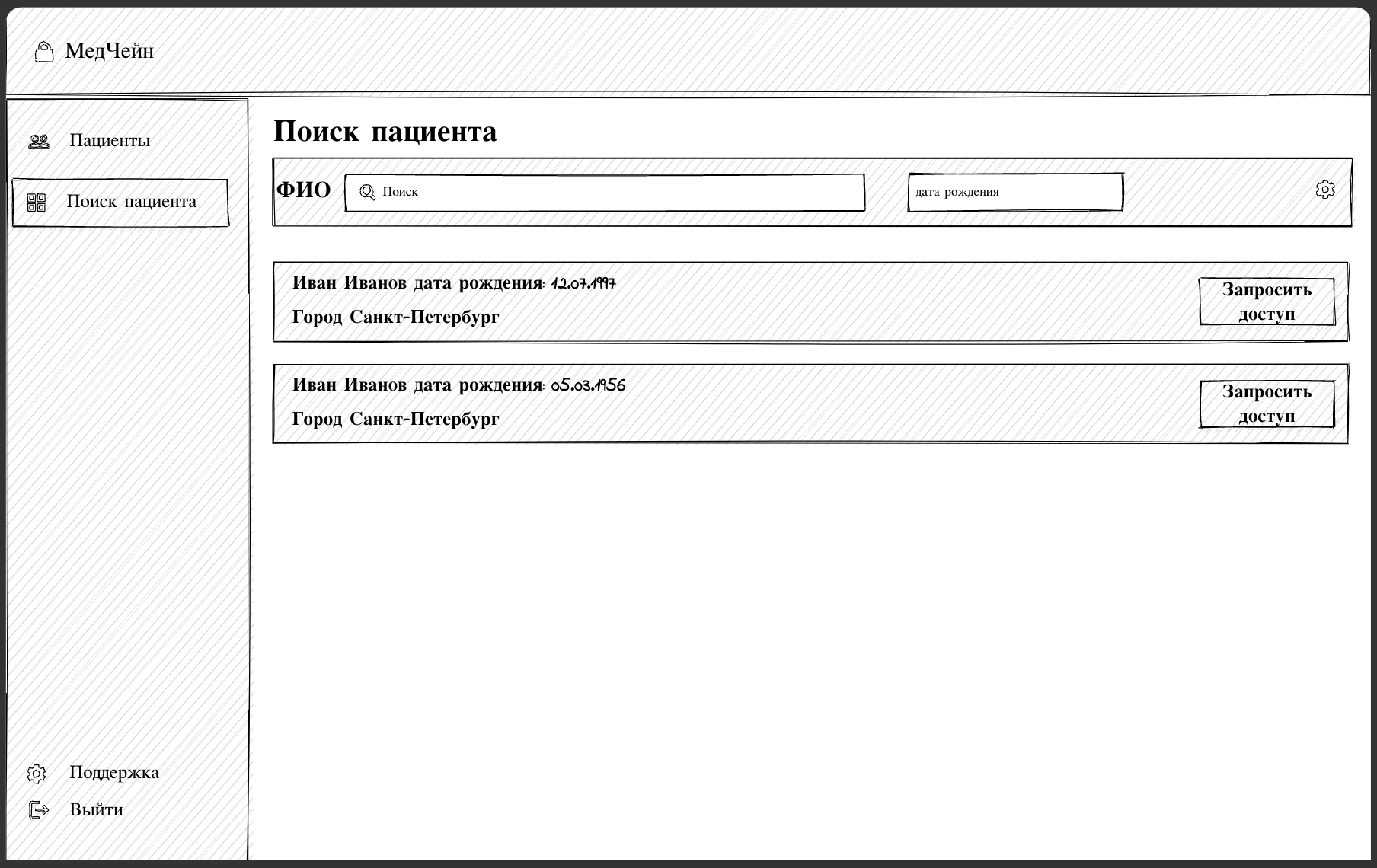
Основные сценарии, которые хотим реализовать, описаны в плане MVP. Откуда следует, что нам нужны следующие страницы:

* страница входа, регистрации
* страница пациента (от лица пациента), где отображается список врачей, которые уже получили доступ к странице и новые запросы
* страница пациента (от лица врача) на которой представлена история мед записей и есть возможность добавить новую
* список пациентов (от лица врача) с возможностью отправить запрос доступа к их страницам

Сильно сырые макеты, просто для понимания, что хочется получить:







# Задачи Front-end

* По макетам сверстать страницы
* Учесть адаптив под мобильные устройства
* Настроить взаимодействие с беком

Страницы, которые хотим получить, описаны чуть выше в задачах дизайна.

В качестве основного фреймворка используем React, можно ознакомиться с документацией <https://ru.legacy.reactjs.org/>

Перед началом работы стоит завести аккаунт в git, работать будем в репозитории <https://github.com/EgoInc/MedChainMVP> и скачать среду разработки (я использую VSCode <https://code.visualstudio.com/> но можно и любую другую)

# Задачи Back-end

## 1. Спроектировать архитектуру БД

Создать ER-диаграммы, чтобы отобразить структуру данных, основные сущности (таблицы) и связи между ними.

### **Данные для хранения на бэкенде:**

#### Информация о пациенте (базовые данные)

* + **Цель**: Хранение основных данных для идентификации пациента и связи с его смарт-контрактом в блокчейне.
  + **Поля**:
    - patient\_id: уникальный идентификатор пациента в системе.
    - name: полное имя пациента.
    - date\_of\_birth: дата рождения пациента.
    - contract\_address: адрес смарт-контракта пациента в блокчейне.

#### Информация о враче

* + **Цель**: Хранение данных для идентификации врача и подтверждения его прав доступа, включая его публичный ключ для верификации.
  + **Поля**:
    - doctor\_id: уникальный идентификатор врача в системе.
    - name: полное имя врача.
    - organization\_id: идентификатор медицинского учреждения, к которому привязан врач.
    - public\_key: публичный ключ врача, который используется для проверки его подписи и аутентификации при доступе к данным пациента.

#### Информация о медицинских учреждениях

* + **Цель**: Хранение базовой информации о медучреждениях, к которым привязаны врачи.
  + **Поля**:
    - organization\_id: уникальный идентификатор медучреждения.
    - name: название учреждения.
    - address: адрес учреждения.
    - contact\_info: контактная информация (телефон, электронная почта).

#### Запросы на доступ к данным пациента

* + **Цель**: Отслеживание и хранение запросов от врачей на доступ к данным пациента. Само подтверждение запроса будет происходить через блокчейн, но бэкенд будет помогать управлять статусами запросов.
  + **Поля**:
    - request\_id: уникальный идентификатор запроса.
    - doctor\_id: ID врача, запрашивающего доступ.
    - patient\_id: ID пациента, к которому запрашивается доступ.
    - status: статус запроса (ожидание, подтверждено, отклонено).
    - request\_date: дата и время создания запроса.

#### Журнал действий врачей

* + **Цель**: Логирование действий врачей для внутреннего аудита и безопасности, например, для отслеживания успешных запросов и записей.
  + **Поля**:
    - log\_id: уникальный идентификатор записи лога.
    - doctor\_id: ID врача, выполняющего действие.
    - patient\_id: ID пациента, к которому относится действие.
    - action\_type: тип действия (запрос доступа, изучение медкарты, изменение медкарты).
    - action\_date: дата и время действия.

## 2. Разработать схемы API-запросов

С помощью Swagger создать прототип запросов. Почитать можно например: <https://medium.com/django-unleashed/a-beginner-guide-to-implement-swagger-documentation-with-django-0de05fbfae3f>   
Прототип запросов на Swagger поможет быстрее наладить интеграцию фронтенда с бэкендом. Для доступа к документации API по адресу {serverAddress}/docs настройте URL и конфигурации Swagger в проекте Django.

### **Запросы пациентов**

#### Запрос на получение списка запросов доступа

**Описание**: Позволяет пациенту просмотреть все текущие запросы на доступ к его данным.

**Схема эндпоинта**: /patient/{patient\_id}/access-requests

**Тип запроса**: GET

**Входные данные**:  
{

"patient\_id": "ID пациента, для которого запрашиваются запросы на доступ. Тип: integer"

}

**Выходные данные**:  
  
[

{

"request\_id": "ID запроса на доступ. Тип: integer",

"doctor": "ФИО врача, который запрашивает доступ. Тип: string",

"status": "Текущий статус запроса (ожидание, подтверждено, отклонено). Тип: string",

"request\_date": "Дата и время создания запроса. Тип: string (ISO 8601)"

}

]

Заглушка:  
[

{

"request\_id": 0",

"doctor": "Иванов Иван Иванович",

"status": "подтверждено",

"request\_date": "2024-06-15T13:00:27+03:00"

},

{

"request\_id": "02",

"doctor": "Петров Петр Петрович",

"status": "отклонено",

"request\_date": "2024-11-04T10:15:27+03:00"

},

{

"request\_id": "03",

"doctor": "Сергеев Сергей Сергеевич",  
"status": "ожидание",  
"request\_date": "2024-11-05T14:48:27+03:00"

}

]

#### Запрос на подтверждение или отклонение доступа

**Описание**: Позволяет пациенту подтвердить или отклонить запрос на доступ к его данным.

**Схема эндпоинта**: /patient/{patient\_id}/access-request/{request\_id}/respond

**Тип запроса**: POST

**Входные данные**:  
{

"patient\_id": "ID пациента, который обрабатывает запрос. Тип: integer",

"request\_id": "ID запроса, который нужно подтвердить или отклонить. Тип: integer",

"approve": "Ответ пациента на запрос (подтвердить - true, отклонить - false). Тип: boolean"

}

**Выходные данные**:  
{

"message": "Результат операции (Запрос подтвержден или Запрос отклонен). Тип: string"

}

Заглушка:

{

"message": "Запрос подтвержден"

}

#### Запрос на получение списка врачей с доступом

**Описание**: Возвращает перечень врачей, которым пациент предоставил доступ к своим данным.

**Схема эндпоинта**: /patient/{patient\_id}/authorized-doctors

**Тип запроса**: GET

**Входные данные**:  
{

"patient\_id": "ID пациента, для которого запрашивается список врачей с доступом. Тип: integer"

}

**Выходные данные**:  
[

{

"doctor\_id": "ID врача, имеющего доступ. Тип: integer",

"doctor\_name": "Полное имя врача. Тип: string",

"organization\_id": "ID организации, к которой принадлежит врач. Тип: integer",

"organization\_name": "Название организации, к которой принадлежит врач. Тип: string",

"access\_date": "Дата и время предоставления доступа. Тип: string (ISO 8601)"

}

]

Заглушка:

[

{

"doctor\_id": 1,

"doctor\_name": "Иванов Иван Иванович",

"organization\_id": 1,

"organization\_name": "Поликлиника №1",

"access\_date": "2024-06-15T13:00:27+03:00"

},

{

"doctor\_id": 2,

"doctor\_name": "Петров Петр Петрович",

"organization\_id": 25,

"organization\_name": "Санкт-Петербургская Клиническая Больница Российской Академии Наук",

"access\_date": "2024-10-20T13:00:27+03:00"

}

]

#### Запрос на добавление пациента

**Описание**: Позволяет создать запись о новом пациенте в системе и сохранить основные данные для связи с его смарт-контрактом в блокчейне.

**Схема эндпоинта**: /admin/add-patient

**Тип запроса**: POST

**Входные данные**:  
  
{

"name": "Полное имя пациента. Тип: string",

"date\_of\_birth": "Дата рождения пациента. Тип: string (ISO 8601)",

"contract\_address": "Адрес смарт-контракта пациента в блокчейне. Тип: string"

}

**Выходные данные**:  
{

"patient\_id": "ID созданного пациента. Тип: integer",

}

Заглушка:  
{

"patient\_id": 100,

}

### **Запросы для врачей**

#### Запрос на получение данных пациента

**Описание**: Позволяет врачу получить базовые данные о пациенте, чтобы подтвердить его личность перед запросом доступа.

**Схема эндпоинта**: /doctor/{doctor\_id}/patient/{patient\_id}

**Тип запроса**: GET

**Входные данные**:  
{

"doctor\_id": "ID врача, запрашивающего данные. Тип: integer",

"patient\_id": "ID пациента, чьи данные запрашиваются. Тип: integer"

}

**Выходные данные**:  
{

"patient\_id": "ID пациента. Тип: integer",

"name": "Полное имя пациента. Тип: string",

"date\_of\_birth": "Дата рождения пациента. Тип: string (ISO 8601)",

"contract\_address": "Адрес смарт-контракта пациента в блокчейне. 42 символа, начинается с 0x. Тип: string"

}

Заглушка:

{

"patient\_id": 1,

"name": "Иванов Иван Иванович",

"date\_of\_birth": "2000-06-01T00:00:00+03:00",

"contract\_address": "0x0000000000000000000000000000000000000000"

}

#### Запрос на доступ к данным пациента

**Описание**: Позволяет врачу отправить запрос на доступ к данным пациента.

**Схема эндпоинта**: /doctor/{doctor\_id}/request-access

**Тип запроса**: POST

**Входные данные**:  
{

"doctor\_id": "ID врача, запрашивающего доступ. Тип: integer",

"patient\_id": "ID пациента, к которому запрашивается доступ. Тип: integer"

}

**Выходные данные**:  
{

"request\_id": "ID созданного запроса на доступ. Тип: integer"

}

Заглушка:

{

"request\_id": 127

}

#### Поиск пациентов

**Описание**: Позволяет врачу найти пациентов по имени, фамилии или полному ФИО, а также с помощью дополнительных параметров, таких как дата рождения.

**Схема эндпоинта**: /doctor/{doctor\_id}/search-patients

**Тип запроса**: GET

**Входные данные**:  
{

"doctor\_id": "ID врача, выполняющего поиск. Тип: integer",

"name": "Имя пациента. Тип: string",

"date\_of\_birth": "Дата рождения пациента (необязательно, д

ля уточнения результатов). Тип: string (ISO 8601)"}

**Выходные данные**:  
[

{

"patient\_id": "ID найденного пациента. Тип: integer",

"name": "Полное имя пациента. Тип: string",

"date\_of\_birth": "Дата рождения пациента. Тип: string (ISO 8601)",

"contract\_address": "Адрес смарт-контракта пациента в блокчейне. 42 символа, начинается с 0x. Тип: string"

}

]

Заглушка:

[

{

"patient\_id": 1,

"name": "Иванов Иван Иванович",

"date\_of\_birth": "2000-06-01T00:00:00+03:00",

"contract\_address": "0x0000000000000000000000000000000000000000"

},

{

"patient\_id": 101,

"name": "Иванов Иван Алексеевич",

"date\_of\_birth": "2011-08-03T00:00:00+03:00",

"contract\_address": "0x1100000000000000000000000000000000000011"

}

]

#### Запрос "Мои пациенты"

**Описание**: Возвращает список пациентов, доступ к данным которых был подтвержден для данного врача.

**Схема эндпоинта**: /doctor/{doctor\_id}/my-patients

**Тип запроса**: GET

**Входные данные**:  
{

"doctor\_id": "ID врача, для которого запрашивается список пациентов с доступом. Тип: integer"

}

**Выходные данные**:  
[

{

"patient\_id": "ID пациента, к которому у врача есть доступ. Тип: integer",

"name": "Полное имя пациента. Тип: string",

"date\_of\_birth": "Дата рождения пациента. Тип: string (ISO 8601)",

"contract\_address": "Адрес смарт-контракта пациента в блокчейне. 42 символа, начинается с 0x. Тип: string",

"access\_granted\_date": "Дата и время, когда доступ был подтвержден. Тип: string (ISO 8601)"

}

]

Заглушка:

[

{

"patient\_id": 101,

"name": "Иванов Иван Иванович",

"date\_of\_birth": "2000-06-01T00:00:00+03:00",

"contract\_address": "Адрес смарт-контракта пациента в блокчейне. Тип: string",

"access\_granted\_date": "2024-12-24T12:00:00+03:00"

},

{

"patient\_id": 101,

"name": "Петров Петр Петрович",

"date\_of\_birth": "2000-06-01T00:00:00+03:00",

"contract\_address": 0x0000000000000000000000000000000000000000",

"access\_granted\_date": "2024-08-22T15:00:00+03:00"

},

]

### **Запрос для больниц**

#### Запрос на добавление больницы

**Описание**: Позволяет создать запись о новом медицинском учреждении (больнице) в системе.

**Схема эндпоинта**: /admin/add-hospital

**Тип запроса**: POST

**Входные данные**:  
{

"name": "Название медицинского учреждения. Тип: string",

"address": "Адрес учреждения. Тип: string",

"contact\_info": "Контактная информация (телефон, электронная почта). Тип: string"

}

**Выходные данные**:  
{

"organization\_id": "ID созданного учреждения. Тип: integer"

}

Заглушка:

{

"organization\_id": 100

}

#### Запрос на добавление нового врача

**Описание**: Позволяет медучреждению зарегистрировать нового врача в системе.

**Схема эндпоинта**: /organization/{organization\_id}/add-doctor

**Тип запроса**: POST

**Входные данные**:  
  
{

"organization\_id": "ID медицинского учреждения, регистрирующего врача. Тип: integer",

"doctor": "Полное имя врача. Тип: string",

"public\_key": "Публичный ключ врача для аутентификации. Тип: string"

}

**Выходные данные**:  
{

"doctor\_id": "ID созданного врача. Тип: integer"

}

Заглушка:

{

"doctor\_id": 1

}

#### Запрос на получение информации о врачах учреждения

**Описание**: Позволяет получить информацию о всех врачах, связанных с конкретным медицинским учреждением.

**Схема эндпоинта**: /organization/{organization\_id}/doctors

**Тип запроса**: GET

**Входные данные**:  
{

"organization\_id": "ID медицинского учреждения. Тип: integer"

}

**Выходные данные**:  
[

{

"doctor\_id": "ID врача. Тип: integer",

"doctor": "Полное имя врача. Тип: string",

"public\_key": "Публичный ключ врача. 42 символа, начинается с 0x. Тип: string"

}  
]

Заглушка:

[

{

"doctor\_id": 1,

"doctor": "Иванов Иван Иванович",

"public\_key": "0x0000000000000000000000000000000000000000"

},

{

"doctor\_id": 2,

"doctor": "Сергеев Сергей Сергеевич,

"public\_key": "0x0000000000000000000000000000000000000000"

}  
]

## 3. Настроить контейнеризацию и сборку приложения

### **Основные файлы и папки**

#### **1. manage.py**

* **Описание**: Основной файл для управления Django-проектом.
* **Назначение**:
  + Запуск сервера разработки (python manage.py runserver).
  + Применение миграций (python manage.py migrate).
  + Создание приложений (python manage.py startapp).
  + Выполнение команд и скриптов Django.

#### **2. medchain (основная папка проекта)**

* **Описание**: Папка с настройками проекта.
* **Содержит**:
  + **\_\_init\_\_.py**: Делает папку модулем Python. Этот файл часто пуст.
  + **asgi.py**: Настройки для ASGI (асинхронный серверный шлюзовый интерфейс). Используется для запуска асинхронных приложений.
  + **settings.py**: Основные настройки проекта (база данных, приложения, параметры конфигурации).
  + **urls.py**: Основные маршруты (роуты) проекта. Содержит ссылки на файлы маршрутов приложений.
  + **wsgi.py**: Настройки для WSGI (веб-серверный шлюзовый интерфейс). Используется для запуска приложения на продакшн-серверах.

#### **3. medchainapi (папка приложения)**

* **Описание**: Это приложение внутри вашего проекта Django. Django проект может включать несколько приложений.
* **Содержит**:
  + **\_\_init\_\_.py**: Делает папку модулем Python.
  + **admin.py**: Настройки для административной панели Django. Здесь вы можете регистрировать модели для их отображения в панели администратора.
  + **apps.py**: Настройки приложения. Определяет конфигурацию приложения.
  + **models.py**: Определение моделей базы данных. Каждая модель соответствует таблице в базе данных.
  + **migrations/**: Папка с файлами миграций, которые Django создает для управления изменениями структуры базы данных.
  + **serializers.py**: Файл для создания сериализаторов (в вашем случае используется для работы с API).
  + **tests.py**: Файл для написания тестов.
  + **views.py**: Основная бизнес-логика приложения. Определяет функции и классы, которые обрабатывают запросы.

### **Вспомогательные файлы**

#### **4. .gitignore**

* **Описание**: Файл для указания файлов и папок, которые не должны отслеживаться Git.
* **Назначение**: Исключает из репозитория такие файлы, как виртуальные окружения, миграции, логи, статические файлы и скомпилированные файлы Python.

#### **5. .dockerignore**

* **Описание**: Аналог .gitignore, но для Docker.
* **Назначение**: Указывает файлы и папки, которые не нужно копировать в контейнер Docker.

#### **6. docker-compose.yml**

* **Описание**: Конфигурационный файл Docker Compose.
* **Назначение**:
  + Описывает и координирует запуск нескольких сервисов (Django, PostgreSQL).
  + Автоматизирует запуск контейнеров.

#### **7. Dockerfile**

* **Описание**: Скрипт для сборки Docker-образа.
* **Назначение**: Определяет, как упаковать ваш Django-проект в Docker-образ.

#### **8. requirements.txt**

* **Описание**: Файл с зависимостями проекта.
* **Назначение**: Указывает пакеты Python и их версии, которые должны быть установлены для работы проекта.

#### **9. db.sqlite3**

* **Описание**: Файл SQLite-базы данных.
* **Назначение**: Содержит данные вашего проекта, такие как записи моделей, данные пользователей, логи и прочее.

#### **10. migrations/**

* **Описание**: Папка с миграциями (внутри каждого приложения).
* **Назначение**:
  + Миграции — это скрипты для обновления структуры базы данных (например, создание или изменение таблиц).
  + Автоматически создаются Django при изменении моделей.

## 4. Развертка на сервере

Настройте виртуальную машину или сервер и разверните приложение. Настройте веб-сервер (например, Nginx), подключите его к Django через Gunicorn или другой WSGI-сервер. На этом этапе проверьте работу миграций, подключите базу данных, кэш и фоновые задачи, если они требуются.

## Задачи блокчейн

## 0. Познакомиться с блокчейном

Необходимо разобраться с основными понятиями блокчейн-сферы:

* Общее:
  + Блокчейн
  + Узлы блокчейна
  + EVM-блокчейн
  + Транзакции
  + Газ
  + Смарт-контракты
  + Публичный и приватный ключ
* Для разработчиков блокчейнов:
  + Алгоритм консенсуса (понять разницу PoW, PoS)
  + Блоки в блокчейне
* Для разработчиком смарт-контрактов:
  + ABI смарт-контракта
  + Bytecode смарт-контракта
  + Криптокошелек

В этом могут помочь видео:

* **Что такое блокчейн**[What is a Blockchain? (Animated + Examples)](https://www.youtube.com/watch?v=kHybf1aC-jE)

*Там очень быстро расскажут про базовые термины, введут в курс дела*

* ***Что такое смарт-контракты***[*What are Smart Contracts in Crypto? (4 Examples + Animated)*](https://www.youtube.com/watch?v=pyaIppMhuic&list=PLHx4UicbtUoYPDWk2aUwZoVKMkdRKtKWe&index=11&pp=iAQB)

*Именно это мы и будем писать, поэтому рекомендую посмотреть, возможно что-то еще почитать и разобраться*

* ***Что такое газ в Ethereum***[*What is Ethereum Gas? (Examples + Easy Explanation)*](https://www.youtube.com/watch?v=3ehaSqwUZ0s&list=PLHx4UicbtUoYPDWk2aUwZoVKMkdRKtKWe&index=6&pp=iAQ)

*Это важный компонент в экосистеме EVM-блокчейном, поэтому тоже хорошо бы понять что это*

* **Публичные и приватные ключи. RSA**

[Asymmetric Encryption - Simply explained](https://youtu.be/AQDCe585Lnc?si=G_9IiLKXmPBc_qYK)

*Поможет разобраться в чем отличие приватных и публичных ключей и зачем вообще они нужны*

## 1. Запустить частный EVM-блокчейн

Развернуть частный EVM-блокчейн, адаптированный под нужды системы, т.е.:

* Пользователям не нужны реальные деньги чтоб осуществлять транзакции
* Может хранить много данных в рамках одного смарт-контракта
* Должен работать на слабых компьютерах и не требовать мощного железа, которого нет у больниц

## 2. Написать смарт-контракты

Написать два стандарта смарт-контрактов:

### **Смарт-контракт пациентов**

* Структуры данных:
  + MedicalRecord – структура для хранения данных о конкретной записи в истории болезни (дата, врач, диагноз, жалобы);
  + Patient – структура для хранения данных о пациенте и его медицинской истории.
* Функции:
  + addDoctor и removeDoctor - функции для управления списком авторизованных врачей, только владелец контракта (пациент) может добавлять или удалять врачей;
  + addMedicalRecord – функция для добавления новой записи в медицинскую историю пациента, доступна только авторизованным врачам;
  + getMedicalHistory – функция для получения списка записей в медицинской истории пациента, доступна только авторизованным врачам;
  + getPatientData – функция получения информации о пациенте (ФИО, дата рождения).

### **Смарт-контракт администратор:**

* Структуры данных:
  + patientContracts – ассоциативный массив (mapping), который хранит адрес смарт-контракта пациента для каждого пациента, ключом является адрес пациента, значением — адрес смарт-контракта;
  + allPatients – массив, содержащий адреса всех пациентов, этот массив используется для получения списка всех пациентов в системе.
* Функции:
  + createPatientContract – функция создает новый смарт-контракт пациента и его адрес сохраняется в patientContracts, адрес пациента также добавляется в allPatients;
  + getPatientContract – функция возвращает адрес смарт-контракта пациента по его адресу и позволяет другим пользователям системы (например, врачам) находить контракт пациента для доступа к его медицинской информации;
  + getAllPatients – функция возвращает массив адресов всех пациентов, может быть полезна для администраторов системы или для анализа данных.

## 3. Развернуть смарт-контракты в блокчейне

Развернуть смарт-контракты:

**А.** В локальной сети, т.ч. частном развернутом на шаге 1 блокчейне

**Б.** В публичном тестнете

## 4. Написать примеры взаимодействия с блокчейном для фронтенда

С помощью библиотеки ethers.js написать скрипты для взаимодействия с написанными смарт-контрактами, которые затем будут интегрированы в код фронтендеров