**Министерство науки и высшего образования Российской Федерации**

**ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ**

**«НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО»**

**(Университет ИТМО)**

Факультет **Прикладной информатики**

Направление подготовки **09.03.03 Прикладная информатика**

Образовательная программа **Мобильные и сетевые технологии**

**К У Р С О В О Й   П Р О Е К Т**

Тема: «Разработка прототипа серверной части системы управления медицинскими данными на основе блокчейн технологий»

Обучающийся: Колпаков Артем Сергеевич, К3139

Санкт-Петербург 2024

# **СОДЕРЖАНИЕ**

[СОДЕРЖАНИЕ 2](#_8by5fiwjzupc)

[ВВЕДЕНИЕ 3](#_njmobwiljf97)

[1 Описание проекта](#_z0eyjnrq151m) 5

[2 Процессы работы над проектом](#_utlos4xpurnk) 7

[3 Поставленные задачи и их решения](#_voevpzicvhhd) 12

[4 Анализ работы 15](#_a1y8v87shhty)

[5 Взаимодействие с командой](#_k753klaxehhn) 16

[6 Взаимодействие с руководителем проекта и оценка его деятельности 17](#_atqm315ic8ic)

[ЗАКЛЮЧЕНИЕ 18](#_pkofja9i6d6m)

[СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ 19](#_ibt3mkrc8sto)

[ПРИЛОЖЕНИЕ 20](#_1qftbwysw55r)

# **ВВЕДЕНИЕ**

В современном мире наблюдается значительное увеличение объемов медицинских данных, что обусловлено активным внедрением цифровых технологий, ростом числа медицинских исследований и повсеместным использованием электронных медицинских карт. Управление такими объемами информации становится все более сложной задачей, особенно с учетом требований к обеспечению безопасности, доступности и прозрачности данных. Традиционные централизованные системы хранения и обработки медицинской информации зачастую не способны справиться с этими вызовами, что приводит к рискам утечек данных, недостоверности записей и снижению уровня доверия пациентов к медицинским учреждениям.

Одной из ключевых проблем современного здравоохранения является защита конфиденциальности медицинских данных. Пациенты все чаще озабочены безопасностью своей личной информации, особенно в условиях роста кибератак и увеличения числа утечек данных. Блокчейн-технологии предлагают инновационный подход к решению этой проблемы. Их децентрализованная структура обеспечивает защиту от несанкционированного доступа, а неизменяемость записей создает условия для максимальной прозрачности и надежности. Использование смарт-контрактов, встроенных в блокчейн-системы, позволяет автоматизировать процессы управления данными и повысить уровень доверия между пациентами и медицинскими учреждениями.

Актуальность разработки прототипа серверной части системы управления медицинскими данными на основе блокчейн-технологий заключается в стремлении создать безопасную и удобную инфраструктуру, которая удовлетворяет потребности всех участников медицинской экосистемы. Такая система позволит пациентам контролировать доступ к своей информации, а медицинским учреждениям — минимизировать риски ошибок и утечек данных. В условиях цифровизации здравоохранения и роста требований к безопасности медицинской информации разработка подобного проекта представляет собой важный шаг к созданию более эффективной и надежной системы управления данными.

Целью данного проекта является разработка прототипа серверной части системы управления медицинскими данными на основе блокчейн-технологий. Система будет обеспечивать безопасное хранение медицинских записей, управление доступом к ним и автоматизацию процессов взаимодействия между пациентами и врачами.

Для достижения поставленной цели необходимо решить следующие задачи:

* разработка архитектуры системы: Создание серверной блокчейн-архитектуры для хранения данных и интерфейсы для взаимодействия пользователей,
* реализация функционала хранения и обработки данных: Разработка механизмов для безопасного хранения медицинских записей с использованием блокчейн-технологий,
* создание пользовательских интерфейсов: Разработка интерфейсов для пациентов и врачей, которые позволят им легко взаимодействовать с системой,
* тестирование прототипа: осуществление проверки системы с целью выявления потенциальных недостатков и оптимизации ее функциональности.

# **1 Описание проекта**

Проект представляет собой инновационное решение, направленное на создание платформы для безопасного хранения и управления медицинской информацией. В условиях постоянно растущего объема данных о пациентах и их медицинской истории возникает потребность в надежных системах, обеспечивающих защиту, доступность и целостность информации.

Основная идея проекта заключается в применении блокчейн-технологий для разработки децентрализованной системы, которая позволит безопасно взаимодействовать с медицинскими данными как пациентам, так и врачам. Благодаря неизменяемости и прозрачности записей, которые предоставляет блокчейн, достигается высокий уровень доверия между всеми участниками: пациентами, врачами и медицинскими учреждениями. В рамках проекта создается серверная часть системы, которая интегрируется с интерфейсами пользователя и базой данных, обеспечивая надежное управление доступом к медицинским записям.

Особенностью системы является возможность пациентов контролировать доступ к своей медицинской информации. Пациенты смогут предоставлять или ограничивать доступ врачам, что повышает уверенность в безопасности данных. Врачи, в свою очередь, получат доступ к инструментам для поиска пациентов и работы с их медицинскими записями, что упрощает и ускоряет оказание медицинских услуг.

Проект также предусматривает разработку смарт-контрактов для автоматизации процессов доступа к данным. Эти контракты помогут исключить человеческий фактор при выполнении условий взаимодействия, минимизировать ошибки и повысить эффективность работы системы.

Завершающим этапом проекта станет создание прототипа, который будет протестирован на соответствие функциональным требованиям. Развертывание прототипа на удаленном сервере позволит провести внешнее тестирование, собрать обратную связь от пользователей и внести необходимые доработки.

Ожидается, что успешная реализация проекта будет способствовать улучшению качества медицинского обслуживания и укреплению доверия к медицинским учреждениям, благодаря созданию надежной и безопасной системы управления медицинскими данными.

# **2 Процессы работы над проектом**

## **2.1 Планирование и распределение задач, обучение**

На первом этапе работы мы детально изучили проект, его цели и задачи, распределили обязанности между участниками команды, а также определили стек технологий, который будет использоваться в процессе разработки.

Далее мы приступили к обучению. С этой целью изучались статьи и видео по блокчейну, предоставленные руководителем проекта. Для закрепления полученных знаний был создан тестовый смарт-контракт библиотеки, представленный на рисунке 1.

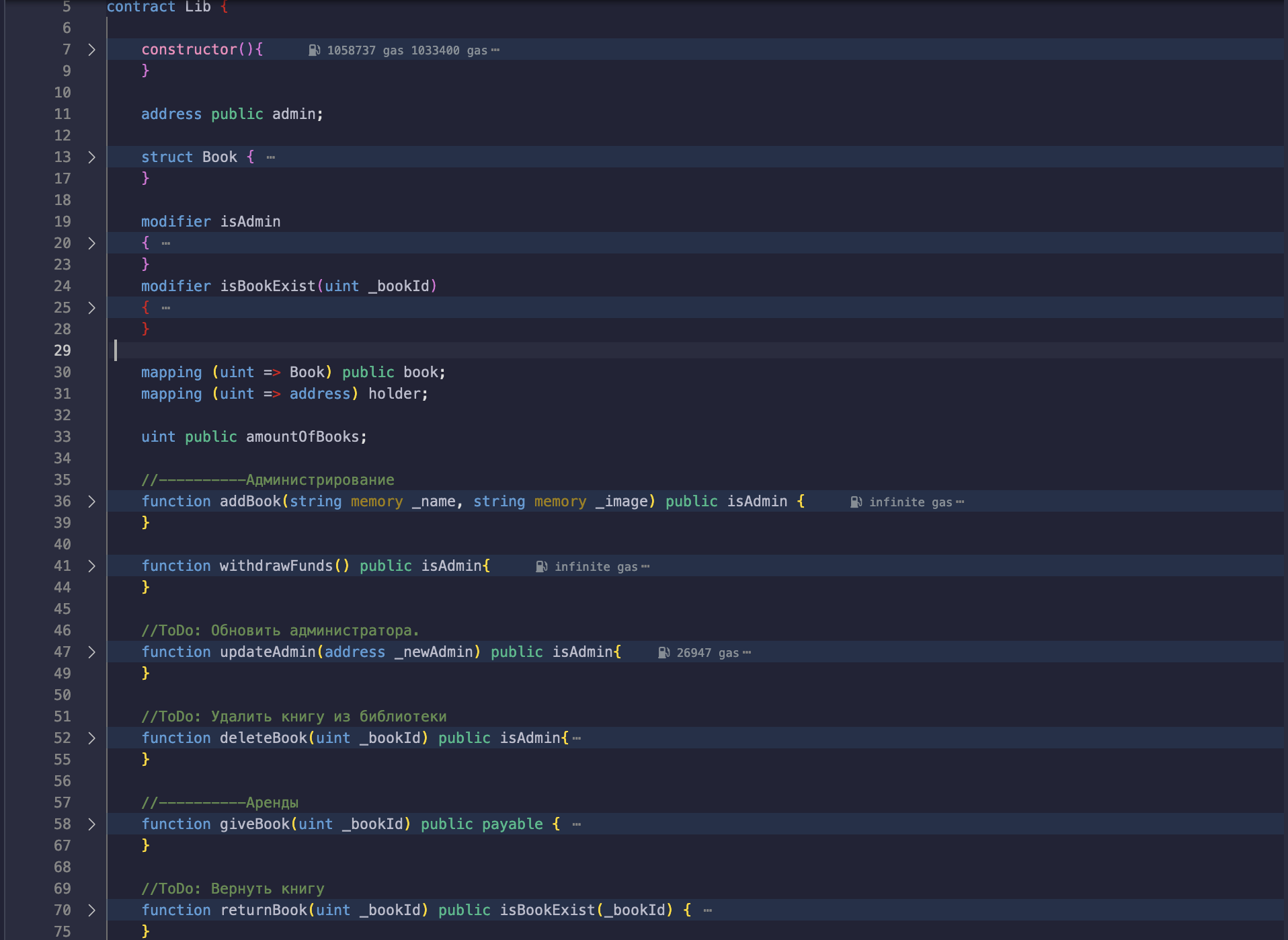


Рисунок 1 — смарт-контракт библиотеки.

Данный контракт позволяет администратору контракта управлять книгами, а именно удалять и добавлять нужные книги в библиотеку, а также предоставляет функции для аренды и возврата книг пользователям системы. В контракте есть структура данных Book, содержащая название, изображение и статус доступности книги. Были реализованы такие функции, как добавление и удаление книги администратором, функции аренды и возвращения книги пользователем, если нужная книга есть в наличии, а также написали собственные модификаторы, которые проверяют, является ли пользователь администратором, а также есть ли определенная книга в библиотеке.

Работа над этим смарт-контрактом позволила мне освоить множество важных навыков и получить ценный опыт. В первую очередь, я изучил основы программирования на языке Solidity, который используется для создания смарт-контрактов на платформе Ethereum. Также я получил базовое понимание работы блокчейна, включая принципы транзакций, плату за их выполнение и взаимодействие с контрактами. Проектирование и реализация структур данных, а также проработка аспектов безопасности смарт-контрактов помогли мне осознать важность тщательной проверки условий для предотвращения ошибок и возможных злоупотреблений.

Во-вторых, я научился проектировать архитектуру контракта, тестировать функции, используя интерфейс Remix IDE и управлять проектом через планирование и документирование кода. Таким образом, написание этого смарт-контракта дало мне комплексный опыт в области разработки блокчейн-приложений.

Этот этап оказался особенно полезным, так как я смог получить практические навыки работы с Solidity и разобраться в структуре работы блокчейн-технологий.

## **2.2 Прототипирование**

На данном этапе команда сосредотачивается на создании четкого видения системы, ее функционала и архитектуры, что закладывает прочную основу для последующей работы.

Начало этапа посвящено анализу требований. Мы изучаем потребности конечных пользователей — пациентов и врачей, активно обсуждаем, какие функции будут наиболее полезными и какие проблемы должна решить система. Это помогает сформировать перечень функциональных требований, включая возможности подтверждения или отклонения запросов на доступ к медицинским данным, а также управления профилями пользователей.

Далее разрабатывается архитектура системы. На этом этапе выбирается стек технологий, включая PostgreSQL и Python с Django для бэкенда, React.js для фронтенда. Команда дизайна создавала макеты пользовательского интерфейса с использованием таких инструментов, как Figma.

Конкретно наша команда использовала для разработки смарт контрактов язык программирования Solidity, Remix IDE в связке с EVM для написания, развертки и тестирования контрактов. В дальнейшем мы также научились разворачивать блокчейн локально с помощью технологии Hardhat и писать скрипты-примеры взаимодействия с блокчейн технологиями с помощью библиотеки ethers.js.

Этап прототипирования завершается подготовкой к следующему этапу — непосредственной реализации системы. К этому моменту мы должны быть готовы воплотить все запланированные функции и возможности, опираясь на разработанные материалы и накопленные знания. В конце мы сошлись на следующей схеме взаимодействия пользователя с блокчейном, представленной на рисунке 2.

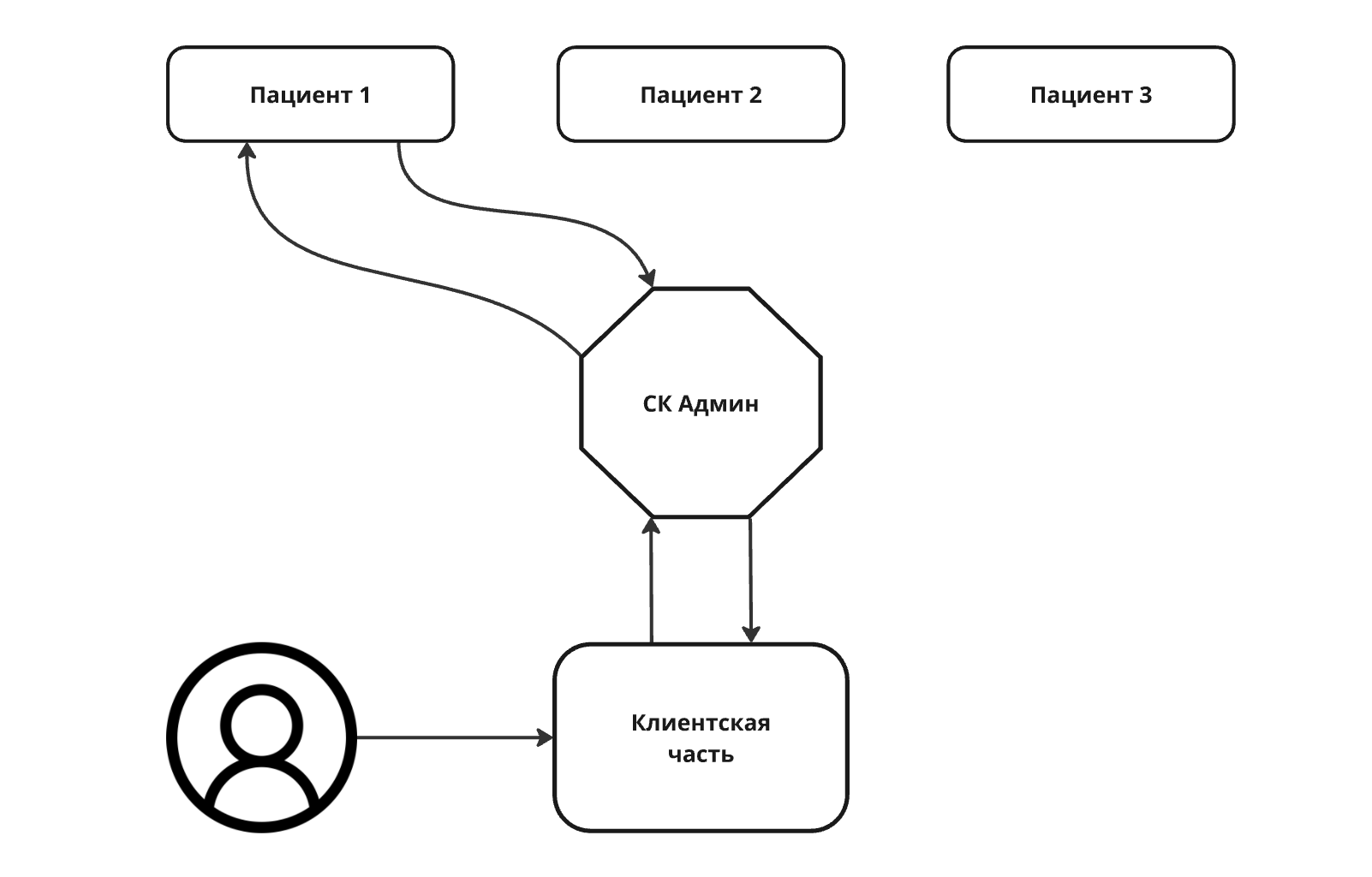


Рисунок 2 — схема взаимодействия пользователя с блокчейном.

На данном рисунке представлено, что клиентская часть взаимодействует непосредственно со смарт-контрактом алминистратора, который в свою очередь, зная адрес смарт-контракта каждого пациента, обращается к нужному и возвращает полученные данные. Такая схема взаимодействия оптимальна для нашего проекта и обеспечивает быстрый и надежный обмен данных между блокчейном и пользователем.

## **2.3 Разработка**

На этом этапе выполнялся широкий спектр задач, включая программирование, тестирование и документирование, что потребовало активного участия каждого члена команды.

В начале работы мы сосредоточились на создании серверной части системы с использованием Python и фреймворка Django. Для хранения данных о пациентах и врачах была интегрирована база данных PostgreSQL, что обеспечило надежность и высокую скорость обработки запросов.

Параллельно велась разработка клиентской части системы на основе React.js. Одним из ключевых направлений стало внедрение блокчейн-технологий. Мы разработали смарт-контракты на языке Solidity, которые автоматизировали процессы доступа к медицинским данным.

После завершения реализации основных функций начался этап тестирования. Мы провели функциональное и интеграционное тестирование, чтобы убедиться, что все компоненты системы работают корректно и безошибочно взаимодействуют друг с другом. Особое внимание было уделено документированию API с использованием Swagger, что сделало процесс интеграции и тестирования системы удобным для сторонних разработчиков. Кроме того, мы стремились к максимально качественной документации кода, чтобы облегчить дальнейшую работу с проектом.

Завершающим шагом этапа стало развертывание системы на удаленном сервере для внешнего тестирования. Это был важный момент, который позволил увидеть результаты наших усилий в действии. Этап разработки дал мне возможность применить полученные знания на практике, значительно улучшить навыки программирования и работы с блокчейн-технологиями, а также получить бесценный опыт командной работы.

# **3 Поставленные задачи и их решения**

Задачей команды блокчейн была непосредственная разработка смарт-контрактов на языке программирования Solidity. Мы разработали смарт-контракт пациента, представленный на рисунке 3.

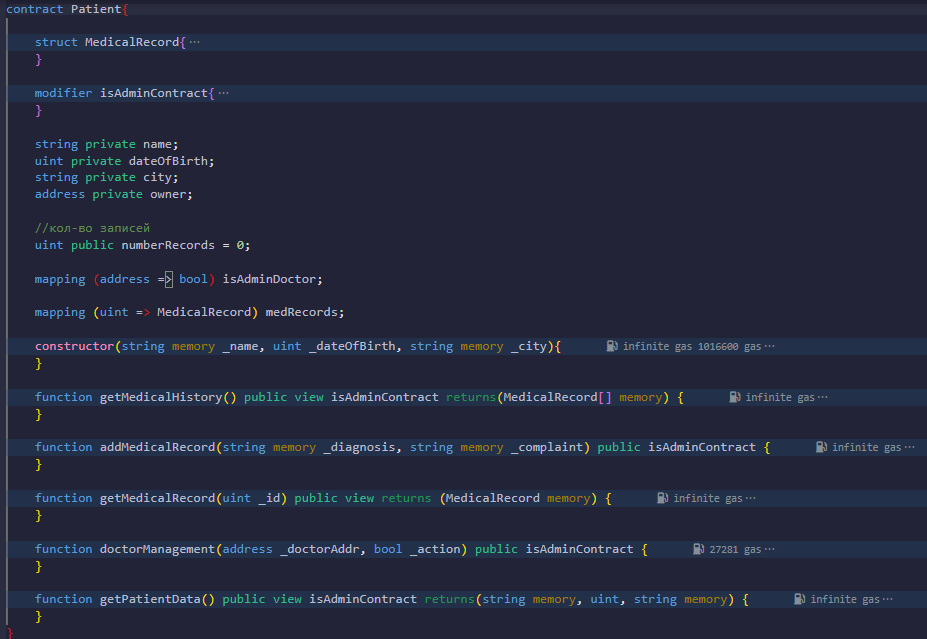


Рисунок 3 — смарт-контракт пациента.

В нем представлена структура пациента, которая содержит в себе поля фио, даты рождения и города. Также создана структура медицинских данных, которая содержит в себе атрибут даты диагноза, адрес смарт контракта доктора, непосредственно диагноз и жалобу пациента. Также создан массив медицинских данных пациента и функции взаимодействия с медицинскими записями: удаление, добавление, просмотр всех и конкретного. Важно отметить, что взаимодействие с данными пациента доступно только доверенным врачам.

Также был реализован смарт-контракт администратора, представленный на рисунке 4.

Рисунок 4 — смарт-контракт администратора.

В данном смарт-контракте реализован массив пациентов, который содержит в себе адреса контрактов всех пациентов системы. Также реализованы функции получения данных о пользователе и добавления пациента в систему.

В конце мы написали примеры скриптов взаимодействия с блокчейном для команды фронтенда с применением библиотеки ethers.js, представленных на рисунке 5.

Рисунок 5 — пример скриптов взаимодействия с блокчейном.

В конце мы развернули смарт-контракты на локальной машине с использованием технологии Hardhat, которая позволяла локально настроить работающий блокчейн. Также мы протестировали разработанные смарт-контракты локально с помощью скриптов, написанных на основе Hardhat.

# **4 Анализ работы**

Анализируя свою работу над проектом "Разработка прототипа серверной части системы управления медицинскими данными на основе блокчейн-технологий", я могу отметить как положительные, так и отрицательные аспекты.

Среди достижений особенно важно выделить освоение языка программирования Solidity и среды разработки Remix IDE, которая значительно упростила процесс написания смарт-контрактов и их ручного тестирования. Я также приобрел опыт работы с такими ключевыми библиотеками, как Hardhat и Ethers.js, которые широко используются в современной блокчейн-разработке. Кроме того, я научился эффективно работать в команде, используя инструменты контроля версий git и платформу GitHub, что, безусловно, станет полезным навыком в моих будущих проектах.

Однако в процессе работы я столкнулся с трудностями. Несмотря на значительное внимание, уделенное изучению теоретических основ блокчейн-технологий, мне порой не хватало знаний для реализации некоторых функций. Это замедляло процесс разработки, но благодаря поддержке компетентного руководителя и слаженной работе команды, все сложности удавалось оперативно преодолеть. Эти вызовы позволили мне не только глубже понять особенности блокчейн-разработки, но и улучшить навыки решения сложных задач в команде.

# 

# **5 Взаимодействие с командой**

После распределения на команды, руководителями проекта сразу был создан чат в телеграмме, который включал в себя всех участников проекта. Внутри чата были созданы отдельные чаты для команд фронтенда, бэкенда, дизайн и блокчейна, а также общие чаты для важной информации.

После создания чата и первой встречи, руководители прислали необходимые материалы для изучения основ используемых технологий. В дальнейшем мы на регулярной основе организовывали онлайн встречи, на которых обсуждали уже реализованные задачи, а также продумывали дальнеший план действий. При возникновении вопросов, мы всегда могли обратиться в чат за помощью, где мы могли обсудить проблему и спросить совета.

# 

# **6 Взаимодействие с руководителем проекта и оценка его деятельности**

Наш руководитель, Лаврова Анастасия, проявила себя как высококвалифицированный и отзывчивый руководитель. Она всегда была готова помочь разобраться со сложными терминами и технологиями, которые неоднократно встречались в процессе работы над проектом. Благодаря ее четкой формулировке задач и строгому контролю сроков выполнения, проект был реализован полноценно и включал все заранее оговоренные функции.

Анастасия также уделяла большое внимание обратной связи: она хвалила за удачные решения, давала ценные рекомендации по улучшению работы и предлагала направления для доработки. Такой подход не только способствовал развитию наших профессиональных навыков, но и мотивировал нас к достижению лучших результатов.

Я считаю, что Лаврова Анастасия - замечательный руководитель и настоящий профессионал своего дела, и она безусловно заслуживает наивысшей оценки.

# **ЗАКЛЮЧЕНИЕ**

В рамках проекта "Разработка прототипа серверной части системы управления медицинскими данными на основе блокчейн-технологий" удалось успешно достичь поставленных целей и задач. Основной целью проекта было создание функционального прототипа системы, обеспечивающей безопасное хранение и управление медицинскими данными с использованием блокчейн-технологий.

Мой вклад в проект включал активное участие в разработке смарт-контрактов. В процессе разработки возникали сложности с пониманием работы блокчейн технологии и создания оптимизированных смарт-контрактов, которые удалось преодолеть благодаря изучению дополнительных материалов. Участие в проекте позволило мне углубить знания в области блокчейн-разработки и приобрести навыки работы с такими инструментами, как Solidity и Remix IDE.

Работа над проектом стала для меня не только важным этапом профессионального роста, но и ценным опытом командной работы. Несмотря на возникшие сложности, наша команда достигла всех поставленных целей, а конечный результат демонстрирует успешность реализации проекта. Я горжусь тем, что смог внести свой вклад в разработку системы, которая имеет практическое значение и потенциал для дальнейшего развития.

# **СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ**

1. Что такое блокчейн <https://www.youtube.com/watch?v=kHybf1aC-jE>
2. Что такое смарт-контракты <https://www.youtube.com/watch?v=pyaIppMhuic&list=PLHx4UicbtUoYPDWk2aUwZoVKMkdRKtKWe&index=11&pp=iAQB>
3. Форум с вопросами по Ethereum и блокчейну в целом <https://ethereum.stackexchange.com/>
4. Документация Solidity <https://docs.soliditylang.org/en/v0.8.28/>
5. Документация Hardhat <https://hardhat.org/docs>
6. Сравнение блокчейн-технологий <https://habr.com/ru/articles/751670/>

# **ПРИЛОЖЕНИЕ. ТЕХНИЧЕСКОЕ ЗАДАНИЕ**

Техническое задание

Сервис для управления медицинскими данными на основе блокчейн

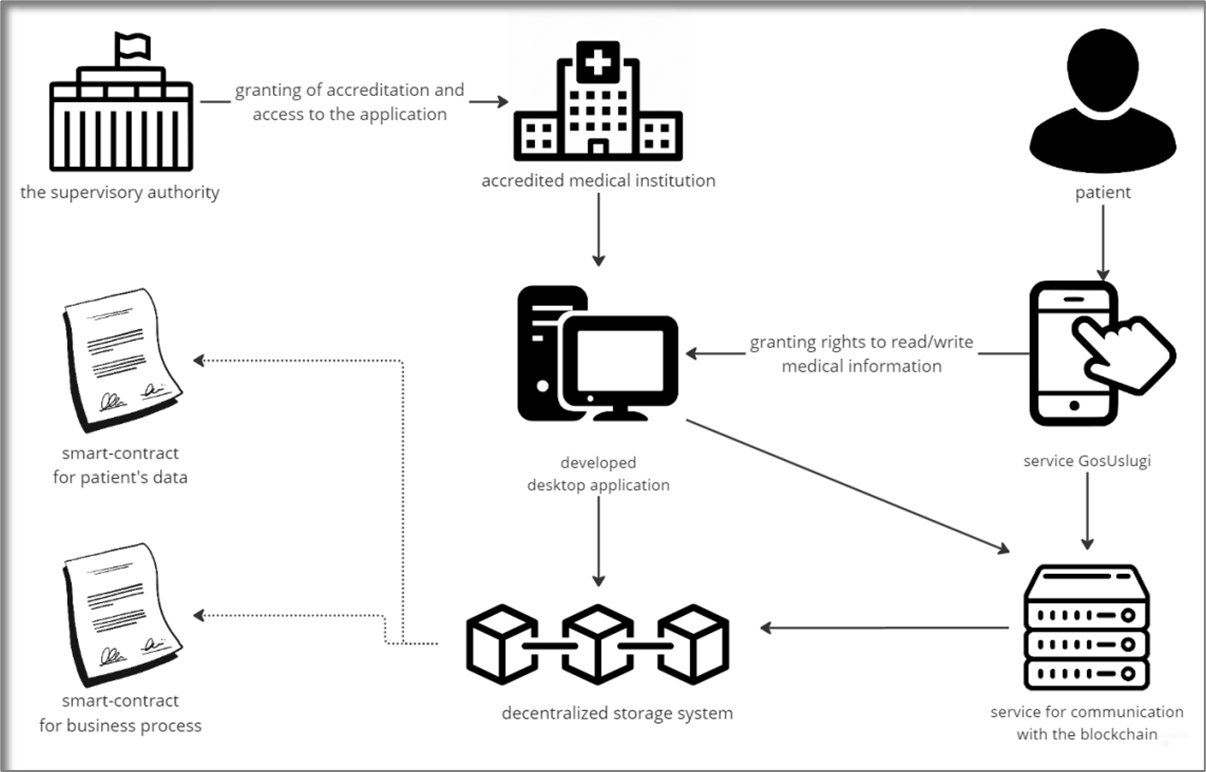
Клиентская часть: Алмазова Л.

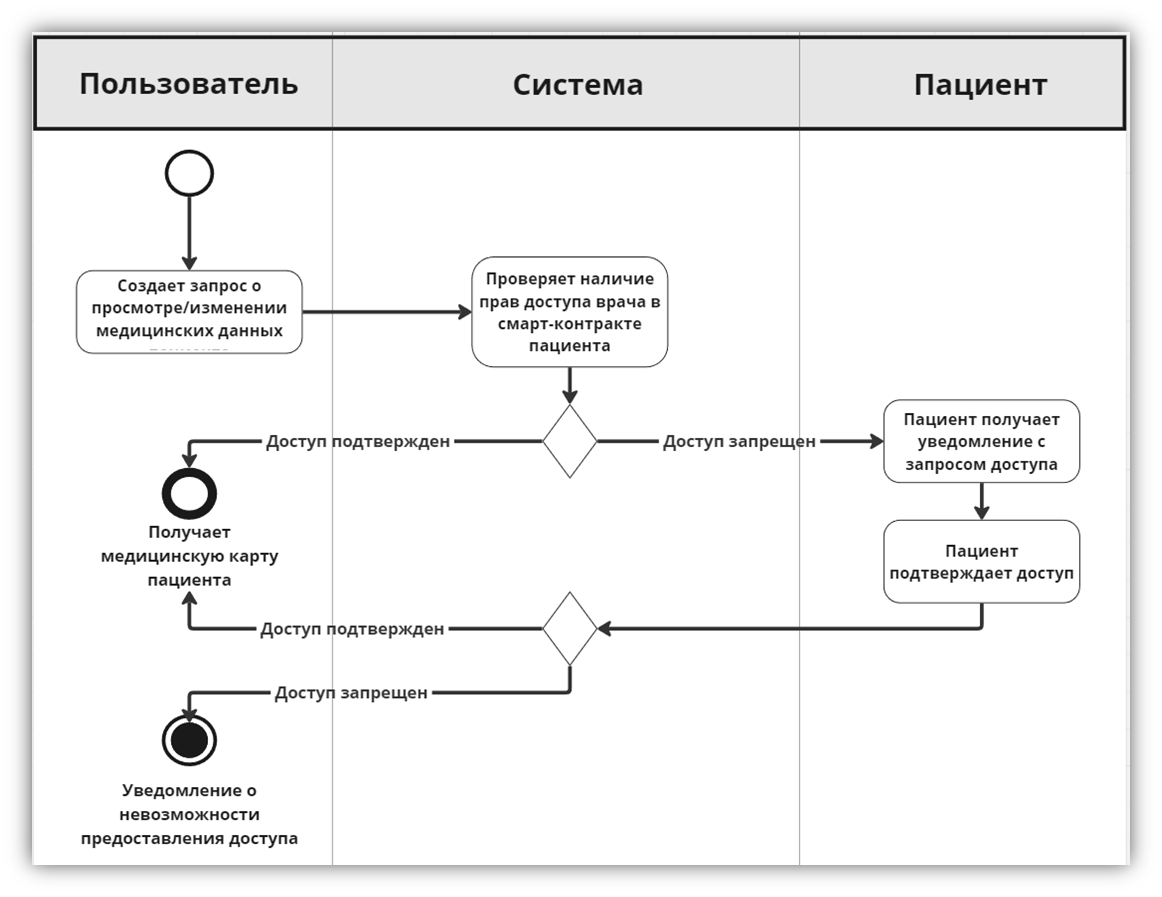
Серверная часть: Лаврова А.К.

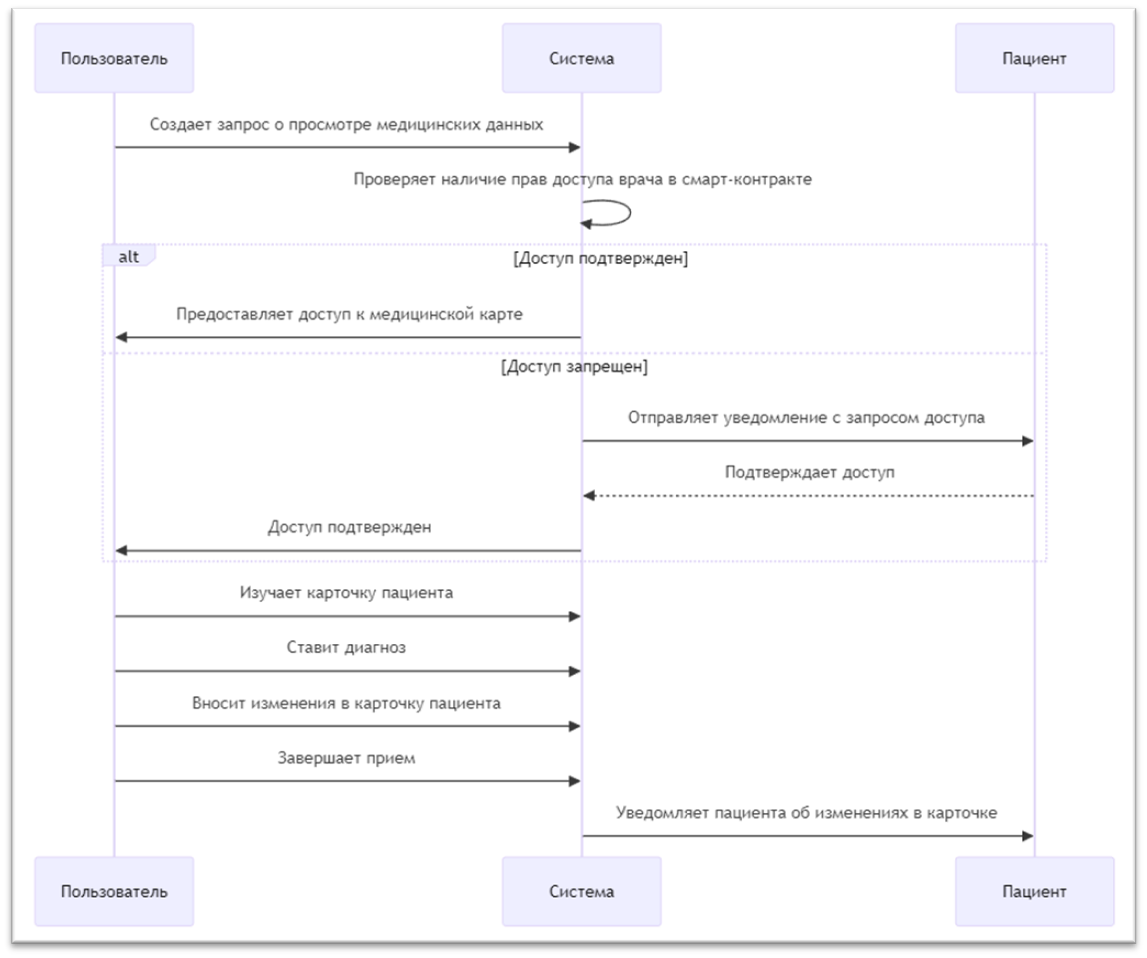
# 1. Общее описание проекта

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*

# 

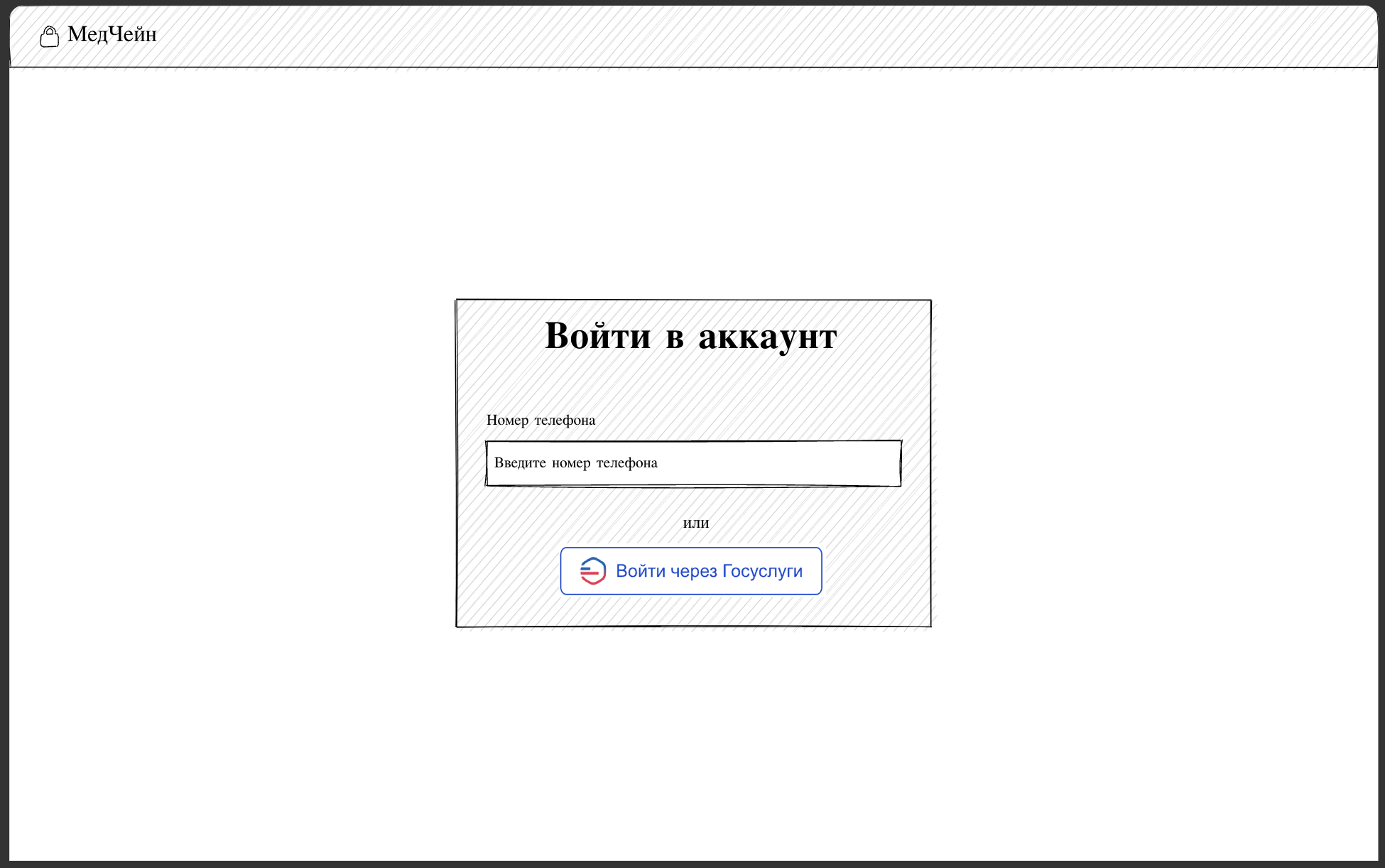
# Задачи дизайн

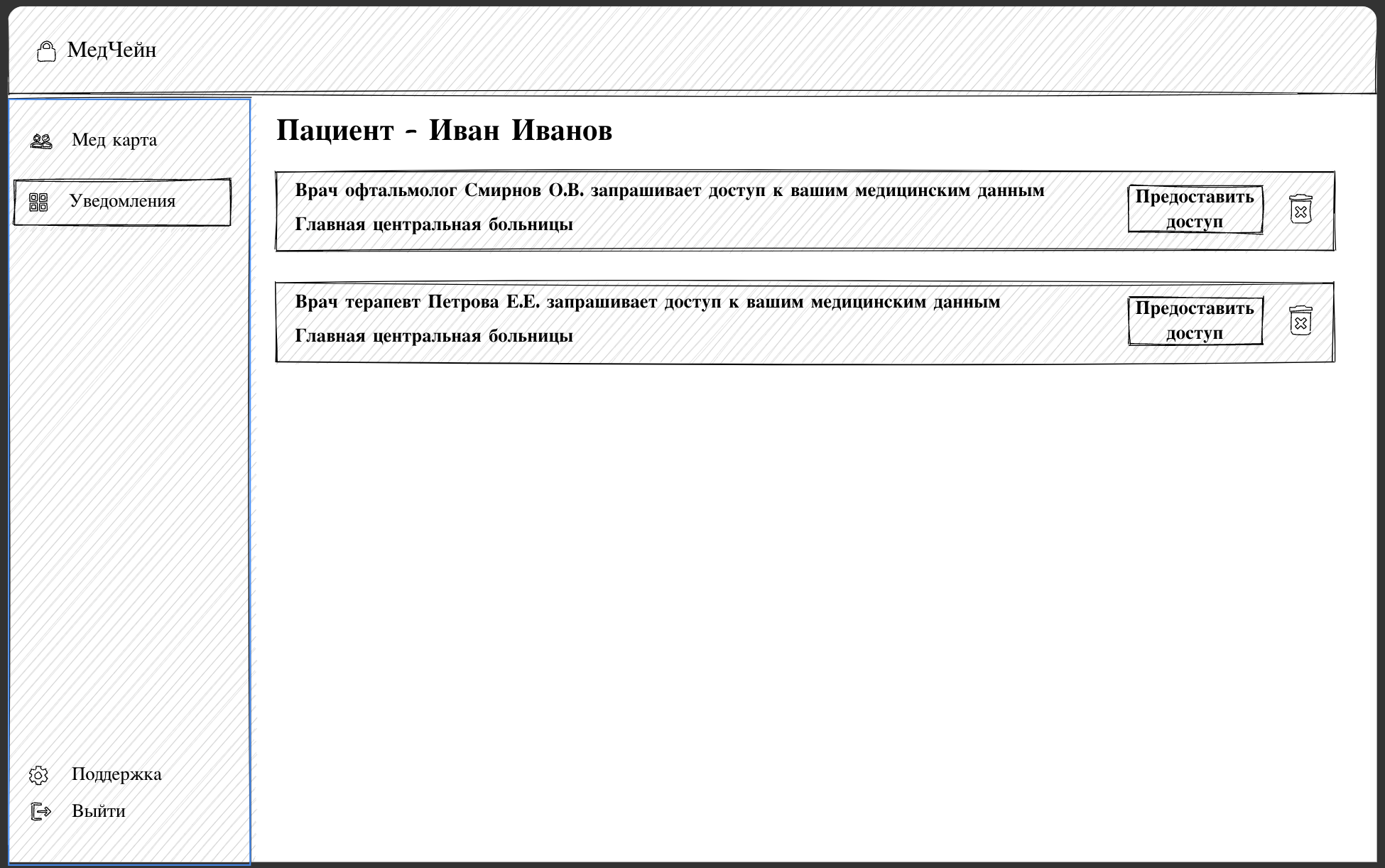
* Проработать цветовую палитру и стилистику
* Создать макеты страниц в figma
* Создать макеты адаптива под мобильные устройства

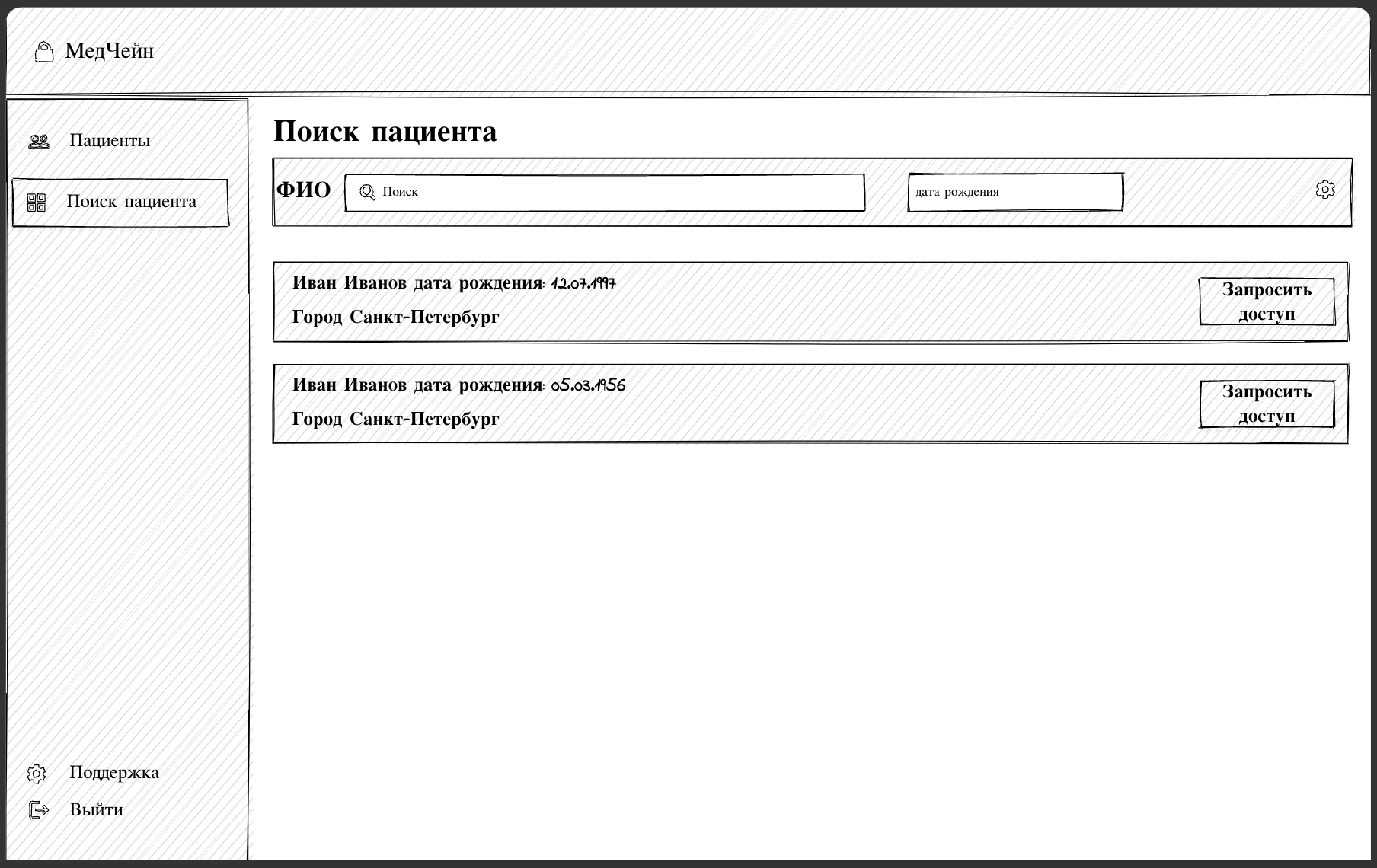
Основные сценарии, которые хотим реализовать, описаны в плане MVP. Откуда следует, что нам нужны следующие страницы:

* 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