

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**

КУРСОВОЙ ПРОЕКТ

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

Обучающийся: Чуприн Сергей Александрович, К3139

Санкт-Петербург 2024

СОДЕРЖАНИЕ

ВВЕДЕНИЕ	3
Актуальность темы	3
Цель проекта	3
Задачи проекта	4
РАБОТА НАД ПРОЕКТОМ	6
1. Описание проекта	6
2. Процессы работы над проектом.	8
1. Обсуждение целей и задач, обучение	8
2. Прототипирование	9
3. Разработка	10
3. Поставленные задачи и их решения	12
4. Анализ работы	14
5. Взаимодействие с командой	15
6. Взаимодействие с руководителем, оценка его деятельности	16
ЗАКЛЮЧЕНИЕ	17
СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ	18
ПРИЛОЖЕНИЕ. ТЕХНИЧЕСКОЕ ЗАДАНИЕ	19

ВВЕДЕНИЕ

Актуальность темы

В последние годы наблюдается значительный рост объемов медицинских данных, что связано с развитием технологий, увеличением числа диагностических исследований и внедрением электронных медицинских карт. В условиях современного здравоохранения, где информация о пациентах становится все более разнообразной и объемной, возникает необходимость в надежных системах управления данными, которые обеспечивают безопасность, доступность и прозрачность. Традиционные системы хранения данных часто не справляются с этими требованиями, что может приводить к утечкам информации, ошибкам в диагнозах и снижению качества медицинского обслуживания.

Одной из ключевых проблем является обеспечение конфиденциальности медицинских данных. Пациенты должны быть уверены в том, что их личная информация защищена от несанкционированного доступа. Блокчейн-технологии предоставляют уникальные возможности для решения этой проблемы. Они обеспечивают децентрализованное хранение данных, что минимизирует риски утечки информации и позволяет пациентам контролировать доступ к своим данным. Использование смарт-контрактов позволяет автоматизировать процессы доступа к информации, что также повышает уровень безопасности.

Кроме того, актуальность разработки системы управления медицинскими данными на основе блокчейн-технологий обусловлена необходимостью повышения уровня доверия между пациентами и медицинскими учреждениями. Система, основанная на прозрачных и неизменяемых записях, может значительно улучшить коммуникацию между врачами и пациентами, позволяя последним легко отслеживать доступ к своей информации и управлять им. Это особенно важно в условиях растущей обеспокоенности пациентов по поводу конфиденциальности их данных.

Таким образом, результаты данного проекта будут востребованы как среди медицинских учреждений, стремящихся повысить уровень безопасности и качества обслуживания своих пациентов, так и среди самих пациентов, которые хотят иметь контроль над своей медицинской информацией. Внедрение блокчейн-технологий в управление медицинскими данными может стать важным шагом к созданию более безопасной и эффективной системы здравоохранения.

Цель проекта

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.

Задачи проекта

Для достижения поставленной цели необходимо решить следующие задачи:

- разработка архитектуры системы: Создание структуры, которая будет включать блокчейн 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,
- обеспечение безопасности данных: Внедрение криптографических методов для защиты информации и управления доступом к данным,

- тестирование прототипа: Проведение тестирования системы для выявления возможных недостатков и улучшения функциональности

1 Описание проекта

Проект "Разработка прототипа серверной части системы управления медицинскими данными на основе блокчейн-технологий" представляет собой комплексное решение, направленное на создание инновационной платформы для хранения и управления медицинской информацией. В условиях современного здравоохранения, где объем данных о пациентах и их медицинской истории постоянно растет, существует настоятельная необходимость в надежных и безопасных системах, которые могут обеспечить защиту, доступность и целостность информации.

Суть проекта заключается в использовании блокчейн-технологий для создания децентрализованной системы, которая будет позволять пациентам и врачам безопасно взаимодействовать с медицинскими данными. Блокчейн обеспечивает прозрачность и неизменность записей, что критически важно для обеспечения доверия между всеми участниками процесса — пациентами, врачами и медицинскими учреждениями. В рамках проекта разрабатывается серверная часть системы, которая будет интегрирована с фронтенд-решениями и базой данных, обеспечивая эффективное управление доступом к медицинским записям.

Ключевым аспектом проекта является возможность пациентов контролировать доступ к своим данным. Они смогут подтверждать или отклонять запросы врачей на доступ к своей медицинской информации, что значительно повышает уровень их уверенности в безопасности личных данных. Врачи, в свою очередь, получают инструменты для поиска пациентов и управления их медицинскими записями, что упростит процесс оказания медицинских услуг.

Проект также включает в себя разработку смарт-контрактов, которые автоматизируют процессы доступа к данным и обеспечивают выполнение условий взаимодействия между пациентами и врачами. Это позволит минимизировать ошибки и повысить эффективность работы системы.

Реализация данного проекта предполагает создание прототипа, который будет протестирован на предмет функциональности и безопасности. Прототип должен быть развернут на удаленной машине для внешнего тестирования, что позволит собрать отзывы пользователей и внести необходимые улучшения.

В результате успешной реализации проекта планируется создать систему, которая не только улучшит качество обслуживания пациентов, но и повысит уровень доверия к медицинским учреждениям за счет обеспечения надежности и безопасности управления медицинскими данными.

2 Процессы работы над проектом

2.1 Обсуждение целей и задач, обучение

На первом этапе необходимо было познакомиться с проектом и с используемыми технологиями в целом.

В начале проекта мы провели “подготовительный” этап, на котором обсуждали цели и задачи. Это был важный момент для формирования общего понимания того, что мы хотим достичь. Мы изучили требования к системе, определили ключевые функции и технологии, которые будем использовать.

Следующим шагом стало обучение. Мы изучали статьи и видео по блокчейну, которые нам предоставил руководитель (например видео [1]-[3], а также [7]), а также разработали тестовый смарт-контракт библиотеки. Этот контракт позволяет добавлять, удалять и управлять книгами, а также предоставляет функции для аренды и возврата книг пользователями. В контракте есть структура данных Book, содержащая название, изображение и статус доступности книги. Мы реализовали основные функции, такие как добавление новой книги, вывод средств администратором, обновление адреса администратора и удаление книг из библиотеки. Пользователи могут арендовать книги, если они доступны и отправляют достаточную сумму, а также возвращать арендованные книги.

Работа над этим смарт-контрактом позволила мне развить множество важных навыков и получить ценный опыт. Во-первых, я научился программировать на Solidity, что является основным языком для разработки смарт-контрактов на платформе Ethereum. Я также получил понимание принципов работы блокчейна, включая транзакции и взаимодействие с контрактами. Работа со структурами данных и безопасностью смарт-контрактов помогла мне осознать важность проверки условий для предотвращения ошибок и злоупотреблений.

Во-вторых, я развил проектные навыки, такие как проектирование архитектуры контракта, тестирование функций и управление 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 понять, как работают смарт-контракты.

2.2 Прототипирование

На этом этапе команда сосредотачивается на формировании четкого представления о системе, ее функциональности и архитектуре, что позволяет заложить прочный фундамент для дальнейшей работы.

В начале этапа мы проводим анализ требований, который включает в себя изучение потребностей конечных пользователей — пациентов и врачей. Мы активно обсуждаем, какие функции будут наиболее полезны, и какие проблемы система должна решить. Это позволяет нам создать список функциональных требований, таких как возможность подтверждения или отклонения запросов на доступ к медицинским данным, а также управление профилями пользователей.

Следующим шагом является разработка архитектуры системы. Мы создаем схемы взаимодействия между компонентами, определяя, как будет организовано хранение данных в блокчейне и как пользователи будут взаимодействовать с системой. Это включает выбор технологий, таких как использование PostgreSQL для базы данных и React.js для фронтенда, а также Python с Django для бэкенда.

Создание макетов пользовательских интерфейсов с помощью инструментов дизайна, таких как Figma, является важной частью прототипирования. Мы разрабатываем визуальные представления страниц системы, что помогает команде понять, как будет выглядеть конечный продукт и как пользователи будут взаимодействовать с ним.

На этом этапе также продолжается изучение блокчейн-технологий. Мы разрабатываем тестовые смарт-контракты на языке Solidity для автоматизации процессов доступа к медицинским данным. Это обучение позволяет нам лучше понять принципы работы блокчейна и подготовить почву для его интеграции в систему.

Кроме того, мы уделяем внимание документации требований и архитектуры системы, что обеспечивает согласованность в команде и упрощает последующую разработку. Важно зафиксировать все решения и спецификации, чтобы каждый участник проекта имел доступ к актуальной информации.

Этап прототипирования завершается подготовкой к следующему этапу — непосредственной разработке системы. Мы должны быть готовы реализовать все запланированные функции и возможности на основе полученных знаний и документации.

2.3 Разработка

Этап разработки в проекте "Разработка прототипа серверной части системы управления медицинскими данными на основе блокчейн-технологий" представляет собой ключевую фазу, в которой команда начинает реализовывать все идеи и концепции, заложенные на этапе прототипирования. Этот этап охватывает множество процессов, от программирования до тестирования, и требует активного участия всех членов команды.

В начале этапа мы сосредоточились на разработке бэкенда системы с использованием Python и фреймворка Django. Мы также интегрировали PostgreSQL в качестве базы данных для хранения информации о пациентах и врачах. Это позволило нам обеспечить надежное и быстрое выполнение запросов к данным.

Параллельно с разработкой бэкенда шла работа над фронтендом, который создавался с использованием React.js. Одним из важных аспектов

этапа разработки стало интегрирование блокчейн-технологий. Мы разрабатывали смарт-контракты на языке Solidity, которые обеспечивают автоматизацию процессов доступа к медицинским данным. Я принимал участие как в разработке, так и в тестировании этих контрактов, что дало мне возможность увидеть их работу в реальных условиях и понять, как они взаимодействуют с остальными компонентами системы.

После завершения разработки основных функциональных компонентов начался этап тестирования системы. Мы проводили как функциональное, так и интеграционное тестирование, чтобы убедиться, что все части системы работают корректно и взаимодействуют друг с другом без ошибок. Также важным моментом была документация API с использованием Swagger. Это позволило нам создать удобный интерфейс для тестирования и интеграции системы другими разработчиками. Мы стремились обеспечить высокую степень документированности нашего кода, чтобы упростить дальнейшую работу над проектом.

В заключение этап разработки завершился развертыванием готовой системы на удаленной машине для внешнего тестирования. Это был захватывающий момент — видеть нашу работу в действии. Этап разработки стал для меня не только возможностью применить теоретические знания на практике, но и ценным опытом командной работы, который значительно расширил мои навыки в области программирования и разработки программного обеспечения.

3 Поставленные задачи и их решения

Передо мной была поставлена задача обучения языку Solidity и разработки на нем смарт-контрактов для системы управления медицинскими данными. Основная цель заключалась в создании смарт-контрактов для двух ролей: пациента и администратора, а также в их тестировании.

Первым шагом в решении этой задачи стало изучение языка Solidity. Я начал с изучения базовых концепций, таких как синтаксис, структуры данных и основные конструкции языка. Я использовал статьи и документацию, чтобы получить теоретические знания, а также практиковался на написании простых смарт-контрактов. Также я смотрел различные видео, чтобы получить более общее представление о блокчейне в целом. Это обучение дало мне уверенность в том, что я смогу реализовать более сложные контракты для нашего проекта.

После того как я освоил основы, я приступил к разработке смарт-контракта для пациента. Этот контракт должен был включать функции для управления доступом к медицинским данным, такие как возможность подтверждения или отклонения запросов врачей на доступ к информации. Я реализовал функции, которые позволяли пациенту видеть список врачей, имеющих доступ к его данным, и управлять этим доступом. Также я сделал функционал, позволяющий врачам вносить правки в медицинскую историю пациента (добавлять новую медицинскую запись, которая состоит из жалобы пациента, диагноза врача, информации о враче, который создал данную медицинскую запись), а также получать список всех медицинских записей. В процессе разработки я также учитывал требования безопасности, чтобы гарантировать защиту личной информации пациентов. Доступ к медицинским записям есть только у лечащих врачей.

После разработки контракта пациента я начал работать над созданием смарт-контракта для администратора. Этот контракт должен был обеспечивать управление профилями пользователей и предоставлять

администраторам возможность добавлять или удалять врачей и пациентов из системы. Я реализовал функции для создания и управления учетными записями врачей и пациентов, а также для проверки их прав на доступ к данным пациентов. Так же как и в контракте пациента, особое внимание было уделено вопросам безопасности.

После завершения разработки обоих контрактов я приступил к тестированию. Для этого я использовал инструменты Hardhat и Ethers.js, которые позволяют разрабатывать и тестировать смарт-контракты в локальной среде блокчейна. Я создал тестовые сценарии, которые имитировали различные действия пользователей — от запроса доступа до подтверждения или отклонения этих запросов. Это помогло мне выявить возможные ошибки и недочеты в логике контрактов.

4 Анализ работы

Анализируя свою работу над курсовым проектом "Разработка прототипа серверной части системы управления медицинскими данными на основе блокчейн-технологий", я могу выделить как положительные моменты, так и трудности, с которыми столкнулся в процессе выполнения.

В первую очередь, мне удалось успешно освоить язык Solidity и разработать смарт-контракты для пациента и администратора. Эти контракты обеспечивают автоматизацию процессов управления доступом к медицинским данным и взаимодействие между пользователями. Я также смог реализовать основные функции, например такие как отправка запросов на доступ, подтверждение или отклонение этих запросов, а также создание и управление профилями пользователей. Учитывая что до этого я никогда не сталкивался с блокчейном и разработкой на Solidity, это был значительный шаг вперед в моем обучении. Также способствовало прогрессу и то, что я не только изучил теорию, но и применил знания на практике.

Однако в процессе работы возникли и определенные трудности. Одной из основных проблем стало понимание концепций блокчейн-технологий и их интеграция в проект. Несмотря на то что я потратил время на изучение материалов и документации, некоторые аспекты оставались сложными для восприятия, особенно в контексте разработки смарт-контрактов. Это замедлило мой прогресс на начальных этапах работы над проектом.

Работа над проектом не всегда шла планомерно. Иногда возникали задержки из-за необходимости дополнительного изучения технологий или исправления ошибок в коде. Например, тестирование смарт-контрактов выявило несколько недочетов в логике, что потребовало времени на их исправление.

5 Взаимодействие с командой

После распределения на команды, мы сразу создали единый чат всего проекта, где были разделения по командам уже внутри проекта: фронтенд, бэкенд, дизайн и блокчейн. Моя команда (блокчейн) работала в основном отдельно от других, не особо взаимодействуя с другими командами, так как такова специфика нашей задачи.

Как только был создан чат, наша руководитель скинула все необходимые материалы для изучения. С самого начала проекта мы организовали регулярные встречи, на которых обсуждали цели и задачи, а также распределяли роли и обязанности среди участников. Также, если возникали какие-либо трудности при решении поставленных задач, на совещаниях обязательно обсуждали проблему и подсказывали как ее решить.

6 Взаимодействие с руководителем проекта и оценка его деятельности

Наш руководитель, Лаврова Анастасия, всегда очень понятно формулировала задачи, оперативно отвечала на все вопросы, помогала, если кто-то сталкивался с какими-либо трудностями. Также всегда давала обратную связь по проделанной работе, хвалила за правильные решения, подсказывала что можно исправить или доработать.

Она четко структурирует процесс работы над проектом, что очень помогало нам следовать установленному графику и не терять фокус на задачах. Анастасия всегда открыта для вопросов и обсуждений, что создавало атмосферу доверия и уверенности в своих силах. Я считаю, что Лаврова Анастасия прекрасный руководитель и она абсолютно заслуживает наивысшей оценки.

ЗАКЛЮЧЕНИЕ

В заключении можно подвести итоги выполнения проекта "Разработка прототипа серверной части системы управления медицинскими данными на основе блокчейн-технологий". Основная цель проекта заключалась в создании функционального прототипа системы, которая обеспечивала бы безопасное хранение и управление медицинскими данными с использованием блокчейн-технологий. Являясь частью команды, я могу сказать, что цели и задачи проекта, поставленные в техническом задании, были достигнуты, выполнены в полном объеме.

Мой вклад в достижение цели проекта заключался в разработке смарт-контрактов и их тестировании. Этот опыт не только обогатил мои технические навыки, но и научил работать в команде, что является важным аспектом успешной разработки программного обеспечения.

В итоге проект стал важным первым шагом в моем освоении блокчейн-разработки. Я горжусь тем, что стал частью этой команды и смог внести свой вклад в достижение общей цели.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Что такое блокчейн
<https://www.youtube.com/watch?v=kHybflaC-jE>
2. Что такое смарт-контракты
<https://www.youtube.com/watch?v=pyaIppMhuic&list=PLHx4UicbtUoYPDWk2aUwZoVKMkdRKtKWe&index=11&pp=iAQB>
3. Что такое газ в Ethereum
<https://www.youtube.com/watch?v=3ehaSqwUZ0s&list=PLHx4UicbtUoYPDWk2aUwZoVKMkdRKtKWe&index=6&pp=iAQB>
4. Форум с вопросами по Ethereum и блокчейну в целом
<https://ethereum.stackexchange.com/>
5. Документация Solidity <https://docs.soliditylang.org/en/v0.8.28/>
6. Документация Solidity на русском
<https://github.com/ethereum/wiki/wiki/%5BRussian%5D-%D0%A0%D1%83%D0%BA%D0%BE%D0%B2%D0%BE%D0%B4%D1%81%D1%82%D0%B2%D0%BE-%D0%BF%D0%BE-Solidity>
7. Статья про ethers.js
<https://dev.to/ajcwebdev/a-first-look-at-ethers-and-hardhat-5848>

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.

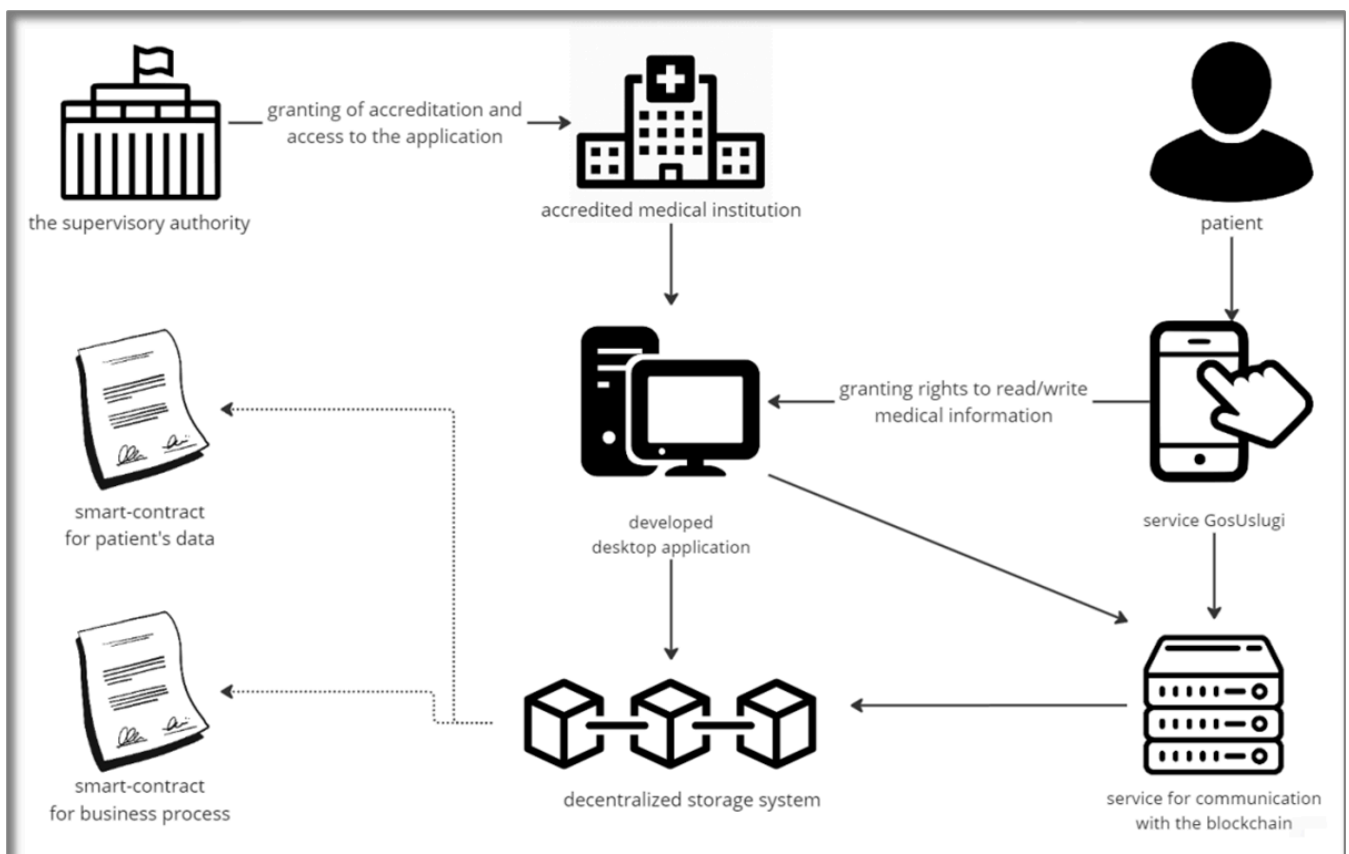
2. Отсутствие прозрачности доступа к данным

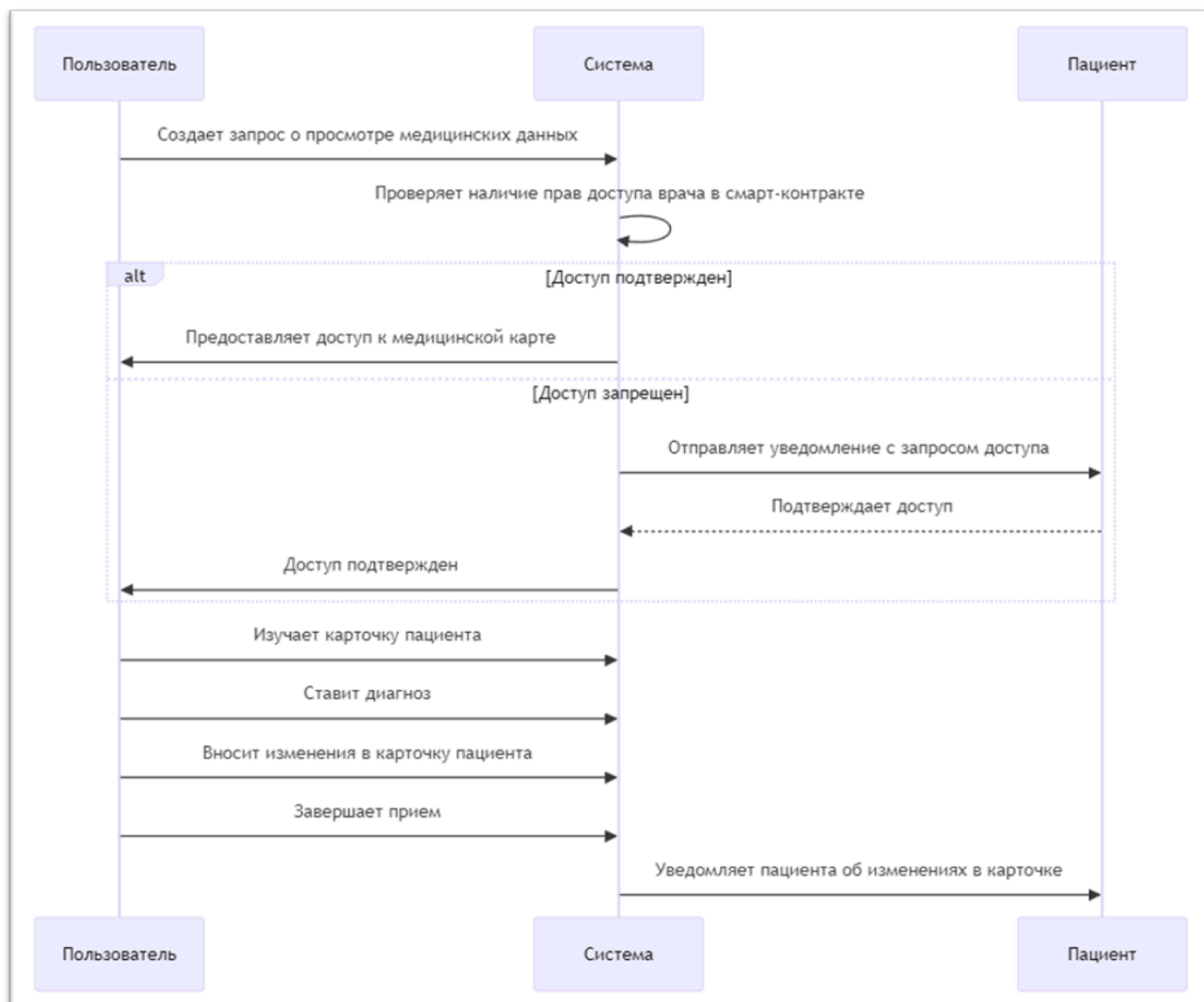
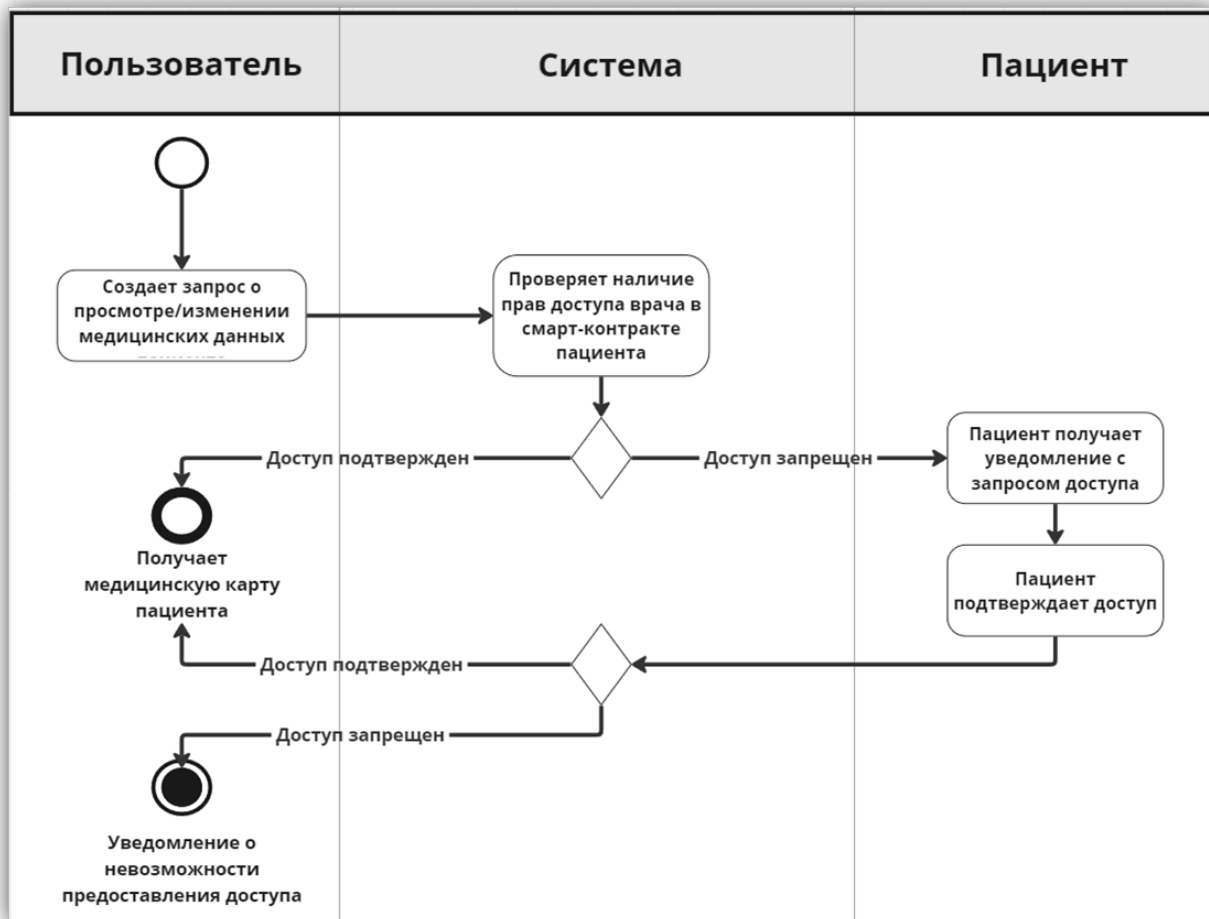
В текущей системе пациенты не имеют контроля над тем, кто и когда просматривает их медицинские данные. Часто такие доступы происходят без ведома пациента, что ставит под угрозу его право на приватность и защиту данных. К тому же пациенты не имеют уверенности в подлинности внесённых записей, так как нет 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 ознакомливается 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:

МедЧейн

Войти в аккаунт

Номер телефона

или

 Войти через Госуслуги

МедЧейн

Мед карта

Уведомления

Поддержка

Выйти

Пациент - Иван Иванов

Врач офтальмолог Смирнов О.В. запрашивает доступ к вашим медицинским данным

Главная центральная больницы

Предоставить доступ

Врач терапевт Петрова Е.Е. запрашивает доступ к вашим медицинским данным

Главная центральная больницы

Предоставить доступ

МедЧейн

Пациенты

Поиск пациента

Поддержка

Выйти

Поиск пациента

ФИО

Поиск

дата рождения

Иван Иванов дата рождения: 42.03.1987

Город Санкт-Петербург

Запросить доступ

Иван Иванов дата рождения: 05.03.1956

Город Санкт-Петербург

Запросить доступ

Задачи Front-end

- По макетам сверстать страницы
- Учесть адаптив под мобильные устройства
- Настроить взаимодействие с бэком

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:

1. 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.

2. 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.

3. 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).

4. 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.

5. 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

1. 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"
  }
]
```

2. 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"
}
```

3. 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"
  }
]

```

4. 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

1. 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"
}
```

2. 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
}
```

3. 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, для 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":
"0x0000000000000000000000000000000000000000000000000000000000000000"
  },
  {
    "patient_id": 101,
    "name": "Иванов Иван Алексеевич",
    "date_of_birth": "2011-08-03T00:00:00+03:00",
    "contract_address":
"0x1100000000000000000000000000000000000000000000000000000000000011"
  }
]

```

4. 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":
0x0000000000000000000000000000000000000000000000000000000000000000",
    "access_granted_date": "2024-08-22T15:00:00+03:00"
  },
]
```

]

Запрос для больниц

1. 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
}
```

2. 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
}
```

3. 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)**
Там очень быстро расскажут про базовые термины, введут в курс дела
- **Что такое смарт-контракты**
 **What are Smart Contracts in Crypto? (4 Examples + Animated)**
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)**
Это важный компонент в экосистеме EVM-блокчейном, поэтому тоже хорошо бы понять что это
- **Публичные и приватные ключи. RSA**
 **Asymmetric Encryption - Simply explained**
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:

1. 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).

2. 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