

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 клиентской части системы управления медицинскими данными на основе технологии блокчейна»

Обучающийся: Барецкий Максим Степанович

Санкт-Петербург 2024

СОДЕРЖАНИЕ

ВВЕДЕНИЕ	3
1 Описание проекта	4
1.1 Контекст и проблемы в сфере хранения медицинских данных	4
1.2 Идея решения: Децентрализованная система хранения медицинских данных	5
1.3 Основные функции системы.....	6
1.4 План MVP	7
1.5 Требования к стеку технологий.....	7
1.6 Архитектура системы.....	8
1.7 Риски и вызовы проекта.....	9
1.8 Заключение	9
2 Процесс работы над проектом.....	10
3 Суть проблемы, поставленной передо мной	11
4 Как я решал поставленную задачу	13
4.1 Разработка формы логина и регистрации:	13
4.2 Валидация данных:.....	14
4.3 Ошибки на странице логина:	14
4.4 Адаптивный дизайн:.....	14
4.5 Реализация по мере поступления проблем:	14
5 Анализ своей работы.....	15
ЗАКЛЮЧЕНИЕ	19
СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ	21
ПРИЛОЖЕНИЕ.....	22

ВВЕДЕНИЕ

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.

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

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.

1 Описание проекта

1.1 Контекст и проблемы в сфере хранения медицинских данных

На текущий момент в сфере здравоохранения существует несколько основных проблем, связанных с хранением и управлением медицинскими данными пациентов:

1.1.1 Разрозненное хранение данных
Медицинские данные пациентов хранятся в разных учреждениях и системах, которые не синхронизированы между собой. Например, каждый пациент имеет отдельную карточку в разных медицинских организациях, и получить полную картину его здоровья становится сложной задачей. Это затрудняет диагностику, повышает вероятность ошибок и снижает качество обслуживания.

1.1.2 Отсутствие прозрачности доступа к данным
В текущей системе пациенты не имеют полного контроля над тем, кто и когда просматривает их медицинские данные. Доступ к данным может быть предоставлен без ведома пациента, что нарушает его право на приватность. Кроме того, часто отсутствует возможность отслеживать изменения в медицинских записях, что может вызвать сомнения в их точности и подлинности.

1.1.3 Централизованное хранение с риском утечек данных
В большинстве случаев данные хранятся на централизованных серверах, которые могут быть уязвимыми для взломов. К тому же многие медицинские учреждения имеют ограниченные бюджеты и не могут обеспечить должный уровень безопасности данных. Это повышает риск утечек конфиденциальной информации, о которых пациент может даже не узнать.

1.2 Идея решения: Децентрализованная система хранения медицинских данных

Проект предполагает создание системы, в которой медицинские данные будут храниться на блокчейне. Это позволит решить многие из вышеупомянутых проблем:

– Хранение данных на блокчейне обеспечит большую безопасность и контроль над медицинскими записями, исключая возможность их потери или подделки.

– Прозрачность доступа будет гарантирована с помощью записи всех операций с данными в блокчейн, где пациенты смогут отслеживать, кто и когда просматривал их данные.

– Контроль доступа пациентом — пациенты смогут самостоятельно разрешать или запрещать доступ врачам к своим данным через интерфейс.

1.3 Основные функции системы

1.3.1 Регистрация и аутентификация пользователей
Пациенты, врачи и медицинские учреждения должны иметь возможность зарегистрироваться в системе с помощью безопасной аутентификации, например, через двухфакторную аутентификацию.

1.3.2 Управление доступом к данным
Пациент может видеть запросы на доступ к его данным от врачей и медицинских учреждений, а также подтверждать или отклонять эти запросы. Врачи могут запрашивать доступ к данным пациента, а пациенты могут управлять этими запросами.

1.3.3 Шифрование данных
Все данные пациента (медицинская карта, результаты анализов, история болезни и т. д.) будут зашифрованы с использованием современных алгоритмов шифрования для обеспечения их безопасности.

1.3.4 История изменений и доступность данных
Все изменения в данных пациента (например, добавление новых анализов или записей) будут фиксироваться в блокчейне. Это обеспечит прозрачность и предотвратит несанкционированные изменения.

1.3.5 Система поиска и фильтрации данных
Врачи смогут искать пациентов по имени, дате рождения или другим ключевым параметрам. Пациенты смогут отслеживать, кому был предоставлен доступ к их данным, и могут просматривать историю запросов.

1.4 План MVP

На стадии MVP планируется реализовать следующие основные сценарии работы системы:

1.4.1 Пациент:
Пациент заходит на свою страницу и видит запрос на доступ со стороны врача. Он может либо подтвердить, либо отклонить запрос. Пациент также может просматривать список врачей, которым уже был предоставлен доступ к его данным, и управлять этими доступами.

1.4.2 Врач:
Врач ищет пациента в базе и запрашивает доступ к его медицинским данным. После получения доступа врач может ознакомиться с медицинской картой пациента. Врач может вносить записи в медицинскую карту пациента, например, обновлять информацию о результатах анализов и назначениях.

1.4.3 Технологические взаимодействия:
Данные будут храниться в блокчейне, а доступ к ним будет регулироваться с помощью криптографических методов. Все операции, связанные с доступом и изменением данных, будут фиксироваться в блокчейне для обеспечения прозрачности.

1.5 Требования к стеку технологий

1.5.1 Дизайн: Figma
Использование Figma для проектирования интерфейса, создания макетов и прототипов. Дизайн должен быть ориентирован на удобство пользователя и соответствовать медицинским стандартам UX/UI.

1.5.2

Front-end:

React.js

Использование React.js для разработки интерактивных пользовательских интерфейсов. React.js обеспечит быстроедействие и модульность компонентов, что важно для масштабирования системы. Компоненты интерфейса будут взаимодействовать с API для обмена данными с сервером и блокчейном.

1.5.3

Back-end:

Python

(Django)

Django будет использоваться для создания RESTful API, обработки запросов от фронтенда, а также для аутентификации и авторизации пользователей. Django обеспечит стабильную и безопасную серверную сторону системы с возможностью масштабирования. Для шифрования и хранения данных будут использованы библиотеки Python, такие как `bcrypt` и `django-cryptfield`.

1.5.4 Блокчейн: Solidity (Hardhat, Ethers.js)

- Solidity используется для написания смарт-контрактов, которые управляют доступом к медицинским данным.
- Hardhat — среда разработки для тестирования смарт-контрактов и развертывания их в приватной сети Ethereum.
- Ethers.js будет использоваться для взаимодействия с блокчейном на фронтенде (взаимодействие с кошельками, подписывание транзакций и запросы данных). Все запросы и операции с данными будут записываться в блокчейн для обеспечения их безопасности и прозрачности.

1.6 Архитектура системы

1.6.1

Front-end

Интерфейс пациента и врача с возможностью управления доступом, просмотра истории данных и редактирования информации. Компоненты для взаимодействия с API и отображения статусов транзакций.

Django API для обработки запросов, аутентификации и взаимодействия с базой данных. Хранение метаданных о пользователях и запросах на доступ, интеграция с блокчейном для записи транзакций.

Смарт-контракты для управления доступом к медицинским данным. Хранение хэшей медицинских данных в блокчейне для обеспечения их неизменности и прозрачности.

1.7 Риски и вызовы проекта

Необходимо обеспечить защиту персональных данных пациентов и правильную реализацию механизмов шифрования и аутентификации.

Важно обеспечить возможность масштабирования системы для работы с большим количеством пользователей и запросов, а также оптимизировать взаимодействие с блокчейном для повышения производительности.

Следует учитывать законодательные требования в области медицинской информации, такие как HIPAA (США) или GDPR (ЕС), при разработке и реализации системы.

1.8 Заключение

Проект на основе блокчейна представляет собой решение для безопасного хранения и управления медицинскими данными, обеспечивая прозрачность и контроль доступа для пациентов и врачей. Это позволит улучшить качество обслуживания и защитить права пациентов в контексте цифровизации здравоохранения.

2 Процесс работы над проектом

Процесс работы над проектом включал несколько ключевых этапов, каждый из которых был организован в рамках методологии Agile, с гибким подходом к изменяющимся требованиям и задачам. На старте проекта команда провела несколько созвонов, на которых нам объяснили основы React.js, что позволило выровнять уровень знаний всех участников и подготовиться к дальнейшей работе.

После этого проект был разбит на итерации, и задачи распределялись между участниками на основании их навыков и интересов. Взаимодействие между фронтенд и бэкенд-разработчиками, а также с командой, работающей над интеграцией блокчейн-части, было обеспечено через регулярные совещания и синхронизации. Я был частью фронтенд-команды, которая занималась реализацией интерфейса. Моя задача — создание форм для регистрации/логина, валидацию данных и адаптивность для различных экранов.

Рисунок 1 – форма входа

Рисунок 2 – форма регистрации

3 Суть проблемы, поставленной передо мной

Задача, поставленная передо мной, заключалась в разработке форм регистрации и логина для веб-приложения, обеспечении их функциональности, безопасности данных и удобства для пользователей. В дополнение к этому требовалось реализовать:

- Валидацию вводимых данных (корректность email, сложность пароля и т.д.).
- Обработку ошибок, например, если пользователь ввёл неверный логин или пароль.

Вход в аккаунт

[Нет аккаунта?](#)[Регистрация](#)

Возникли проблемы?

Возникли проблемы со входом?
Отправьте обращение с подробным
описанием проблемы:

Введите Email/Номер телефона

Опишите проблему

Отправить

Данные для входа

+7 (012) 121-21-21

.....

☐ Врач ☒ Пациент

Не удалось
войти

Войти

Рисунок 3 – форма ошибки

Кроме того, важно было, чтобы страницы регистрации и логина выглядели одинаково хорошо как на мобильных устройствах, так и на десктопах, обеспечивая отзывчивый (адаптивный) дизайн. В дополнение к этим задачам, необходимо было проработать систему отображения ошибок на странице логина — это особенно важно для повышения удобства использования и предотвращения недоразумений со стороны пользователей.

The image displays two mobile application registration screens. Both screens have a header 'Регистрация' (Registration) with a logo 'MD MS' in the top right corner. Below the header, there are two buttons: 'Уже есть аккаунт?' (Already have an account?) and 'Вход' (Login).

The left screen, titled 'Данные для регистрации' (Registration data), contains the following input fields:

- ФИО (Full Name)
- mm/dd/yyyy (Date of Birth)
- Серия и номер паспорта (Passport Series and Number)
- Номер СНИЛС (SNILS Number)
- Номер страхового полиса (Insurance Policy Number)

 A 'Далее >' (Next) button is located at the bottom right.

The right screen, also titled 'Данные для регистрации' (Registration data), contains the following input fields:

- Введите номер телефона (Enter phone number)
- Введите email (Enter email) - This field has a red error message: 'Please fill out this field.'
- Введите Вашу поликлинику (Enter your clinic)
- Пароль (Password) - Represented by a yellow bar with dots.
- Повторите пароль (Repeat password)

 Below the fields is a checkbox labeled 'Соглашаюсь на обработку персональных данных' (I agree to the processing of personal data). An 'Отправить' (Send) button is at the bottom right.

Рисунок 4, 5 – форма регистрации на мобильном устройстве

Таким образом, основная проблема заключалась в разработке удобных и функциональных форм, которые бы не только корректно работали, но и обеспечивали хороший пользовательский опыт.

4 Как я решал поставленную задачу

Для решения поставленных задач я использовал React.js, основы которого я изучал в процессе работы над проектом с помощью видео на YouTube[1][2], также я отмечал свой прогресс на дорожной карте[3]. После того как я изучил основы, я сделал тестовый проект[4], чтобы показать то, чему я научился руководителю проекта. По мере того как дизайнеры разрабатывали макеты, я начинал реализацию компонентов с использованием React, адаптируя их под требования макета.

4.1 Разработка формы логина и регистрации:

Форма регистрации/логина: Основной задачей было создание формы с полями для ввода логина и пароля. Для этого я использовал компоненты React, управляемые с помощью состояния (state). При отправке формы проверялись введенные данные и в случае ошибок отображались сообщения, что обеспечивало хороший UX[5].

4.2 Валидация данных:

Валидация ввода: Для валидации данных (например, проверка на корректность email, длину пароля, обязательность всех полей) использовал регулярные выражения и условные операторы. Была настроена логика, которая уведомляла пользователя о несоответствиях (например, "Пароль должен содержать не менее 8 символов").

4.3 Ошибки на странице логина:

Обработка ошибок: Важной частью работы было создание компонента для отображения ошибок на странице логина. Этот компонент показывал пользователю, если введенные данные были некорректными (например, неверный логин или пароль). Это позволило обеспечить более ясную и понятную обратную связь пользователю.

4.4 Адаптивный дизайн:

Адаптивность страниц: Используя CSS-медиа-запросы и технику гибкой вёрстки (flexbox, grid), я сделал страницы адаптивными. Это позволило интерфейсу корректно отображаться на различных устройствах (смартфонах, планшетах, десктопах) с разными разрешениями экрана.

4.5 Реализация по мере поступления проблем:

По мере реализации и тестирования страниц я сталкивался с рядом технических и дизайнерских проблем. Например, были случаи, когда валидация данных некорректно обрабатывала определенные сценарии, или

компоненты не работали на разных разрешениях экрана. Каждую такую проблему я решал по мере поступления: изучал причины и исправлял их. Например, для некоторых ошибок использовался кастомный компонент ошибок, а также проводилась оптимизация CSS для корректного отображения на разных экранах.

Каждый шаг разработки был поддержан тесным взаимодействием с другими членами команды, чтобы обеспечить правильную интеграцию с серверной частью и блокчейн-решением.

5 Анализ своей работы

В ходе выполнения курсового проекта все поставленные передо мной задачи были успешно выполнены. Я смог создать формы регистрации и логина, реализовать валидацию данных и обработку ошибок, а также обеспечить адаптивность этих страниц для различных устройств. Каждая из задач была решена в установленный срок, и все компоненты работали корректно в рамках требований проекта.

Что получилось:

- Форма регистрации и логина: Я успешно разработал компоненты для ввода данных (логин, пароль, email) и связал их с состоянием компонента с использованием `useState` в `React`. Формы работали стабильно, вводимые данные сохранялись в состоянии, и была реализована логика отправки формы.

- Валидация данных: Я внедрил валидацию для полей ввода, используя регулярные выражения для email и пароля, что позволило исключить ввод некорректных данных. В случае ошибок форма выводила сообщения, которые информировали пользователя о неправильных данных.

– Обработка ошибок: При неверном вводе данных на странице логина я реализовал отображение сообщений об ошибках. Если пользователь вводил неправильный логин или пароль, они сразу же получали уведомление с подробным объяснением ошибки.

– Адаптивность: Благодаря использованию CSS media queries и технологий Flexbox и Grid, я обеспечил адаптивность страницы. Все элементы корректно отображались на различных устройствах и экранах, от мобильных телефонов до десктопов.

Основные трудности:

– Адаптивность на всех устройствах: Несмотря на то, что страницы корректно отображались на большинстве устройств, возникали некоторые сложности с точной настройкой отображения на экранах с нестандартным разрешением. На мобильных устройствах возникали проблемы с выравниванием элементов формы при использовании большого количества полей ввода. Это требовало дополнительных настроек в CSS и тестирования на различных устройствах.

– Обработка ошибок валидации: На начальных этапах разработки возникли сложности с правильным отображением сообщений об ошибках. Проблемы заключались в том, как синхронизировать данные с состоянием компонента и корректно отображать сообщения в случае, если форма была отправлена с некорректными данными.

Планомерная работа и её трудности:

Процесс работы был в целом планомерным, но возникали моменты, когда приходилось отклоняться от изначального плана, чтобы решить возникающие проблемы. Например, на этапе адаптивной верстки мне пришлось несколько раз переработать структуру компонента, чтобы

обеспечить правильное отображение на мобильных устройствах. Однако, благодаря этому я научился лучше управлять состоянием и выстраивать взаимодействие между компонентами, а также находить решения для оптимальной верстки.

Что мешало планомерной работе:

Основными факторами, которые замедляли процесс, были проблемы с адаптивной версткой, а также необходимость дополнительного тестирования и правки ошибок валидации. Иногда на решение проблем с дизайном или пользовательским интерфейсом уходило больше времени, чем планировалось, что затрудняло переход к следующим этапам разработки.

Чему я научился:

– Глубже изучил React.js: В ходе проекта я значительно улучшил свои знания и навыки работы с React. Особенно полезными были знания о работе с состоянием компонентов, хуках (например, `useState`, `useEffect`) и принципах их использования в реальных проектах.

– Улучшил навыки работы с CSS: Процесс адаптивной верстки дал мне больше опыта в использовании CSS media queries, Flexbox и Grid, что позволило более уверенно подходить к решению задач, связанных с дизайном.

– Работа с ошибками и валидацией: Я научился правильно обрабатывать и выводить ошибки на стороне клиента, что значительно улучшило пользовательский опыт. Понимание того, как настроить валидацию на стороне фронтенда, стало важным достижением для меня.

– Интеграция с макетами: Взаимодействие с дизайнерами и корректировка кода в соответствии с макетами стало важным навыком. Я научился учитывать детали дизайна, такие как расположение элементов и их адаптивность для разных разрешений экранов.

В целом, все поставленные задачи были выполнены, и я успешно завершил проект, следуя плану, решая возникающие проблемы и улучшая свои навыки в процессе работы.

ЗАКЛЮЧЕНИЕ

Цель проекта была достигнута в полном объеме. Все ключевые задачи, поставленные на начальном этапе, были успешно выполнены. Формы регистрации и логина были реализованы с использованием React.js, обеспечив необходимую функциональность и адаптивность для различных устройств. Также была реализована валидация данных, обеспечивающая корректность ввода информации пользователями, а форма ошибки на странице логина была продумана и выполнена с учётом всех требуемых сценариев. Все задачи, включая адаптивность страниц, были выполнены в срок и соответствовали требованиям технического задания.

В процессе работы я решал несколько ключевых задач: создание форм регистрации и логина, обеспечение валидации данных, обработка ошибок на странице логина и адаптивность страниц для различных устройств. Эти задачи были решены с учётом всех требований, а результат работы полностью соответствовал ожиданиям.

Не возникло ситуаций, когда какая-либо из поставленных задач осталась невыполненной. Были моменты, когда возникали сложности с адаптивной версткой и взаимодействием с сервером, но они были решены оперативно, не повлияв на общий ход работы и сроки сдачи проекта.

В процессе выполнения проекта не возникло серьёзных проблем, которые могли бы помешать достижению цели. Единственные трудности, которые были, связаны с настройкой адаптивности, но все они были решены на этапе реализации.

Мой вклад в достижение цели был значительным. Я отвечал за разработку ключевых элементов фронтенда, включая формы регистрации и логина, валидацию данных и обеспечение адаптивности страниц. Кроме того, я активно участвовал в решении возникающих технических проблем, помогал коллегам и взаимодействовал с другими членами команды для достижения

общей цели. Моя работа позволила нам успешно завершить проект и достичь всех поставленных целей.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Learn React Js – Full Course for Beginners – Tutorial 2019 // youtube.com
URL: <https://www.youtube.com/watch?v=DLX62G4lc44&t=1s>
2. React Full Course for free (2024) // youtube.com URL:
<https://www.youtube.com/watch?v=CgkZ7MvWUAA>
3. React Developer Roadmap: Learn to become a React developer // roadmap.sh
URL :<https://roadmap.sh/react>
4. House // figma.com URL:
<https://www.figma.com/design/SjHvI8W1yzwJjzyUrCPpsI/House?node-id=3-838>
5. What are the most effective UX design principles for creating a seamless user experience? // linkedin.com URL: <https://www.linkedin.com/advice/3/what-most-effective-ux-design-principles>

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

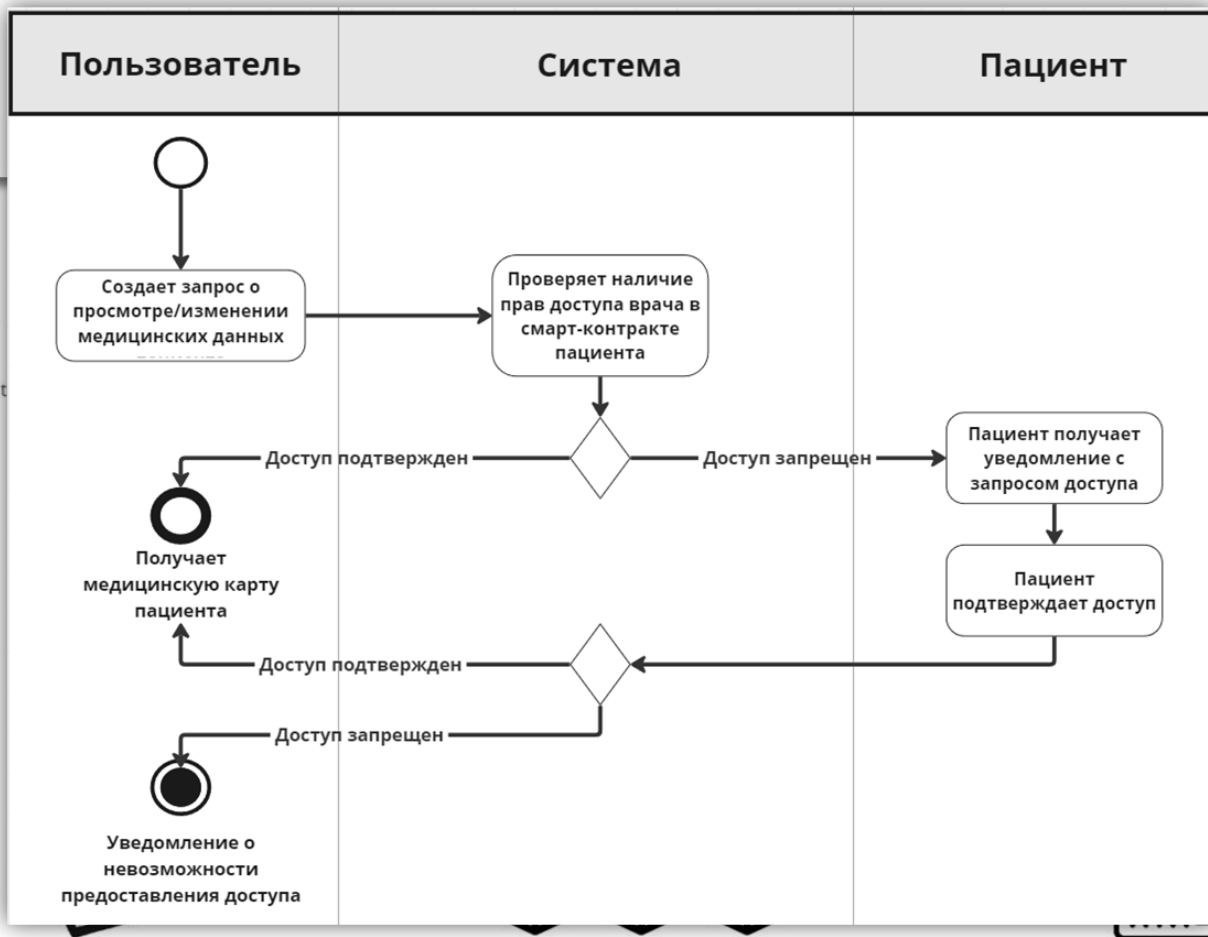
В большинстве медицинских учреждений данные пациентов хранятся в локальных базах данных. Каждый пациент имеет отдельные карточки в разных учреждениях, что создаёт серьёзные сложности при попытке собрать полную медицинскую историю. Эта разрозненность препятствует быстрой и точной диагностике, а также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

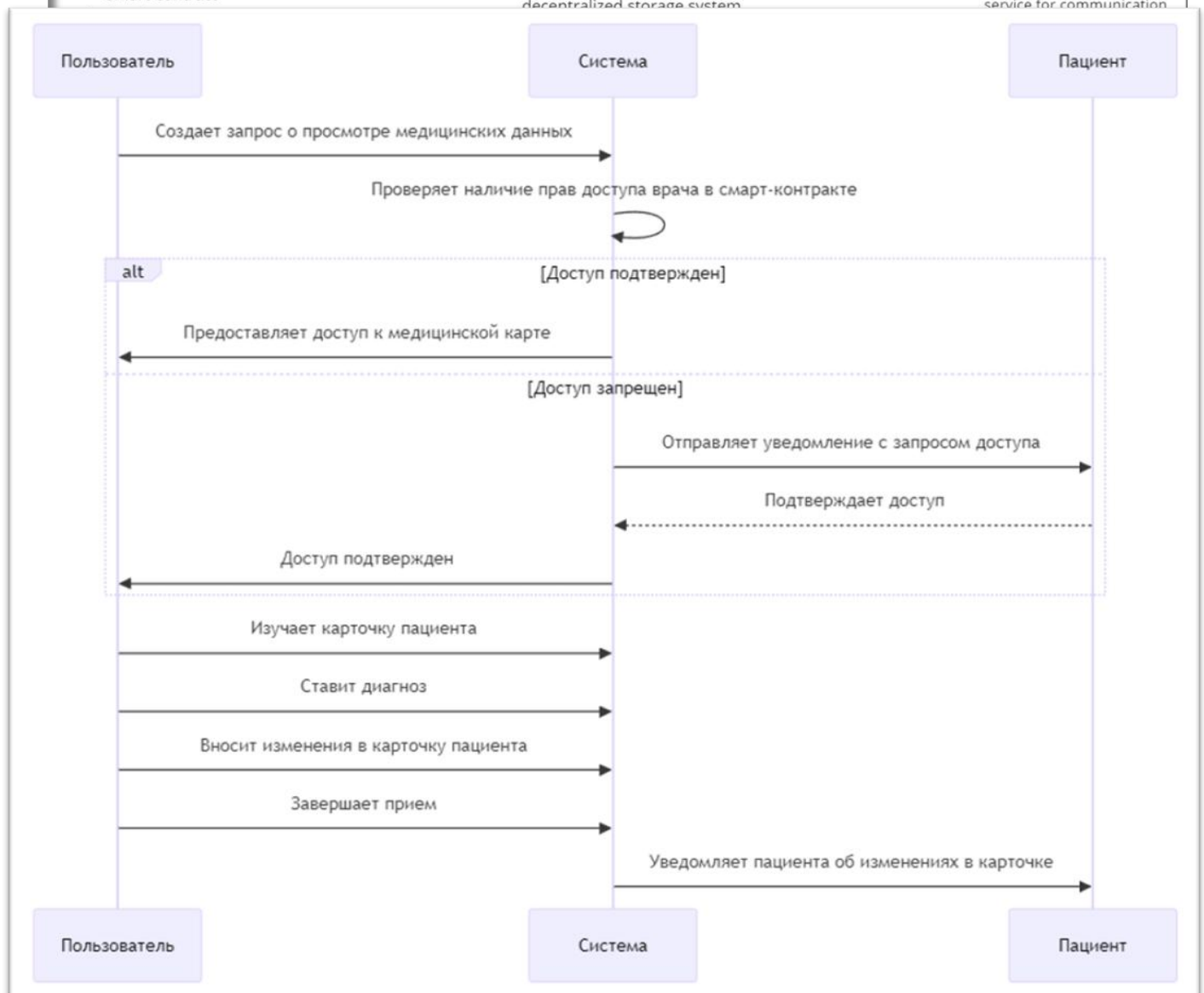
В большинстве медучреждений данные хранятся на централизованных серверах, которые часто подвергаются риску взломов и утечек. Учитывая, что многие учреждения сталкиваются с ограничениями по бюджету, уделять внимания мерам информационной безопасности оказывается недостаточным. Это создаёт условия для возможных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



smart-contract

decentralized storage system

service for communication



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 ознакоми́вается 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

Введите 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: 12.07.1999
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",
}
```



```

        "doctor_id": 2,
        "doctor": "Сергеев Сергей Сергеевич",
        "public_key": "0x0000000000000000000000000000000000000000000000000000000000000000"
    }
]

```

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