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 **45.03.04 Интеллектуальные системы в гуманитарной сфере**

Образовательная программа **Языковые модели и искусственный интеллект**

К У Р С О В О Й П Р О Е К Т

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

Обучающийся: Ломакина Мария Сергеевна, группа К3161

Санкт-Петербург 2024

СОДЕРЖАНИЕ

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

Наш проект занимался разработкой системы управления медицинскими данными. Мы создали прототип сервиса, предназначенного специально для медицинских учреждений. Основная задача этого сервиса — улучшить качество обслуживания пациентов, обеспечить безопасное хранение медицинских данных и сделать поиск нужной информации для врачей и пациентов более удобным и быстрым.

На этапе проектирования интерфейса мы использовали сервис Figma. Это удобный инструмент для создания дизайна, который поддерживает командную работу над проектом. С его помощью был разработан макет удобного интерфейса для пользователей. В этом макете дизайнеры постарались сделать максимально понятными связи и переходы между страницами сайта, чтобы у фронтенд-разработчиков не было проблем с оформлением сайта и переносом в него данных.

Для разработки клиентской части веб-сайта был выбран React.js. Этот фреймворк позволяет создавать кроссплатформенные приложения, что важно для обеспечения работы системы на разных устройствах. Кроме того, React.js упрощает работу в команде, так как каждый разработчик может сосредоточиться на своей части кода, не боясь случайно что-то испортить в других компонентах.

Серверная часть была написана на Python. Python помог нам быстро реализовать основные функции сервера и интегрировать их с другими компонентами системы.

Для хранения данных мы использовали PostgreSQL. Это удобная и гибкая система управления базами данных с открытым исходным кодом. Она поддерживает множество типов данных и хорошо подходит для проектов, где важно быстро разработать прототип и при этом обеспечить стабильную работу.

В итоге, сочетание React.js, Figma, Python и PostgreSQL позволило нам создать прототип сервера для системы управления медицинскими данными, который работает быстро, поддерживает разные платформы и имеет удобный интерфейс. Такой подход помог нам не только ускорить процесс разработки, но и заложить основы для дальнейшего улучшения и масштабирования системы.

2 Распределение обязанностей

2.1 Задачи команды

Первым этапом работы была разработка плана действий для каждого участника. После обсуждения технического задания были предприняты следующие шаги для комфортного сосуществования всех участников команды в работе над проектом:

- Были распределены роли и обязанности между всеми членами команды.
- Были установлены четкие дедлайны выполнения задач для всех этапов работы каждого из участников
- Каждый участник вовремя выполнял свою часть работы, а все проблемы решал путем взаимодействия с командой или руководителем
- После выполнения работы участники подготовились к защите своего проекта, оформив все результаты и проанализировав свой вклад в этот проект
- После защиты каждый из участников проекта должен был написать итоговый индивидуальный отчет о проделанной работе

2.2 Мои задачи

Работа над нашей частью большого проекта разработки клиентской части системы управления медицинскими данными делится на фронтенд разработку и Дизайн. Я в команде занимаю роль одного из фронтенд разработчиков.

У фронтенд-разработчиков было несколько основных задач:

- По макетам сверстать страницы сайта

- Учесть адаптив под мобильные устройства

- Настроить взаимодействие сайта с бэкэндом

В проекте я отвечаю за аккаунт пациента. Соответственно, я была ответственна за следующие страницы сайта:

- Страница аккаунта пациента, на которой отображаются данные о нем, его аватар, история медицинской карты, список врачей и новые заявки от врачей

- Страница списка врачей пациента

- Страница аккаунта врача, на которой отображаются данные о нем и его аватар

- Страница поиска врачей

- Страница новых заявок от врачей с возможностью их принять или отклонить

- Страница медицинской карты пациента

3 Ход моей работы над проектом

3.1 Подготовка к работе

Поскольку я не была знакома со всеми сервисами, языками и технологии, которые мне было необходимо использовать для работы над проектом, первым этапом для меня стало постепенное ознакомление с языками JavaScript, HTML, CSS и библиотекой ReactJS.

Для изучения библиотеки ReactJS я обратилась к форуму на habr[1], а также непосредственно к документации ReactJS[2], а языки я изучала с помощью видео на YouTube[3][4][5]

По мере того как дизайнеры разрабатывали макеты проекта, я начала реализацию компонентов с использованием ReactJS, адаптируя их под требования макета.

3.2 Работа над компонентами

Следующим этапом после ознакомления со всеми языками и необходимой библиотекой стало непосредственно создание компонентов для будущего сайта. Сначала я писала каждый компонент с помощью JavaScript, где указывалось содержимое компонента. Затем я писала для каждого компонента оформление с помощью CSS

Я написала следующие компоненты:

- Кнопка фильтра

Этот компонент позволяет фильтровать список данных по выбранным критериям.

- Меню критериев фильтра

Это список условий, которые должны выполняться у фрагмента из списка данных (в случае, если эти условия выбрал пользователь), чтобы после запуска процесса фильтрации эти компоненты продолжили отображаться.

- Кнопка поиска

Этот компонент позволяет вбить в строку поиска некоторое буквосочетание и найти среди списка данных те фрагменты, которые подходят под критерий поиска.

- Кнопка “переворачивающейся” стрелки

Этот компонент позволяет визуализировать действия пользователя. если он нажимает кнопку, то она визуально поворачивается, при этом происходит изменение на экране: либо пользователя перекидывает на следующую страницу, либо открывается расширенное окно информации (например, дополнительная информация о блоке из медицинской карты)

- Аватар пациента

- Данные пациента

Это компонент, который содержит ФИО, айди и дата рождения пациента

- Кнопка списка врачей пациента

Это кнопка, при нажатии на которую пользователя перебрасывает на страницу списка его врачей

- Кнопка медицинской карты

Это кнопка, при нажатии на которую пользователя перебрасывает на страницу его медицинской карты

- Кнопка новых заявок от врачей

Это кнопка, при нажатии на которую пользователя перебрасывает на страницу новых заявок, поступивших ему от врачей

- Аватар врача

- Данные врача

Этот компонент содержит в себе ФИО, больницу, контактную информацию конкретного врача (опционально этот компонент может содержать причину, по которой врач послал заявку пациенту)

- Список врачей

Компонент, где выводится список врачей пациента, причем каждый фрагмент этого списка также является компонентом

- Медицинская карта

Содержит все данные медицинской карты пациента (заметки врачей, результаты анализов и т.п.)

- Компонент заметки в медицинской карте

3.3 Работа над страницами

После написания компонентов следующим шагом было составление страниц из этих компонентов. Я написала следующие страницы:

- Страница аккаунта пациента (рисунок 1), на которой отображаются данные о нем, его аватар, кнопка медицинской карты, кнопка списка его врачей и кнопка новых заявок от врачей

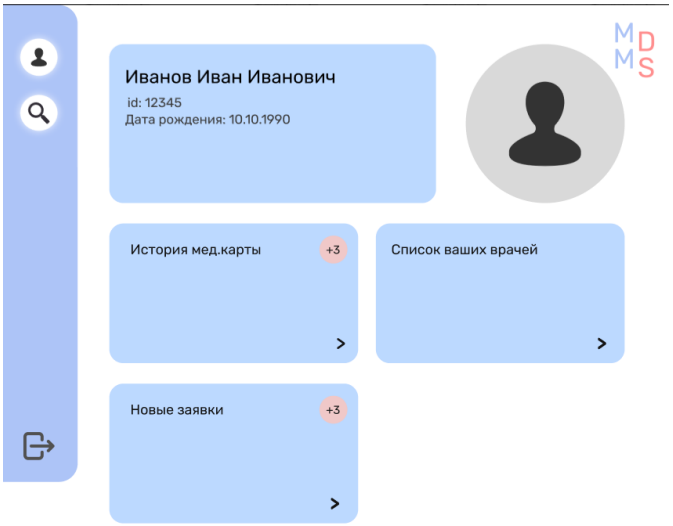


Рисунок 1 - Страница аккаунта пациента

- Страница списка врачей пациента (рисунок 2), на которой отображается меню поиска, а также компоненты врачей, при нажатии на которые пользователь перебрасывается на страницы аккаунтов этих врачей

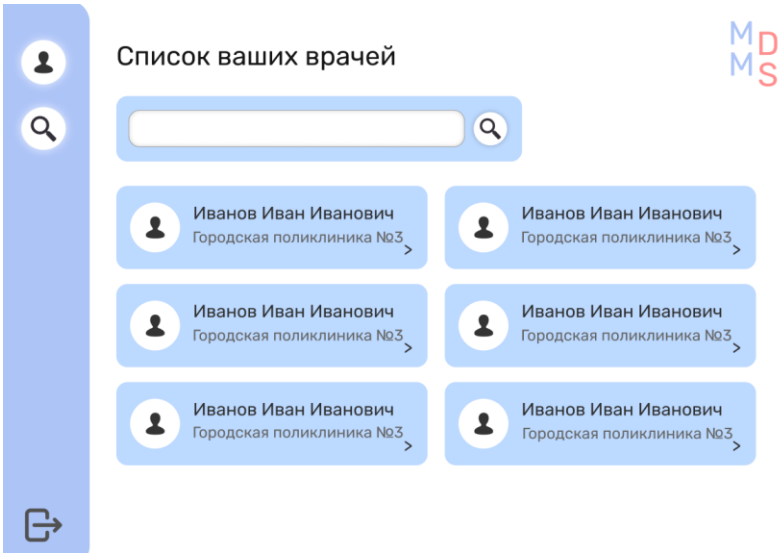


Рисунок 2 - Страница списка врачей пациента

- Страница аккаунта врача (рисунок 3), на которой отображаются данные о нем и его аватар

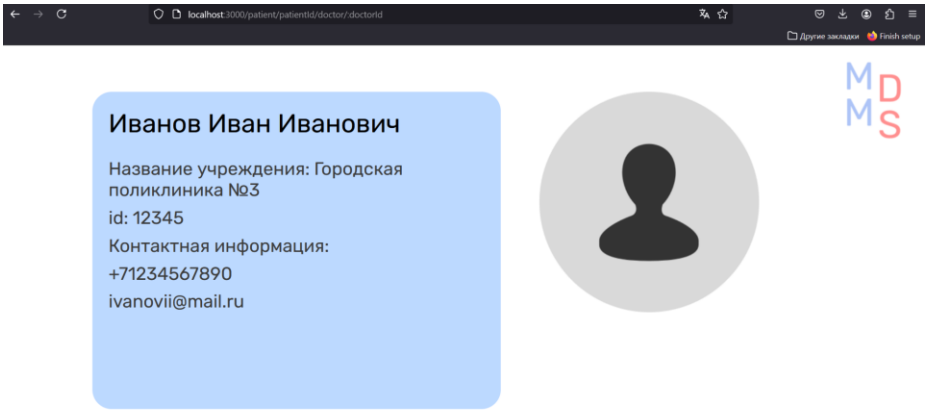


Рисунок 3 - Страница аккаунта врача

- Страница поиска врачей (рисунок 4), на которой отображается меню поиска, а также компоненты врачей, при нажатии на которые пользователь перебрасывается на страницы аккаунтов этих врачей

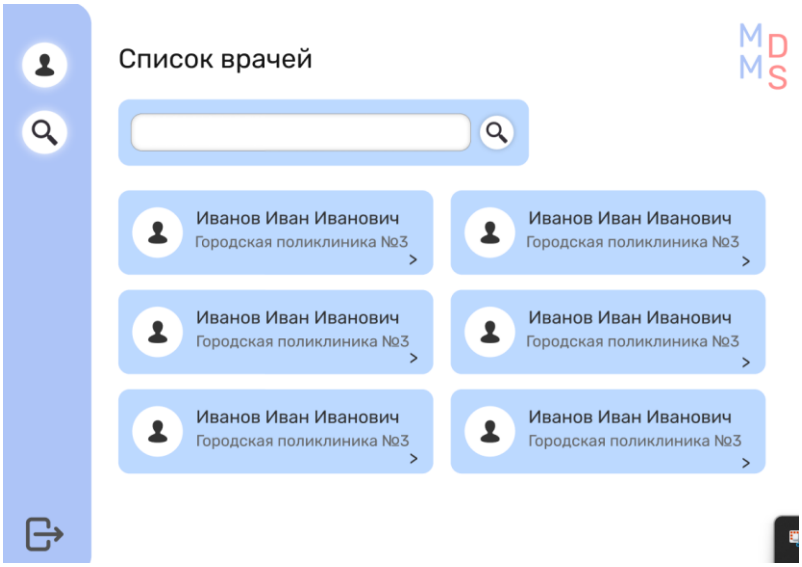


Рисунок 4 - Страница поиска врачей

- Страница новых заявок от врачей (рисунок 5) с возможностью их принять или отклонить. Эта страница также содержит компоненты поиска, фильтра

и меню фильтрации, а также компоненты врачей, при нажатии на которые пользователь перебрасывается на страницы аккаунтов этих врачей

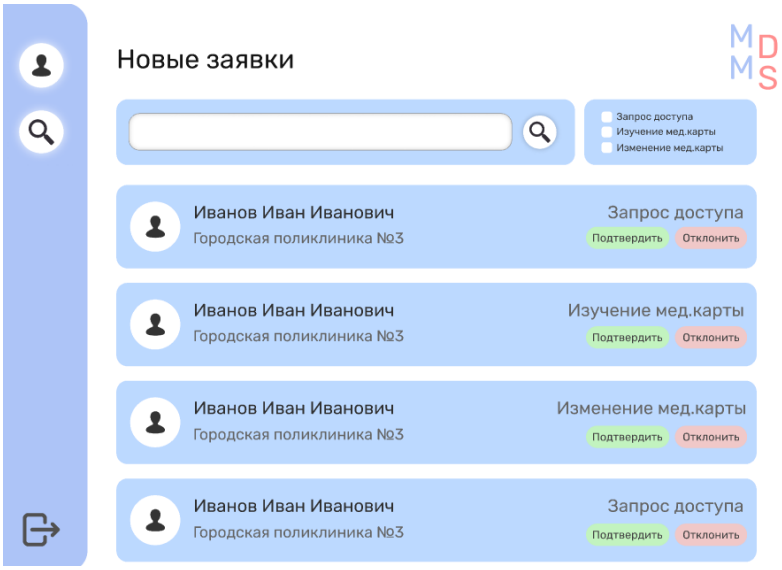


Рисунок 5 - Страница новых заявок от врачей

- Страница медицинской карты пациента (рисунок 6), на которой отображаются заметки врачей, а также есть компоненты фильтра и меню фильтра

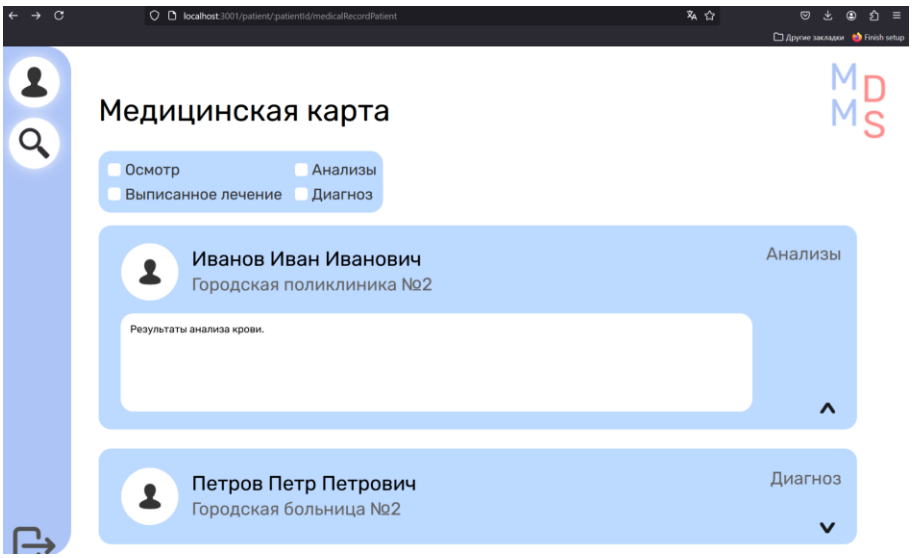


Рисунок 6 - Страница медицинской карты пациента

3.4 Оценка моей работы

Я планомерно и в установленные сроки выполнила ту часть работы, которая касалась создания компонентов и страниц для сайта. Однако по причине болезни я не смогла закончить свою часть работы над проектом, связанную с реализацией адаптива для мобильных устройств и подключения к коду API для входа и регистрации от бэкэнд разработчиков

3.5 Взаимодействие с командой и руководителем

Все участники команды ответственно подходили к поставленным перед ними задачам и почти всегда выполняли их в сроки. Чтобы не терять связь с коллегами, был создан общий чат, где каждый мог задать интересующий его вопрос по любой теме, связанной с проектом.

Мы своей командой достаточно часто проводили конференции совместно с руководителем, чтобы лучше понимать, что от нас требуется. Наш руководитель всегда была готова прийти на помощь и объяснить непонятные моменты или помочь исправить места, в которых возникали ошибки.

ЗАКЛЮЧЕНИЕ

Нам удалось достигнуть цели проекта и разработать клиентскую часть сервиса управления медицинскими данными. В процессе работы были выполнены практически все поставленные задачи, и теперь наш сайт предоставляет весь необходимый функционал.

По итогу был разработан сайт управления медицинскими данными с минималистичным дизайном и удобным интерфейсом, интуитивно понятным каждому пользователю. Сервер понятно и удобно выполняет свои функции, благодаря чему обеспечивается комфортное взаимодействие пользователей между собой.

Мой вклад в проект заключался в активном участии по части фронтенд разработки. В рамках этой задачи я сверстала следующие страницы: страница аккаунта пациента, страница списка врачей пациента, страница аккаунта врача, страница поиска врачей, страница новых заявок от врачей с возможностью их принять или отклонить, страница медицинской карты пациента. Таким образом, мой вклад в проект можно оценить как разработку интерфейса сайта для пользователей, которые являются пациентами.

Во время работы над проектом у меня возникали сложности с пониманием большого количества нового материала и применением его на практике. В эти моменты мне очень помогал наш руководитель - она подробно объясняла мне все нюансы и старалась сделать так, чтобы я могла самостоятельно устранять ошибки.

Таким образом, несмотря на некоторые трудности, общий результат работы команды и достижение всех целей проекта свидетельствуют о его успешной реализации.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. https://habr.com/ru/articles/335244/
2. https://react.dev/learn
3. https://itproger.com/course/html
4. https://itproger.com/course/css
5. https://itproger.com/course/javascript

ПРИЛОЖЕНИЕ

Техническое задание

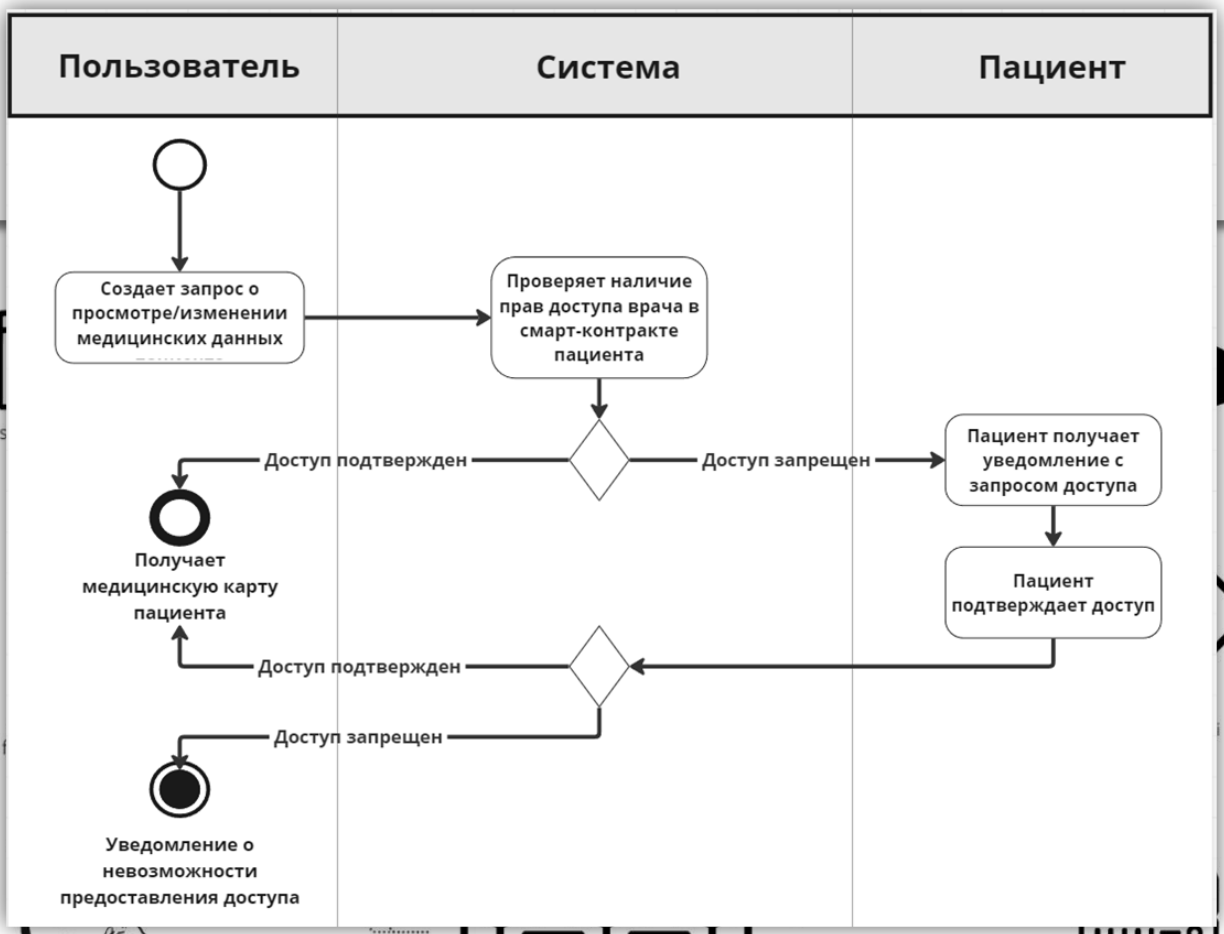
Сервис для управления медицинскими данными на основе блокчейн

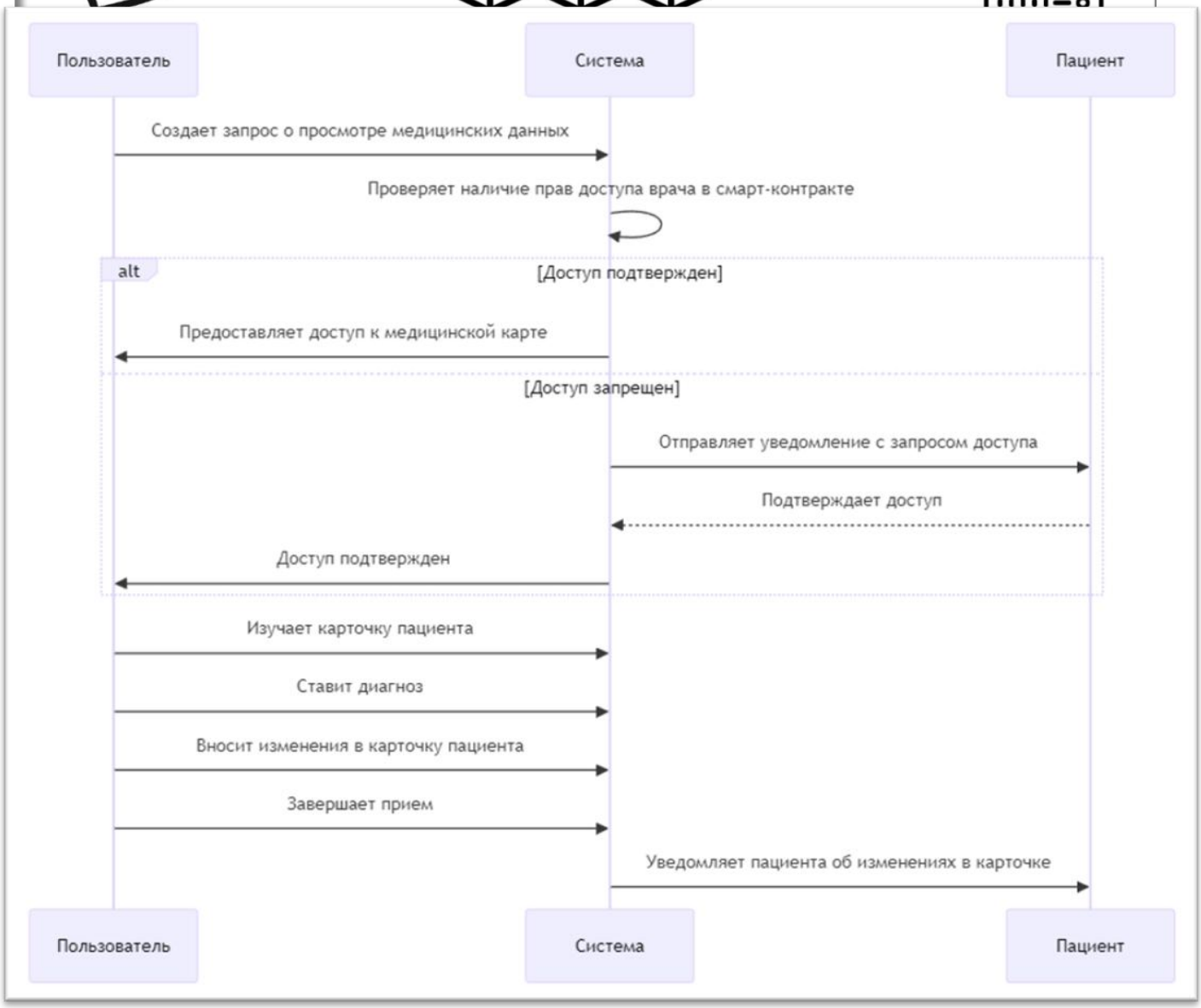
Клиентская часть: Алмазова Л.
Серверная часть: Лаврова А.К.

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.


Пользователь
Система
Пациент
Создает запрос о просмотре/изменении медицинских данных
Проверяет наличие прав доступа врача в смарт-контракте пациента
Доступ подтвержден
Доступ запрещен
Пациент получает уведомление с запросом доступа
Получает медицинскую карту пациента
Пациент подтверждает доступ
Доступ подтвержден
Доступ запрещен
Уведомление о невозможности предоставления доступа


Пользователь
Система
Пациент
Создает запрос о просмотре медицинских данных
Проверяет наличие прав доступа врача в смарт-контракте
alt
[Доступ подтвержден]
Предоставляет доступ к медицинской карте
[Доступ запрещен]
Отправляет уведомление с запросом доступа
Подтверждает доступ
Доступ подтвержден
Изучает карточку пациента
Ставит диагноз
Вносит изменения в карточку пациента
Завершает прием
Уведомляет пациента об изменениях в карточке
Пользователь
Система
Пациент

Требования

План MVP

Реализовать следующие сценарии:

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

Требования к стеку

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

Полезные ссылки

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

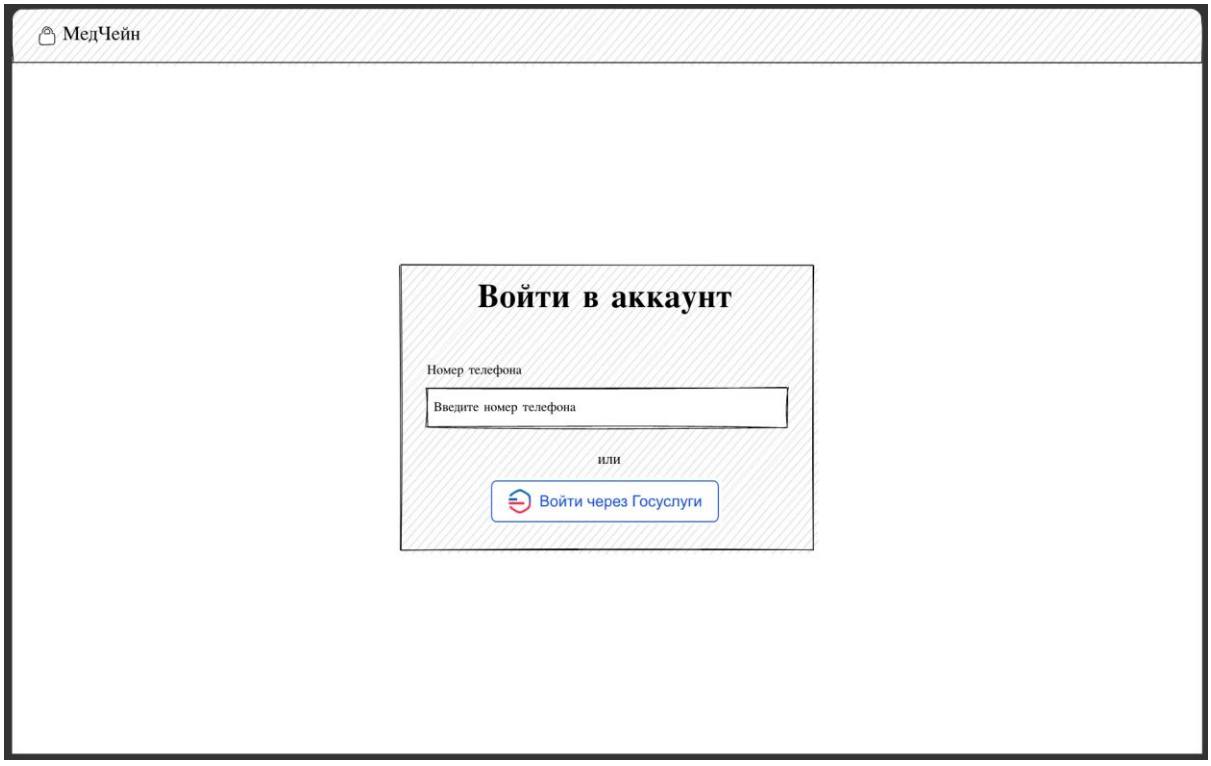
Задачи дизайн

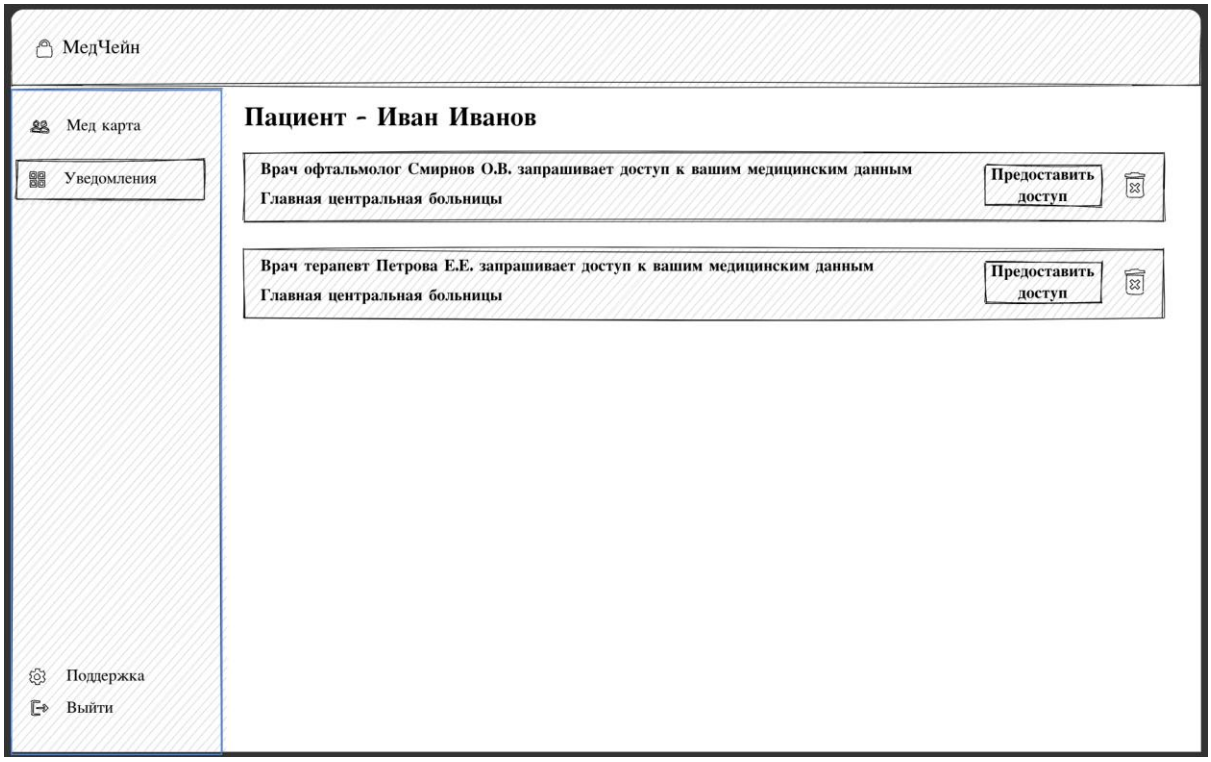
- Проработать цветовую палитру и стилистику
- Создать макеты страниц в figma
- Создать макеты адаптива под мобильные устройства

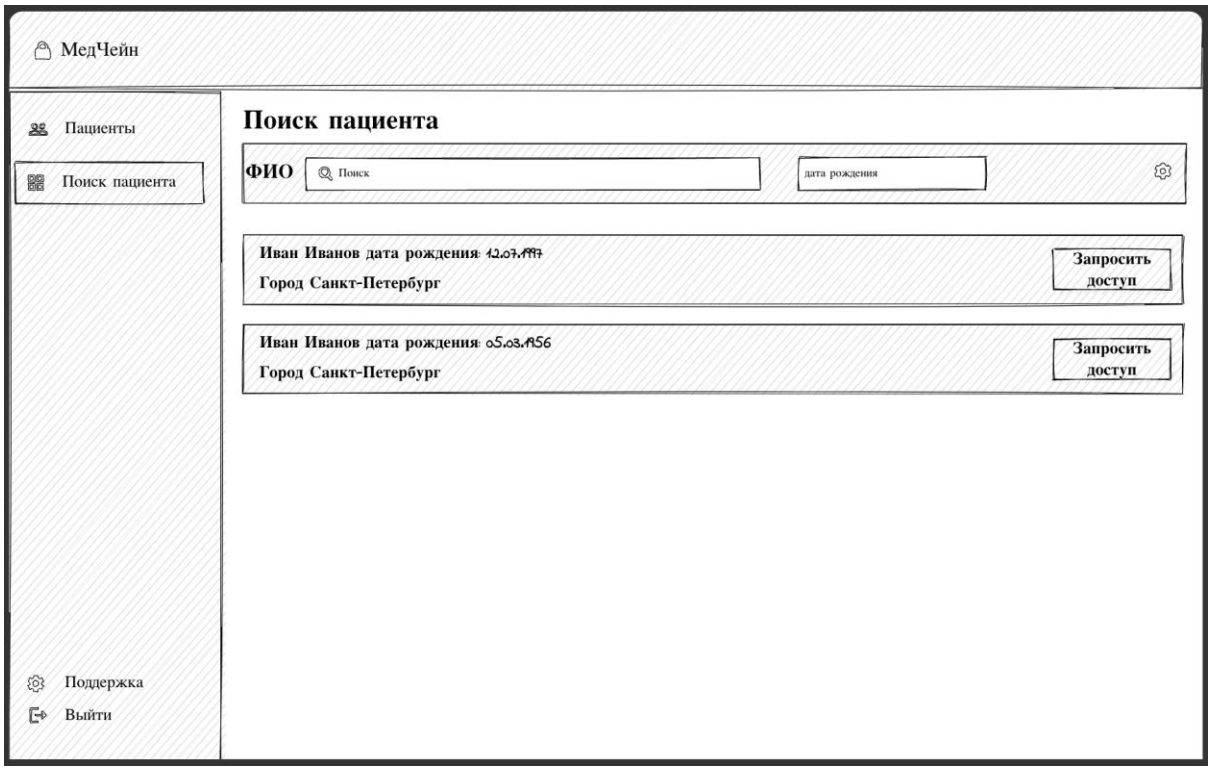
Основные сценарии, которые хотим реализовать, описаны в плане MVP.
Откуда следует, что нам нужны следующие страницы:

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

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




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: 12.07.1997
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 беком

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

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

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

Задачи Back-end

1. Спроектировать архитектуру БД

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

Данные для хранения на бэкенде:

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

Выходные данные:
```
[
  {
    "patient_id": "ID найденного пациента. Тип: integer",
    "name": "Полное имя пациента. Тип: string",
    "date_of_birth": "Дата рождения пациента. Тип: string (ISO
8601)",
    "contract_address": "Адрес смарт-контракта пациента в
блокчейне. 42 символа, начинается с 0x. Тип: string"
  }
]
```

Заглушка:

```
[
  {
```

```
    "patient_id": 1,
    "name": "Иванов Иван Иванович",
    "date_of_birth": "2000-06-01T00:00:00+03:00",
    "contract_address":
"0x0000000000000000000000000000000000000000"
  },
  {
    "patient_id": 101,
    "name": "Иванов Иван Алексеевич",
    "date_of_birth": "2011-08-03T00:00:00+03:00",
    "contract_address":
"0x1100000000000000000000000000000000000011"
  }
]
```

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":
0x0000000000000000000000000000000000000000",
    "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