

**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**

**КУРСОВОЙ ПРОЕКТ**

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

**Обучающийся: Журавлева Анна Андреевна, К3161**

Санкт-Петербург 2024

## СОДЕРЖАНИЕ

СОДЕРЖАНИЕ.....	2
ВВЕДЕНИЕ .....	3
1 Описание проекта .....	5
2 Работа над проектом .....	7
2.1 Этапы работы команды .....	7
2.2 Поставленные передо мной задачи .....	7
2.3 Выполнение поставленных задач.....	8
2.4 Взаимодействие с командой и руководителем проекта.....	13
ЗАКЛЮЧЕНИЕ.....	14
СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ .....	15
ПРИЛОЖЕНИЕ .....	16

## **ВВЕДЕНИЕ**

На данный момент в сфере хранения медицинских данных существует ряд проблем:

- 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;

- 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 внесенных записей, так как нет прозрачного механизма отслеживания изменений;

- 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 создает условия для возможных утечек конфиденциальной информации, о которых пациент зачастую даже не будет знать.

Создание системы управления медицинскими данными на основе блокчейн-технологий позволит преодолеть перечисленные трудности и обеспечит безопасность информации пользователя, поскольку такой подход имеет ряд преимуществ:

- децентрализованное хранение данных, которое снижает риски утечек и взломов;

- фиксация всех внесенных в карту пациента изменений в блокчейне, что делает их удаление или изменение задним числом невозможным. Это обеспечивает прозрачность данных;

- единое хранилище медицинской информации на базе блокчейна, которое упростит процесс взаимодействия между пациентами и врачами, повысит эффективность их работы благодаря возможности получения полных сведений о здоровье пациента в кратчайшие сроки;

- смарт-контракты позволяют пациенту самостоятельно принимать решения относительно того, кому открыть доступ к карте, а у кого отозвать его.

Все вышеперечисленное улучшит качество обслуживания в медицинских учреждениях.

Целью проекта является разработка прототипа клиентской части системы управления медицинскими данными на основе блокчейн технологий.

В ходе работы над проектом необходимо было выполнить следующие задачи:

- создание пользовательского интерфейса для пациентов. Он должен включать отображение всех внесенных в карточку пациента изменений, список врачей, обладающих доступом к карте, для обеспечения прозрачности хранимой информации, а также список запросов на доступ от врачей с возможностью их принятия или отклонения для повышения безопасности данных пользователя;

- разработка интерфейса для врачей. Он должен поддерживать возможности поиска пациентов в базе данных, отправления запросов на доступ к медицинской карте, ознакомления с карточкой и внесения в нее изменений, что упростит работу врачей и позволит им получать актуальную информацию о состоянии пациента.

## 1 Описание проекта

Система управления медицинскими данными является современной платформой, которая объединяет передовые технологии для обеспечения удобства и безопасности работы пользователей. Веб-сайт разработан с учетом потребностей как пациентов, так и врачей, что делает его универсальным инструментом, способствующим упрощению взаимодействия в медицинской сфере.

Особое внимание было уделено созданию удобного интерфейса, что делает использование системы интуитивно понятным даже для лиц, впервые открывших сайт. Доступные функции, такие как управление доступом, возможность внесения/отслеживания изменений, позволяют легко выполнять повседневные задачи.

Для комфортной работы пользователей (как врачей, так и пациентов) с веб-сайтом необходимо было реализовать следующие сценарии взаимодействия с системой:

- регистрация/авторизация для врачей и пациентов;
- отображение запроса врача на доступ к медицинской карте пациента с возможностью его принятия или отклонения;
- возможность просмотра пациентом списка врачей, обладающих доступом к его карте;
- просмотр пациентом внесенных в карту изменений с возможностью их фильтрации;
- функции поиска пациента в базе и отправления заявки на внесение изменений или просмотр данных (для врачей);
- возможность ознакомления врача с карточкой пациента;
- внесение врачом изменений в карточку пациента.

Веб-сайт создавался с использованием современных технологий, что позволило обеспечить его высокую производительность и значительно

упростило работу участников команды. Были использованы следующие инструменты:

- Figma. Данная программа использовалась при разработке дизайна интерфейса. Она позволяет командам работать совместно в реальном времени, а также поддерживает создание прототипов, которые помогают быстро тестировать взаимодействие пользователей с интерфейсом, что делает ее удобным и незаменимым инструментом;

- React.js использовался для верстки интерфейса. Его преимуществами являются высокая производительность, кроссплатформенность, компонентный подход, который упрощает разработку и поддержку кода;

- Python. Данный язык программирования использовался при разработке серверной части веб-сайта. Он известен своей простотой, высокой читаемостью, большим количеством библиотек, поддержкой работы с различными API и возможностью интеграции с блокчейн-технологиями;

- PostgreSQL. Данная СУБД применялась при разработке базы данных. Она популярна благодаря своей надежности и большому функционалу. Система поддерживает сложные запросы и возможность работы с большими объемами данных, обеспечивает высокую степень их согласованности.

Использование данных технологий в проекте обеспечивает создание высококачественного веб-сайта. Интеграция Figma для дизайна, React для фронтенда, Python для бэкенда и PostgreSQL для хранения данных позволило разработать современное, безопасное и удобное решение для пользователей.

## **2 Работа над проектом**

### **2.1 Этапы работы команды**

Работа над проектом включала следующие этапы:

- разработка и корректировка технического задания. На данном этапе происходило обсуждение и утверждение технического задания с командой;
- предварительная подготовка. В тот момент происходило распределение ролей и обязанностей в команде, выбор технологий и инструментов для работы, изучение материалов по теме для качественного выполнения каждым из участников своей части проекта;
- проработка дизайна. В ходе данного этапа происходили создание визуального дизайна интерфейса, обсуждение цветовой палитры, шрифтов, иконографии, разработка интерактивных прототипов для наглядной демонстрации взаимодействия пользователя с системой, а также обсуждение и утверждение макетов с руководителем проекта;
- верстка. На данном этапе разработанные дизайн-макеты превращались в готовый продукт, а также обеспечивалось корректное отображение интерфейса на мобильных устройствах;
- интеграция. Было реализовано взаимодействие между фронтендом и бэкендом, настроены маршруты и обработчики запросов;
- тестирование. На данном этапе осуществлялась проверка всех функций веб-сайта на соответствие требованиям технического задания;
- подготовка к защите проекта и его защита. Была создана презентация, проведена репетиция выступления команды и получена обратная связь по ней от руководителя. Осуществлена сама защита проекта.

### **2.2 Поставленные передо мной задачи**

Я состояла в проекте на роли дизайнера. Передо мной были поставлены следующие задачи:

- создание общего стиля системы, включая цветовую палитру, типографику, иконографику и правила оформления интерфейсов;

- разработка дизайна десктопной версии для страниц личного кабинета пациента, регистрации и авторизации;
- разработка дизайна адаптивных версий (для мобильных устройств и планшетов) с учетом особенностей отображения элементов на меньших экранах.

### **2.3 Выполнение поставленных задач**

Работа над проектом началась с изучения материалов по теме. Необходимо было ознакомиться с функциональностью Figma для создания адаптивного и интерактивного дизайна, а также с инструментами для совместной работы и прототипирования. В этом мне помогли обучающие статьи на сайте самого инструмента [1], а также видеокурс [2]. После изучения информации можно было приступать к разработке дизайна.

Для начала необходимо было определиться с цветовой гаммой, которая бы соответствовала назначению веб-сайта. Для этого пришлось обратиться к видеоуроку по теории цвета в дизайне [3] и статье о психологии цвета [4]. В итоге была создана цветовая палитра, которая учитывает психологические аспекты восприятия цветов. Дизайн разработан в голубых тонах, что способствует снижению уровня стресса у пользователей и создает доверительную атмосферу.

Также нужно было подобрать типографику. Для того, чтобы лучше разобраться в данной теме, был просмотрен видеоурок по типографике в дизайне [5]. В качестве основного шрифта был выбран шрифт “Rubik”, обладающий высокой читабельностью, что обеспечивает четкость и легкость восприятия текста на экране.

Была разработана иконография, включающая в себя набор иконок для основных функций системы (выход, поиск, возвращение в личный кабинет), а также логотип самого веб-сайта. Иконки подбирались исходя из их простоты и понятности, чтобы пользователи могли легко ориентироваться в интерфейсе.



После утверждения общего стиля системы можно было переходить к разработке дизайна десктопной версии. Был тщательно продуман интерфейс личного кабинета пациента. Он включает в себя несколько разделов:

- отображение личных данных пациента, таких как ФИО, персональный идентификатор в системе, дата рождения;
- история медицинской карты, предоставляющая доступ к полной информации о состоянии здоровья пациента. В данном разделе добавлена возможность фильтрации записей по их видам, включая анализы, диагнозы, осмотры, выписанное лечение;
- список заявок врачей на доступ к медицинской карте. Пациенты могут легко принимать или отклонять запросы, а также фильтровать их по целям: изучение карты, изменение карты, запрос доступа;
- список врачей, обладающих доступом к карте. У пациента есть возможность получить подробную информацию о специалисте, включая его ФИО, место работы, контактные данные и специализацию что повышает уровень доверия со стороны пользователей.

Для упрощения взаимодействия пользователя с системой на страницах с заявками и списком врачей была добавлена возможность поиска. Также веб-сайт оповещает пациента о новых заявках на доступ и изменениях в медицинской карте с помощью значков, указывающих на количество обновлений.

Как можно заметить, интерфейс был разработан с учетом минималистичного отображения информации, что делает его легким для восприятия. Четкая структура и логичная навигация способствуют быстрому доступу к нужным данным, что создает положительный опыт использования системы.

Разработанный дизайн представлен на рисунке 1.

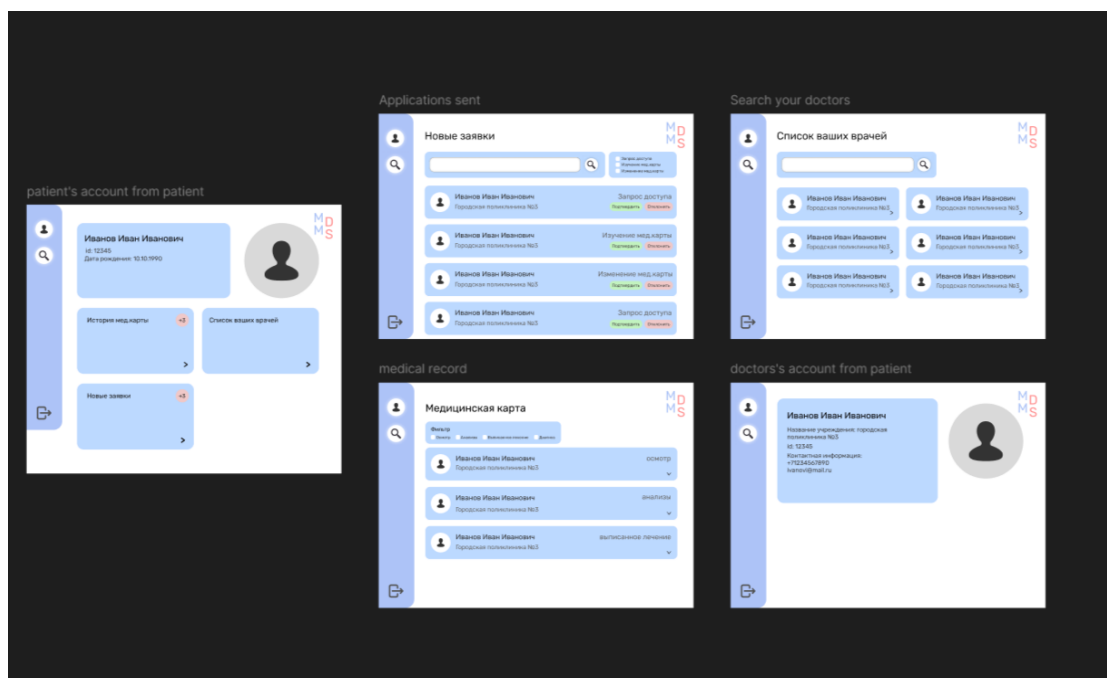


Рисунок 1 – Дизайн десктопной версии личного кабинета пациента

Далее был разработан дизайн страниц авторизации и регистрации. Данные страницы были созданы с акцентом на безопасность и простоту использования. В страницу регистрации включены все необходимые поля для ввода данных, а также окно для подтверждения регистрации. На страницу авторизации добавлены возможность выбора роли пользователя (врач/пациент), а также окно, появляющееся при возникновении проблем со входом в систему, что делает взаимодействие пользователя с системой более комфортным.

На рисунке 2 представлен дизайн страниц регистрации и авторизации.



Рисунок 2 – Дизайн страниц регистрации и авторизации

На следующем этапе необходимо было разработать дизайн адаптива для мобильных устройств. Нужно было адаптировать интерфейс, учитывая разнообразие размеров экранов и специфику взаимодействия пользователей с данным типом устройств. Этап включал в себя переработку элементов интерфейса для их корректного отображения на меньших дисплеях, а также для удобства их нажатия.

Можно выделить следующие аспекты разработки адаптивной версии:

- оптимизация кнопок и ссылок путем увеличения интерактивных элементов для удобства нажатия;
- упрощение навигации благодаря добавлению выпадающих меню для минимизации сложности интерфейса;
- акцент на самых важных функциях и информации, которая должна быть легко доступна пользователю.

На рисунке 3 представлен дизайн личного кабинета пациента для мобильных устройств.

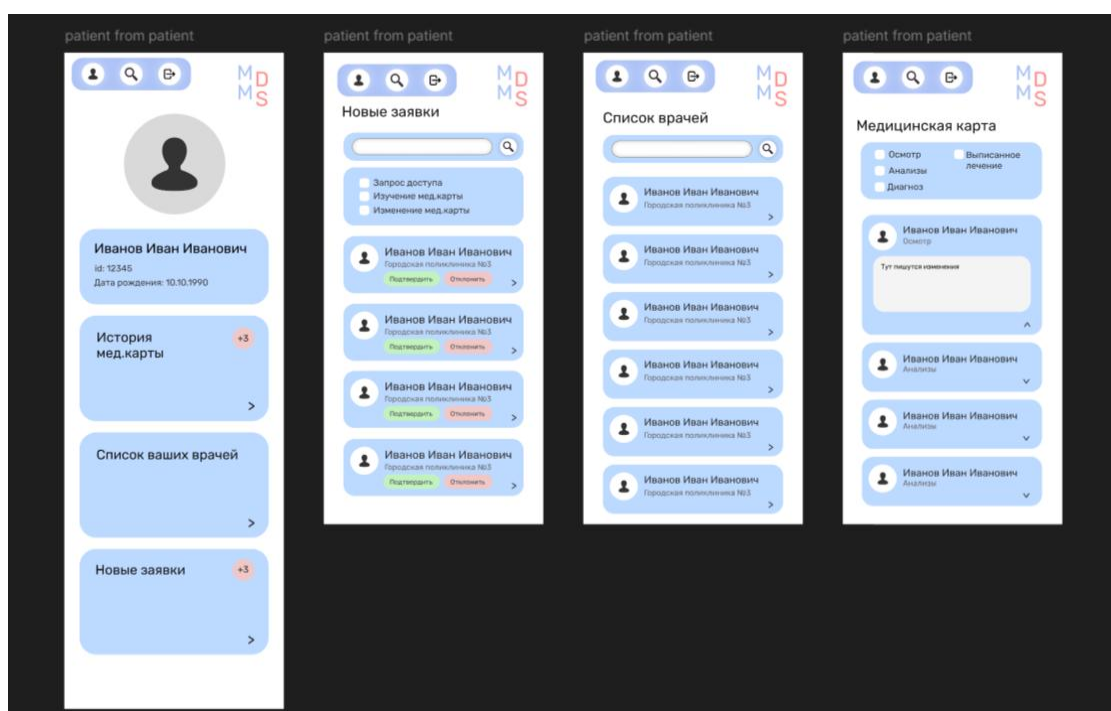


Рисунок 3 – Дизайн личного кабинета пациента для мобильных устройств

На рисунке 4 представлен дизайн страниц регистрации и авторизации для мобильных устройств.

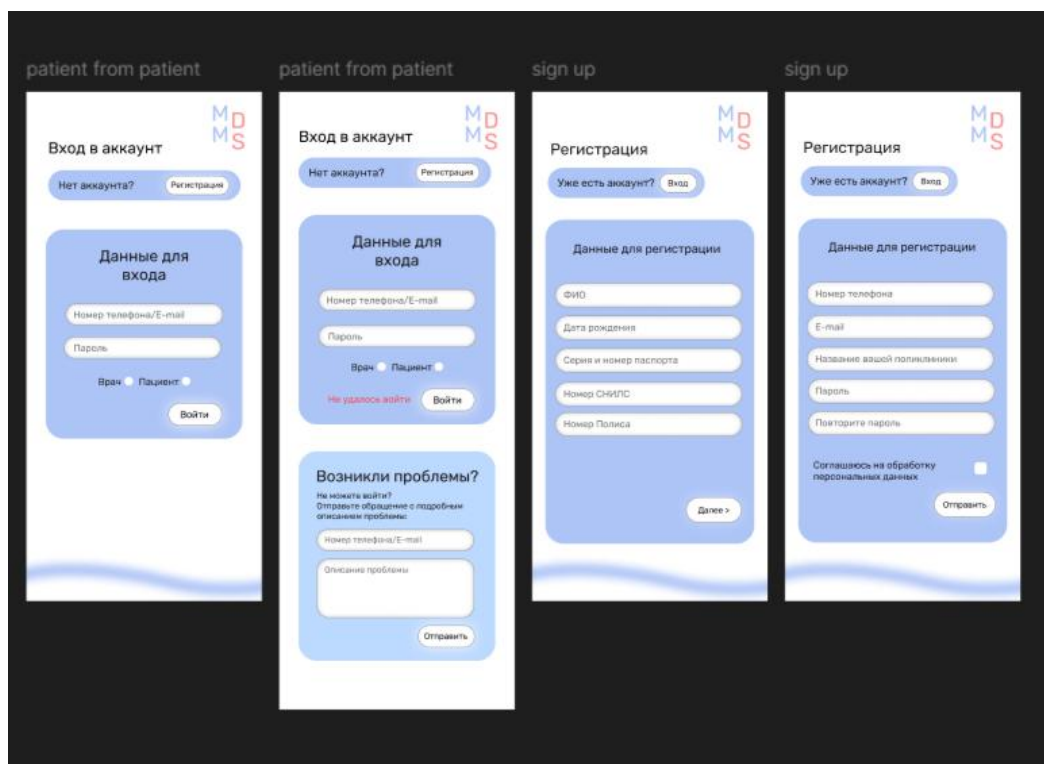


Рисунок 4 – Дизайн страниц регистрации и авторизации для мобильных устройств

В ходе работы над проектом были выполнены все поставленные передо мной задачи, за исключением разработки адаптивной версии для планшетов. В силу относительно сжатых сроков, реализация данной версии не являлась приоритетной задачей проекта. Однако отсутствие специальной версии для такого типа устройств не мешает корректной работе веб-сайта на них благодаря блочной структуре интерфейса.

Основные сложности возникли на начальном этапе разработки дизайна: трудности с подбором цвета и определением общей стилистики интерфейса. Однако консультации с руководителем проекта и изучение обучающих материалов помогли с решением данной проблемы.

На мой взгляд, мне удавалось работать планомерно, так как на решение каждой отдельной задачи руководителем отводилось определенное количество времени, что стимулировало выполнять все в срок.

В ходе создания проекта я научилась грамотно создавать дизайны веб-сайтов, а также работать в команде, поскольку над проектом работало большое количество человек (серверная и клиентская части).

## **2.4 Взаимодействие с командой и руководителем проекта**

Во время выполнения проекта я общалась со вторым дизайнером, с которым согласовывали все изменения, а также с фронтенд разработчиками, отвечала на их вопросы по верстке страниц пациента для обеспечения корректной работы интерфейса и сохранения логики работы с ним.

На протяжении всей работы над проектом руководитель помогал с выполнением поставленных задач, оперативно отвечал на возникающие вопросы и консультировал по грамотному оформлению дизайна в Figma для облегчения дальнейшей работы фронтенд разработчиков. Он всегда поддерживал открытый и конструктивный диалог, что способствовало созданию доверительной атмосферы в команде. Руководитель прислушивался к мнению каждого члена команды и учитывал их при принятии решений, был готов оказать поддержку, что повышало уровень мотивации и вовлеченности участников команды.

В целом, работа руководителя была эффективной, способствовала успешному решению поставленных задач. Я считаю, что он достоин лучшей оценки.

## **ЗАКЛЮЧЕНИЕ**

На мой взгляд, проект можно считать успешным, так как поставленная цель была достигнута. В результате был получен прототип веб-сайта, который

обладает всем необходимым функционалом для обеспечения эффективного взаимодействия между пользователями. Большая часть задач была выполнена в соответствии с установленными требованиями и в рамках обозначенных сроков.

Одной из особенностей проекта стало отсутствие адаптивной версии для планшетов. Однако это не оказывает значительного влияния на работу веб-сайта на данном типе устройств. За счет минималистичного дизайна и блочной структуры система корректно отображается и на них. Реализация адаптивной версии была признана менее приоритетной, учитывая ограниченные сроки на финальных этапах проекта.

Мой вклад в достижение цели заключался в создании общей стилистики интерфейса. Были подобраны цветовая палитра, типографика, а также иконографика. Кроме того, был разработан дизайн десктопной версии страниц личного кабинета пациента, включающий медицинскую карту, список заявок врачей, список врачей пациента, а также страниц авторизации и регистрации. Также была создана адаптивная версия данных страниц для их корректного отображения на различных устройствах.

Несмотря на все трудности, приложенные усилия каждого участника проекта позволили создать прототип клиентской части системы управления медицинскими данными, полностью соответствующий поставленной цели.

## **СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ**

- 1 Обучающие статьи на сайте Figma – <https://www.figma.com/@learn>

- 2 Видеокурс по работе с Figma —  
<https://www.youtube.com/watch?v=BOt3MNB71gI&list=PLjiHFwhbHYlEmPhn68XdG2p2k4X47XR-8>
- 3 Видеоурок по теории цвета в дизайне —  
<https://yandex.ru/video/preview/9038336185450170325>
- 4 Статья от Skillbox о психологии цвета —  
[https://skillbox.ru/media/design/psy\\_of\\_colour/](https://skillbox.ru/media/design/psy_of_colour/)
- 5 Видеоурок о типографике в дизайне —  
<https://rutube.ru/video/9bff0c3b2a05d01263ce675902dbbf38/>

## **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**

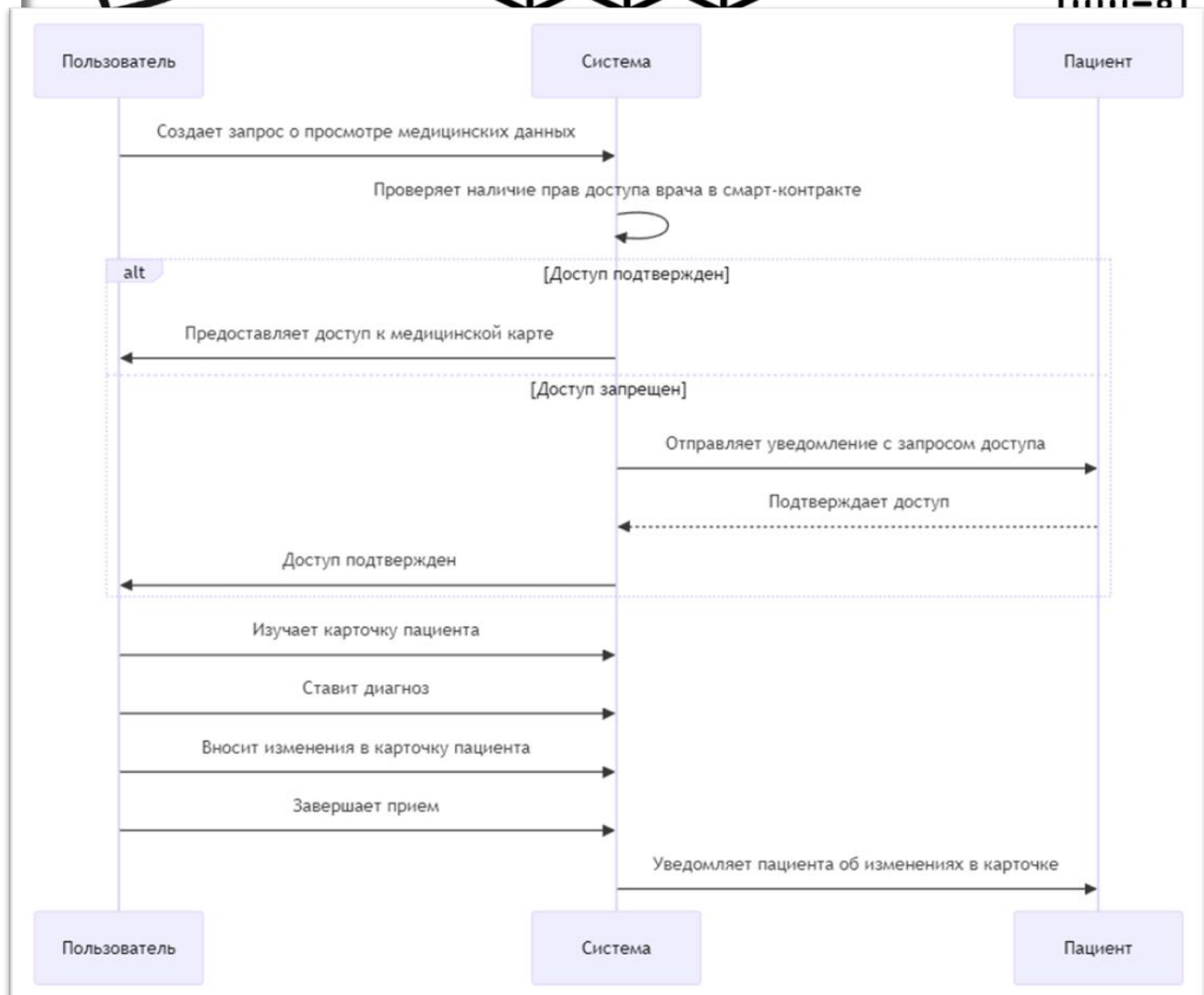
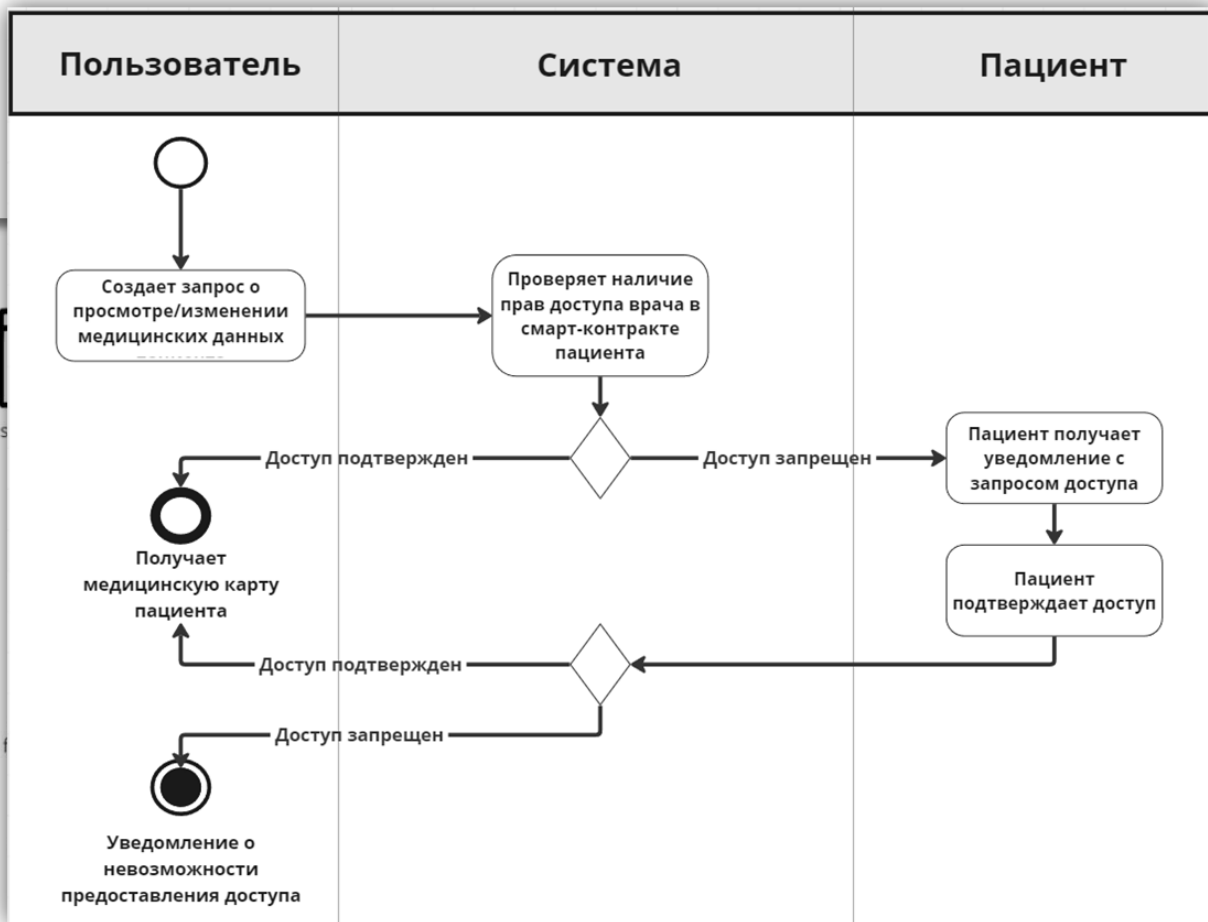
В большинстве медицинских учреждений данные пациентов хранятся в локальных базах данных. Каждый пациент имеет отдельные карточки в разных учреждениях, что создаёт серьёзные сложности при попытке собрать полную медицинскую историю. Эта разрозненность препятствует быстрой и точной диагностике, а также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



## 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 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 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":  
"0x0000000000000000000000000000000000000000000000000000000000000000"  
}
```

## 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":
"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": "0x0000000000000000000000000000000000000000000000000000000000000000"
  },
  {
    "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.