Министерство науки и высшего образования РФ
Федеральное государственное автономное
образовательное учреждение высшего образования
«СИБИРСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»

Институт космических и информационных технологий

Кафедра вычислительной техники

УТВЕРЖДАЮ
Заведующий кафедрой
_____О.В. Непомнящий
"___"_____2023 г.

БАКАЛАВРСКАЯ РАБОТА

090301 Информатика и вычислительная техника

Веб-сайт “Piglet”

Руководитель	_____	_____	доцент, канд. техн. наук	С.Н. Титовский
	подпись	*дата*	*должность, ученая степень*	
Выпускник	_____	_____		А.Р. Саппык
	подпись	*дата*		
Нормоконтролёр	_____	_____		С.Н. Титовский
	подпись	*дата*		

Красноярск 2023

РЕФЕРАТ

Выпускная квалификационная работа на тему «Наименование работы» содержит 49 страницы текстового документа, 3 приложения, 4 использованный источник, презентацию из 15 слайдов.

ИС ДЛЯ СТОМАТОЛОГИИ, MERN, ВЕБ ПРОГРАММИРВОАНИЕ, ИНФОРМАЦИОННАЯ СИСТЕМА, JAVASCRIPT, ВЕБСАЙТ.

Целью выпускной квалификационной работы является разработка вебсайта, позволяющая пациентам стоматологических клиник получать актуальную информацию о своих зубах и о плане лечения для дальнейшего посещения стоматологической клинки.

Задача решенные в ходе выполнения бакалаврской работы:

– Выполнено исследование на основе открытых данных для поиска аналогов, также исследование было дополнено опросом. Учитывая данные из исследования, определили цели и задание разработки;

– На основе целей и задания, выбраны технологии позволяющие разработать вебсайт с указанными возможностями вебсайта из задания, а также позволяющие разработать вебсайт в указанный срок – дата защиты бакалаврской работы;

– Выполнено проектирование с помощью диаграмм использования, диаграмм последовательности и диаграмм прецедентов;

– На основе проектирования, был разработан вебсайт и помещен на виртуальный сервер для доступа к вебсайту.

Общие результаты и выводы: в ходе работы над бакалаврской работой, была затронута проблема, не решенная на данный момент ни одной компанией, на основе требований был разработан вебсайт позволяющий решить описанную в бакалаврской работе проблему.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

На данный момент сервис в стоматологических клиниках выстроен так, что пациент может взаимодействовать с клиникой только очно, что заставляет тратить дополнительное время на дорогу и встречи.

Взаимодействие включает в себя такие процедуры как: лечение, консультация, эстетическая стоматология. Каждая из них комплексная задача, детали которой необходимо донести до пациента.

На данный момент информация о процедурах передается устно обученным человеком – куратором, либо в отсутствие куратора процедура объясняется лечащим врачом, также информацию о процедуре можно передать на бумажном носителе – название, описание и т.д.

Соответственно появляется задача создания сервиса, где информация о процедурах может быть доступна пациенту в любое время в доступной форме. Для доступности информации, можно использовать графическое представление, представление в виде графиков и т.д.

1 Общие сведения

1.1 Исследование актуальности

Для проверки актуальности, было решено провести опрос среди клиник Красноярска.

По результатам опроса, выяснилось, что среди клиник крупного размера, востребованность в сервисе была незначительной, так как обладали ресурсами для найма куратора.

Средние клиники уже были заинтересованы, так как не владели необходимыми ресурсами для найма куратора, но также терапевты обладали некоторым временем для доведения до пациентов информации.

Малые клиники, которые не обладали ресурсами для куратора, также и не обладали достаточным временем врача, так как они отчасти занимаются операционной деятельностью.

Также частным случаем считается государственная клиника, так как они не обладают возможностью найма квалифицированных сотрудников на позицию куратора.

В результате, актуальность работы подтверждается спросом на сервис у государственных, средних и малых клиник, для освобождения сотрудников от непрофильных задач и уменьшения издержек на найм работников.

Так как существует актуальная проблема, то необходимо разработать сервис, целью которой будет улучшение взаимодействия пациента и клиники.

Для достижения цели работы должны быть решены следующие задачи:

– провести анализ предметной области, выделив отличительные черты существующих аналогов, проведя обзор существующих современных средств разработки;

– спроектировать веб-сайт;

– реализовать веб-сайт.

1.2 Постановка задачи разработки

Специфика стоматологических клиник состоит в том, что им не так часто пользуются, это не услуги ежедневого пользования, значит для пациента будет легче посещать вебсайт, не скачивая приложение, которое будет занимать свободное пространство на устройстве пациента.

При разработке вебсайта, необходимо определить роли пользователей, так как речь идет о стоматологической клинике, то там существуют такие лица как пациент, врач, клиника.

Функционал исходя из цели состоит из функций, привязанных к действующим лицам:

– У пациента функции просмотра текущих процедур, просмотра плана лечение, обзора зубов с проблемами;

– У доктора функции по добавлению процедур, удалению и просмотру у конкретного пациента по ID;

– У клиники добавление информации о себе.

1.3 Требования к вебсайту

Определив задачи, определим требования к функциям. Для вебсайта должна быть разработана серверная часть, клиентская часть. Для работы различных групп пользователей, необходимо создать функционал деления пользователей на роли, где необходима возможность регистрации и входа на страницу пользователей. Важно отметить, что пациент лицо не компетентное и не может редактировать информацию о своих процедурах, это говорит о том, что пользователи ограничены в правах.

Так как вебсайт предполагает использование различных клиник, то появляется возможность обращения пациента к различным клиникам, для этого необходимо разработать функционал поиска процедур у разных клиник и возможность сортировать по цене процедур.

1.4 Сравнительный анализ существующих аналогов

В настоящий момент нет подобных сервисов, решающих постановленную задачу. Но существуют подобные веб-сайты:

– Яндекс.Здоровье — это веб-сайт, с помощью которого можно получить онлайн консультацию, без дальнейшего плана и лечения;

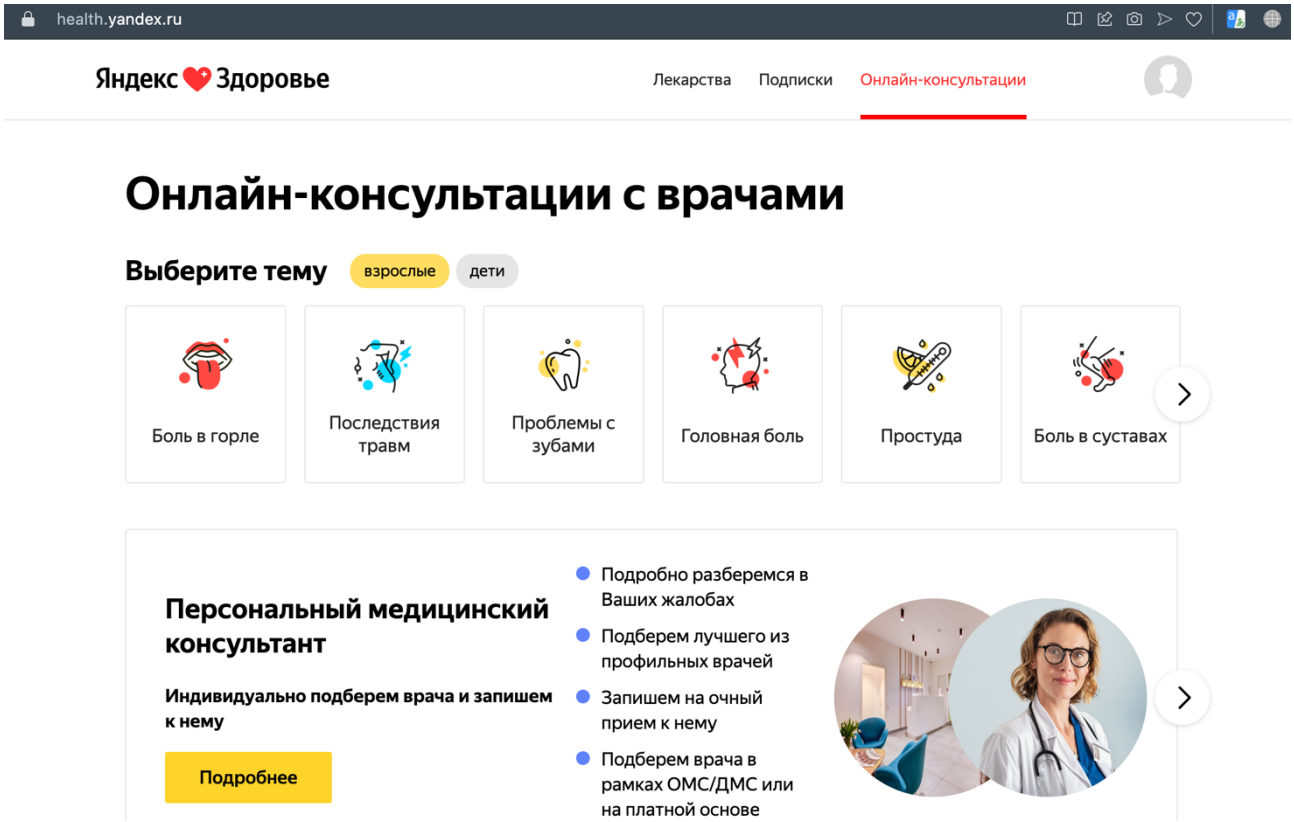

Рисунок 1 – Снимок экрана сайта Яндекс.Здоровье

– ФСНКЦ Личный кабинет — веб-сайт, где можно записаться на прием к врачу, выбрав поликлинику и специалиста, можно получить доступ к истории посещения врачей;

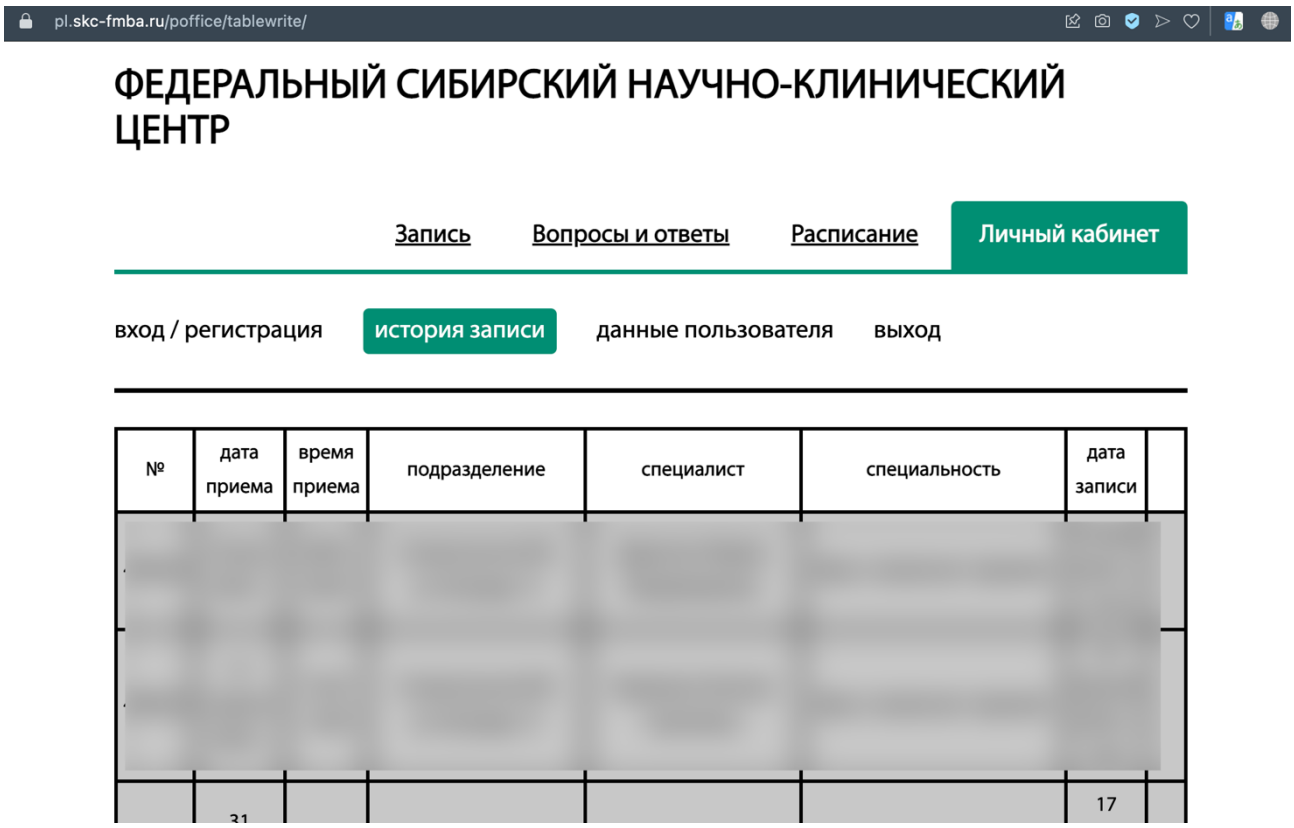

Рисунок 2 - Снимок экрана сайта ФСНКЦ

– Amazon Clinic — это веб-сайт где можно выбрать интересующую медицинскую помощь и получить оценку предлагаемого специалиста после которого можно получить план лечения.

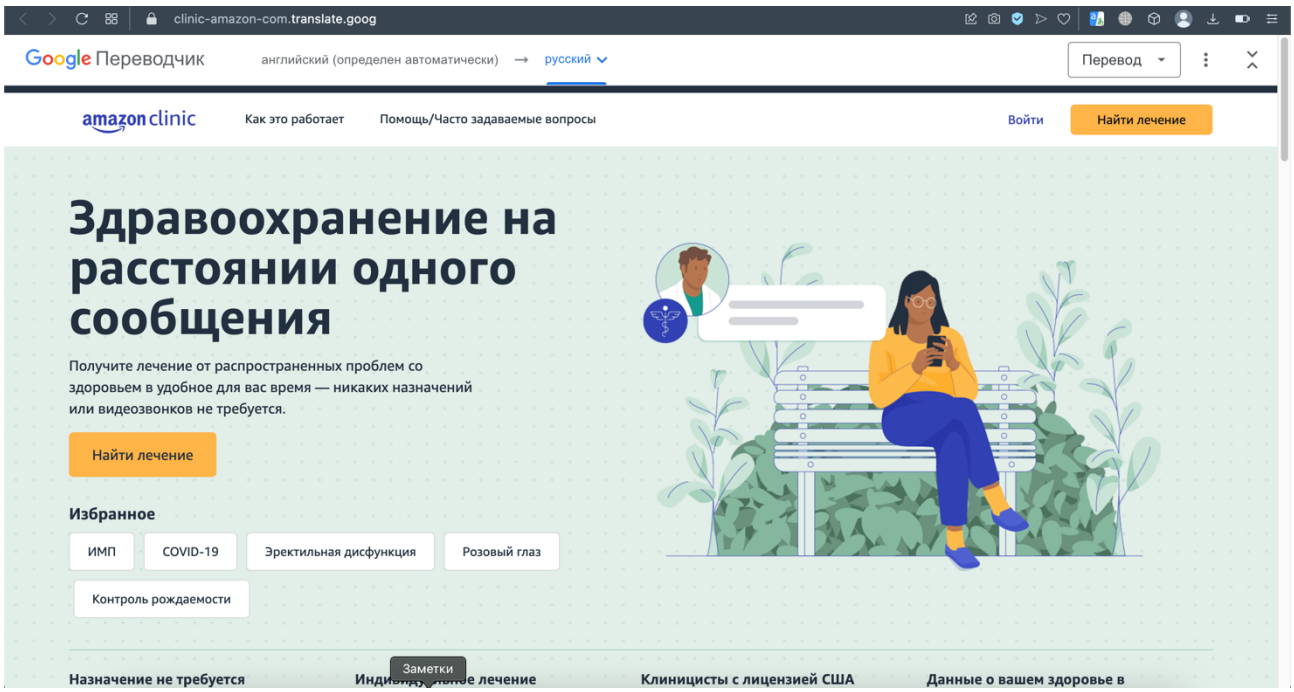

Рисунок 3 - Снимок экрана сайта Amazon Clinic

Данные веб-сайты, которые призваны помочь в лечении, имеют недостатки функционала в виде поиска выгодного предложения, построения роудмапа лечения и отсутствия личной карты пациента, позволяющая хранить информацию о посещениях врача и противопоказаниям, которые в данный момент человек может лишь сам записать или не записать без прямого совета врача.

2 Анализ средств разработки веб-сайтов

2.1 Клиентская часть

Клиентская часть — это все, что пользователь видит и с чем может взаимодействовать при помощи браузера.

Для разработки клиентской части используются базовые технологии, такие как:

– HTML - для создания базовой структуры страниц и контента;

– CSS - для стилизации внешнего вида.

Для расширения функционала, используются скрипты.

2.1.1 JS

С ее помощью можно повысить общую интерактивность сайта. Он позволяет моделировать анимированные компоненты пользовательского интерфейса, такие как: слайдеры, всплывающие окна, расширенные меню навигации по сайту и многое другое.

Веб-страницы, разработанные с помощью JavaScript, реагируют на действия пользователей и обновляются динамически. Благодаря JavaScript этот процесс не требует перезагрузки страниц, чтобы отобразить изменения.

Например, в данной работе можно создавать всплывающие окна, формы ввода с прослушкой, для уменьшения действий пользователем.

2.1.2 TypeScript

Строго типизированный надмножество JavaScript, которое компилируется в обычный JavaScript. Альтернатива для JS в клиентской части, из-за статической типизации улучшает поддержку кода и предотвращает ошибки. Но также из-за статической типизации может растянуть процесс разработки.

2.1.3 Mustache

В процессе разработки появляются повторяющиеся элементы в страницах, например:

– header – где содержатся основные ссылки;

– footer – где содержится дополнительная информация о сайте.

Для решения данной проблемы можно применить шаблонизаторы, позволяющие хранить куски кода в отдельных файлах и подключать их в необходимом месте в конкретной странице. Один из них – mustache.

Mustache.js это движок создания шаблонов JavaScript, который позволяет генерировать HTML-разметку на основе данных. Он разработан таким образом, чтобы быть простым. Например, можно скопировать повторяющийся код в отдельный .js файл и ссылаться на него для отрисовки.

2.1.4 Pug

В отличие от mustache pug также поддерживает условные выражения, циклы и другие функции для более сложной шаблонизации. Однако его синтаксис отличается от привычного HTML, что может потребовать некоторого времени для адаптации.

2.1.5 Итоги клиентской части

Кроме озвученных стандартных технологий HTML, CSS, появляются потребности в добавлении более сложных функций, таких как всплывающие окна, прослушки форм ввода и т.д. Так как разработка проекта ограничена по времени, рационально выбрать более быстрые средства разработки как JavaScript, для уменьшения работы с типами данных и фокусирования работы на разработке наиболее важного функционала.

При разработке вебсайтов, появляется проблема повторяющегося кода, преимущественно это элементы интерфейса которая должна быть на каждой странице, например header и footer. Для решения этой задачи, необходимо повторяющийся код разместить на отдельном файле и подключать в необходимых страницах. На данном этапе, нам не нужны дополнительные функции при работе с шаблонами, поэтому технология mustache подходит для наших целей.

2.2 Серверная часть

Серверная часть - включает в себя работу с приложениями, базами данных и серверами для обработки прикладной логики и функциональности управления данными веб-приложения. Эти технологии взаимодействуют с интерфейсной частью, формируя полный технологический стек.

Серверное приложение обрабатывает бизнес-логику, необходимую для того, чтобы кнопки, формы и другие интерактивные функциональные возможности на интерфейсе действительно работали. Например, когда пользователь вводит свое имя пользователя и пароль для входа в веб-приложение, эта информация отправляется в серверную часть для аутентификации. Затем серверная часть проверит базу данных, содержащую учетные данные пользователя, чтобы убедиться в правильности информации для входа, и отправит ответ с подтверждением интерфейсу.

2.2.1 Node.js

Node.js - среда выполнения, программная инфраструктура, которая выполняет код и взаимодействует с операционной системой. Node.js запускает код JavaScript на стороне сервера, используя компиляцию just-in-time.

Node.js чаще всего используется вместе с Express.js это небольшой фреймворк, который работает поверх Node.js функциональность веб-сервера для упрощения его API-интерфейсов и добавления новых полезных функций.

Это упрощает организацию функциональности приложения с помощью промежуточного программного обеспечения и маршрутизации. Это добавляет полезные утилиты к Node.js HTTP-объекты и облегчает рендеринг динамических HTTP-объектов.

2.2.2 PHP

PHP - считается очень доступным для веб-разработки по многим причинам. Его легко и дешево настроить, что упрощает развертывание запущенных серверных приложений.

Изначально PHP разработан для беспрепятственной работы с наиболее часто используемыми языками баз данных, такими как MySQL.

Главным образом мы перестаем работать с JavaScript, отсутствие опыта в разработке с языком увеличивает время разработки.

2.2.3 Python

Python высокоуровневый язык программирования которую принято считать простой и быстрой для разработки. Но есть существенные минусы, такая как синхронность, что может быть препятствием при обработке большого количества запросов или в ситуациях, требующих масштабируемости.

2.2.4 MongoDB

MongoDB - документально-ориентированная база данных, направленная на простоту работы с базой данных. Простота заключается в том, что хранение и извлечение данных не в виде таблиц, что позволяет быстро менять содержание баз данных, что очень важно для быстроизменяющихся проектов.

Как мы знаем, MongoDB — это сервер баз данных, и данные хранятся в этих базах данных. Или, другими словами, среда MongoDB предоставляет сер-

вер, который можно запустить, а затем создать на нем несколько баз данных с помощью MongoDB. У данного типа БД есть следующие преимущества:

– Благодаря базе данных NoSQL данные хранятся в коллекциях и документах;

– База данных MongoDB содержит коллекции точно так же, как база данных MYSQL содержит таблицы;

– Данные, хранящиеся в MongoDB, представлены в формате документов BSON. Здесь BSON расшифровывается как двоичное представление документов в формате JSON. Или, другими словами, в серверной части сервер MongoDB преобразует данные JSON в двоичную форму, известную как BSON, и этот BSON хранится и запрашивается более эффективно.

Также необходимо выделить, что MongoDB работает вместе с mongoose. Mongoose - средство отображения объектных документов. Его также называют инструментом моделирования объектов. Он построен поверх драйвера MongoDB для MongoDB и Node.js. Это помогает разработчикам моделировать свои данные, определять схему для документов внутри коллекции и управлять связями между данными.

2.2.5 MySQL

MySQL — это реляционная база данных с открытым исходным кодом. Она известна своей хорошей производительностью. MySQL обладает обширным сообществом и широкой поддержкой, а также предлагает множество инструментов и функций для управления данными и выполнения запросов. Отличия от NoSQL, состоят в том, что это реляционная модель данных: MySQL основан на реляционной модели данных, где данные организованы в виде таблиц, состоящих из строк и столбцов. Высокая производительность - MySQL предлагает эффективные алгоритмы обработки запросов, кэширование данных, индексы и другие оптимизации, которые помогают обеспечить быстрое выполнение запросов и обработку большого объема данных.

Так как база данных MySQL высокопроизводителен за счет жесткой связки таблиц, это становится и минусом, так как следует приложить больше усилий в сравнении с NoSQL для изменения структуры базы данных, например, при добавлении новых функций.

2.2.6 Итоги серверной части

В результате сравнения основных технологий для сборки серверной части вебсайта, пришли к тому, что Node.js позволит нам в кратчайшие сроки разработать вебсайт, за счет масштабируемости, быстрого старта работы, так как раннее изучали данную технологию. Так как вебсайт предполагает хранение данных пользователей, то необходимо выбрать базы данных, для которого был выбран MongoDB, за счет скорости развертывания и легкости к изменениям данных.

2.3 Создание сервера

Кроме разработки серверной и клиентской части сайта, также необходимо развернуть его сервере, чтобы любой пользователь интернете смог получить доступ к сайту. Для развертывания сервера рассмотрим технологии.

2.3.1 Nginx

Nginx - это веб-сервер с открытым исходным кодом, который с момента своего первоначального успеха в качестве веб-сервера теперь также используется в качестве обратного прокси, HTTP-кэша и средства балансировки нагрузки.

Nginx создан таким образом, чтобы обеспечить низкое использование памяти и высокий уровень параллелизма. Вместо того чтобы создавать новые процессы для каждого веб-запроса, Nginx использует асинхронный, управляемый событиями подход, при котором запросы обрабатываются в одном потоке.

С помощью Nginx один главный процесс может управлять несколькими рабочими процессами. Мастер поддерживает рабочие процессы, в то время как рабочие выполняют фактическую обработку. Поскольку Nginx является асинхронным, каждый запрос может выполняться работником одновременно, не блокируя другие запросы.

2.3.2 Apache

HTTP-сервер Apache был разработан в первую очередь статические веб-страницы. Он использует модули для расширения своих основных функций. Большинство функций, необходимых для базового статического сайта, включены в установку по умолчанию. Для поддержки динамических веб-сайтов могут быть добавлены дополнительные модули. Вот отличительные черты Apache:

– Представление. Он поддерживает современные функции повышения производительности, включая прокси-серверы, кэширование и балансировку нагрузки. При правильной настройке кластер серверов Apache может обрабатывать десятки тысяч подключений. Nginx считается более производительным в сценариях с чрезвычайно высоким спросом, хотя последние обновления Apache сократили этот разрыв в производительности. Регулирование трафика и отказоустойчивость могут быть восстановлены после ухудшения качества обслуживания;

– Открытый и выдвижной. HTTP-сервер Apache является кроссплатформенным, со сборками для основных операционных систем, таких как Linux и Windows. Он бесплатный и с открытым исходным кодом, поэтому его можно легко включить в качестве основы или зависимости для других проектов. Он также может быть легко дополнен другими модулями в соответствии с любыми потребностями. Одним из популярных примеров является LAMP - Linux, Apache, MySQL, PHP - объединяющий свободное программное обеспечение

для размещения современного динамического веб-сайта в виде единого веб-стека.

2.3.3 Итоги обзора сервера

В результате сравнения технологий для создания сервера, был выбран Nginx, вот ключевые моменты:

– Высокая производительность: Nginx известен своей высокой производительностью и эффективностью обработки запросов. Он имеет неблокирующую архитектуру, которая позволяет обрабатывать большое количество одновременных подключений с минимальным потреблением ресурсов системы. Это делает его идеальным выбором для высоконагруженных веб-приложений и сайтов;

– Эффективное использование ресурсов: Nginx оптимизирован для работы с ограниченным количеством ресурсов. Он потребляет меньше памяти и процессорного времени по сравнению с Apache, что особенно важно при работе с большим количеством одновременных подключений;

– Легковесность и масштабируемость: Nginx является легковесным веб-сервером, который быстро масштабируется и адаптируется к изменяющимся потребностям приложений. Он может быть использован как в качестве самостоятельного веб-сервера, так и в сочетании с другими серверами приложений, такими как Node.js, для обеспечения более гибкой и эффективной инфраструктуры;

– Высокая стабильность и надежность: Nginx славится своей стабильностью и надежностью. Он имеет встроенные механизмы обнаружения и автоматического восстановления от сбоев, что обеспечивает более непрерывную работу приложений;

– Стек технологий, как мы узнали, Apache больше подходит на проектов со стеком LAMP, использование PHP исходя из разработки клиентской и сер-

верной части не предоставляется возможным, поэтому Nginx для нас лучшее решение.

3 Проектирование

3.1 Подготовка

После оформления общих требований к вебсайту, анализа уже существующих аналогов и выбора технологий, осуществляется процесс проектирования.

Проектирование веб-сайта в отношении разработки с использованием различных диаграмм разработки — это процесс создания технической и функциональной архитектуры веб-сайта, с помощью различных диаграмм, которые помогают разработчикам лучше понять взаимодействие между компонентами системы.

Некоторые из наиболее распространенных диаграмм разработки, используемых при проектировании веб-сайтов, включают:

– Диаграммы прецедентов;

– Диаграммы пригодности;

– Диаграммы последовательности.

Использование диаграмм разработки помогает разработчикам лучше понимать взаимодействие между компонентами системы, предоставляя им возможность разрабатывать более эффективные и надежные решения веб-сайтов.

3.2 Диаграмма прецедентов

Диаграмма Use Case (диаграмма прецедентов) является графическим представлением функциональности системы, которая определяет, какие действия могут выполнять актеры (users) в сотрудничестве с системой. Это важный инструмент в разработке программного обеспечения, который позволяет определить требования к системе и ее функциональные возможности.

Мы будем использовать эту диаграмму для сбора требований, проектирование и тестирование. Она также послужит нам основой для создания дальней-

ших моделей системы и способствовать более эффективной коммуникации между членами команды разработчиков и заказчиком.

Для создания этой диаграммы необходимо описать сценарии, у нас получается так:

Любой пользователь может:

– Регистрироваться по различным ролям;

– Входить в систему;

– Искать услуги.

Пациент может:

– Просматривать историю процедур;

– Обозревать зубы;

– Обозревать план лечения.

Врач может:

– Просматривать историю процедур пациента;

– Добавлять процедуры;

– Удалять процедуры.

Клиника может:

– Добавлять информацию о себе.

Диаграммы должны содержать функции значимые для сценариев, менее значимые как открытие всплывающего окна не должны отражаться в нем.

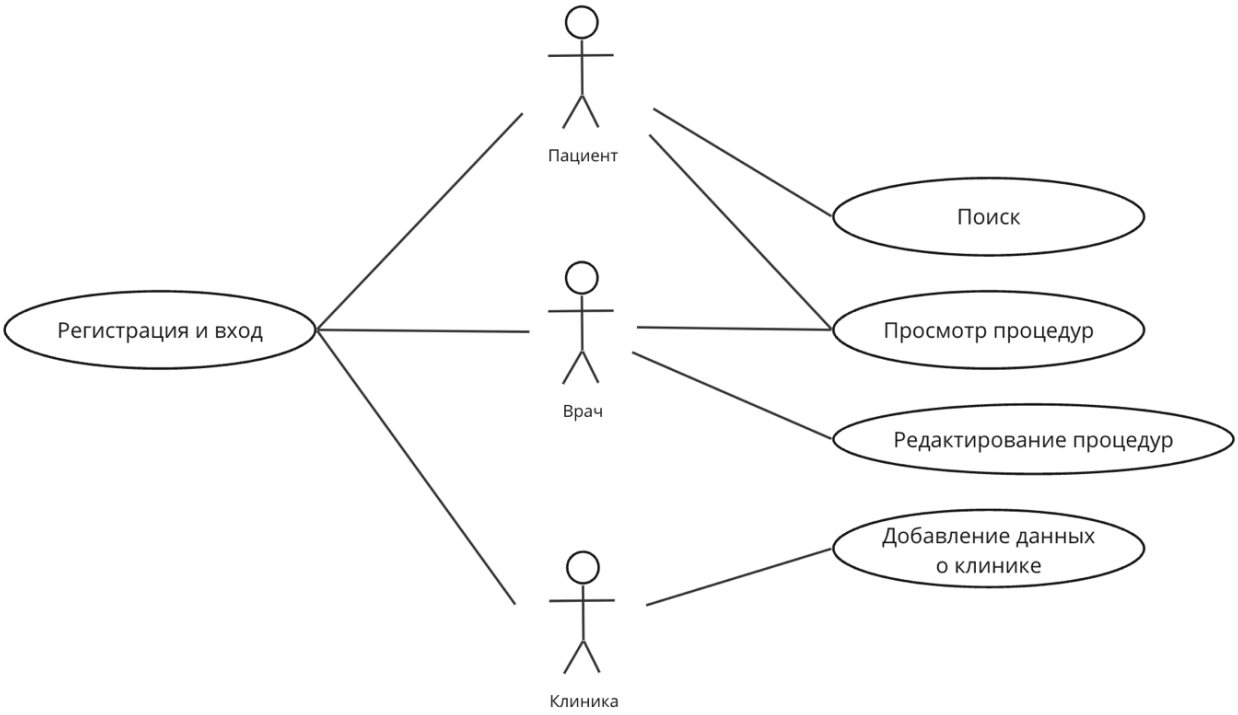

Рисунок 4 – Диаграмма прецедентов

3.3 Диаграмма пригодности

Диаграмма пригодности (Fitness Landscape Diagram) – это графическое представление пространства поиска решений для определенной задачи оптимизации. Эта диаграмма отображает зависимость функции цели от значений параметров, используемых для решения задачи.

Для создания диаграммы пригодности, функция цели, которая определяет качество решения, представляется в виде поверхности или ландшафта с разными уровнями высоты. Различные значения параметров решения могут быть представлены на графике как точки. Цветовая шкала обычно используется для отображения уровня пригодности решения, где более темный цвет соответствует более высокой пригодности.

Диаграмма пригодности может быть полезна для определения наилучшего решения задачи оптимизации и для исследования свойств пространства поиска решений. Она может помочь оптимизаторам понять, как различные пара-

метры влияют на пригодность решений, и какие комбинации параметров дают наилучший результат. Кроме того, диаграмма пригодности может использоваться для определения возможных локальных оптимумов и для выявления сложностей в поиске оптимального решения.

Для описанных в предыдущем разделе прецедентов, создадим диаграммы пригодности. Так как все пользователи получают возможность регистрации, укажем актора в диаграмме как пользователя (user), в результате для прецедента регистрации оно примет следующий вид:

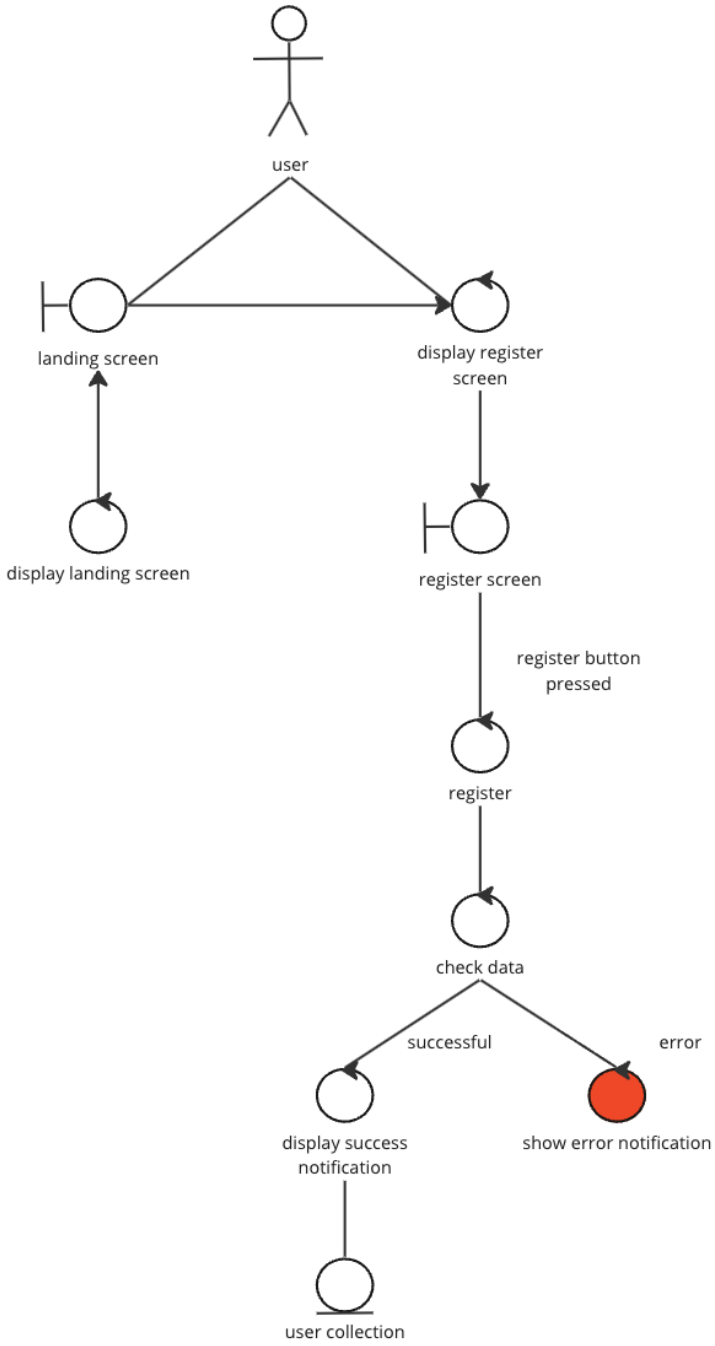

Рисунок 5 – Диаграмма пригодности регистрации для всех ролей пользователей

Рассмотрим диаграмму пригодности для входа в систему, она также примет обезличенный вид, так как функция не отличается от роли пользователя. Следовательно, диаграмма примет вид как на рисунке ниже:

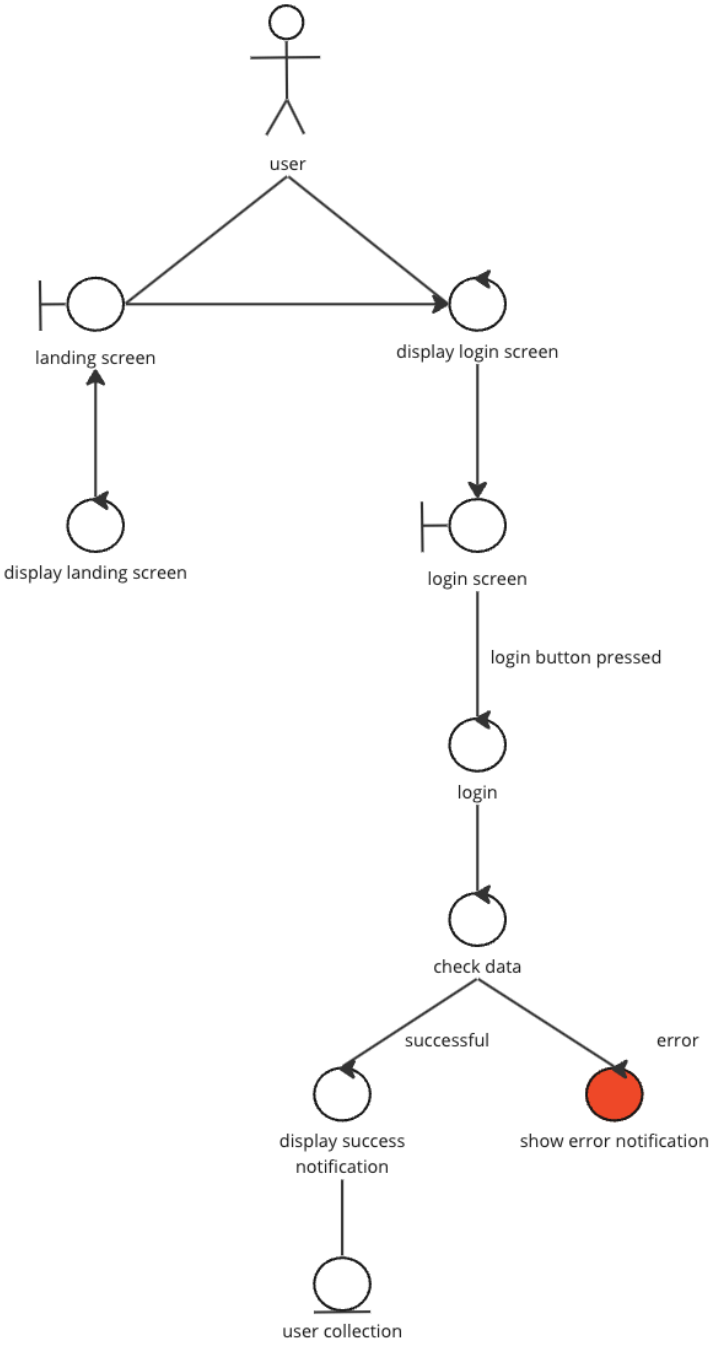

Рисунок 6 – Диаграмма пригодности входа всех пользователей

Рассмотрим диаграмму пригодности для функции поиска, следуя тому, что к функции поиска подпущены все пользователи, также укажем актора как user:

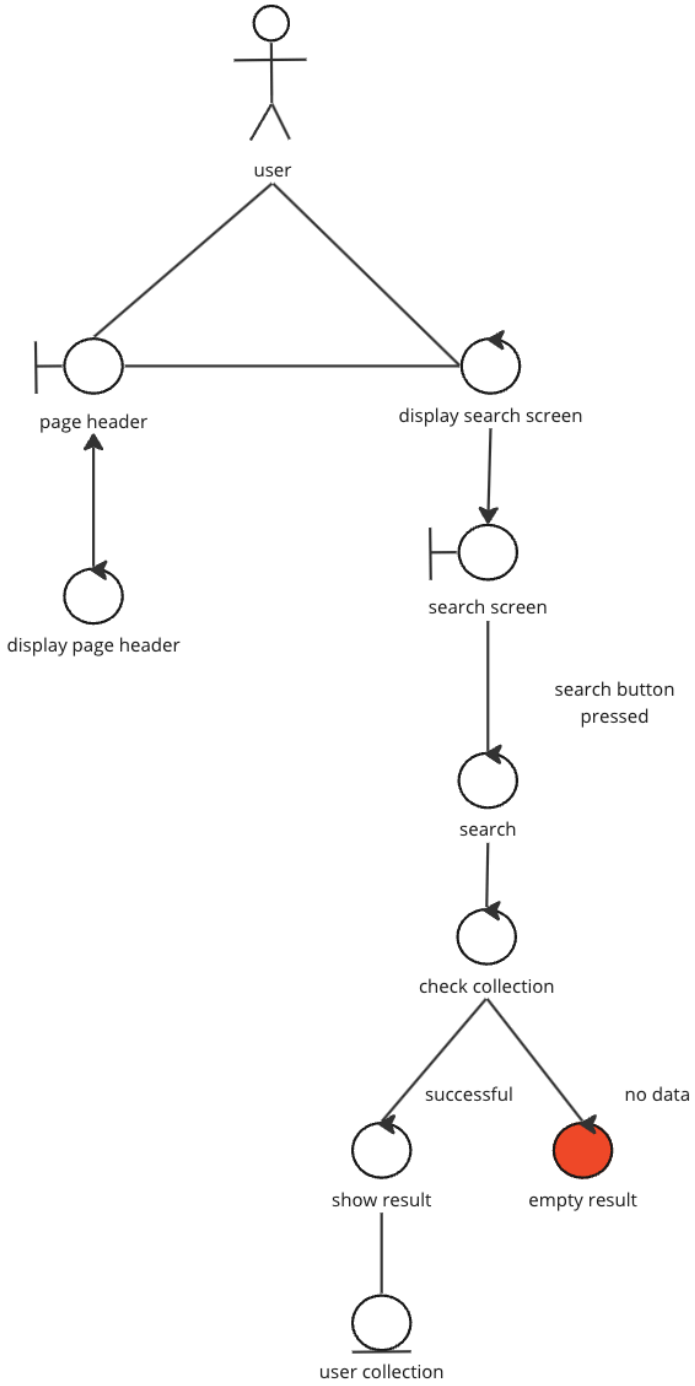

Рисунок 7 – Диаграмма пригодности поиска

Теперь рассмотрим диаграмму пригодности для функции просмотра процедур у пациента пациентом:

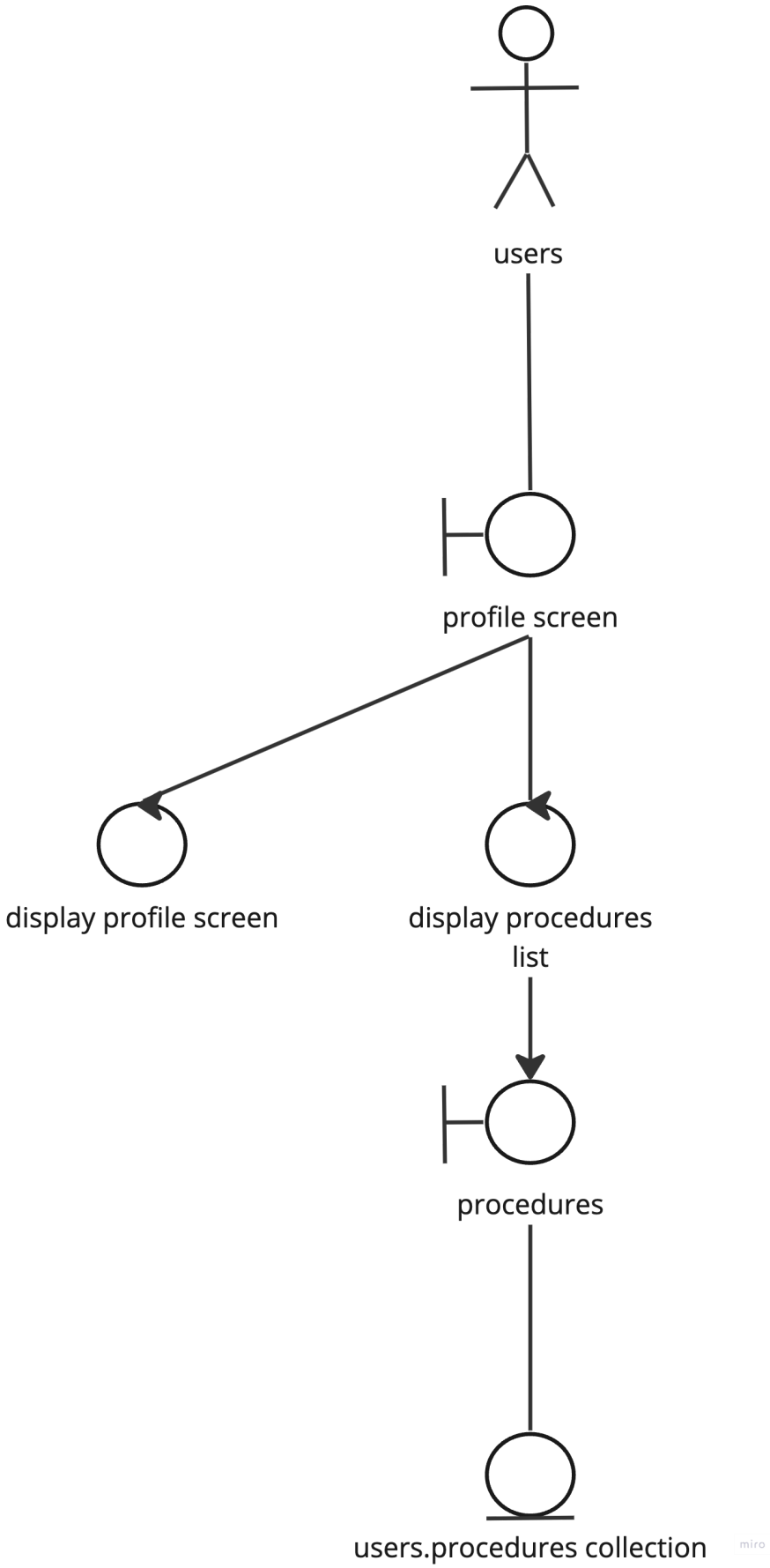

Рисунок 8 – Диаграмма пригодности просмотра процедур у пациента пациентом

Далее рассмотрим диаграмму просмотра процедур пациента врачом, он примет следующий вид:

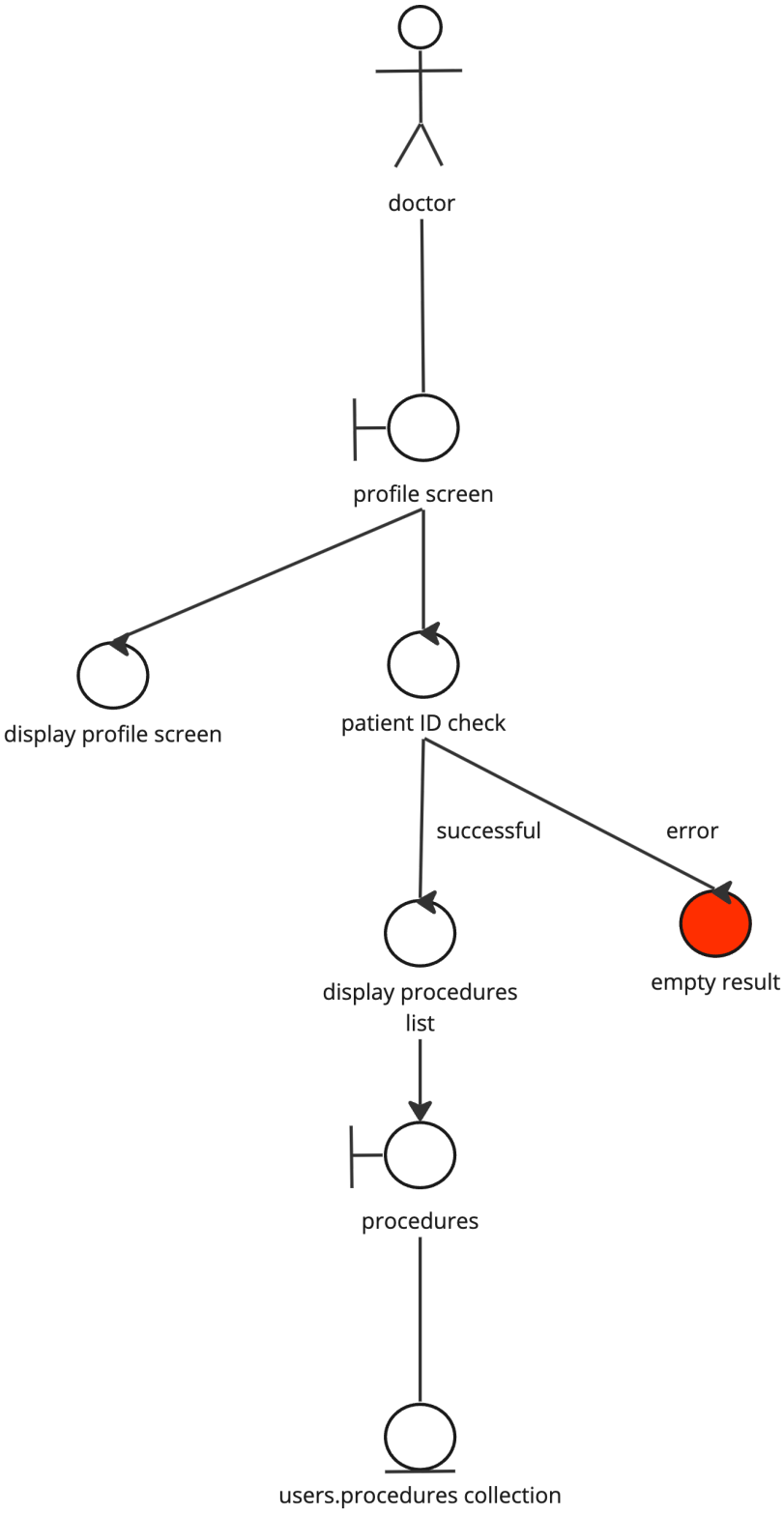

Рисунок 9 – Диаграмма пригодности просмотра процедур у врача

Рассмотрим теперь прецедент удаления процедур и создадим для него диаграмму пригодности:

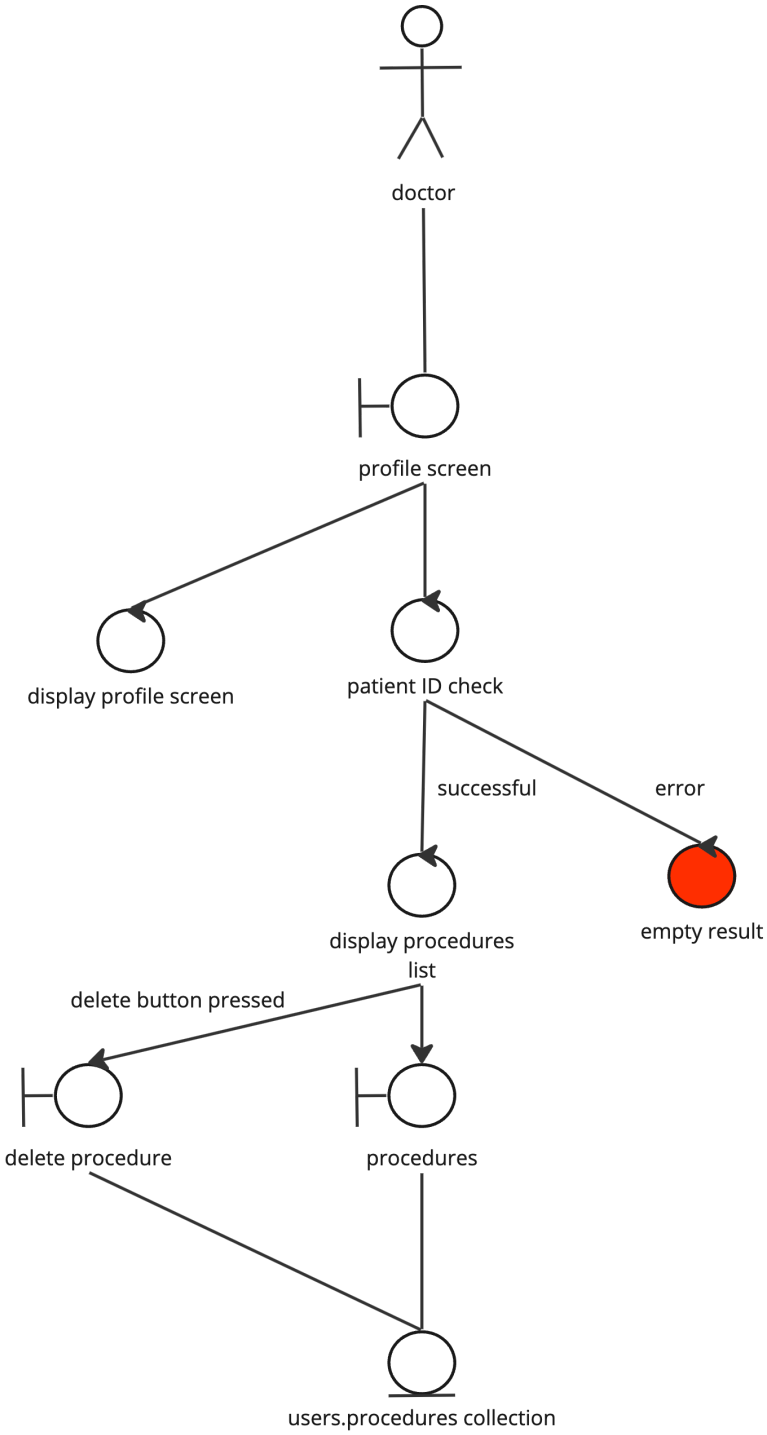

Рисунок 10 – Диаграмма пригодности для функции удаления процедур

Рассмотрим прецедент добавления данных клиники, диаграмма пригодности будет выглядеть следующим образом:

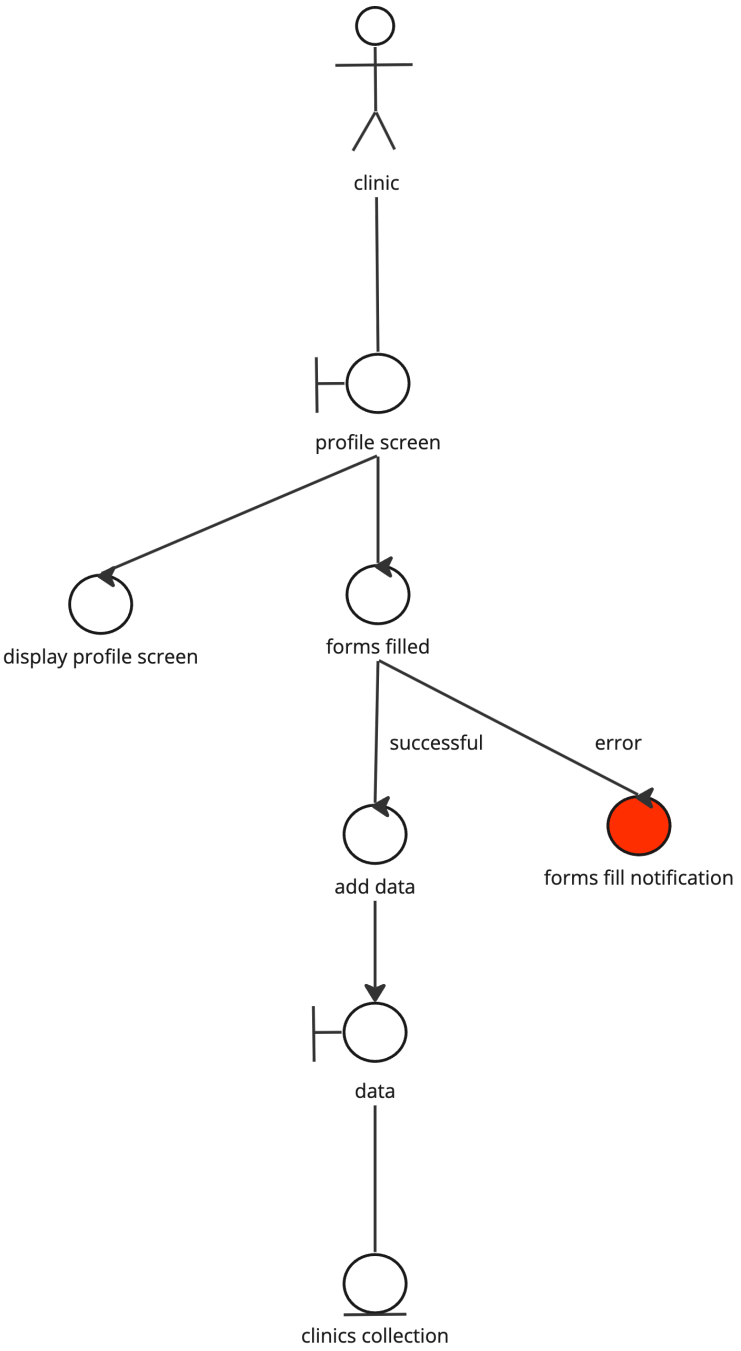

Рисунок 11 – Диаграмма пригодности добавления данных клиники

3.4 Диаграммы последовательности

Рассмотрим диаграмму последовательности для входа, исходя из диаграммы пригодности:

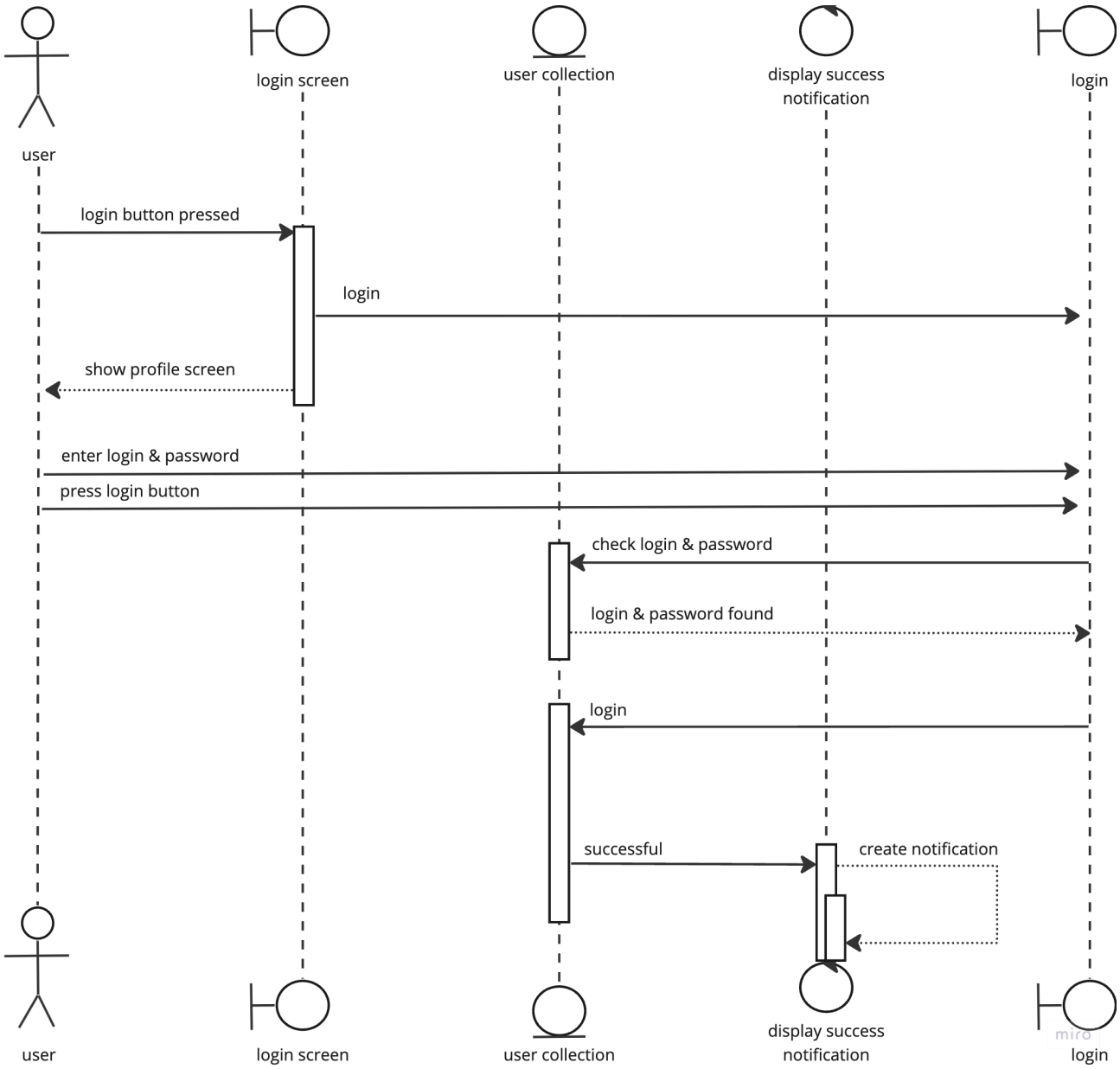

Рисунок 12 – Диаграмма последовательности для функции входа

Рассмотрим диаграмму последовательности для регистрации:

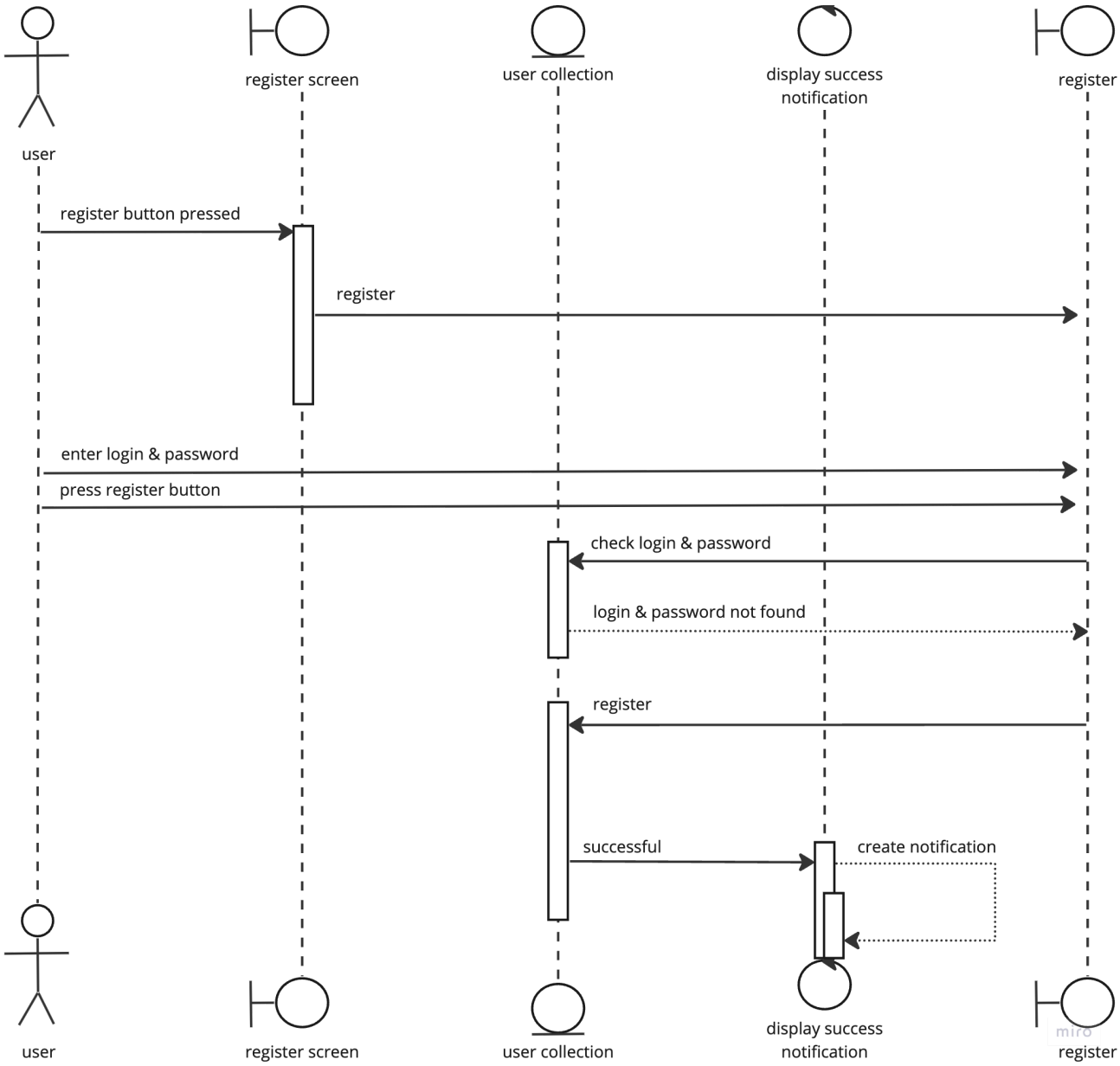

Рисунок 13 – Диаграмма последовательности для функции регистрации

Рассмотрим функцию поиска и построим диаграмму последовательности:

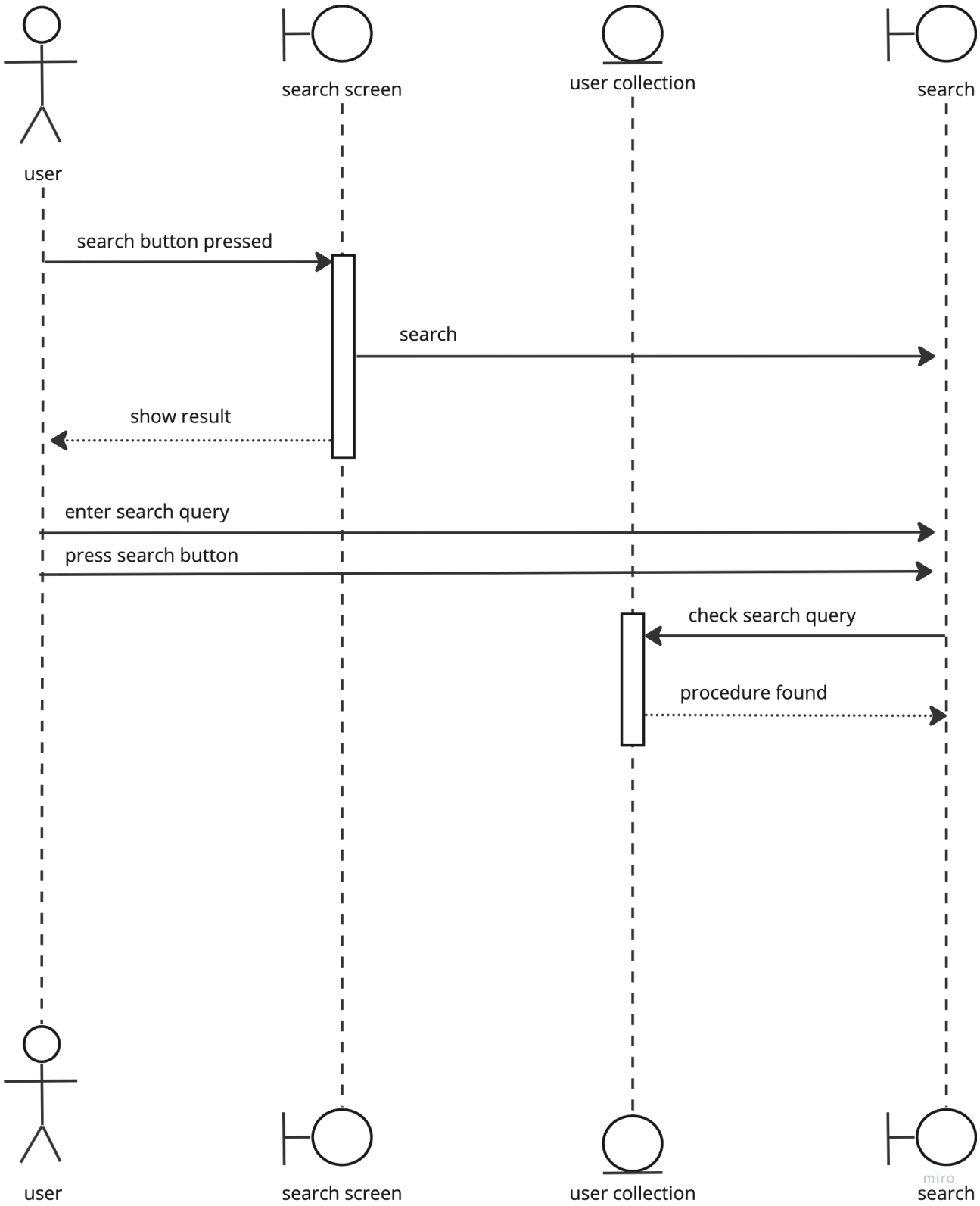

Рисунок 14 – Диаграмма последовательности для функции поиска

Рассмотрим функцию просмотра процедур пациента врачом, создадим диаграмму:

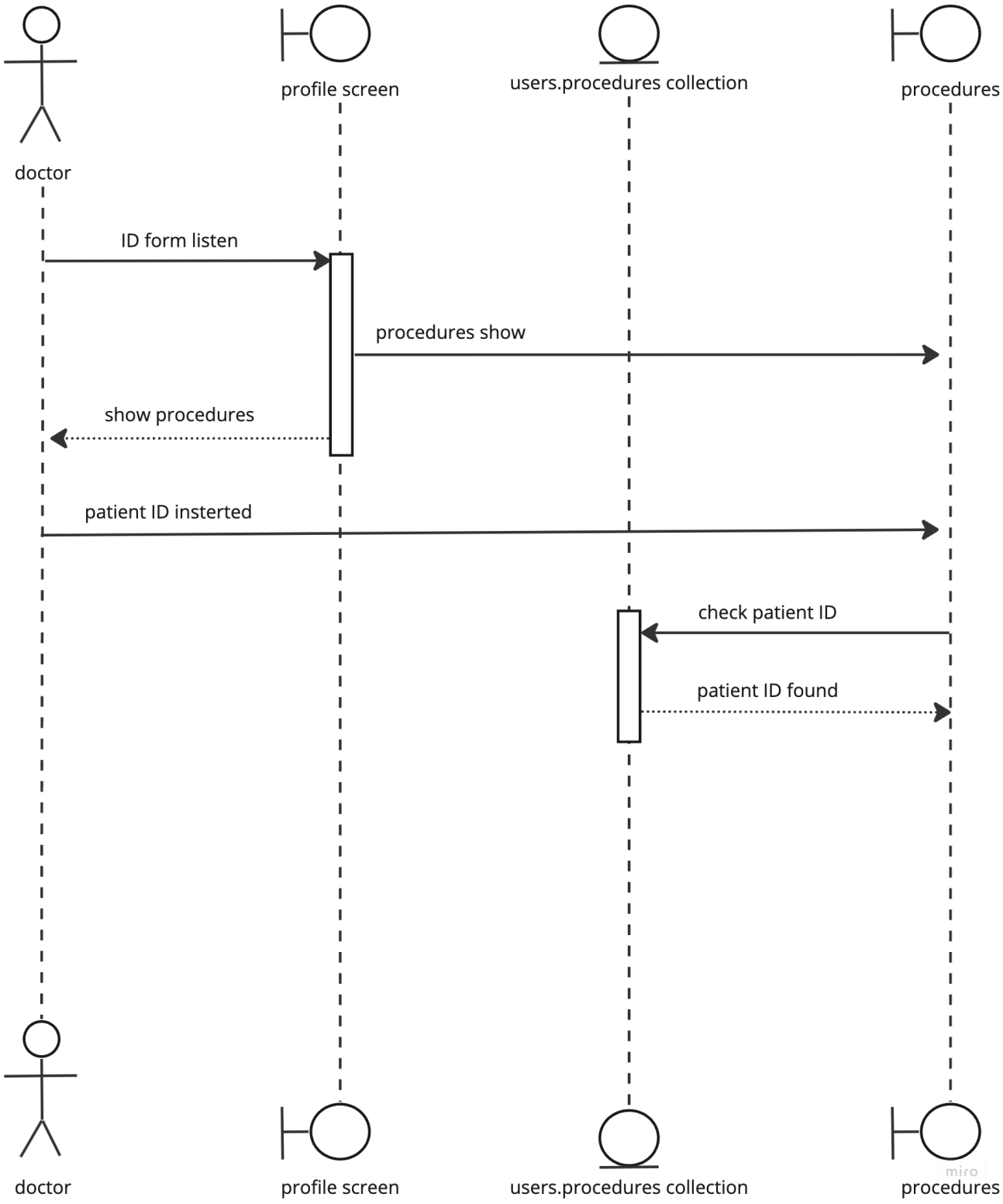

Рисунок 15 – Диаграмма последовательности для функции просмотра процедур врачом

Рассмотрим функцию просмотра своих процедур пациентом:

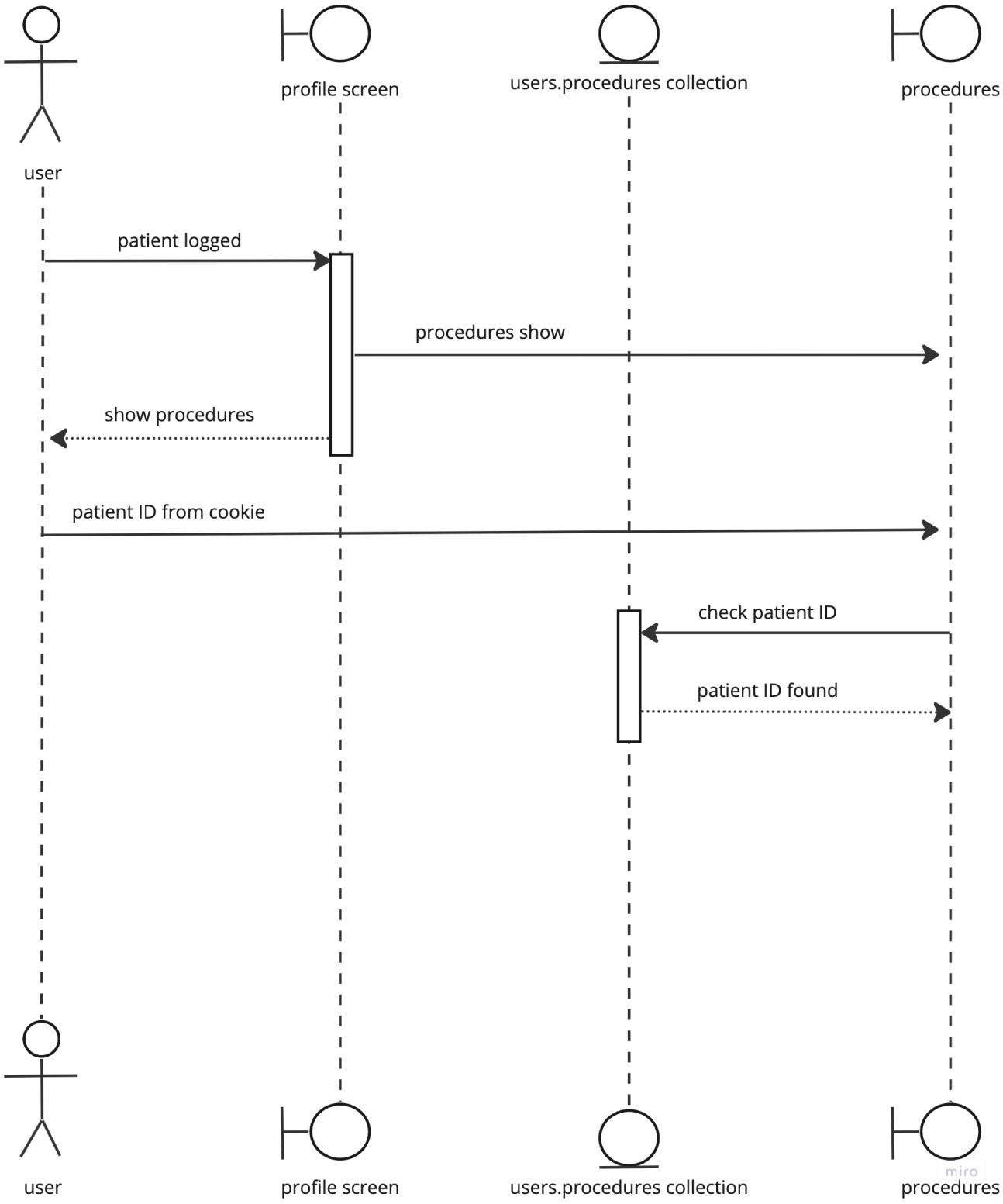

Рисунок 16 – диаграмма последовательности для функции просмотра процедур пациентом

Рассмотрим функцию удаления процедуры врачом:

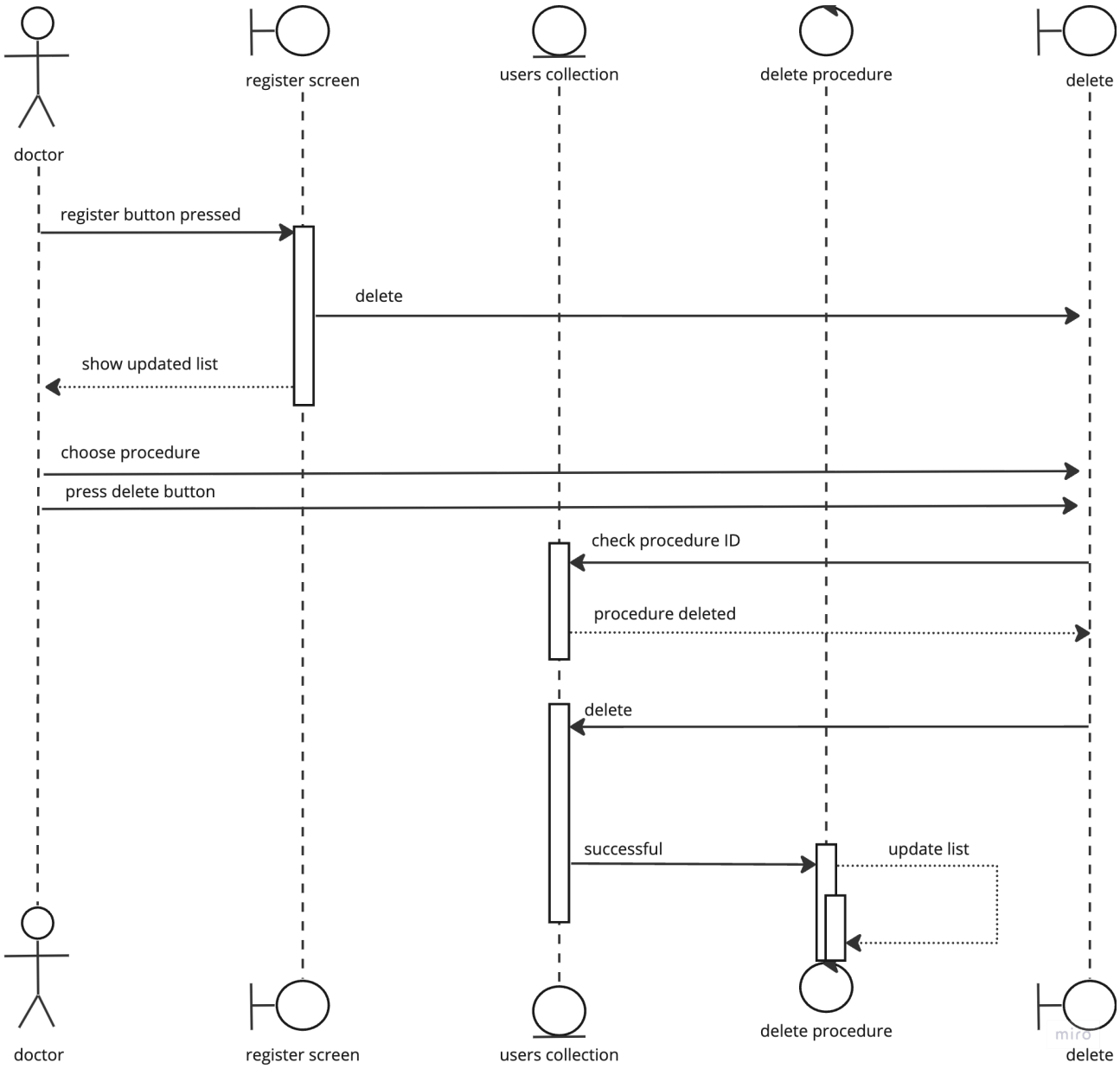

Рисунок 17 – Диаграмма последовательности для функции удаление процедуры врачом

Рассмотрим функцию обновления данных клиники о себе:

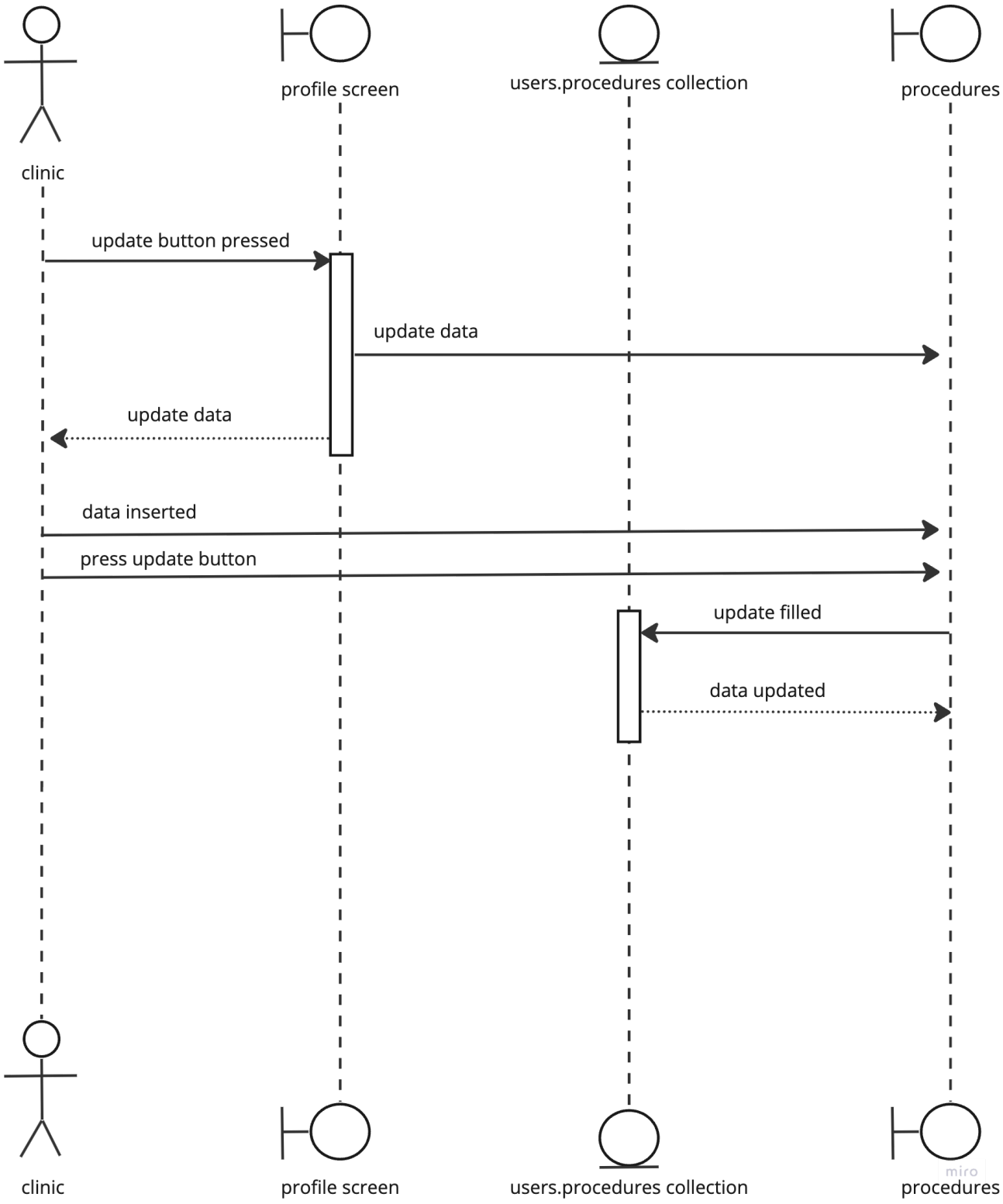

Рисунок 18 – Диаграмма последовательности для функции обновления данных клиники

4 Разработка

4.1 Серверная часть

Серверная часть веб-сайта работает на Node.js и использует фреймворк Express.js для создания HTTP-сервера.

4.1.1 Маршрутизация

Серверная часть слушает HTTP-запросы, которые приходят на различные URL-адреса (например, /profile, /search), и обрабатывает их, вызывая соответствующие функции обратного вызова. Эти функции могут получать данные от пользователя, извлекать данные из базы данных MongoDB и отправлять данные обратно на клиент в виде HTML-страниц или JSON-ответов. Разработанные маршруты:

– Маршрут '/' отвечает на GET запрос и отображает шаблон 'index';

– Маршрут '/clinics' отвечает на GET запрос и отображает шаблон 'clinics';

– Маршрут '/settings' отвечает на GET запрос и отображает шаблон 'settings';

– Маршрут '/search' отвечает на GET запрос и выполняет поиск клиник по тексту, полученному из запроса;

– Маршрут '/community' отвечает на GET запрос и отображает шаблон 'community';

– Маршрут '/login' отвечает на GET запрос и отображает шаблон 'login';

– Маршрут '/register' отвечает на GET запрос и отображает шаблон 'register'.

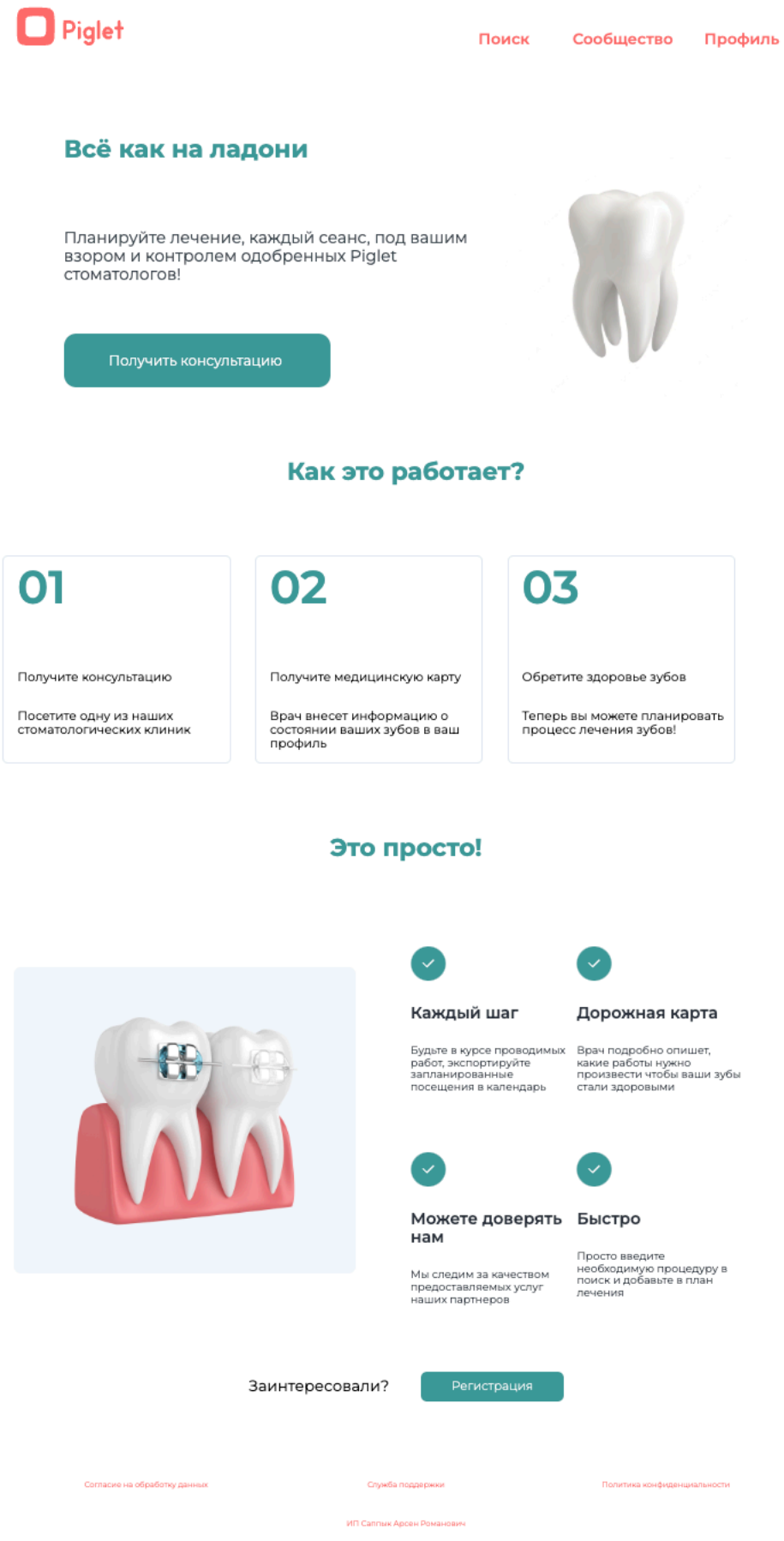

Рисунок 19 – Главная страница

4.1.2 База данных

Сервер взаимодействует с базой данных MongoDB, используя библиотеку Mongoose для создания моделей данных и выполнения операций просмотра и удаления. База данных используется для хранения информации о пользователях, процедурах и клиниках. В данном случае врач может просматривать и удалять процедуры пациента:

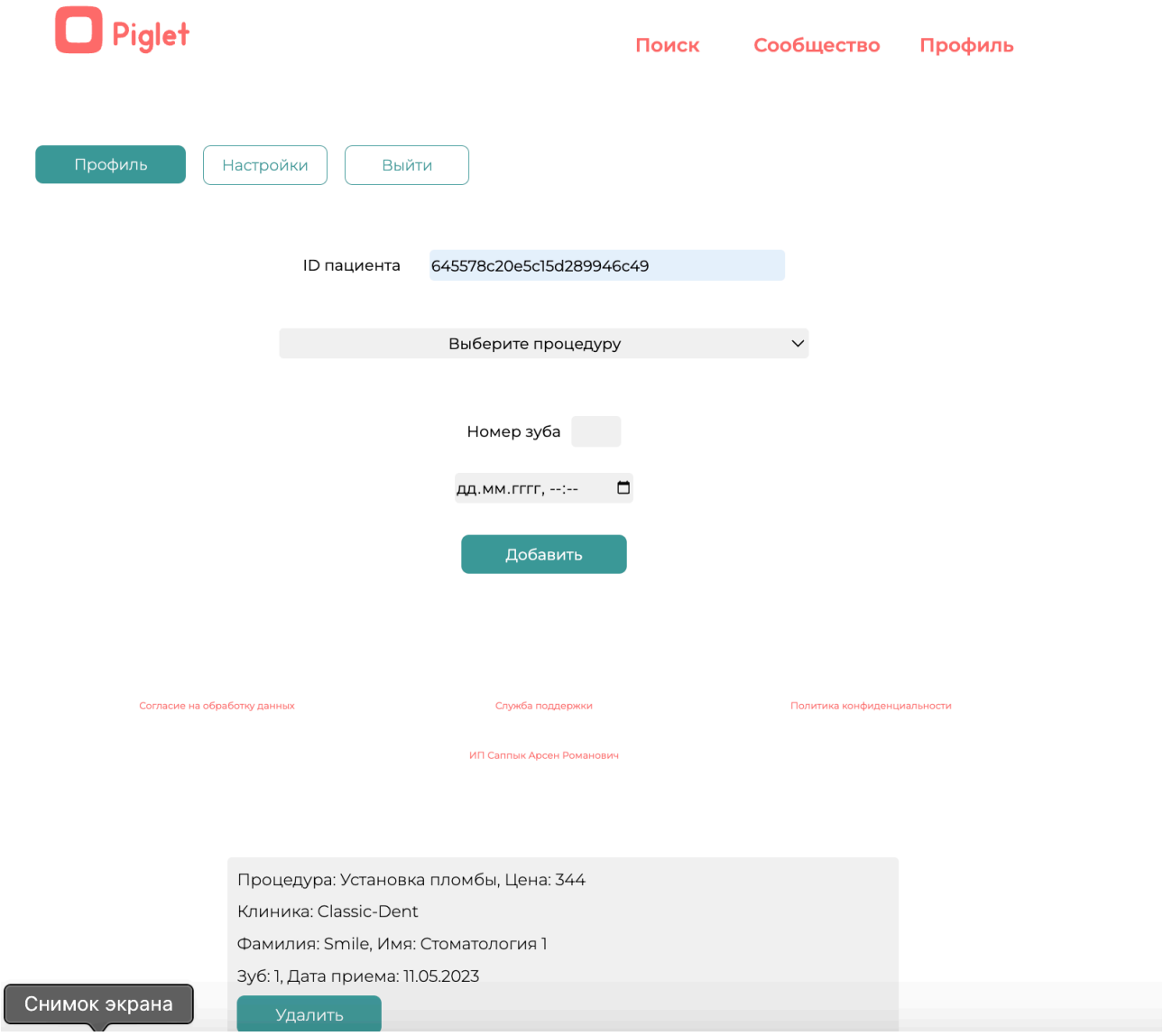

Рисунок 20 – Страница доктора

У пользователя исходя из ТЗ есть возможность просматривать план лечения и все процедуры как показано ниже.

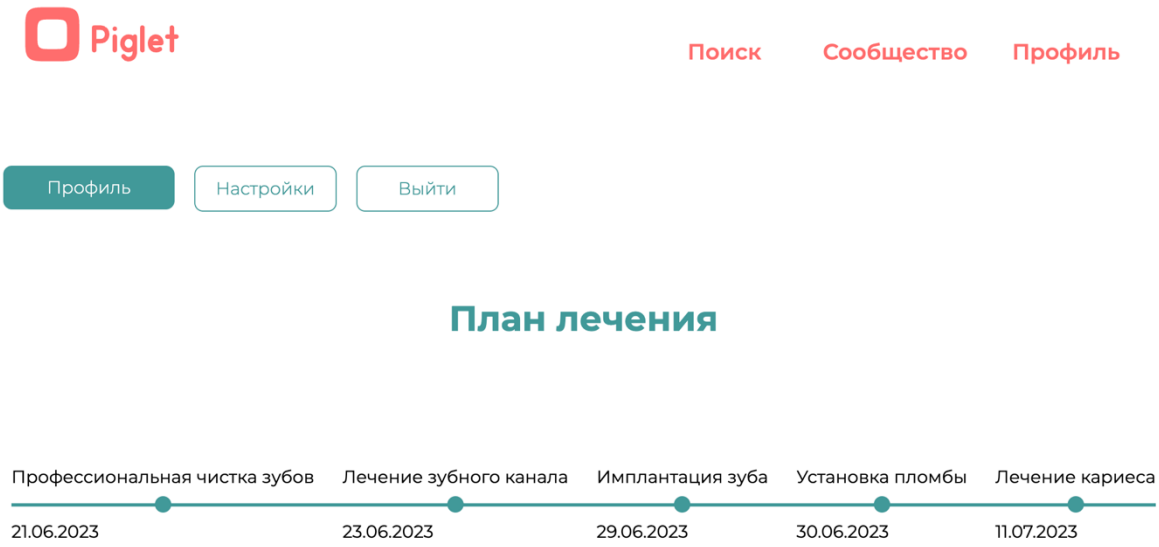

Рисунок 21 – План лечения

4.1.3 Аутентификация

Серверная часть поддерживает аутентификацию пользователей, используя пароли, которые хранятся в базе данных в хэшированном виде.

В проекте используется модуль express-session для управления сессиями пользователей. Когда пользователь успешно аутентифицируется или зарегистрируется, его идентификатор сохраняется в сессии (req.session.userId = user._id). Затем при доступе к защищенным маршрутам, таким как /profile, проверяется наличие идентификатора пользователя в сессии. Если идентификатор присутствует, то пользователь аутентифицирован и его профиль может быть отображен. Если идентификатор отсутствует или недействителен, то возвращается ошибка авторизации. Также по идентификатору мы можем посмотреть тип пользователя, пациент, врач или клиника. Тип помогает работать с защитой маршрутов от других пользователей.

При выходе из системы (/logout) сессия пользователя удаляется (req.session.destroy()), и он больше не считается аутентифицированным.

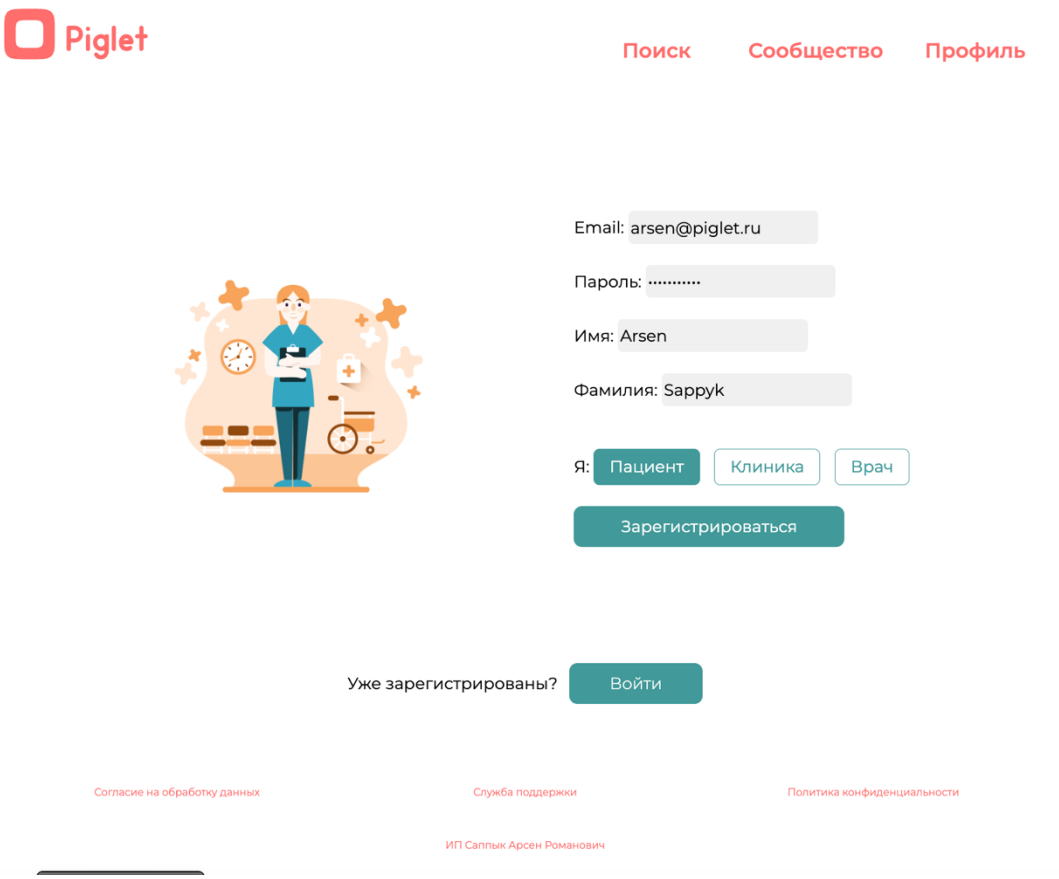

Рисунок 22 - Регистрация

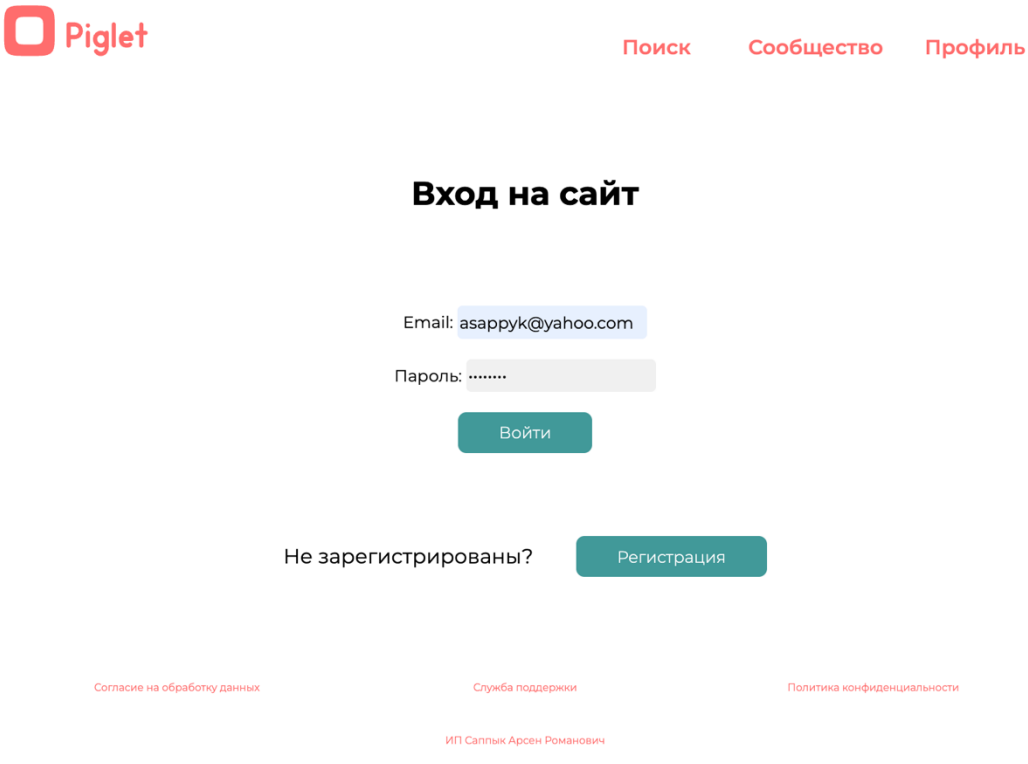

Рисунок 23 - Вход

4.1.4 Полнотекстовый поиск

Серверная часть поддерживает полнотекстовый поиск, используя функцию текстового поиска MongoDB.

Когда пользователь отправляет форму поиска (submit событие на searchForm), происходит следующее:

– Происходит отмена стандартного действия браузера с помощью event.preventDefault(), чтобы страница не перезагружалась;

– Получается текст поиска из элемента формы с помощью searchForm.elements.text.value;

– Отправляется асинхронный запрос (fetch) на сервер с использованием текста поиска в URL-параметре: /search?text=${searchText}.

Ответ от сервера получается в формате JSON с помощью response.json().

Если полученный ответ (json) не содержит результатов поиска (длина равна 0), то выводится сообщение "Ничего не найдено" в элемент searchResults и происходит возврат из функции.

Если полученный ответ содержит результаты поиска, то они сохраняются в переменную searchResultsArray.

Вызывается функция displayResults(json), которая отображает результаты поиска на странице.

Рисунок 24 – Страница поиска

4.2 Клиентская часть

Клиентская часть сайта написан на языке JavaScript и использует HTML и CSS для структурирования и оформления контента. Вот основные аспекты его работы:

4.1.5 Шаблонизация

Шаблонизация ускоряет разработку и облегчает поддержку сайта, так как повторяющиеся элементы преобразуются в одну строку вида {{> header}} которая вставляет шаблон header. По такому же принципу мы отображаем footer. Результат работы ниже:

Рисунок 25 – Шаблон header

Рассмотрим шаблон footer:

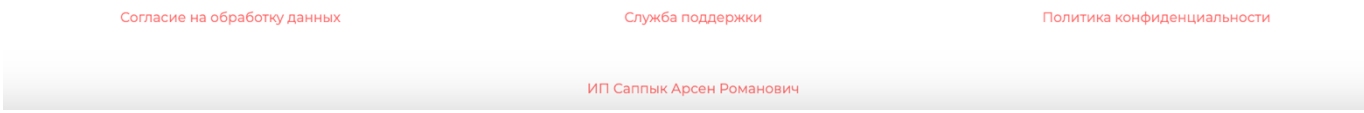

Рисунок 26 – Шаблон footer

4.1.6 Асинхронные запросы

Клиентская часть выполняет асинхронные запросы к серверной части, используя AJAX или Fetch API, для обновления частей страницы без необходимости полной перезагрузки.

AJAX запросы используются например, при показе всех процедур пациента, не требуя дополнительного ввода ID пациента.

4.1.7 Размещение

Также для работы вебсайта необходимо разместить код вебсайта (и серверный и клиентский) на выделенном сервере.

Для этого воспользуемся виртуальным выделенным сервером VPS, где мы можем размещать код с помощью репозитория GitHub.

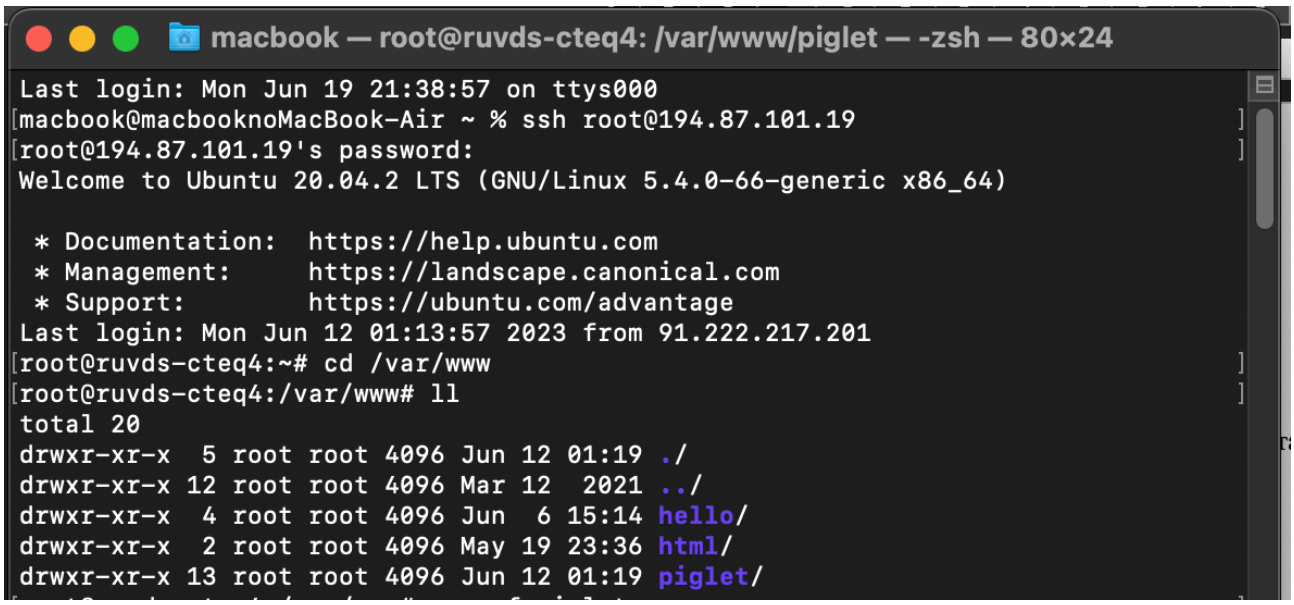

Рисунок 27– Размещенный код на виртуальном выделенном сервере

После размещения кода, необходимо установить удобное доменное имя, после регистрации доменного имени, вебсайт будет доступен не по IP адресу, а по доменному имени piglett.ru.

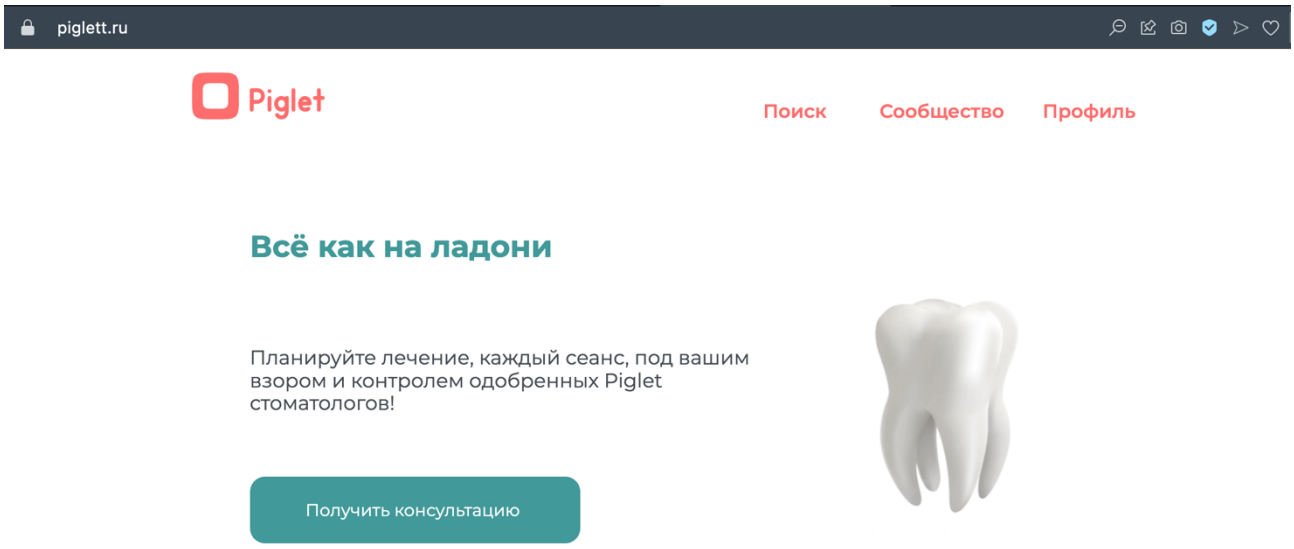

Рисунок 28 – Запрос в браузере piglett.ru и отображение страницы

ЗАКЛЮЧЕНИЕ

В ходе выполнения бакалаврской работы была затронута актуальность стоматологических сервисов, после утверждения темы проанализировали существующие аналоги в результате чего прямых аналогов не было найдено, но были учтены косвенные аналоги, имеющий частично схожий функционал.

Также с исследованием аналогов, было решено провести опрос среди стоматологических клиник г. Красноярск, после чего актуальность была подтверждена.

Исходя из поставленной задачи, выбрали стек технологий, позволяющая выполнить данную работу в срок сдачи бакалаврской работы, были учтены плюсы и минусы описанных технологий.

После утверждения темы работы и стека технологий, на основе задачи были спроектированы диаграммы, показывающие работу вебсайта.

Далее исходя из диаграмм и требований к вебсайту, был разработан вебсайт в соответствии с требованиями и целями бакалаврской работы.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Python: официальный сайт. — URL://www.python.org (дата обращения: 15.02.2023).

2. Habr: Разработка вебсайтов. — URL: https://habr.com/ru/hub/webdev/ (дата обращения: 15.02.2023).

3. Блог программиста: проектирование. — URL: https://pro-prof.com/archives/category/design_patterns (дата обращения: 10.03.2023).

4. Figma: Основа дизайна. — URL: https://help.figma.com/hc/en-us/sections/13148565160471-Figma-design-basics (дата обращения: 15.03.2023).

ПРИЛОЖЕНИЕ А

Код профиля

```
// Защищенный маршрут
app.get('/profile', requireAuth('patient'), async (req, res) => {
  try {
    const user = await User.findOne({ email: req.session.user.email });
    if (!user) {
      throw new Error('User not found');
    }

    // Сначала получим все процедуры для этого пользователя
    const proceduresForUser = user.procedures;

    // Теперь мы будем использовать Promise.all, чтобы получить все имена процедур одно-
временно
    const proceduresWithNames = await Promise.all(proceduresForUser.map(async procedure => {
        const procedureData = await Procedure.findById(procedure.procedure_id);  // Используйте
свою модель Mongoose здесь
        return {
            ...procedure._doc,
            appointmentDate: moment(procedure.appointmentDate).format('DD.MM.YYYY HH:mm'),
            procedure_name: procedureData ? procedureData.name : 'Unknown Procedure' // исполь-
зуйте правильное поле из ProcedureModel здесь
        };
    }));

    // Сортируем процедуры по дате
    proceduresWithNames.sort((a, b) => new Date(a.appointmentDate) - new
Date(b.appointmentDate));

    res.render('profile', { procedures: proceduresWithNames });
  } catch (err) {
    console.error(err);
    res.status(500).send(err.message);
  }
});

//маршрут для выдачи данных о зубе
app.get('/procedures/:toothNumber', async (req, res) => {
  const { toothNumber } = req.params;
  const user = await User.findById(req.session.user._id);
  const procedures = user.procedures.filter(procedure => procedure.toothNumber === Num-
ber(toothNumber));
  const procedureDetails = await Promise.all(procedures.map(async (procedure) => {
    const procedureData = await Procedure.findById(procedure.procedure_id);
    return {
      ...procedureData._doc,
```

```
      appointmentDate: procedure.appointmentDate // добавим дату в ответ сервера
    };
  }));
  res.json(procedureDetails);
});
```

ПРИЛОЖЕНИЕ Б

Код поиска

```
app.get('/search', async (req, res) => {
  const text = req.query.text;

  const clinics = await db.collection('clinics').aggregate([
    {
      $unwind: '$products'
    },
    {
      $lookup: {
        from: 'procedures',
        localField: 'products.product_id',
        foreignField: '_id',
        as: 'procedure_info'
      }
    },
    {
      $match: {
        'procedure_info.name': { $regex: text, $options: 'i' }
      }
    },
    {
      $project: {
        _id: 1,
        name: 1,
        procedure_name: { $arrayElemAt: ['$procedure_info.name', 0] },
        procedure_desc: { $arrayElemAt: ['$procedure_info.desc', 0] },
        procedure_price: '$products.price',
        rate: 1,
        contact: 1
      }
    }
  ]).toArray();

  res.json(clinics);
});
```

ПРИЛОЖЕНИЕ В

Код удаления и добавления процедур доктором

```
app.post('/doctor', requireAuth('doctor'), async (req, res) => {
const { userId, procedureId, toothNumber, appointmentDateTime, price } = req.body;

console.log(`-----------------------------------------------------`, price);  // Выводим price в консоль
console.log(price);  // Выводим price в консоль

const clinic = await Clinic.findById(req.session.user.clinicId);
const clinName = clinic.name;

const doctor = await Doctor.findById(req.session.user._id);

const firstName = doctor.firstName;
const lastName = doctor.lastName;

const appointmentDate = new Date(appointmentDateTime);

const procedure = {
  procedure_id: new mongoose.Types.ObjectId(procedureId),
  toothNumber,
  appointmentDate,
  price: parseFloat(price),
  clinicName: clinName,
  firstName,
  lastName
};

try {
  await User.updateOne(
    { _id: new mongoose.Types.ObjectId(userId) },
    { $push: { procedures: procedure } }
  );
  res.redirect('/doctor');
} catch (error) {
  console.error(error);
  res.status(500).send('Server error');
}
});


app.get('/user/:id/procedures', async (req, res) => {
  const user = await User.findById(req.params.id);

  const procedures = await Promise.all(user.procedures.map(async (procedure) => {
    const fullProcedure = await Procedure.findById(procedure.procedure_id); // здесь используется
Procedure вместо Procedures
    return {
```

```
      ...procedure._doc,
      procedure_name: fullProcedure.name
    };
  }));

  res.send(procedures);
});

app.delete('/user/:userId/procedures/:procedureId', async (req, res) => {
  const { userId, procedureId } = req.params;
  try {
    await User.findOneAndUpdate(
      { _id: userId },
      { $pull: { procedures: { _id: procedureId } } }
    );
    res.status(200).send();
  } catch (error) {
    console.error(error);
    res.status(500).send('Server error');
  }
});
```

Министерство науки и высшего образования РФ
Федеральное государственное автономное
образовательное учреждение высшего образования
«СИБИРСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»

Институт космических и информационных технологий

Кафедра вычислительной техники

УТВЕРЖДАЮ
Заведующий кафедрой
_____ О.В. Непомнящий
"16" 06 2023 г.

БАКАЛАВРСКАЯ РАБОТА

090301 Информатика и вычислительная техника

Вебсайт “Piglet”

Руководитель	_____ подпись	16.06.23 дата	доцент, канд. техн. наук должность, ученая степень	С.Н. Титовский
Выпускник	Саппык подпись	16.06.23 дата		А.Р. Саппык
Нормоконтролёр	_____ подпись	16.06.23 дата		С.Н. Титовский

Красноярск 2023