МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ

УЧРЕЖДЕНИЕ ОБРАЗОВАНИЯ «ВИТЕБСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ ИМЕНИ П.М. МАШЕРОВА»

Факультет математики и информационных технологий

Кафедра информатики и управления бизнесом

Допущен к защите

«\_\_»\_\_\_\_\_\_\_20\_\_г.

Заведующий кафедрой

\_\_\_\_\_\_\_\_С.А.Ермоченко

ДИПЛОМНЫЙ ПРОЕКТ

РАЗРАБОТКА WEB -ПРИЛОЖЕНИЯ В СФЕРЕ ЗДРАВОХРАНЕНИЯ

Специальность: 1-31 03 07 Прикладная информатика

Мащёнский Дмитрий Юрьевич,

4 курс, 41 группа

Научный руководитель:

Осипов Александр Владимирович,

Старший преподаватель

Витебск, 2022

**РЕФЕРАТ**

Дипломный проект представлен следующим образом. Электронные носители: 1 компакт-диск. Пояснительная записка составляет 51 страниц, 32 рисунка, 3 таблицы, 17 литературных источников, 3 приложения.

Ключевые слова: ReactJS, Redux, Saga, JavaScript, Node JS.

Предметная область: автоматизация и учёт пациентов в сфере здравоохранении.

Объект разработки: web-приложение для автоматизации и учета пациентов в сфере здравоохранении.

Целью разработки является web-приложение, которое предоставит возможность автоматизировать процесс докторов. При разработке веб-приложения были использованы следующие среды разработки и сторонние программы: Visual Studio Code, Chrome, Postman. Языки программирования, которые были использованы при разработке: ReacteJS, JavaScript, NodeJS База данных – MySQL.

Разработанный продукт может использоваться работниками медучреждения для упрощения работы с личными делами пациентов, записи пациентов, записи нового пациента, поиска существующей истории болезни пациента, обновления данных последующего обследования пациента, записи времени посещения пациента

Задачи, поставленные перед реализацией дипломного проекта, были выполнены. В перспективе возможно расширение функционала новыми возможностями.

**СОДЕРЖАНИЕ**

[ВВЕДЕНИЕ 5](#_Toc103540010)

[1 ТЕОРИТИЧЕСКАЯ 6](#_Toc103540011)

[1.1 Обзор аналогов 6](#_Toc103540012)

[1.1.1 Online-zapis 6](#_Toc103540013)

[1.1.2 МЕDODS 7](#_Toc103540014)

[1.2 Обзор средств разработки 8](#_Toc103540015)

[1.2.1 Языки программирования 8](#_Toc103540016)

[1.2.2 Обзор инструментов разработки клиентской части приложения 9](#_Toc103540017)

[1.2.5 Обзор систем управления базами данных 15](#_Toc103540018)

[1.3 Постановка задачи 15](#_Toc103540019)

[1.4 Обоснование выборки средств разработки 16](#_Toc103540020)

[2 СИСТЕМНОЕ ПРОЕКТИРОВАНИЕ 17](#_Toc103540021)

[2.1 Модуль работы backend 17](#_Toc103540022)

[2.2 Модуль работы frontend 17](#_Toc103540023)

[2.3 Модуль работы с данными 17](#_Toc103540024)

[2.4 Модуль аутентификации 18](#_Toc103540025)

[3 ФУНКЦИОНАЛЬНОЕ ПРОЕКТИРОВАНИЕ 19](#_Toc103540026)

[3.1 Модуль работы backend 19](#_Toc103540027)

[3.3 Модуль реляционной базы данных 20](#_Toc103540028)

[3.4 Модуль работы frontend 22](#_Toc103540029)

[4 РАЗРАБОТКА ПРОГРАММНЫХ МОДУЛЕЙ 24](#_Toc103540030)

[4.1 Аутентификация 24](#_Toc103540031)

[4.2 Список пациентов 24](#_Toc103540032)

[4.3 Список докторов 25](#_Toc103540033)

[4.4 Добавление пациентов 25](#_Toc103540034)

[4.4 Тайм менеджер 26](#_Toc103540035)

[4.5 Mail 27](#_Toc103540036)

[4.6 Цены 28](#_Toc103540037)

[5 ПРОГРАММА И МЕТОДИКА ИСПЫТАНИЙ 29](#_Toc103540038)

[6 РУКОВОДСТВО ПОЛЬЗОВАТЕЛЯ 32](#_Toc103540039)

[6.1 Руководство пользователя для использования веб-приложением 32](#_Toc103540040)

[6.1.1 Страница входа 32](#_Toc103540041)

[6.2 Список пациентов 33](#_Toc103540042)

[6.3 Добавление пациентов 35](#_Toc103540043)

[6.4 Тайм менеджер 38](#_Toc103540044)

[6.4 Mail 40](#_Toc103540045)

[6.5 Список докторов 41](#_Toc103540046)

[6.6 Цены 42](#_Toc103540047)

[ЗАКЛЮЧЕНИЕ 46](#_Toc103540048)

[СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ 47](#_Toc103540049)

# ВВЕДЕНИЕ

Сегодняшняя эпоха - это эпоха информатизации. С развитием технологий и научной теории традиционная медицина, основой которой является биотехнология, постепенно начала оцифровываться. И появилось современное здравоохранение, включающее в себя новое поколение информационных технологий. "Современное здравоохранение" - это не просто технический прогресс, но и всесторонние, многоуровневые изменения. Это изменение воплощено в следующем: изменения медицинской модели (от ориентированной на болезнь помощи к ориентированной на пациента), изменения в конструкции информатизации (от клинической информатизации к региональной медицинской информатизации), изменения в медицинском управлении (от общего управления к персонализированному управлению) и изменения в концепции профилактики и лечения (от сосредоточение внимания на лечении заболеваний, чтобы сосредоточиться на профилактическом здравоохранении).Эти изменения направлены на удовлетворение индивидуальных потребностей людей при одновременном повышении эффективности медицинской помощи, что значительно повышает качество медицинского обслуживания и медицинского обслуживания и представляет собой будущее направление развития современной медицины. Этот обзор начнется с концепции интеллектуального здравоохранения, затем кратко представит ключевые технологии, поддерживающие интеллектуальное здравоохранение, и объяснит достижения и проблемы ИТ, проанализировав статус применения этих технологий в важных областях медицины, прежде чем, наконец, выдвинуть будущие перспективы интеллектуального здравоохранения.

С точки зрения принятия решений, создание интегрированной платформы управления может реализовать такие функции, как распределение ресурсов, анализ качества и анализ эффективности, и может снизить медицинские расходы, максимально эффективно использовать ресурсы, и помогать больницам принимать решения относительно развития. Что касается удобства работы с пациентами, то пациенты могут получить доступ к множеству функций, таких как системы физического осмотра, онлайн-записи на прием и взаимодействие между врачом и пациентом. Эти автоматизированные системы делают процессы лечения пациентов более лаконичными. Пациенты ждут меньше времени и получают более гуманное обслуживание. В целом, интеграция, усовершенствование и автоматизация - это будущие направления развития "современных больниц ".

Таким образом, целью дипломного проекта является разработка web-приложение, которое предоставит возможность автоматизировать процесс докторов.

# 1 ТЕОРИТИЧЕСКАЯ

## 1.1 Обзор аналогов

Разрабатываемый программный продукт, конечно же имеет аналоги. Существует множество решений, которые предоставляют возможность автоматизировать процесс докторов.

Найти универсальный продукт, который смог бы решить все необходимые задачи, очень сложно. Максимально нецелесообразно переплачивать за продукт, в котором есть избыточный функционал, часть из которого останется непригодным. Поэтому стоит задача разработать продукт, соответствующий всем требованиям и реализующий все необходимые задачи.

В качестве аналогов были рассмотрены следующие программные продукты:

- Online-zapis;

- Medods;

### 1.1.1 Online-zapis

Online-zapis (рисунок 1.1) – это готовое решение для автоматизации любых заведений, которые работают с клиентами.

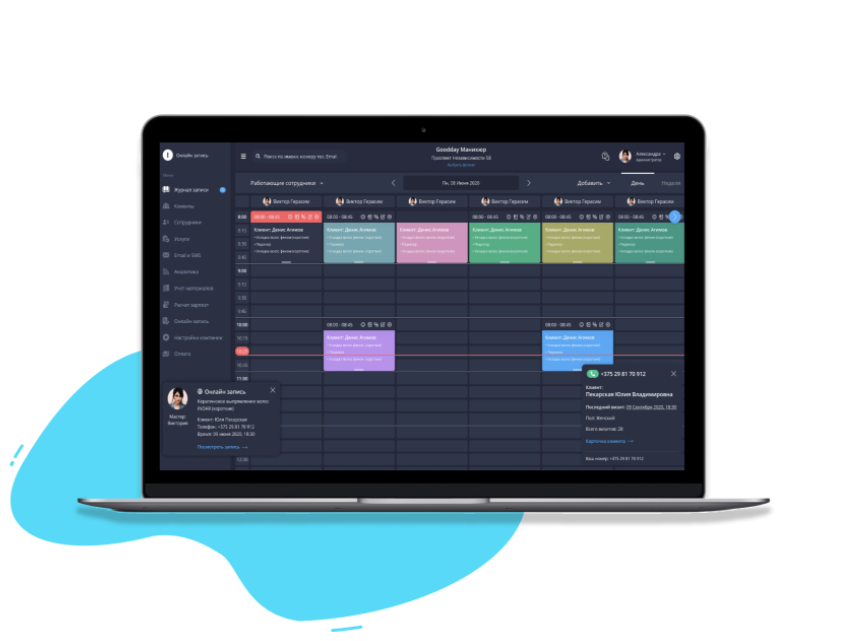


Рисунок 1.1 – Online-zapis

Решения, реализованные в Online-zapis:

* онлайн-запись для медицинского центра;
* онлайн-запись для салона красоты;
* онлайн-запись для шиномонтажа;
* онлайн-запись для юристконсульта;
* онлайн-запись для стоматологии;
* онлайн-запись для барбершопа;
* онлайн-запись для автомойки;
* онлайн-запись для коворкинга;
* онлайн-запись для лаборатории;
* онлайн-запись для спа-салона;
* онлайн-запись для СТО;
* онлайн-запись для фотостудии;
* онлайн-запись для поликлиники;
* онлайн-запись для массажной студии;
* онлайн-запись для техосмотра;
* онлайн-запись для персонального тренера;
* онлайн-запись для косметологии;

Компания Online-zapis позваляет внедрять клиентов через социальные сети, а также производить запись через TikTok, Facebook, Instagram.

### 1.1.2 МЕDODS

МЕDODS (рисунок 1.2) – это готовое решение для автоматизации медзаведения.



Рисунок 1.2 – МЕDODS [2]

Возможности, реализованные в МЕDODS:

* Онлайн-запись;
* ЭМК;
* Кассы по 54-ФЗ;
* СМС;
* Телефония;
* Лаборатории;
* МДЛП;
* Агрегаторы;
* Безопасность;

В основном это приложение для руководителей, маркетологов, администраторов, врачей. Приложение избавляет от заполнения бумаг, так как уже есть: гибкие и легкие шаблоны. Важным плюсом приложения, является быстрый поиск пациента и онлайн-уведомления о приходе пациента. Почему выбирают – это приложение дешевле большинство конкурентов, не нужно разбираться, можно сразу начинать работу, шифрование данных при передаче и хранении.

## 1.2 Обзор средств разработки

Для разработки веб-приложения используются языки программирования и фреймворки, которые облегчают разработку. В процессе разработки для отдельных модулей проекта может использоваться несколько языков программирования.

### 1.2.1 Языки программирования

**1.2.1.1 NODE JS**

*Node JS* –это платформа, основанный на движке JavaScript V8 Chrome.

Преимущества использования Node JS для разработки веб-приложений:

*Мощный стек технологий*:JavaScript, безусловно, является одним из самых мощных языков программирования, используемых во всем мире. Поэтому любой, кто решит использовать Node JS, естественно, получит полный доступ ко всем функциям JavaScript. Это означает отличную скорость и производительность приложений, легко обрабатываемые и повторно используемые коды, а также беспрепятственный доступ к многочисленным бесплатным инструментам. Это, естественно, помогает убедиться, что веб-компании и по всему миру имеют гораздо больше гибкости, когда дело доходит до работы над проектом. Фактическая работа по разработке также становится намного менее трудоемкой. Разработчики, имеющие навыки работы с внешним JavaScript, могут выполнять разработку на стороне сервера с минимальными усилиями.

*Гарантирует быстрые результаты обработки***:** Одна из лучших особенностей использования Node JS заключается в том, что он действительно очень быстрый. Движок, предназначенный для реализации Node JS, изначально создавался для браузера Chrome. Версия Chrome V8 была написана на C++ и предназначена для компиляции в машинный режим функций, написанных на JavaScript. Он может управлять всеми связанными процессами на чрезвычайно высоких скоростях. Кроме того, такие функции, как асинхронная обработка запросов и неблокирующий ввод/вывод, могут помочь Node JS быстро обрабатывать запросы без каких-либо задержек. Он также поставляется с моделью, основанной на событиях, которая обеспечивает быструю синхронизацию, если один и тот же язык используется как для клиентов, так и для серверной части. Это действительно может пригодиться при создании приложений реального времени с функциями, основанными на событиях.

*Богатая экосистема NodeJS* : Node JS имеет очень богатую экосистему и включает в себя более 836000 библиотек. Он также имеет менеджер пакетов по умолчанию, называемый npm, который обеспечивает быстрое улучшение инструментов JavaScript. Более 97% всех современных веб-приложений используют модули npm, и это делает Node JS идеальным решением для обширных требований к разработке веб-приложений.

*Отличная масштабируемость*: приложения, разработанные с помощью Node JS, чрезвычайно масштабируемы, что обеспечивает лучшую производительность и желаемый отклик приложений. Это одна из основных причин, почему разработчики Node Js настоятельно рекомендуют работать с Node JS.

### 1.2.2 Обзор инструментов разработки клиентской части приложения

*React* –это библиотека JavaScript, разработанная Facebook, которая, среди прочего, использовалась для создания Instagram.com. Его цель — позволить разработчикам легко создавать быстрые пользовательские интерфейсы для веб-сайтов и приложений. Основная концепция React.js — виртуальный DOM. Это дерево, основанное на компонентах JavaScript, созданных с помощью React, которое имитирует дерево DOM. Он делает минимально возможное количество манипуляций с DOM, чтобы поддерживать ваши компоненты React в актуальном состоянии. Разработчику достаточно описать, как части приложения выглядят в разных состояниях, а React будет обновлять их, когда они изменятся. Декларативное представление делает код более предсказуемым.

Будучи частью языка JavaScript, использование React дает множество преимуществ. Продукты, созданные с помощью React, легко масштабировать, единый язык, используемый на серверной/клиентской/мобильной стороне, обеспечивает выдающуюся производительность, существуют шаблоны рабочего процесса для удобной командной работы, код пользовательского интерфейса удобочитаем и удобен в сопровождении и многое другое. Ведущие мировые компании использовали React и другие технологии JS в некоторых продуктах, определяющих рынок (наиболее яркими примерами являются Instagram, Reddit и Facebook).

Одной из основных причин использования React.js для веб-разработки является оптимизированный интерфейс библиотеки и язык кодирования. Таким образом, облегченный API React усилен возможностями высокой производительности для обеспечения беспрепятственного и быстрого рабочего процесса разработки. Компоненты и концепции React действительно просты для понимания, поэтому здесь не нужно много времени для обучения.

В отличие от других популярных фреймворков, таких как Vue и Angular, здесь нет множества дополнительных HTML-атрибутов (созданных, когда JavaScript «втиснут» в HTML — стандартная практика для традиционных фреймворков и решений библиотек JS). В конечном счете, помещая JSX в JavaScript (буквально наоборот), React предоставляет гораздо более чистый, лучше читаемый и более полный код.

Использование React.js для веб-разработки может оказаться чрезвычайно удобным, потому что React — это один из тех случаев, когда вы осваиваете одну технологию, чтобы легко использовать ее на разных платформах. И все из-за того, что это библиотека по своей природе, основной целью которой является создание отдельных элементов и компонентов веб-дизайна (все, что угодно, от кнопок и меток до сеток и интерактивных функций).

Вдобавок ко всему, значительный вклад вносит обширное, давно сложившееся сообщество. Текущая экосистема React настолько обширна, что позволяет разработчикам создавать настольные решения и мобильные приложения, создавать статические веб-сайты, обрабатывать серверный рендеринг и интегрировать передовые технологические концепции (такие как виртуальная реальность и 360-градусные обзоры) с веб-решениями — и все это с помощью аналогичной простой веб-разработки. руководства и философии.

Виртуализируя и сохраняя DOM в памяти, React предоставляет невероятно быстрые возможности рендеринга, при этом все изменения представления легко отражаются в виртуальном DOM. Специализированный алгоритм сравнения сопоставляет прежние и существующие состояния виртуального DOM, вычисляя наиболее эффективный способ применения новых изменений, не требуя слишком большого количества обновлений. Затем вводится минимальное количество обновлений для достижения максимального времени чтения/записи, что приводит к общему повышению производительности.

Особый спрос на React для веб-разработки также продиктован возможностями Flux и Redux, которые он предлагает «из коробки». Создатели Facebook впервые представили программную архитектуру на основе Flux, которая дополнила стандартные компоненты React возможностями однонаправленного потока данных и предложила более оптимальную структуру действий.

Таким образом, центральный диспетчер используется для управления созданными действиями и хранилищами обновлений. Затем он обновляет представления в соответствии с изменениями в магазине. При этом все данные остаются в хранилищах — дубликаты не генерируются, что помогает поддерживать синхронизацию всех данных модели во всем приложении, не уходя далеко.

Помимо прочего, наши специалисты лично объясняют, почему мы используем React JS для веб-разработки, рассказывая об образцовом наборе инструментов и технологическом стеке. *Инструменты разработчика React* вместе с *инструментами разработки Redux* предлагают чрезвычайно удобные возможности, в то время как вы можете легко установить и использовать их как обычные расширения Chrome. В частности, с их помощью вы можете эффективно проверять компоненты иерархии на основе React (включая связанные реквизиты и состояния), проверять действия отправки и просматривать изменения состояния сразу в расширении (которые также можно записывать и использовать в качестве резервной копии для отладки в расширении). будущее).

*V8* – движок, который компилирует JavaScript в машинный код, пропуская стадию компиляции в промежуточный байт-код [12].

*ExpressJS* – это популярный, быстрый и гибкий MVC NodeJS фреймворк, который предлагает широкий набор функций для веб-приложений, а также для разработки мобильных приложений [13]. Он имеет API, который позволяет пользователям настраивать маршруты для отправки и получения запросов между frontend и базой данных. Кроме того, Express поставляется с системой просмотра, которая поддерживает более 14 шаблонизаторов и согласование контента.

Redux – это библиотека для управления глобальным состоянием приложения.

**Redux** предлагает решение, гарантируя, что:

* Ваше состояние заключено в хранилище, которое обрабатывает все обновления и уведомляет весь код, который подписывается на хранилище, об обновлениях состояния. Вам больше не нужно передавать состояние через все дерево компонентов, но вы можете подписаться на эти изменения гораздо ближе к тому месту, где необходима информация.
* Все изменения вносятся последовательно, чтобы обеспечить предсказуемый конечный результат без неожиданных эффектов и условий гонки.
* Состояние неизменяемо, а это означает, что каждое изменение состояния приводит к созданию совершенно новой версии состояния, что позволяет нам писать более предсказуемый код или просматривать любую предыдущую версию состояния с помощью **инструментов разработчика Redux**. Это приводит к невероятному опыту отладки.

Библиотека возникла на основе шаблона Flux. Flux – это архитектурный подход, который состоит из 4 компонент:

«Действие» – вспомогательная функция, которая упрощает передачу данных Диспетчеру.

«Диспетчер» – принимает Действия и рассылает данные обработчикам.

«Хранилище» – контейнер, в котором находится состояние приложения.

«Представление» – React-компоненты, которые собирают состояния из хранилищ и передают дочерним компонентам.

Принцип работы Flux представлен на рисунке 1.6.

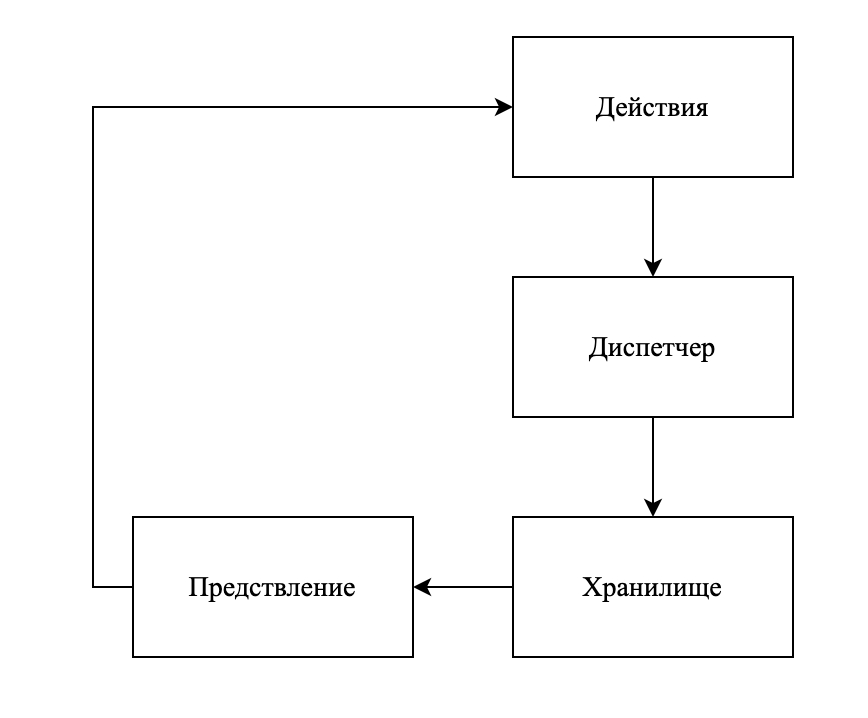


Рисунок 1.6 – Принцип работы Flux

«Действие» – это такая функция, которая вызывается из «Представления», чтобы отправить действие «Диспетчеру». Для различения действий используется константный подход, эта константа посылается вместе с данными действия.

«Диспетчер» – это главный узел процесса. Он получает данные и отправляет их всем зарегистрированным действиям, в свою очередь обработчики вызываются в определенном порядке, а далее действия могут быть обработаны в хранилище и состояние приложения будет изменено. Пример работы «Диспетчера» приведен на рисунке 1.7.

Функции, которые вызываются в «Диспетчерах» должны быть «чистыми».

Чистая функция – это такая функция, которая является детерминированной и не обладает сторонним эффектами.

Детерминированная функция – это такая функция, которая для одного и того же набора входных значений будет возвращен одинаковый результат.

Стоит отметить, что к сторонним эффектам можно отнести такие вещи, как: операции ввода-вывода, реагирование на исключительные ситуации, вызов обработчиков исключительных ситуаций, модификация глобальных переменных.

Существует так же другой тип побочных эффектов – это изменение входных значений в процессе работы функции.

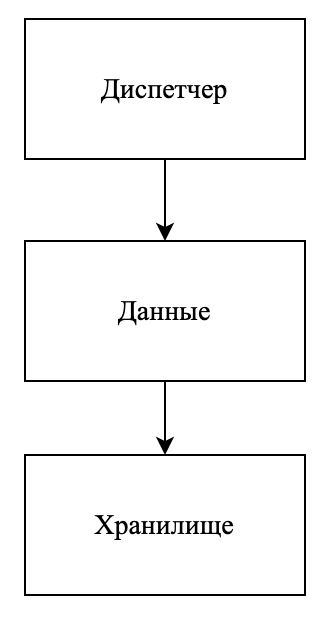


Рисунок 1.7 – Принцип работы «Диспетчера»

Хранилище во Flux занимается управлением состояниями приложения. Хранилище хранит данные и методы для получения данных. Подробная роль Хранилища во Flux представлена на рисунке 1.8.

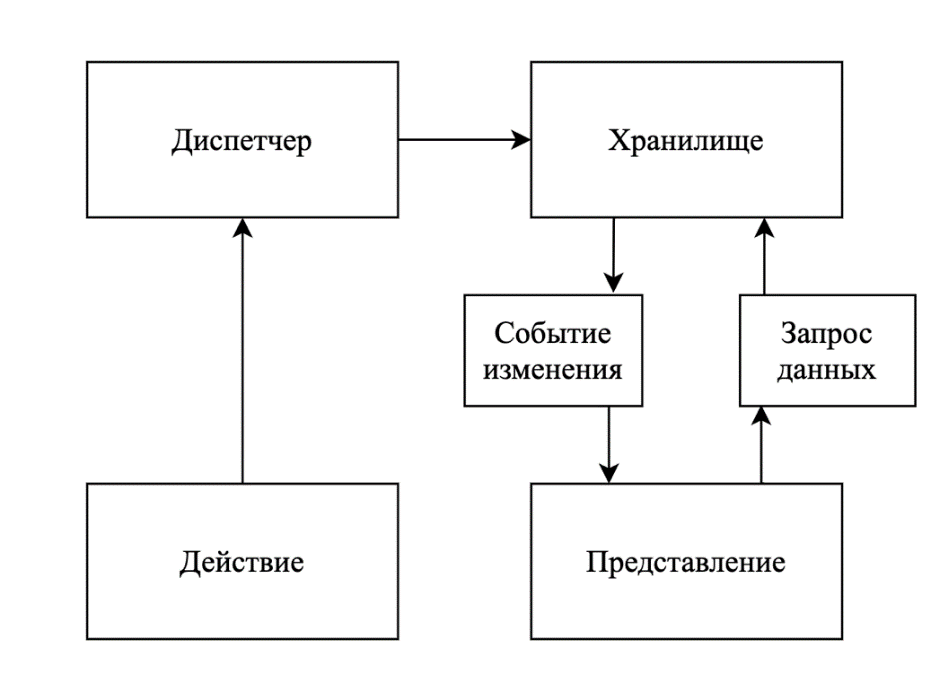


Рисунок 1.8 – Роль Хранилища во Flux

Как видно из рисунка, «Представление» подписано на хранилище. «Действие» вызывает «Диспетчера», в котором происходит обработка данных. Данные изменяются и кладутся в хранилище, как только произошло изменение «Представление» оповещается об этом с помощью события.

«Представление» – это компонента, написанная на React, которые подписываются на события и получают из хранилища состояние приложения, связанные с этим компонентом, каждый раз, когда оно обновляется.

Стоит отметить, что подпоясывание на «Хранилище» происходит в главном представлении, а далее данные передаются по свойствам дочерних «представлений».

Рисунок, с «Представлением» в шаблоне Flux представлен на рисунке 1.9.

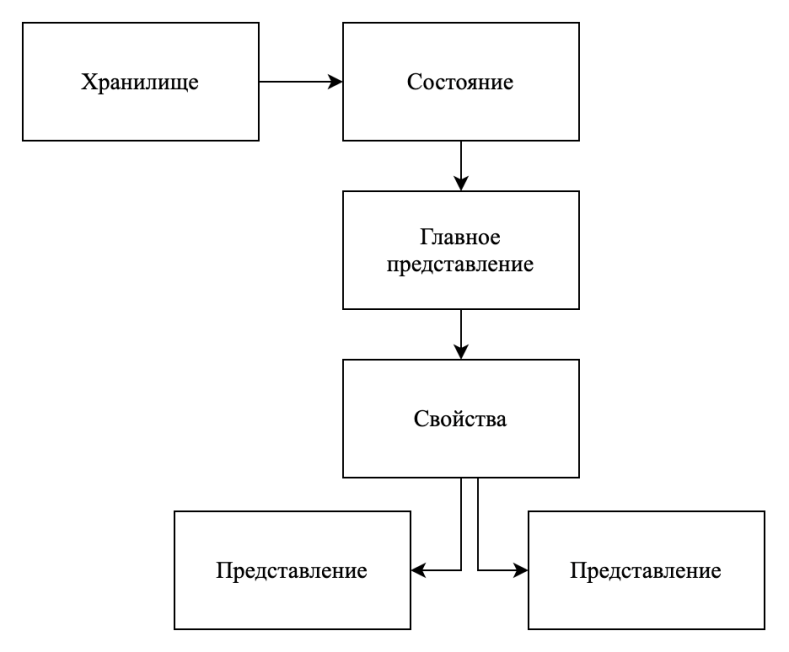


Рисунок 1.9 – Роль «Представления» во Flux

### 1.2.5 Обзор систем управления базами данных

*MySQL* – был запущен еще в 1995 году. С тех пор он претерпел несколько изменений в владельце / руководстве, прежде чем в 2010 году перешел в корпорацию Oracle. Хотя сейчас за это отвечает Oracle, MySQL по-прежнему является программным обеспечением с открытым исходным кодом, что означает, что вы можете свободно использовать и изменять его.

Плюсы реляционной базы данных:

* удобный инструмент для работы приложений в реальном времени;
* отлично подходит для обработки больших объемов данных.

Минусы реляционной базы данных:

* медленный доступ к данным;
* трудоемкость разработки.

Мы используем MySQL для внутренних операций, так как мы являемся разработчиком программного обеспечения. Мы используем этот инструмент в нашем серверном приложении для получения данных. В основном мы используем MySQL для нашей Системы управления реляционными базами данных (СУБД). Мы храним данные наших клиентов определенным образом. Мы должны создать различные уровни схемы. Мы предоставляем данные клиента, такие как имя, номер мобильного телефона, что было куплено и что им нравится или не нравится. Как только мы создадим схему, набор данных, нам нужно сохранить уникальные ограничения. Например, для клиентов мы присвоим каждой записи уникальный идентификатор, который будет первичным ключом. Все эти ограничения SQL мы будем использовать, и не должно быть никаких отклонений или репликации изолированных данных. Когда мы выполняем какие-либо операции обновления, это не должно влиять на какие-либо изолированные данные. Нам нужно позаботиться о многих вещах, таких как согласованность, изоляция и изменчивость при создании любой базы данных. Есть также еще несколько концепций, таких как нормализация и PL / SQL. Например, если я хочу запустить задание в определенное время, продукт будет использовать PL / SQL. Мы как бы внедряем наши навыки кодирования в базы данных. Наши операции осуществляются на месте. Тем не менее, мы находимся в процессе переноса всего в облако.

## 1.3 Постановка задачи

Исходя из выделенных недостатков существующих аналогов и основываясь на цели дипломного проекта, были поставлены следующие задачи:

* построение архитектуры приложения;
* разработка веб-приложения;
* разработка и конфигурация СУБД.

Приложение в конечном виде будет обладать следующими возможностями:

* учётная запись доктора;
* визуализация и обновление существующей записи;
* запись нового пациента;
* данные после осмотра пациентов;
* обновление данных после обновления пациентов;
* обновление данных после обновления пациентов;
* удалить историю болезни пациента;
* список докторов в клинике;
* менеджер для доктора;
* отправка сообщений пациентам

## 1.4 Обоснование выборки средств разработки

Было принято решение реализовать приложение в виде двух отдельных модулей: RESTful API backend сервера и frontend клиента.

Языком программирования для backend выбран Node JS, так как он многопоточен, прост в использовании, очень востребован на данный момент.

Языком программирования для frontend выбран JavaScript, библиотека – React. React выбран в целях ознакомления как одна из самых востребованных технологий, и как самая трендовая технология на StackOverflow.

Реляционной базой выбрана MySQL, так как она не является проприетарной, поддерживает контейнеризацию и пользуется популярностью среди разработчиков.

# 2 СИСТЕМНОЕ ПРОЕКТИРОВАНИЕ

Поставив задачи, а также обосновав выбор средств разработки, разбиваем приложение на модули:

* модуль работы backend;
* модуль работы frontend;
* модуль работы с данными;
* модуль базы данных;

## 2.1 Модуль работы backend

Модуль работы backend сервера выполняет следующие задачи:

* получение и обработка HTTP-запросов;
* взаимодействие с базой данных;
* обработка статистических данных;
* авторизация пользователей.

Модули серверов в разрабатываемом приложении является ключевыми, так как включают в себя многие другие модули, которые позволяют функционировать приложению.

С точки зрения кода программы, он содержит в себе все необходимые для функционирования сервера конфигурационные классы, файлы настроек и каркас API.

При разработке backend сервера был использован Node JS. Автоматическая конфигурация позволяет разработчикам не изобретать колесо каждый раз, когда они создают новое приложение.

## 2.2 Модуль работы frontend

Модуль работы frontend сервера предназначен для отправки HTTP-запросов API серверу и обработки получаемых ответов. Данный модуль содержит в себе конфигурации и файлы настроек, необходимые для функционирования сервера.

При разработке frontend сервера в основу лег такой фреймворк, как ReacteJS, а основная платформа разработки NodeJS.

## 2.3 Модуль работы с данными

База данных содержит в себе всю необходимую для функционирования приложения информацию: пользователи, магазины, терминалы, чеки, товары и т.п.

Модуль взаимодействует с базой данных посредством SQL запросов, отправляемых сервером. В зависимости от запроса, модуль позволяет получать, сохранять, изменять или удалять данные.

Этот программный модуль имеет свой репозиторий (программная абстракция, которая взаимодействует с базой данных) для осуществления всех необходимых операций.

При разработке сервера был сделан выбор в сторону MySQL. MySQL – реляционная система для управления базами данных. В реляционных базах данных информация сведена в таблицы, столбы и строки.

## 2.4 Модуль аутентификации

Модуль авторизации предназначен для идентификации докторов с целью сохранения его активности. Так же для идентификации своих пациентов, своего графика работы и своих email пациентов.

# 

# 3 ФУНКЦИОНАЛЬНОЕ ПРОЕКТИРОВАНИЕ

Целью функционального проектирования является разработка функционала программного продукта. В этот этап входит разработка классов, которые позволят реализовать требуемые функции. Определяются конкретные поля и методы каждого из классов, задачи, которые будут на них возложены.

Раздел определяет функционирование разрабатываемой программы и исчерпывающую информацию о ее структуре с точки зрения описания данных и обрабатывающих их функций. Описывается структура таблиц и связи между ними.

## 3.1 Модуль работы backend

Модуль работы backend содержит функции обработки HTTP-запросов, защиты приложения от несанкционированного доступа. Для HTTP – запросов используются такие методы как GET, PUT, POST.

* followUp – метод обработки PUT запроса, которому соответствует URL ‘/api/update/LatestCheckupInfo’ принимает на вход объект, содержащий данные о осмотре пациента. В случае обновления данных, происходит возвращение ответа со статусом 200 (Ok).
* deleteHistory – метод обработки DELETE запроса, которому соответствует URL ‘/api/delete/:id’ принимает на вход объект, содержащий данные о пациенте. В случае удаления данных, происходит возвращение ответа со статусом 200 (Ok).
* detailInfo – метод обработки GET запроса, которому соответствует URL ‘/api/get/HistoryList/Details/:id’ принимает на вход объект, содержащий данные о пациенте. В случае нахождения данных, происходит получение данных и возвращение ответа со статусом 200 (Ok).
* addPatientPersonalInfo – метод обработки POST запроса, которому соответствует URL ‘/api/PatientPersonalInfo/insert’ принимает на вход дату рождения, пол, адрес, группу крови. В случае нахождения пациента, происходит запись данных и возвращение ответа со статусом 200 (Ok).
* addPatientCheckupData – метод обработки POST запроса, которому соответствует URL ‘/api/PatienCheckUpinfo/insert’ принимает данные о осмотре пациента. В случае нахождения пациента, происходит запись данных и возвращение ответа со статусом 200 (Ok).
* historyList– метод обработки GET запроса, которому соответствует URL ‘/api/get / HistoryList' принимает данные о пациенте. В случае нахождения пациента, происходит отправка данных и возвращение ответа со статусом 200 (Ok).
* doctorList– метод обработки GET запроса, которому соответствует URL ‘/api/get / DoctorList’ принимает данные о докторах. Происходит отправка данных и возвращение ответа со статусом 200 (Ok).
* loginUser– метод обработки POST запроса, которому соответствует URL ‘/api/loginUser’ принимает данные логин и пароль. Происходит проверка данных и возвращение ответа со статусом 200 (Ok) или если таких данных нет 404(Not Found)

## 3.3 Модуль реляционной базы данных

В ходе анализа предметной области приложения были выделены необходимые сущности, их структура перенесена на таблицы реляционной базы данных, описание которых будет представлено ниже.

Таблица patient\_checkup\_info содержит в себе данные о пациентах и имеет следующие поля:

* Id: int – уникальный идентификатор карточки;
* patientId: int – уникальный идентификатор пациента;
* dateofAdmission: varchar(15) – дата посещения больницы;
* chiefComplaint: varchar(200) – жалоба;
* historyofpatientIllness: varchar(200) – история болезни пациента;
* generalAppearance: varchar(200) – общий вид;
* BP: varchar(20) – давление;
* HR: int(11) - частота сердцебиения;
* PR: int(11) - частота дыхания;
* Temperature (float) – температура;
* O2Sat: int(11) – количество кислорода;
* Weight: float – вес;
* Skin: varchar(300) – кожа;
* Heent: varchar(300) – ухо, горло, нос;
* Neck : varchar(300) – шея;
* Chest: varchar(300) – грудь;
* CVS: varchar(300) – пищевод;
* Abdomen: varchar(300) - брюшная полость;
* Gut: varchar(300) – кишечник;
* Extromities: varchar(300)- конечности;
* Neuro :varchar(300) – нервы;
* admittingDiagnosis: varchar(300) – подтверждающий диагноз;
* treatment: varchar(300) - лечение

Таблица doctor\_data содержит в себе данные о докторах. Данная таблица имеет следующую структуру:

* Id: int – уникальный идентификатор доктора;
* name: varchar(200) – фамилия;
* dateOfBirth: varchar(20) – дата рождения;
* gender: varchar(11) –пол;
* specialization: varchar(11) – специализация;
* address: varchar(200) – адрес;
* experience: varchar(15) – опыт работы;

Таблица patient\_data содержит в себе данные о докторах. Данная таблица имеет следующую структуру:

* Id: int – уникальный идентификатор пациента;
* name: varchar(200) – фамилия;
* dateOfBirth: varchar(20) – дата рождения;
* gender: varchar(11) –пол;
* bloodType: varchar(11) – группа крови;
* address: varchar(200) – адрес;
* dateOfLatestAdmission: varchar(15) – дата последнего осмотра;

Таблица time\_data содержит в себе данные о расписании доктора. Данная таблица имеет следующую структуру:

* Id: int – уникальный идентификатор события;
* title: varchar(200) – название таблицы;
* start: TIMESTAMP – дата записи таблицы;
* end: TIMESTAMP –дата окончания события ;
* desc: varchar(11) – описание таблицы;
* address: varchar(200) – адрес;

Таблица table\_data содержит в себе данные о всех ценах назначенные медучреждением. Данная таблица имеет следующую структуру:

* Id: int – уникальный идентификатор строки;
* Name: varchar(200) – название услуги;
* Executor: varchar(200)– содержит в себе ФИО исполнителя услуги;
* Age: int(11) –возраст ;
* Category: int(11) – категория врача;
* Cost: int(11) – цена за услугу.
* service: int(11) – цена за повторную услугу;

## 3.4 Модуль работы frontend

Клиентская часть приложения реализована на JavaScript библиотеке React. Так как основным принципом библиотеки React является построение приложения с помощью компонент, то и функциональное проектирование будет строиться на создании и описании компонент.

Данное приложение было создано с помощью механизма create-react-app, а также дополнялось с помощью пакетного менеджера npm. Данный механизм создает полностью настроенное React приложение. Для учета зависимостей используется файл package.json, в котором сохраняются названия библиотек, способов запуска, настройки линтеров, используемые приложением. Все зависимости устанавливаются в папку node\_moduls. В папке public лежит html-файл, в который будет рендерится приложение. Каталог src отвечает непосредственно за само приложение со всеми компонентами, логикой.

Компонент SignIn предназначен для отображения первой страницы приложения, которая предназначена для аутентификация, с которой пользователю предстоит работать.

Компонент Table предназначен для работы с данными и записи информации. Для таблицы данных требуется значение в виде массива объектов и столбцов, определенных с помощью компонента сolumn. Во всех продукта, содержащий код, название, описание, изображение, категорию, количество, цену, статус инвентаризации и свойства рейтинга, используется для определения объекта, который будет отображаться в таблице данных. Table не имеет никаких ограничений на способ предоставления данных.

Компонент SideBar нужен, чтобы отобразить список пунктов меню и многоуровневых выпадающих меню по обе стороны страницы для навигации по вашему веб-сайту. Компонент боковой панели можно использовать в качестве дополнительного элемента по отношению к панели навигации, отображаемой в левой или правой части страницы, используемой для навигации в вашем веб-приложении, включая пункты меню, многоуровневые выпадающие меню, элементы вызова действий и многое другое.

Koмпонент Header нужен, чтобы отображать информацию и действия, относящиеся к текущему экрану.

Кнопки перехода к определенной части приложения также являются и ссылками, благодаря которым в React реализован роутинг. Основными библиотеками для этого служат react-router и react-router-dom, которые и используются в данном приложении. Роутинг в React удобен тем, что можно с легкостью менять как полностью всю страницу, так и некоторые ее части.

Material-UI — одна из лучших библиотек пользовательского интерфейса для React.js. Будучи проектом с открытым исходным кодом, Material UI содержит компоненты React, которые реализуют Google Material Design. Стремясь создать высококачественный цифровой опыт, он также помогает нам быстро создавать красивые приложения с согласованными макетами без ущерба для производительности.

Кроме того, библиотека предоставляет вам множество настраиваемых и доступных виджетов пользовательского интерфейса, которые можно импортировать и использовать. Разработчику больше не нужно писать код для создания компонентов с нуля, не только экономится значительное количество времени веб-разработки, но и может быть потрачено на разработку важных функций.

# 4 РАЗРАБОТКА ПРОГРАММНЫХ МОДУЛЕЙ

## 4.1 Аутентификация

Аутентификация используется для подтверждения личности пользователя, которое заключается в процессе проверки учетных данных. Если введенные данные совпадают с данными, сохраненными в базе данных, пользователю предоставляется доступ. В противном случае вход и дальнейшее использование сервиса блокируется.

Для аутентификации использовался метод Login который получает на вход логин и пароль пользователя (рисунок 4.1)

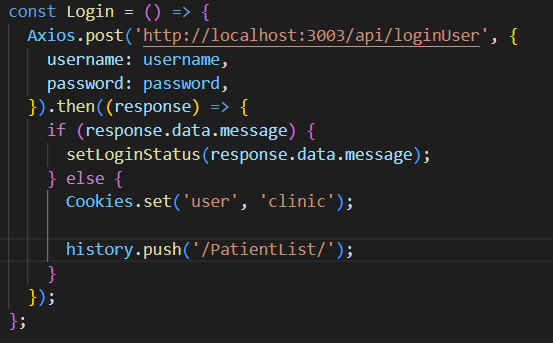


Рисунок 4.1 – аутентификация

Когда сервер получит, пароль и логин. Он проверит существует ли он у него, если он будет, то перейдёт на страницу пациентов, если нет, то получим на странице входа сообщения.

## 4.2 Список пациентов

Список пациентов, является одной из главных страниц приложения. В котором хранится вся информация пациентов и может происходить умный поиск. Так же на этой странице можно удалять пациентов и посмотреть детали осмотра. Поиск происходит при помощи регулярной функции, что позволяет быстро определить пациента без каких-либо усилий. Удаление пациента происходит благодаря функции deleteHistory (рисунок 4.2)

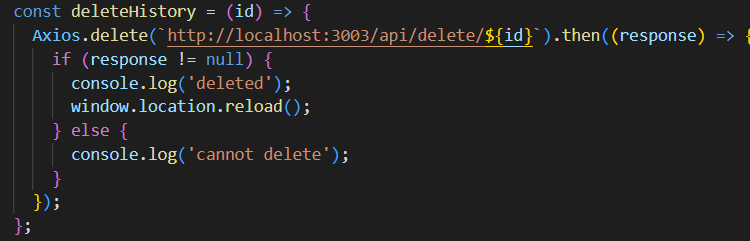


Рисунок 4.2 – функция удаления пациента

По уникальному номеру пациента происходит удаление пациента.

## 4.3 Список докторов

Список докторов – эта страница, предназначенная для поиск своих коллег по работе. В которой хранится вся информация докторов и может происходить умный поиск (рисунок 4.3)

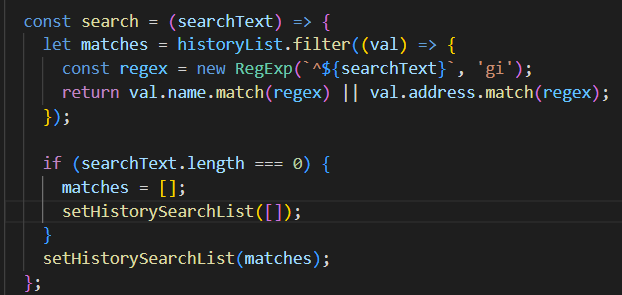


Рисунок 4.3 – функция поиска докторов

## 4.4 Добавление пациентов

Добавление пациентов – эта страница, предназначенная для добавления пациентов доктором. Доктор заполняет все данные пациента и производит осмотр, все данные осмотра и лечения он сохранение при помощи функции addPatientPersonalInfo (рисунок 4.4)

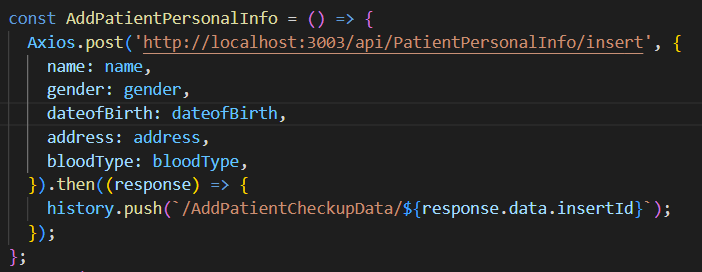


Рисунок 4.3 – функция сохранения информации о пациентах

## 4.4 Тайм менеджер

Тайм менеджер – эта страница, предназначенная для планировки своим временем докторам. Доктор может планировать своё время как на неделю, так и на месяц, год. Он состоит из компонента Calendar (рисунок 4.5)

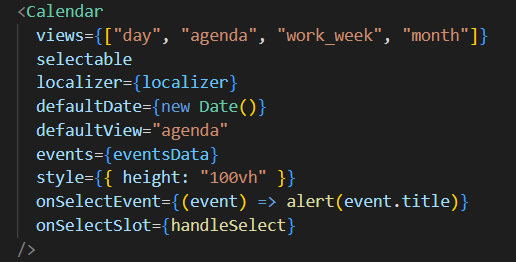


Рисунок 4.4 – компонент Calendar с пропсами

Для получения данных из БД, был разработан хук useEffect который при загрузке страницы отправит запрос(рисунок 4.6)

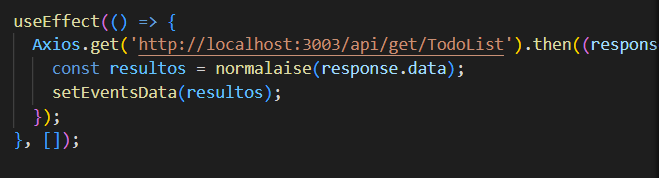


Рисунок 4.4 – получение данных

Для создания события используем функцию handleSelect, которая отправляет данные на создание нового события и добавляет его в тайм-менеджер. (рисунок 4.7)

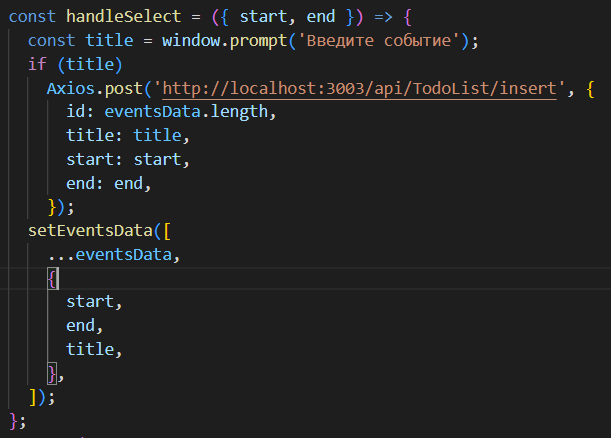


Рисунок 4.7 – отправка данных

## 4.5 Mail

Mail – эта страница, предназначенная для отправки сообщений доктором, своим пациентам. Для этого был разработан компонент ContactUs Для отправки сообщений нам нужен личный номер email, так же название сервиса. Для того, чтобы пользователь понимал, что у него сообщения отправилось был добавлен компонент toast(рисунок 4.5)

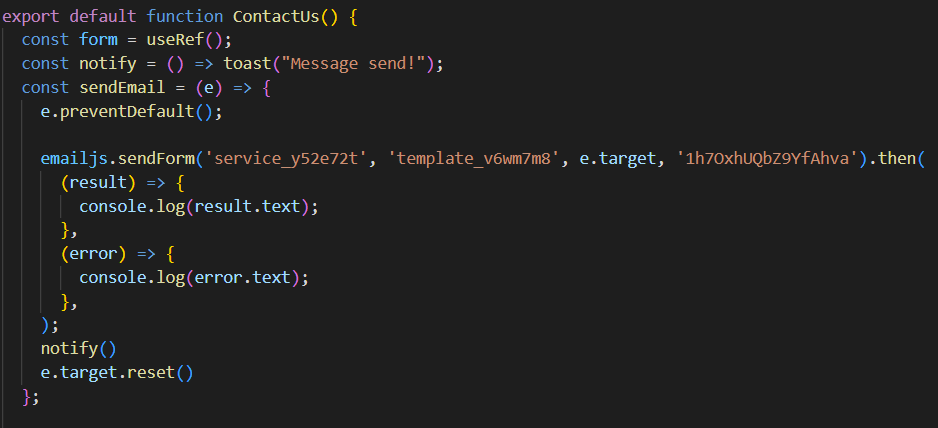


Рисунок 4.5 – компонент СontactUs

## 4.6 Цены

Цены – эта страница, предназначенная для создания, формирования цен за работу доктора. Для этого был разработан компонент EnhancedTable (рисунок 4.6)

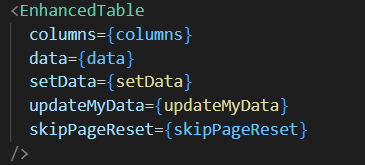


Рисунок 4.6 – компонент EnhancedTable с пропсами

Для получения данных из БД, был разработан хук useEffect который при загрузке страницы отправит запрос (рисунок 4.6)

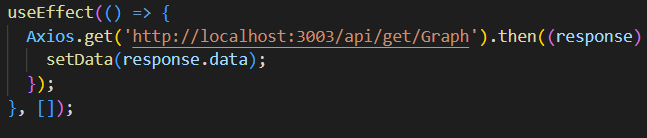


Рисунок 4.4 – получение данных

Для создания цен используем функцию handleAdd, которая отправляет данные на создание новой цены и добавляет его в таблицу. (рисунок 4.7)

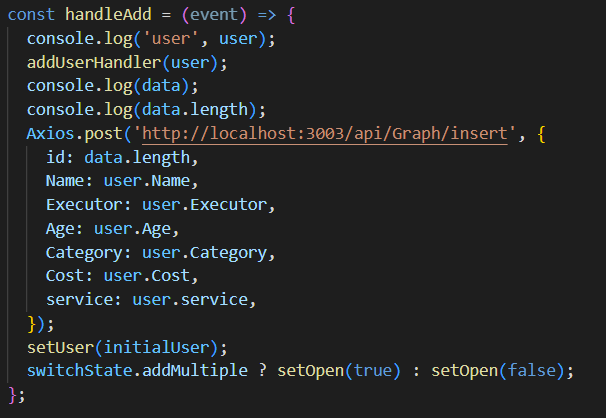


Рисунок 4.7 – отправка данных

# 5 ПРОГРАММА И МЕТОДИКА ИСПЫТАНИЙ

Тестирование — один из самых сложных этапов процесса разработки программного обеспечения. Он требует пристального внимания к деталям и не может быть завершен, если вы не применяете методический подход. Вот почему тестирование программного обеспечения разбито на несколько этапов.

Всего существует четыре этапа тестирования программного обеспечения, которые включают модульное тестирование, интеграционное тестирование, системное тестирование и приемочное тестирование. С учетом сказанного, эти четыре этапа можно разделить на два типа: первые два являются этапами проверки, а последние два являются частью этапа проверки.

Основное различие между ними заключается в том, что тестирование, проводимое на этапах проверки, основано на процессах, используемых во время разработки. Напротив, на этапе проверки проверяется функциональность готового продукта и в конце используются отзывы пользователей.

Кроме того, на первом этапе процедуры тестирования в своих процессах используется тестирование методом «белого ящика». Это означает, что внутренняя структура программного обеспечения не скрыта, и специалисты, проводящие тестирование, должны знать о реализации программного обеспечения.

С другой стороны, при тестировании методом «черного ящика» внутренняя структура программного обеспечения скрыта, и тестировщики используют этот метод на этапе окончательной проверки. Тестировщики должны проверить функциональность приложения на соответствие спецификациям или требованиям, выдвинутым пользователями.

В рамках разрабатываемого проекта тестирование производится по тест-кейсам. Тест-кейс – проверка какой-либо единичной функциональности. Они описывают последовательность действий, направленных на проверку какого-либо функционала, ожидая некий фактический результат.

Функциональное тестирование проекта можно разделить на проверку трех блоков приложения:

* пользовательского интерфейса;
* выполнения CRUD-операций;
* обработка HTTP-запросов.

Вся проверка заключается в формировании тест-кейсов для всех трех блоков вручную. Тестовый набор представлен на таблице 5.1.

Таблица 5.1 – Тестирование системы

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Имя теста | Описание | Шаги | Ожидаемый результат | Статус |
| 1 | 2 | 3 | 4 | 5 |
| Т1 | Успешная аутентификация | Переход на страницу входа, ввод существующих данных, удовлетворяющих валидации, нажатие кнопки Login | Ответ от сервера с кодом 200 (Ok) и подтверждением аутентификации в приложении | ОК |
| Т2 | Ошибка аутентификация | Переход на страницу входа, ввод не- существующих данных, удовлетворяющих валидации, нажатие кнопки Login | Ответ от сервера с кодом 500 (Internal Server Error) и сообщение о не- существующих данных. | ОК |
| Т3 | Успешное создание пациента | Переход на страницу добавления пациента и ввод корректных данных о пациенте, нажатие на Сохранить | Ответ от сервера с кодом 200 (Ok) и подтверждением сохранением в приложении | ОК |
| Т4 | Успешное обновление данных пациента | Переход на страницу добавления пациента , выбираем пациента , нажимаем на кнопку детали, вводим новые данные и нажимаем на кнопку Обновить изменения | Ответ от сервера с кодом 200 (Ok) и подтверждением обновления в приложении | ОК |
| Т5 | Успешное обновление данных пациента | Переход на страницу добавления пациента , выбираем пациента , нажимаем на кнопку детали, вводим новые данные и нажимаем на кнопку Последующие действия. | Ответ от сервера с кодом 200 (Ok) и подтверждением обновления в приложении | ОК |
| Т6 | Успешное обновление данных пациента | Переход на страницу добавления пациента , выбираем пациента , нажимаем на кнопку детали, вводим новые данные и нажимаем на кнопку Последующие действия, заполнить данные и нажать Последующие действия | Ответ от сервера с кодом 200 (Ok) и подтверждением обновления в приложении | ОК |
| Т7 | Удаление пациента | Переход на страницу всех пациентов Удалить. | Ответ от сервера с кодом 200 (Ok) и подтверждением удаления в приложении | ОК |

Продолжение таблицы 5.1

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 1 | 2 | 3 | 4 | 5 |
| Т8 | Успешное добавление менеджера времени. | Переход на страницу Тайм менеджер,  Выбираем дату и записываем события Сохранить. | Ответ от сервера с кодом 200 (Ok) и подтверждением сохранения в приложении | ОК |
| Т9 | Успешная отправка сообщения. | Переход на страницу Mail,вводим данные, нажимаем на кнопку Send. | Ответ от сервера с кодом 200 (Ok) и подтверждением отправки в приложении | ОК |
| Т10 | Успешное создание цены | Переход на страницу Цены , нажимаем на create заполняем форму, нажимаем на add. | Ответ от сервера с кодом 200 (Ok) и подтверждением создания в приложении | ОК |
| Т11 | Успешное удаление цены. | Переход на страницу Цены , выбираем эдемент который хотим удалить, нажимаем на кнопку delete | Ответ от сервера с кодом 200 (Ok) и подтверждением удаления в приложении | ОК |

Как видно из проделанных тест-кейсов, все тесты достигли ожидаемого результата, что говорит о том, что реализованный программный продукт работоспособен и отвечает требованиям, поставленным в спецификации.

# 6 РУКОВОДСТВО ПОЛЬЗОВАТЕЛЯ

Клиентские приложения - веб-приложения, поэтому они не имеют жесткой привязанности к операционной системе, однако рассмотрим некоторые важные системные требования:

* веб-браузер: Yandex, Safari, Safari mobile, Chrome, Chrome mobile, Mozilla Firefox, Mozilla Firefox mobile, Internet Explorer 11+;
* операционная система: Microsoft Windows XP и выше, Apple Mac OS X, Linux;
* рекомендуемая оперативная память устройства: 512 мегабайт и выше.

## 6.1 Руководство пользователя для использования веб-приложением

## 6.1.1 Страница входа

Аутентификация используется для подтверждения личности пользователя, которое заключается в процессе проверки учетных данных. Для входа в систему нужно ввести логин и пароль уже авторизированного пользователя (рисунок 6.1). Если такого, логина или пароля не будет в системе, то система вам выдаст сообщение о некорректности данных (рисунок 6.2)



Рисунок 6.1 – страница входа



Рисунок 6.1 – некорректность данных

## 6.2 Список пациентов

Список пациентов, является одной из главных страниц приложения. В котором хранится вся информация пациентов и может происходить умный поиск. Так же на этой странице можно удалять пациентов и посмотреть детали осмотра.

Рассмотрим каждую возможность по отдельности:

1.Поиск пациентов считается одним из самых важных элементов на странице. Так как благодаря ему происходит поиск личного дела пациента, которое не всегда быстро получается найти медсёстрам в больницах.



Рисунок 6.2.1 – поиск пациентов

2. В каждом личном деле пациента, можно посмотреть содержимое и удалить его.



Рисунок 6.2.2 – личное дело пациента

В личном деле хранятся все записи предыдущих осмотров и заключение врача.

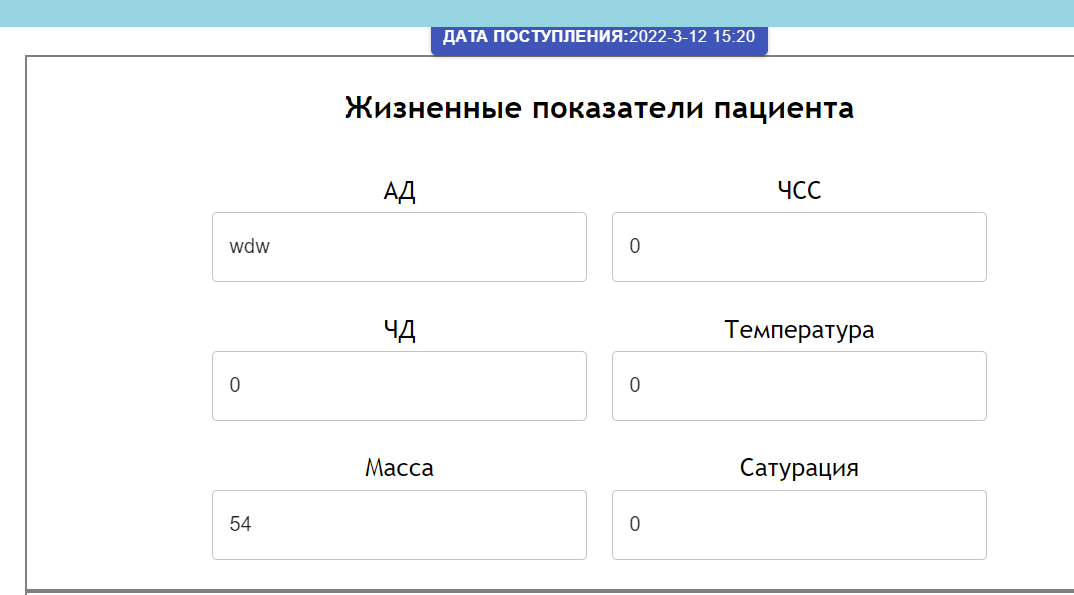


Рисунок 6.2.2 – детали личного дела

Так же можно, посмотреть дату последнего осмотра. Она находиться в правом углу окна и большой плюс, что её не нужно заполнять самостоятельно это делается системой.

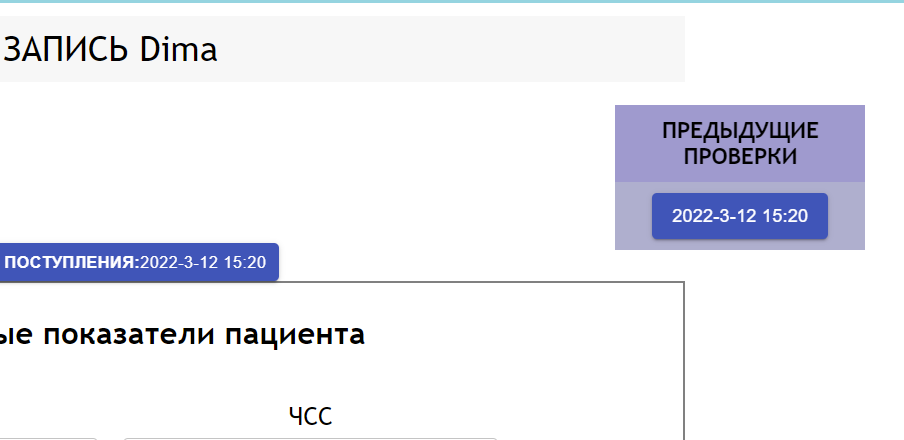


Рисунок 6.2.3 – последняя дата осмотра

После того, как врач ознакомиться с личным делом пациента у него есть возможность, обновить данные после опроса на основе уже существующих данных или создать новую форму.

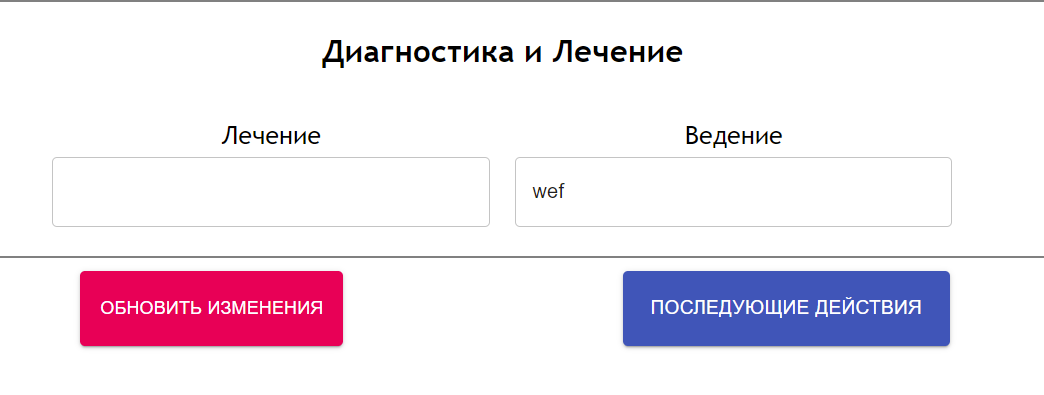


Рисунок 6.2.3 – возможность обновления и создания новой формы

## 6.3 Добавление пациентов

Добавление пациентов – эта страница, предназначенная для добавления пациентов доктором. Доктор заполняет все данные пациента и производит осмотр, все данные осмотра и лечения он сохраняет.

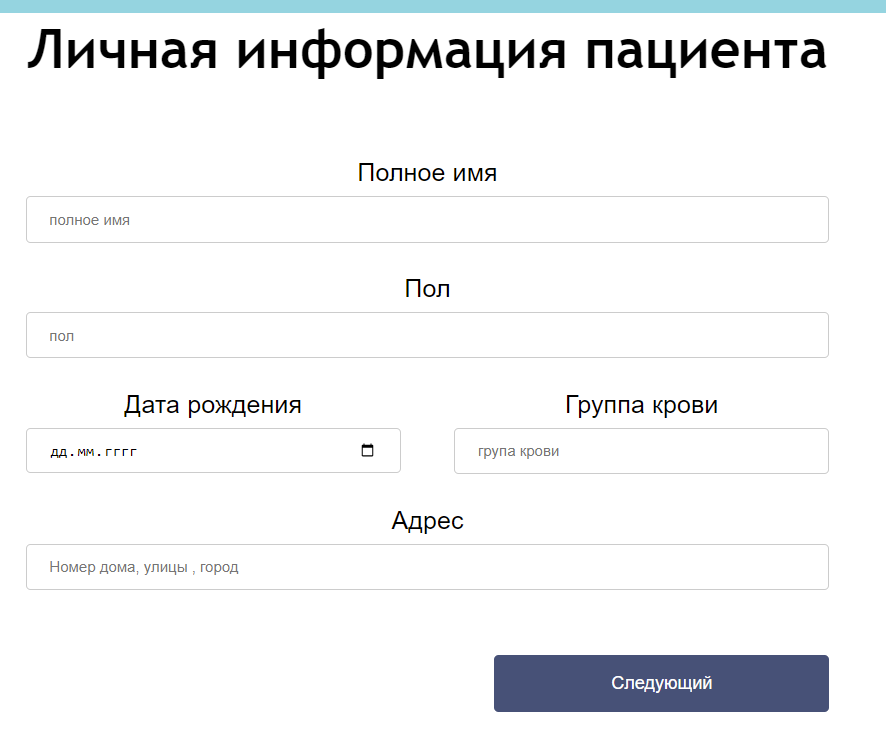


Рисунок 6.3.1 – личная информация пациента

Для заполнения личной информации подобраны элементы благодаря которым можно увеличить работоспособность доктора.

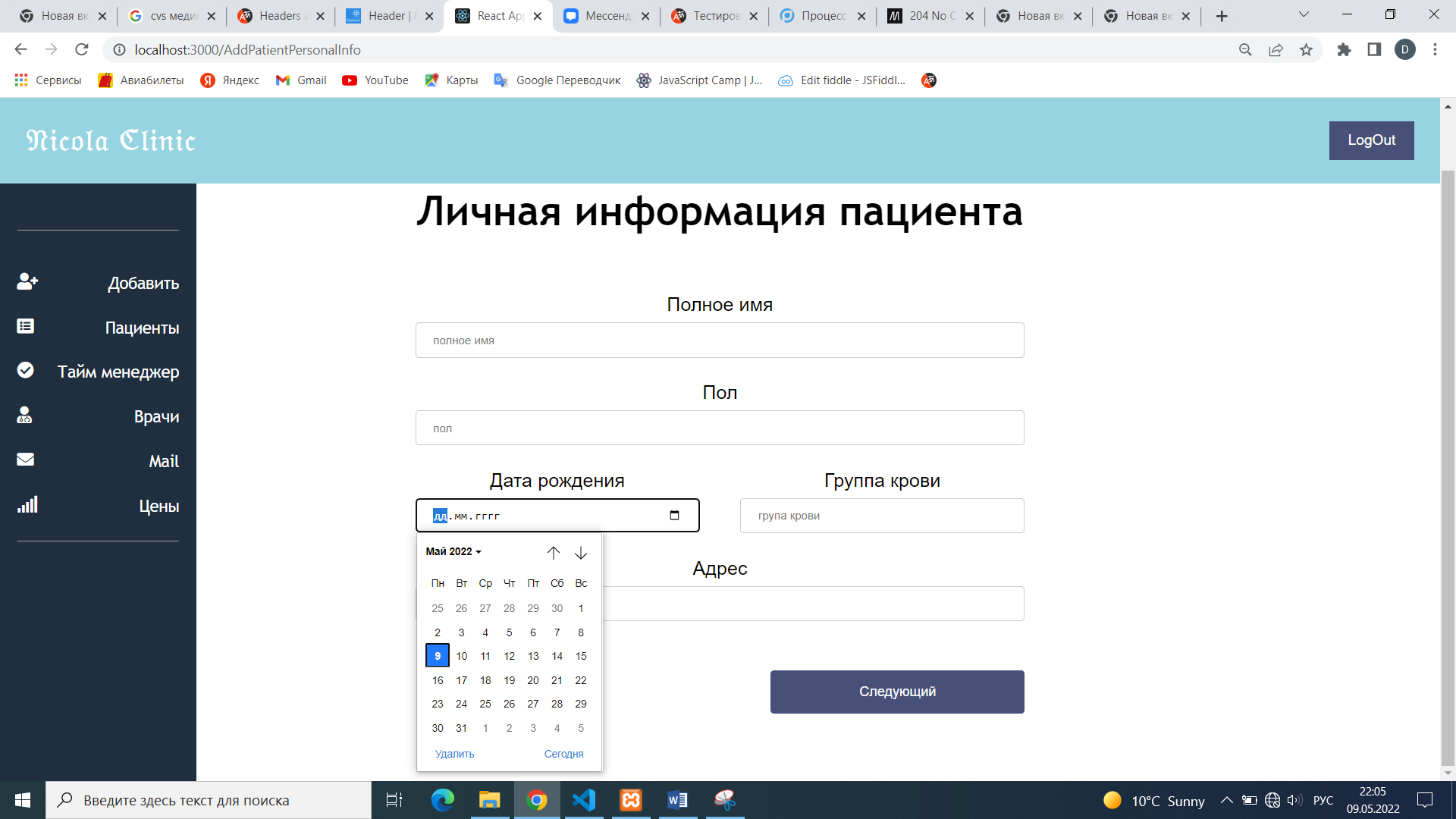


Рисунок 6.3.1 – усовершенствованный календарь

В календаре есть возможность определять текущий день (что будет большим плюсом для некоторых врачей), а также если каким-то образом ввели неправильную дату, то есть возможность удаления.

После заполнения всех данных и нажатия на кнопку Следующий, то мы перейдём на страницу которая соответствует всем нормам осмотра врачей. На которой врач, заполняет данные осмотра, пишет диагноз и подробное лечение. Для автоматизации работы врача был разбит осмотр на 4 категории:

1. Основные данные

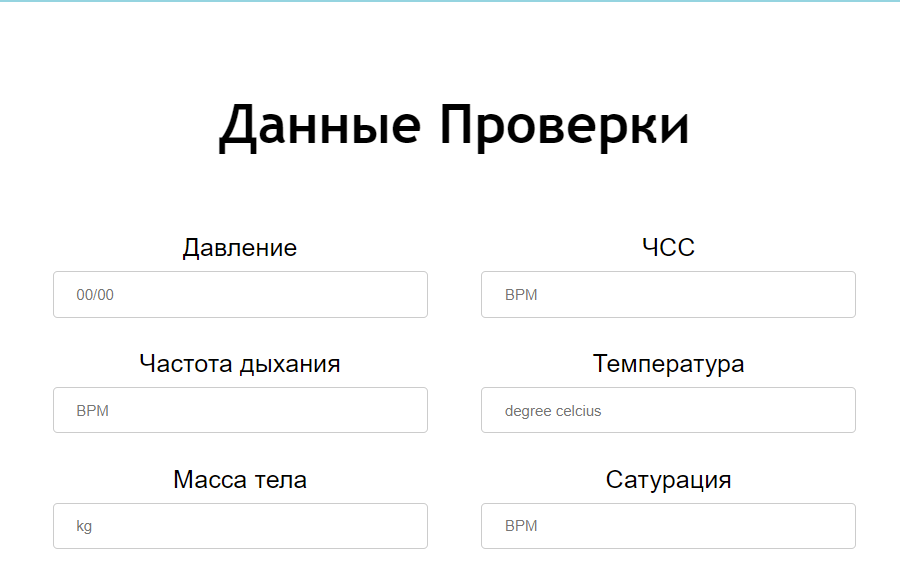


Рисунок 6.3.1 – основные данные

1. Жалоба/История/Внешний вид

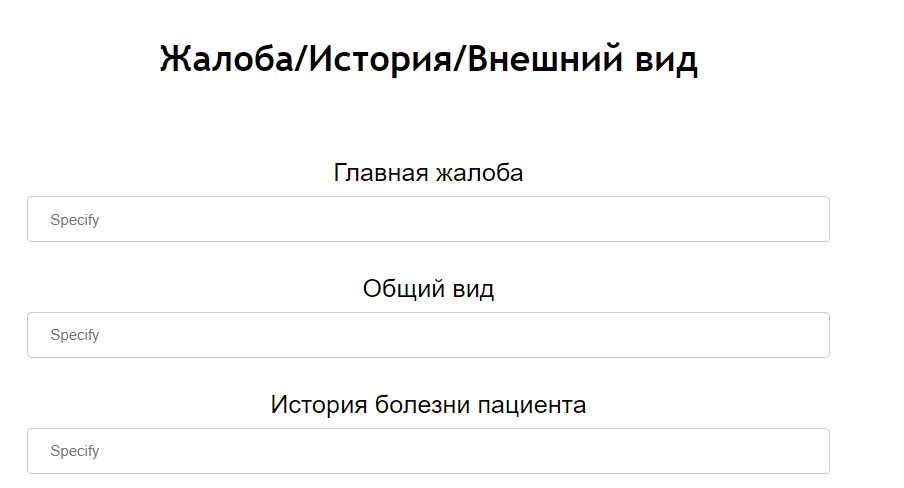


Рисунок 6.3.3 – Жалоба/История/Внешний вид

1. Медицинский осмотр [Да/Нет (Описать)]

Ежегодный медицинский осмотр позволяет вам и вашему врачу оценить, как у вас обстоят дела со здоровьем, независимо от того, чувствуете ли вы симптомы или нет. Это также может помочь вам оценить, какие области вашего здоровья требуют внимания, чтобы в дальнейшем они не вызывали серьезных проблем. Некоторые примеры включают:

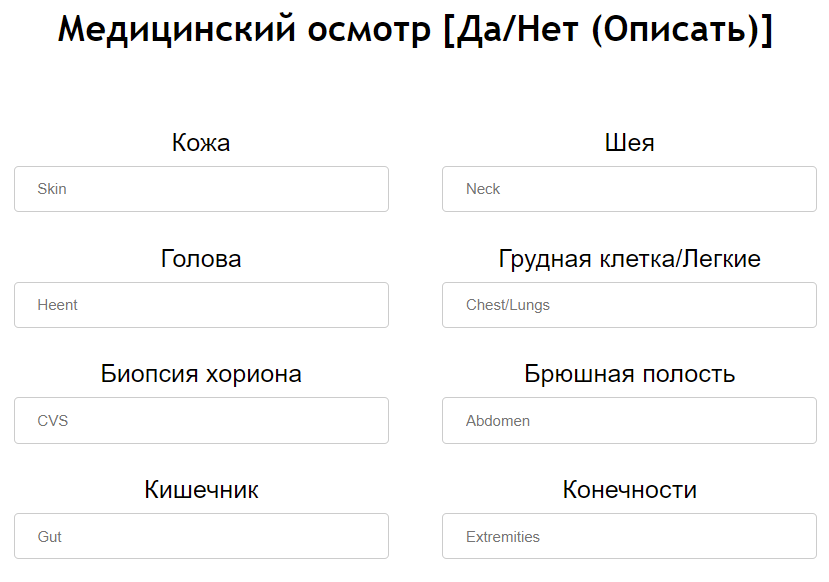


Рисунок 6.3.4 – Медицинский осмотр

1. Постановка диагноза и лечение

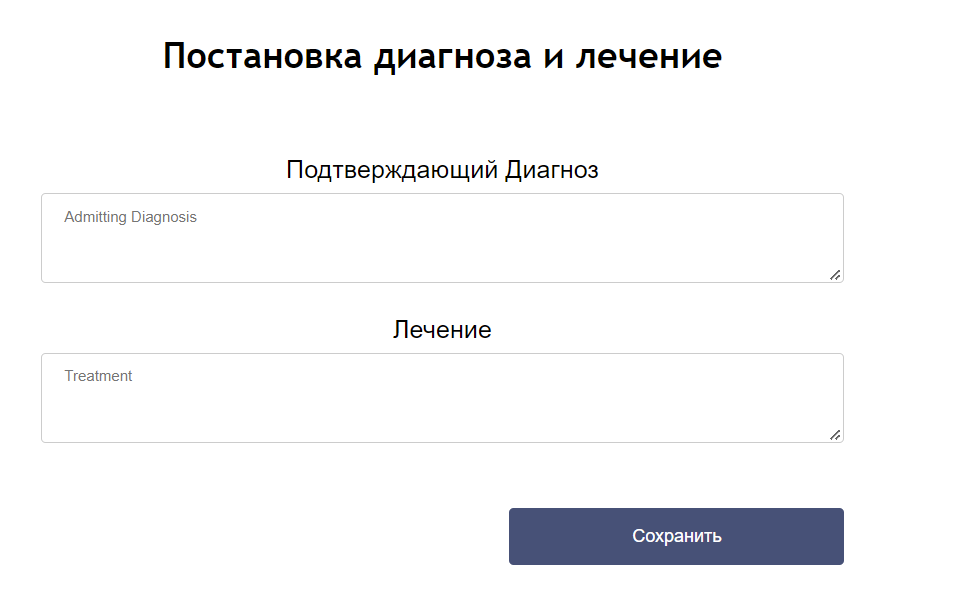


Рисунок 6.3.5 – постановка диагноза и лечение

## 6.4 Тайм менеджер

Тайм менеджер – эта страница, предназначенная для планировки своим временем докторам. Доктор может планировать своё время как на неделю, так и на месяц. Одна из распространенных привычек высокоэффективных профессионалов — часто ставить и пересматривать цели. Эти цели можно разделить на краткосрочные и долгосрочные. Когда вы приходите на работу, может быть сложно взяться за дело, но время, потраченное на чаепитие, блуждание по коридорам или проверку Facebook, потрачено впустую. Запишите конкретные задачи, которые вам нужно выполнять каждый день, и время, к которому вам нужно их выполнить. Проверьте свой список, как только приступите к работе, и разместите его на видном месте. Оставайтесь на правильном пути, часто проверяя свой список и корректируя его по мере необходимости. Вы будете удивлены, как много вы можете сделать, если будете нести ответственность за список дел.

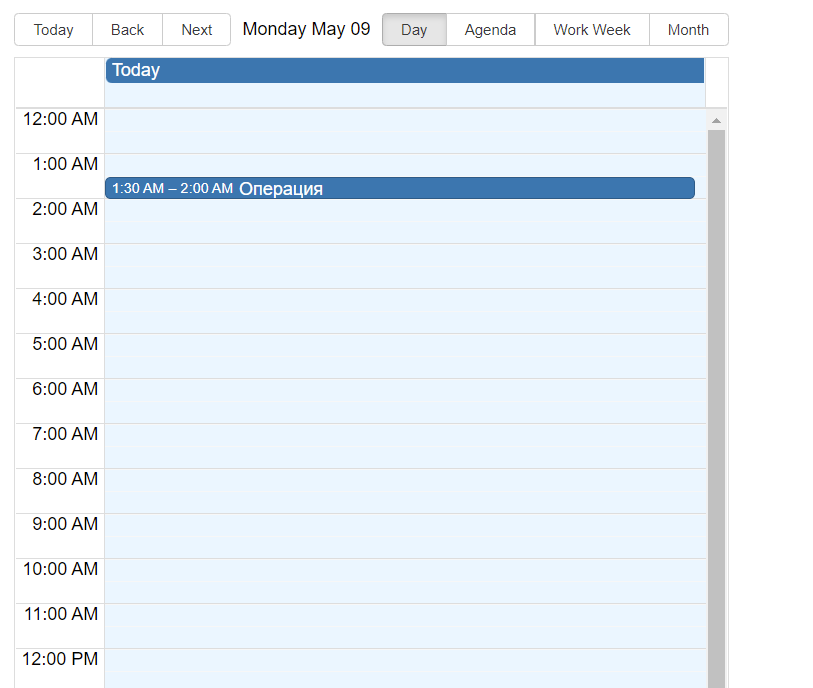


Рисунок 6.3.5 – тайм менеджер

В функционале тайм менеджера есть просмотр текущих, вчерашних и следующего дня задач. Так же мы можем изменять просмотр задач по 4 категориям:

1. По дням (рисунок 6.3.5)
2. По повестке дня

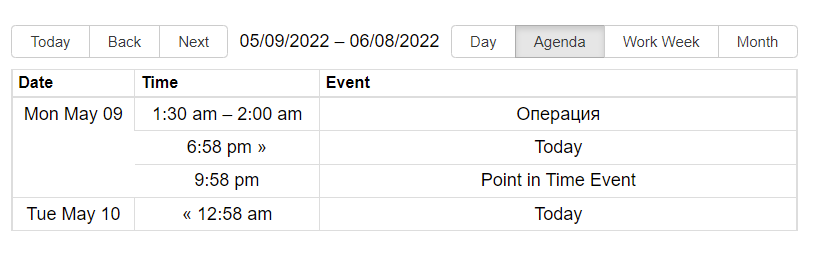


Рисунок 6.3.6 – По повестке дня

1. По рабочей неделе

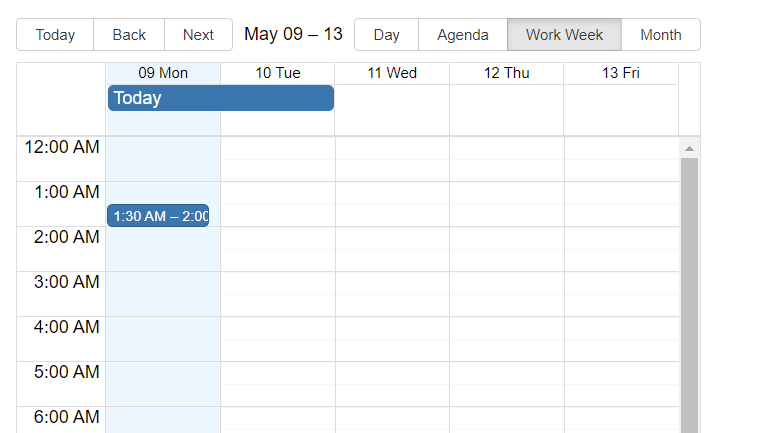


Рисунок 6.3.7 – по рабочей неделе

1. На месяц

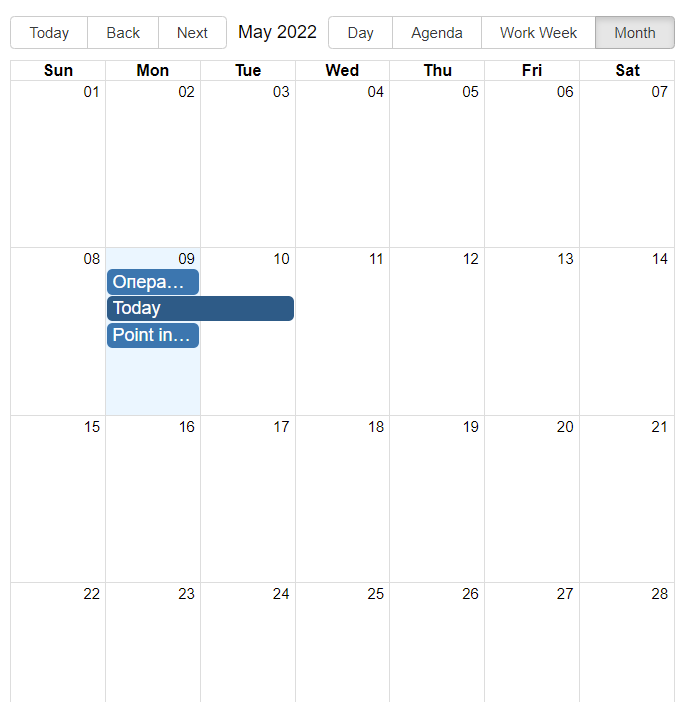


Рисунок 6.3.8 – на месяц

## 6.4 Mail

Mail – эта страница, предназначенная для отправки сообщений доктором, своим пациентам. Это простой и удобный способ поддерживать связь с пациентами. Например, после клиник и исследований обычный сценарий таков, что пациенты ждут (и беспокоятся) в течение нескольких недель, прежде чем они получат свои результаты и информацию о каких-либо последующих действиях или советах. Теперь я сократил это время ожидания до нескольких дней, позволив пациентам отправлять мне текстовые сообщения через три-четыре дня после их исследований. (Мне не нужно брать номера мобильных телефонов пациентов, но они берут карточку с информацией о пациенте, на которой есть и моя.) Затем вы можете отвечать пациентам с их результатами и любым последующим планом действий.

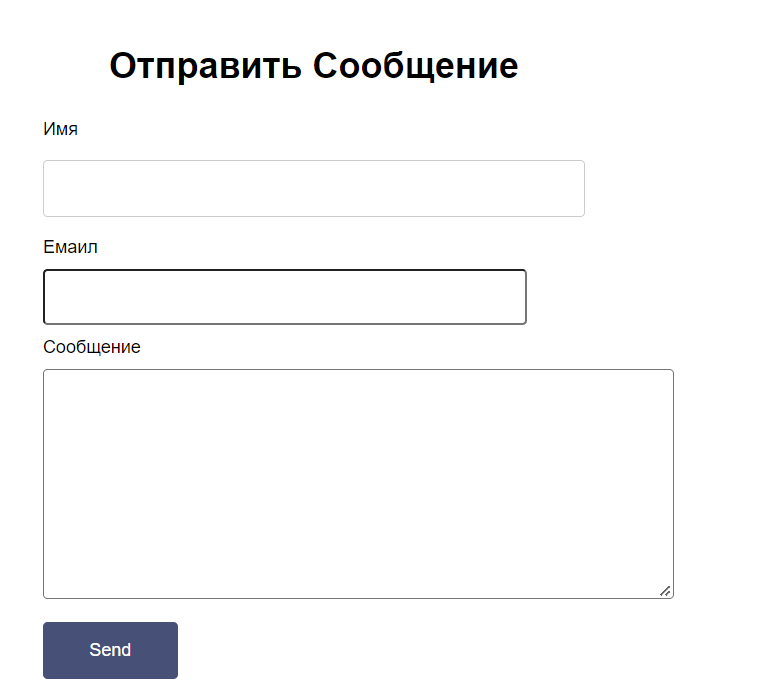


Рисунок 6.4.1 – страница Mail

## 6.5 Список докторов

Список докторов – эта страница, предназначенная для поиск своих коллег по работе. Если вам обратились как к специалисту по поводу болезни сердца или любой другой серьезной проблемы, то любой врач может помочь найти доктора, который соответствует вашим конкретным потребностям.



Рисунок 6.5.1 – Список докторов

## 6.6 Цены

Цены – эта страница, предназначенная для создания, формирования цен за работу доктора. Многие врачи так увлечены своим детищем, что у них нету времени запоминать цены на их услуги. Два недавних исследования показали, что врачи очень мало знают о стоимости. Но это же исследование также подтвердило, что, вопреки распространенному мнению, врачей действительно заботит стоимость их услуг. И для этого было разработана универсальная таблица для записи цен. Рассмотрим основные возможности таблицы

1. Поиск (рисунок 6.6.1)

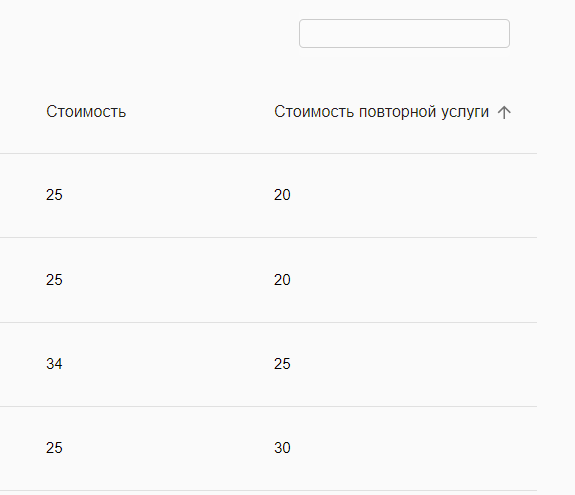


Рисунок 6.6.1 – Поиск

Поиск в таблице происходит по всем категориям таблицы: стоимости, стоимость повторной услуги, категории, возрасту, исполнителю, наименование услуги. В основном - это огромная помощь докторам, которые работают с кучей бумаг, которые порой сложно найти.

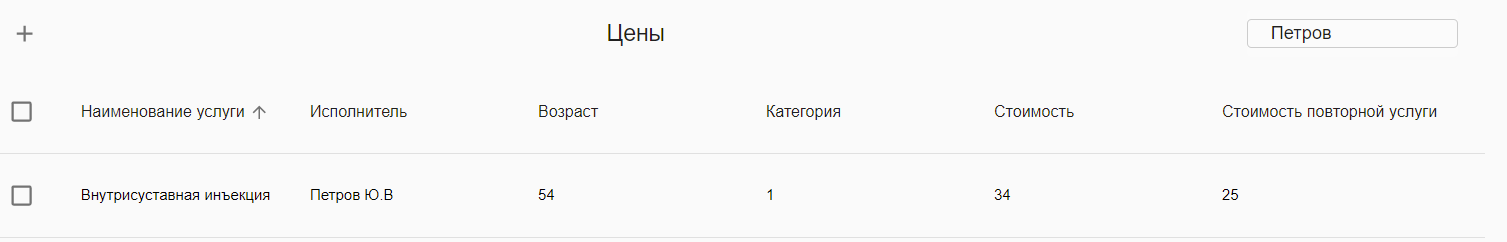


Рисунок 6.6.1 – Поиск по исполнителю

1. Панель выбора строк на странице

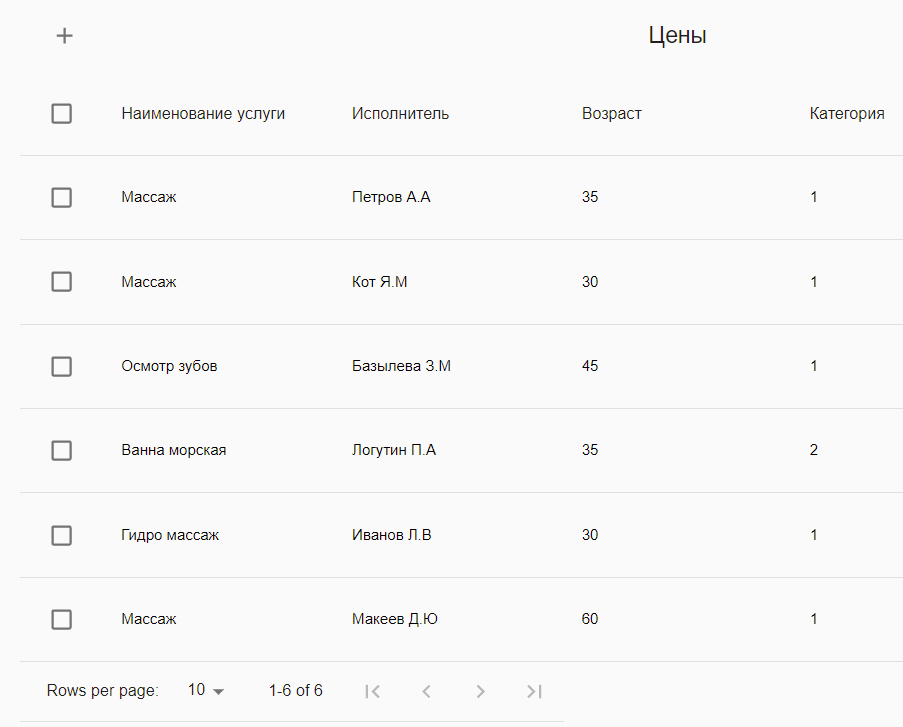


Рисунок 6.6.1 – Панель выбора строк на странице

Эта панель имеет 4 категории контроля строк на странице – это 5, 10, 25, все строки (рисунок 6.6.2)

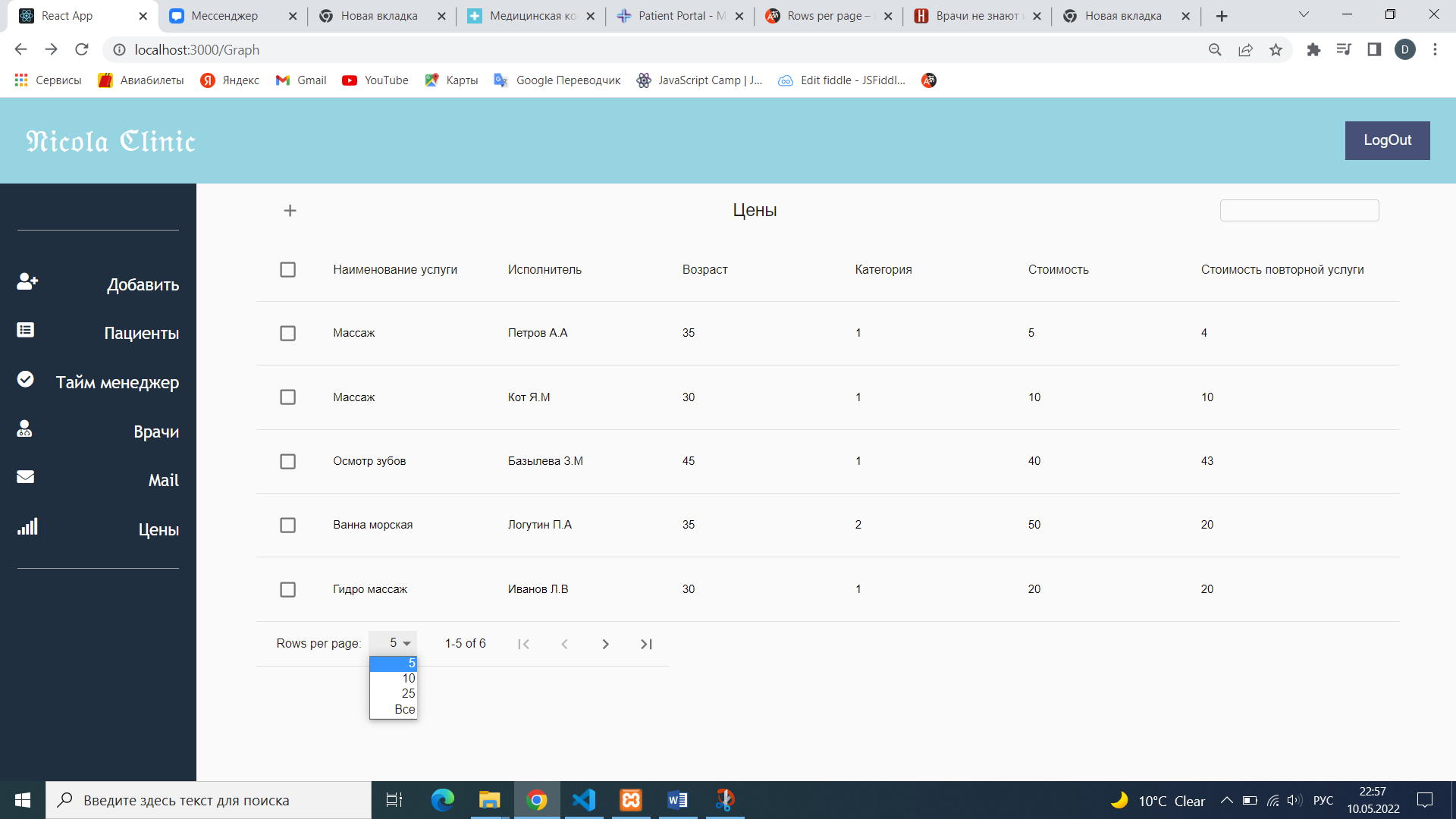


Рисунок 6.6.2 – Панель выбора строк на странице

Так же в панели выбора строк есть быстрый доступ к данным от первого к последнему, а также от одной страницы выбора строк к другой. Что позволяет за короткий промежуток времени производить манипуляции с данными (рисунок 6.6.3)

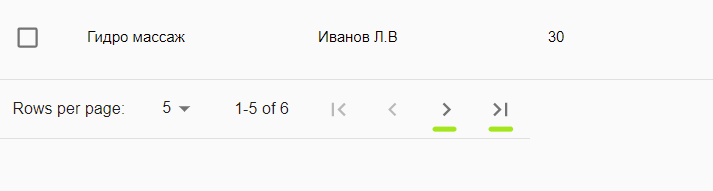


Рисунок 6.6.2 – Панель выбора строк на странице

1. Сортировка

В таблице есть сортировка в каждом заголовке, так как при работе с пациентами важна каждая минута и не знаешь какую из них придётся отсортировать. Сортировка работает по принципу от меньшего большему или от большего к меньшему. А с текстовыми данными по правилу метода sort в JS. (рисунок 6.6.3)

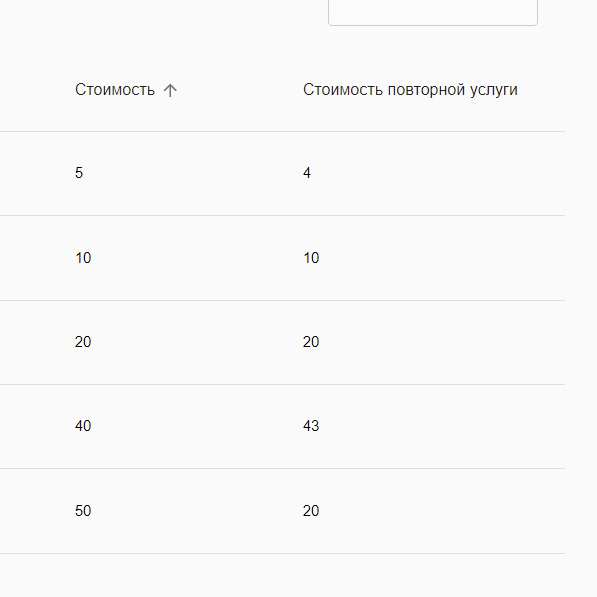


Рисунок 6.6.3 – Сортировка по стоимости

4.Создание цены

Создание происходит путём нажатия на кнопку add на панели таблицы. Затем вам придётся заполнить таблицу данных и сохранить эти данные. Так же в сплывающем модальном окне есть селекторы благодаря которым, мы можем, инкрементировать значение возраста и цены, что вполне будет полезно для пользователя. Так же мы можем отменить все наши вводимые данные и закрыть модальное окно (рисунок 6.6.4)

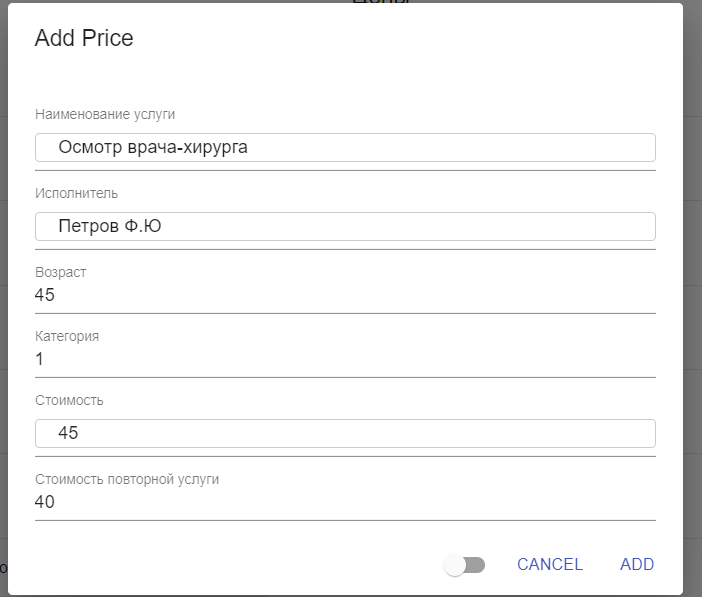


Рисунок 6.6.3 – Модальное окно

1. Выбор и Удаление

Выбор строки происходит путем нажатия на checkbox, затем в вверху панели таблицы загорится строка, на которой будет написано количество выделенных элементов для удаления. При нажатии на иконку “ Корзина” будут удалены все выделенные элементы (рисунок 6.6.4)



Рисунок 6.6.4 – удаление элементов

# ЗАКЛЮЧЕНИЕ

В ходе подготовки к разработке дипломного проекта были изучены теоретические аспекты предметной области, рассмотрены ближайшие аналоги, выявлены их преимущества и недостатки. Выбраны технологии, для написания серверной и клиентской части приложения. Дано краткое описание используемых технологий.

Все заявленные функции в результате были выполнены в полной мере. Проведены тесты, которые выполнились успешно, что подтверждает работоспособность приложения.

Актуальность приложения очень высока, так как интеграция, усовершенствование и автоматизация - это будущие направления развития "современных больниц ". Тем более, в настоящее время создание интегрированной платформы управления может реализовать такие функции, как распределение ресурсов, анализ качества и анализ эффективности, и может снизить медицинские расходы, максимально эффективно использовать ресурсы, и помогать больницам принимать решения относительно развития. Что касается удобства работы с пациентами, то пациенты могут получить доступ к множеству функций, таких как системы физического осмотра, онлайн-записи на прием и взаимодействие между врачом и пациентом.

Таким образом, предусмотренная дипломная работой цель, была достигнута и в соответствии с ней была продемонстрирована востребованность данной работы.

# СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

[1] Веб-сайт компании «ABMcloud» [Электронный ресурс]. – Электронные данные. – Режим доступа: https://abmcloud.com/. – Дата доступа: 02.04.2020.

[2] Веб-сайт программы «CloudShop» [Электронный ресурс]. – Электронные данные. – Режим доступа: https://cloudshop.ru/. – Дата доступа: 02.04.2020.

[3] Веб-сайт программы «FloraPOS» [Электронный ресурс]. – Электронные данные. – Режим доступа: https://florapos.ru/. – Дата доступа: 02.04.2020.

[4] Документация Java [Электронный ресурс]. – Электронные данные. – Режим доступа: <https://docs.oracle.com/en/java/>. – Дата доступа: 02.04.2020.

[5] Современный учебник JavaScript [Электронный ресурс]. – Электронные данные. – Режим доступа: <https://learn.javascript.ru/>. – Дата доступа: 02.04.2020.

[6] Руководство REST API [Электронный ресурс]. – Электронные данные. – Режим доступа: https://restfulapi.net/. – Дата доступа: 02.04.2020.

[7] Документация Spring Framework [Электронный ресурс]. – Электронные данные. – Режим доступа: <https://docs.spring.io/spring/docs/current/spring-framework-reference/index.html>. – Дата доступа: 02.04.2020.

[8] Документация Spring Boot [Электронный ресурс]. – Электронные данные. – Режим доступа: <https://docs.spring.io/spring-boot/docs/current/reference/htmlsingle/>. – Дата доступа: 02.04.2020.

[9] Документация Spring Security [Электронный ресурс]. – Электронные данные. – Режим доступа: <https://docs.spring.io/spring-security/site/docs/current/reference/html5/>. – Дата доступа: 02.04.2020.

[10] Документация React [Электронный ресурс]. – Электронные данные. – Режим доступа: <https://ru.reactjs.org/docs/getting-started.html>. – Дата доступа: 02.04.2020.

[11] Документация Node.js [Электронный ресурс]. – Электронные данные. – Режим доступа: <https://nodejs.org/en/docs/>. – Дата доступа: 02.04.2020.

[12] Документация движка V8 [Электронный ресурс]. – Электронные данные. – Режим доступа: <https://v8.dev/docs>. – Дата доступа: 02.04.2020.

[13] Официальный сайт фреймворка Express [Электронный ресурс]. – Электронные данные. – Режим доступа: <https://expressjs.com/ru/>. – Дата доступа: 02.04.2020.

[14] Официальный сайт PosgreSQL [Электронный ресурс]. – Электронные данные. – Режим доступа: <https://www.postgresql.org/>. – Дата доступа: 02.04.2020.

[15] Документация спецификации JPA [Электронный ресурс]. – Электронные данные. – Режим доступа: <https://docs.spring.io/spring-data/jpa/docs/current/reference/html/#reference>. – Дата доступа: 02.04.2020.

[16] Официальный сайт Hibernate [Электронный ресурс]. – Электронные данные. – Режим доступа: <http://hibernate.org/>. – Дата доступа: 02.04.2020.

[17] Экономическое обоснование проекта по разработке программного обеспечения / В.Г. Горовой, А.В. Грицай, В.А. Пархименко. – Минск: кафедра экономики БГУИР. – 2018. – 12 с.