Федеральное государственное бюджетное образовательное учреждение

высшего образования

«Саратовский государственный технический университет   
имени Гагарина Ю.А.»

Институт прикладных информационных технологий и коммуникаций

Кафедра «Информационно-коммуникационные системы

и программная инженерия»

**КУРСОВАЯ РАБОТА**

по дисциплине

«Современные технологии интернет-программирования»

**ПРОЕКТИРОВАНИЕ МЕДИЦИНСКОЙ ИНФОРМАЦИОННОЙ СИСТЕМЫ «МЕДИЦИНСКАЯ КАРТА»**

Выполнил: студент группы б1-ИВЧТ31

Нуржанов Айшат Бердагалиевич

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

подпись студента

Руководитель:

к.с.н., доцент кафедры ИКСП

Пчелинцева Елена Германовна

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

подпись руководителя

Курсовая работа защищена на оценку

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Саратов 2024

Содержание

[ВВЕДЕНИЕ 3](#_Toc167043654)

[1 Модель взаимодействия с системой 5](#_Toc167043655)

[1.1 Определение ролей системы 5](#_Toc167043656)

[1.2 Способ интеграции в медицинский процесс 7](#_Toc167043657)

[2 Проектирование системы 8](#_Toc167043658)

[2.1 Проектирование базы данных 8](#_Toc167043659)

[2.2 Проектирование серверной части 10](#_Toc167043660)

[2.3 Проектирование клиентской части 11](#_Toc167043661)

[3 Модель взаимодействия с системой 21](#_Toc167043662)

[3.1 Технологии разработки клиента 21](#_Toc167043663)

[3.2 Технологии разработки серверной части 24](#_Toc167043664)

[ЗАКЛЮЧЕНИЕ 26](#_Toc167043665)

[Список используемых источников 27](#_Toc167043666)

# ВВЕДЕНИЕ

В современном мире информационных технологий, где разработка программного обеспечения играет ключевую роль во многих отраслях, вопросы эффективного и качественного контроля данных становятся все более значимыми. Развитие информационных систем контроля данных направлено на ускорение бизнес-процессов и улучшение взаимодействия с клиентами, что вносит важный вклад в современную технологическую эпоху.

Несмотря на широкое распространение существующих информационных систем, появляется потребность в более совершенных и узкоспециализированных решениях, способных эффективно работать в более специфичных отраслях, например, таких как медицина. С увеличением числа людей, связанных с медицинскими учреждениями, возрастает спрос на информационные системы, специально адаптированные под требования медицинской сферы, иными словами, МИС (Медицинская Информационная Система). Бумажный учет больших объемов информации становится все более неудобным, и в этом контексте электронный учет представляется более перспективным и эффективным решением.

Во всех государственных медицинских учреждениях уже интегрирована государственная МИС. В то время как частные клиники вынуждены использовать частные решения.

Медицинские информационные системы предоставляют возможность вести эффективный учет медицинских историй пациентов, быстро находить необходимую информацию. Они обеспечивают врачей, медсестер и административный персонал инструментами для более эффективного взаимодействия с пациентами.

После разработки технического задания подобной системы следует этап проектирования. Для этого этапа необходимо проанализировать выдвигаемые требования, разработать архитектуру приложения и выбрать основные технологии, на которых будут написаны серверная и клиентская части приложения.

Цель курсовой работы: проектирование медицинской информационной системы под названием «Медицинская карта».

Для выполнения поставленной цели необходимо выполнить следующие задачи:

* Детальный анализ и структурирование требований, указанных в техническом задании.
* Проектирование архитектуры МИС «Медицинская карта».
* Выбор основных технологий.
* Проектирование базы данных.
* Проектирование клиентской части.
* Проектирование серверной части.

# 1 Модель взаимодействия с системой

## 1.1 Определение ролей системы

Для реализации функционала разрабатываемой программы необходимо тщательно продумать пользовательские роли и определить их обязанности. На основе анализа технического задания были определены и сформулированы следующие роли: Администратор, Врач, Медрегистратор и Пациент.

* Администратор. Администратор отвечает за управление пользователями, включая создание новых учетных записей, их модификацию и удаление. Администратор назначает и настраивает права доступа, гарантируя, что пользователи имеют доступ только к тем данным и функциям, которые необходимы для выполнения их обязанностей.;
* Врач. Врач обеспечивает введение, обновление и корректировку медицинских данных пациентов. В его обязанности входит запись подробной истории болезни каждого пациента, включая симптомы, диагнозы, результаты анализов и предыдущее лечение. Врач имеет доступ к полной медицинской карте пациента. Врач также назначает лечение и анализы, указывая необходимые процедуры, медикаменты и дальнейшие действия для пациента. Он может записывать результаты текущего лечения и анализов;
* Пациент. Пациент получает доступ к своей личной медицинской информации, включая полную историю болезни, текущие и прошедшие диагнозы, результаты анализов, назначения врачей и проведенные процедуры;
* Медрегистратор: Медрегистратор регистрирует новых пациентов, обновляет сведения пациентов, координирует расписание приемов врачей, обрабатывая запросы на запись от пациентов.

На рисунке 4 изображена диаграмма прецедентов рассматриваемой системы. Администратор, Врач, Медрегистратор и Пациент – это роли системы. Пользователь и Сотрудник используются в качестве обобщения.

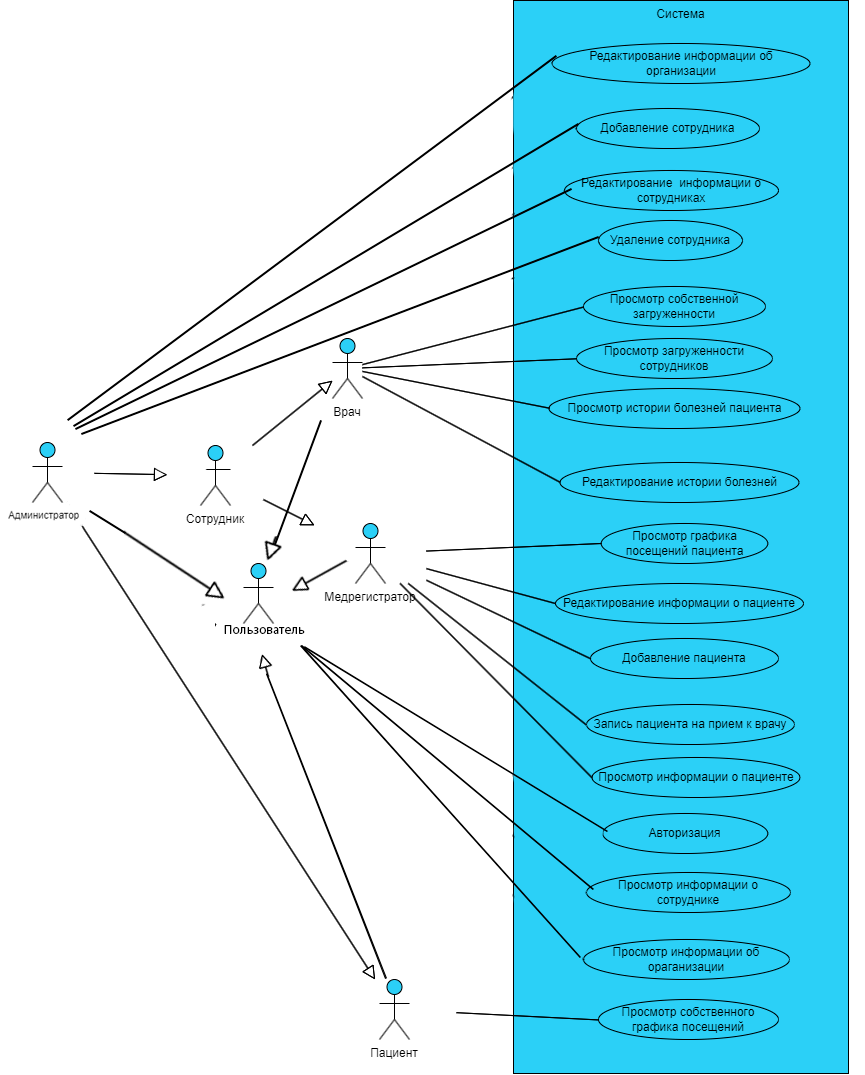


Рисунок 1 – Диаграмма прецедентов

## 1.2 Способ интеграции в медицинский процесс

Система будет интегрирована в медицинский процесс, как собственный сервис: для каждого медицинского учреждения будет создаваться свой домен, на который будет встраиваться клиентская часть системы с индивидуальными данными того или иного медицинского учреждения. Связь с системой будет устанавливаться при помощи индивидуальных номеров, присваиваемых системой автоматически. После регистрации в системе пациентам будет автоматически открыт доступ к их медицинским картам, где будут храниться история болезней, результаты обследований, рецепты и другая медицинская информация. Добавление сотрудников в систему осуществляется администратором. Администратором должен являть главный врач медицинского учреждения, либо любое другое уполномоченное лицо.

# 2 Проектирование системы

Так как МИС «Медицинская карта», исходя из анализа технического задания, подразумевается, как веб-приложение, то логичнее всего использовать клиент-серверную архитектуру. На клиенте реализуется доступный для пользователей интерфейс приложения, на сервере – функционал системы, модели, роли и их права доступа, операции с базой данной

## 2.1 Проектирование базы данных

На рисунке 2 изображена ER-диаграмма базы данных рассматриваемой системы. В ней содержатся следующие сущности:

* Организации (Organizations). В этой сущности содержится информация об организации: название (org\_name), адрес (address), контактный номер (phone\_number) и описание организации (org\_description). В поле org\_description описан профиль медицинского учреждения, например, стоматология. org\_id – первичный ключ;
* Пользователи (Users). Эта сущность содержит аутентификационные данные: ФИО (поля last\_name, first\_name и patronymic соответственно). По внешнему ключу org\_id связана с сущностью Организации. user\_id – первичный ключ;
* Администраторы (Admins), Врачи (Doctors), Пациенты (Patients) и Медрегистраторы (Medregistrators). Это сущности, характеризуют роль пользователя в системе. Связаны по внешнему ключу user\_id с сущностью Пользователи;
* Администратор и Медрегистратор хранят только первичный (admin\_id, medreg\_id соответственно) и внешний ключи;
* Пациенты содержат поля: дата рождения (birthdate), контактный номер (phone\_number), адрес электронной почты (email), адрес проживания (address), пол (gender);
* Врачи содержат поля: специализация (specialization), ссылка на облачное хранилище, где хранятся документы об образовании (documents\_link), стаж работы (work\_experience);
* События. Эта сущность используется для формирования календаря событий. Содержит такие поля: внешние ключи на сущности Врачи и Пациенты (doctor\_id и patient\_id соответственно), дата события (event\_date), ссылка на облачное хранилище с прикреплёнными документами (documents\_link), тип события (event\_type), статус события (event\_status), описание события (event\_description).

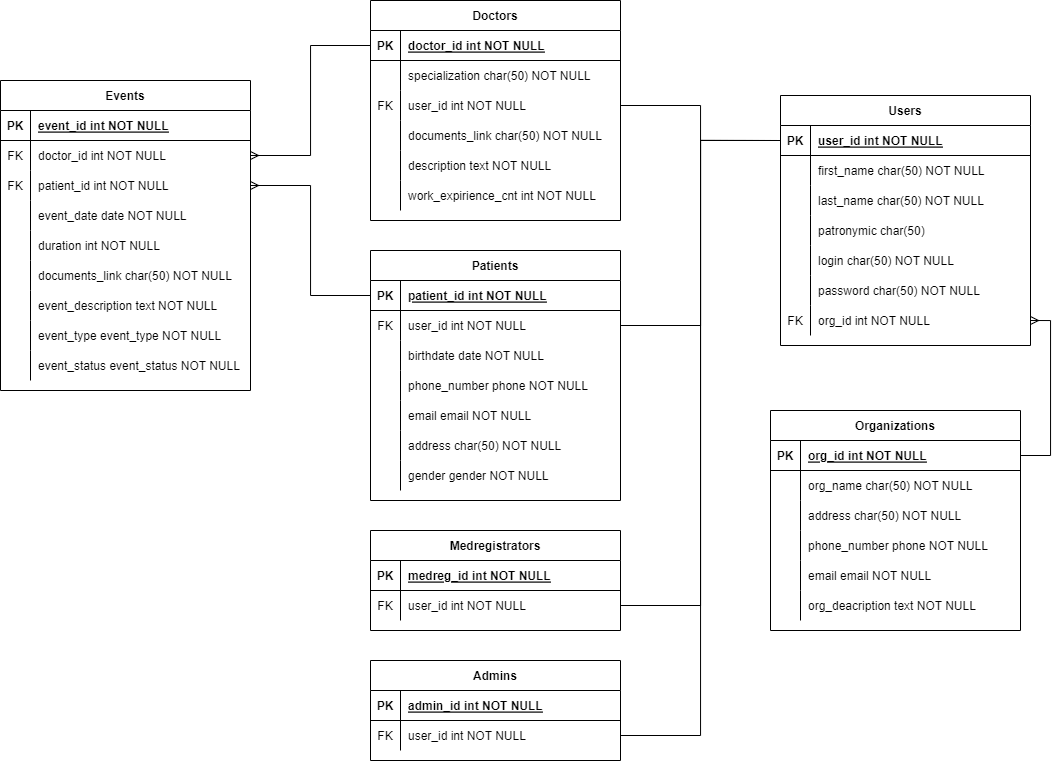


Рисунок 2 – ER-диаграммма

## 2.2 Проектирование серверной части

Проектирование серверной части заключается в продумывании моделей сущностей системы, установка правил их взаимодействия и связи, а также определение функций, которые будет выполнять серверная часть разрабатываемой системы. В ходе проектирования клиентской части был определен примерный функционал системы, исходя из которого была нарисована диаграмма классов.

На рисунке 3 изображена диаграмма классов разрабатываемого приложения

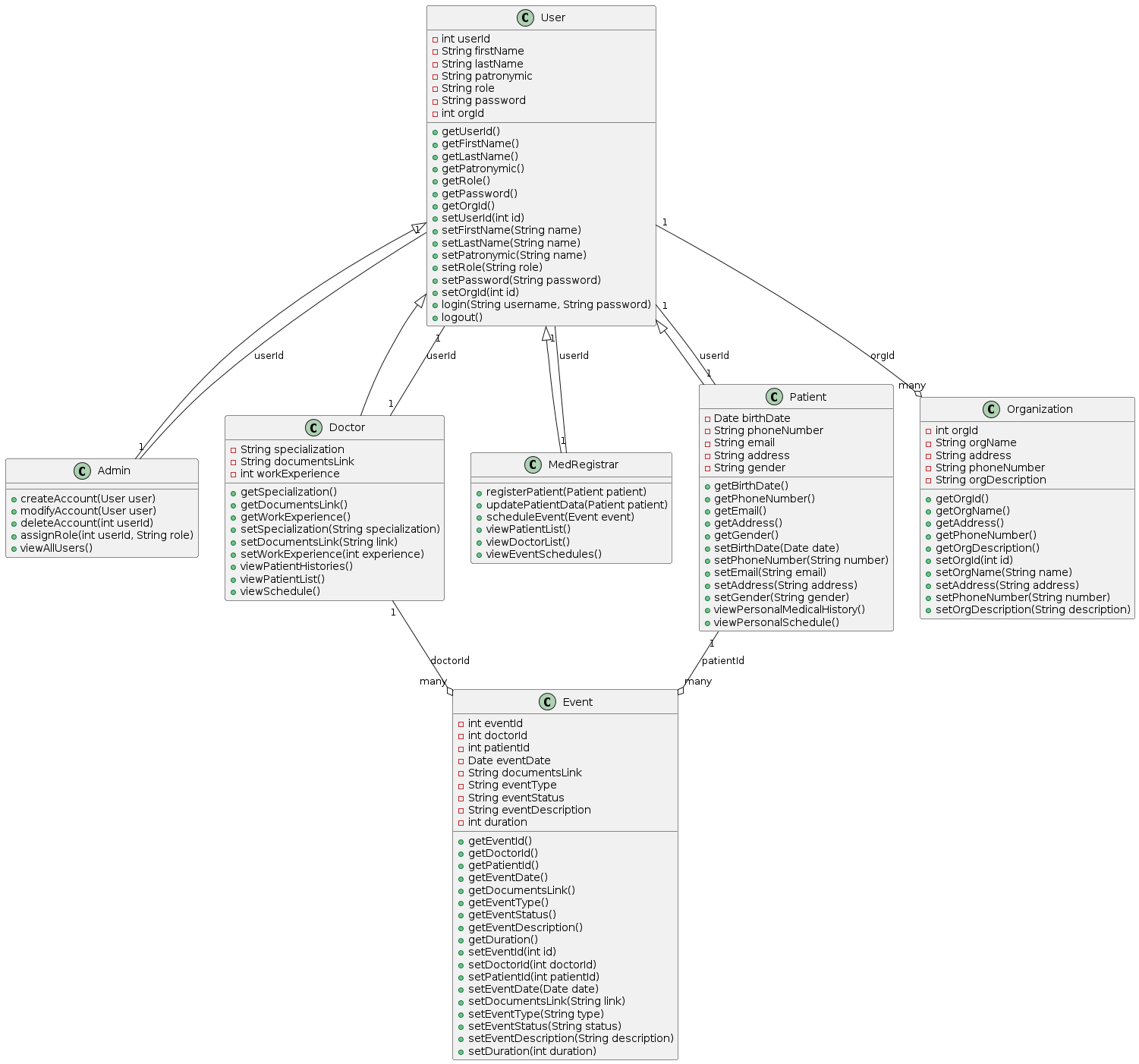


Рисунок 3 – Диаграмма классов

Помимо взаимодействия сущностей, необходимо, чтобы серверная часть приложения выполняла следующие функции:

* Общие функции: любой пользователь может войти в систему и выйти из нее.
* Функции администратора: добавление новых пользователей в систему createAccount(User user), изменение данных о пользователе modifyAccount(User user), удаление пользователей deleteAccount(int userId), назначение роли пользователям assignRole(int userId, String role), просмотр списка пользователей viewAllUsers(), а также функции всех остальных ролей.
* Функции врача: просмотр медицинской истории пациентов viewPatientHistories(), просмотр списка пациентов viewPatientList(), просмотр своего рабочего расписания viewSchedule().
* Функции медрегистратора: регистрация новых пациентов registerPatient(Patient patient), обновление данных пациентов updatePatientData(Patient patient), записывание пациента на прием scheduleEvent(Event event), просмотр списка всех пациентов viewPatientList(), просмотр списка всех врачей viewDoctorList(), просмотр загруженности врачей viewAppointmentSchedules().
* Функции пациента: просмотр собственной медицинской истории viewPersonalMedicalHistory(), просмотр графика посещений врачей viewPersonalSchedule().

## 2.3 Проектирование клиентской части

Проектирование клиентской части МИС «Медицинская карта» включает в себя определение пользовательского интерфейса (UI) и доступа к функциональности, необходимой для взаимодействия пользователей с системой. Взаимодействие пользователей с системой будет происходить через веб-приложение, к которому будет подключена МИС «Медицинская карта». Для лучшего представления устройства структуры сайт были разработаны макеты страниц веб-приложения:

После запуска веб-приложения неавторизованному пользователю отображается главная страница, прототип которой изображен на рисунке.

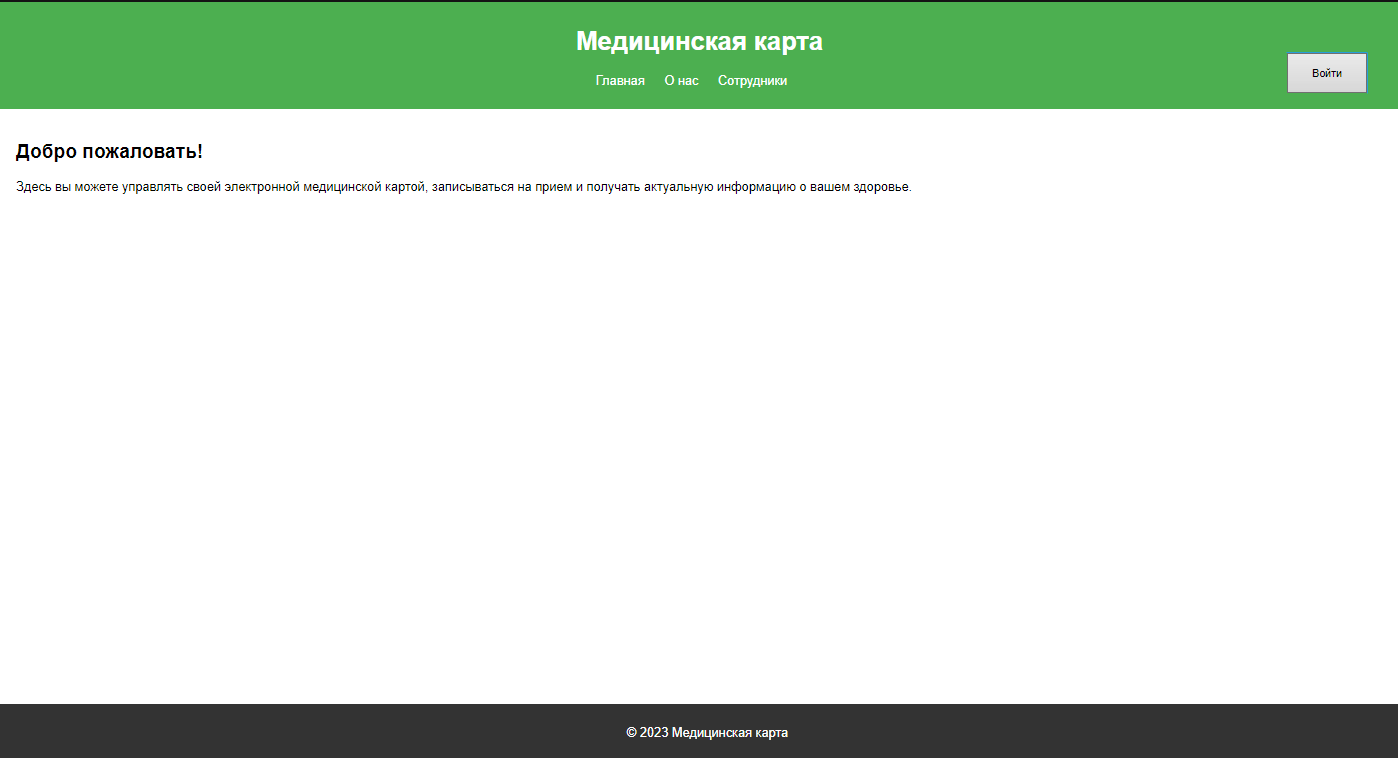


Рисунок 4- главная страница сайта

На главной странице размещены ссылки, перейдя по которым можно узнать:

* Соответствующую информацию о сотрудниках медицинского учреждения (рис. 5);
* Соответствующую информацию о медицинском учреждении (рис. 6)

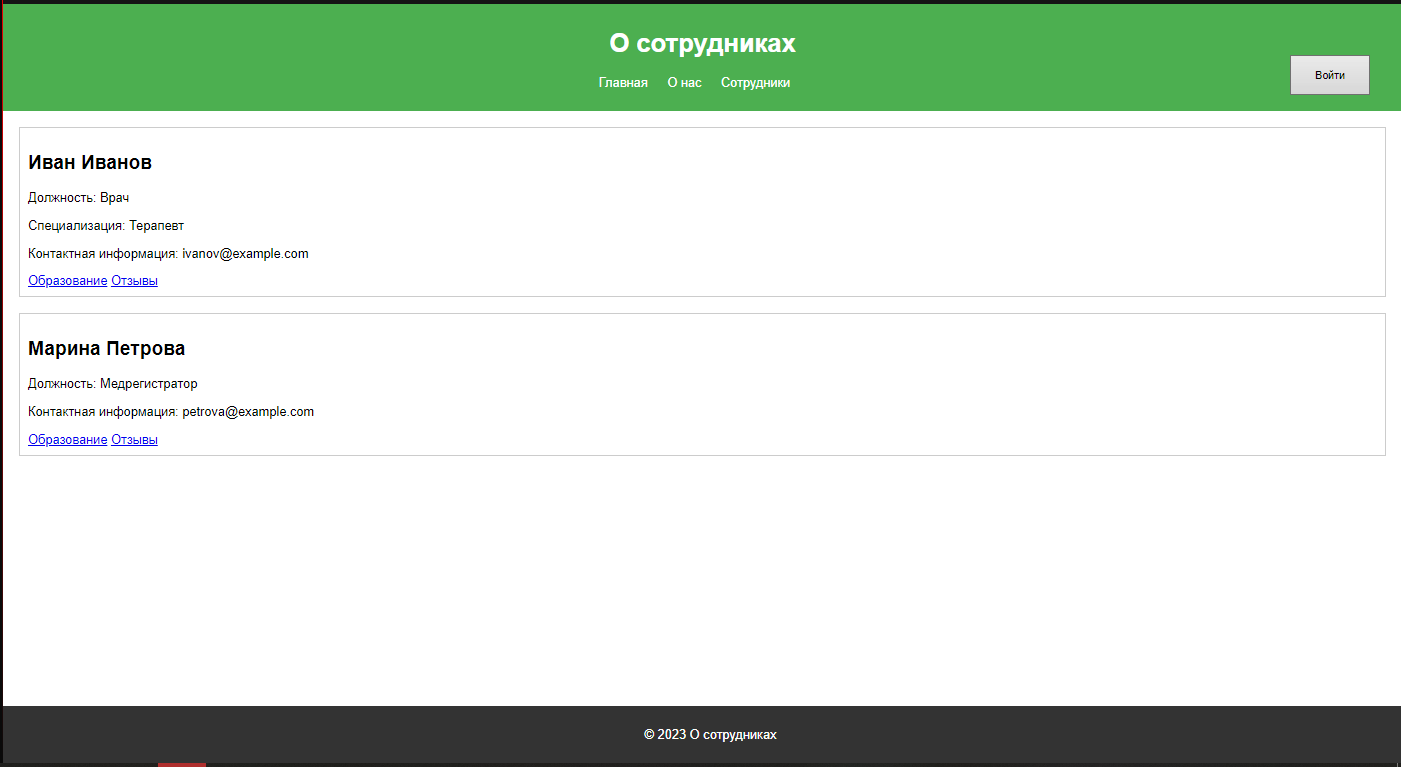


Рисунок 5 – Страница «О сотрудниках»

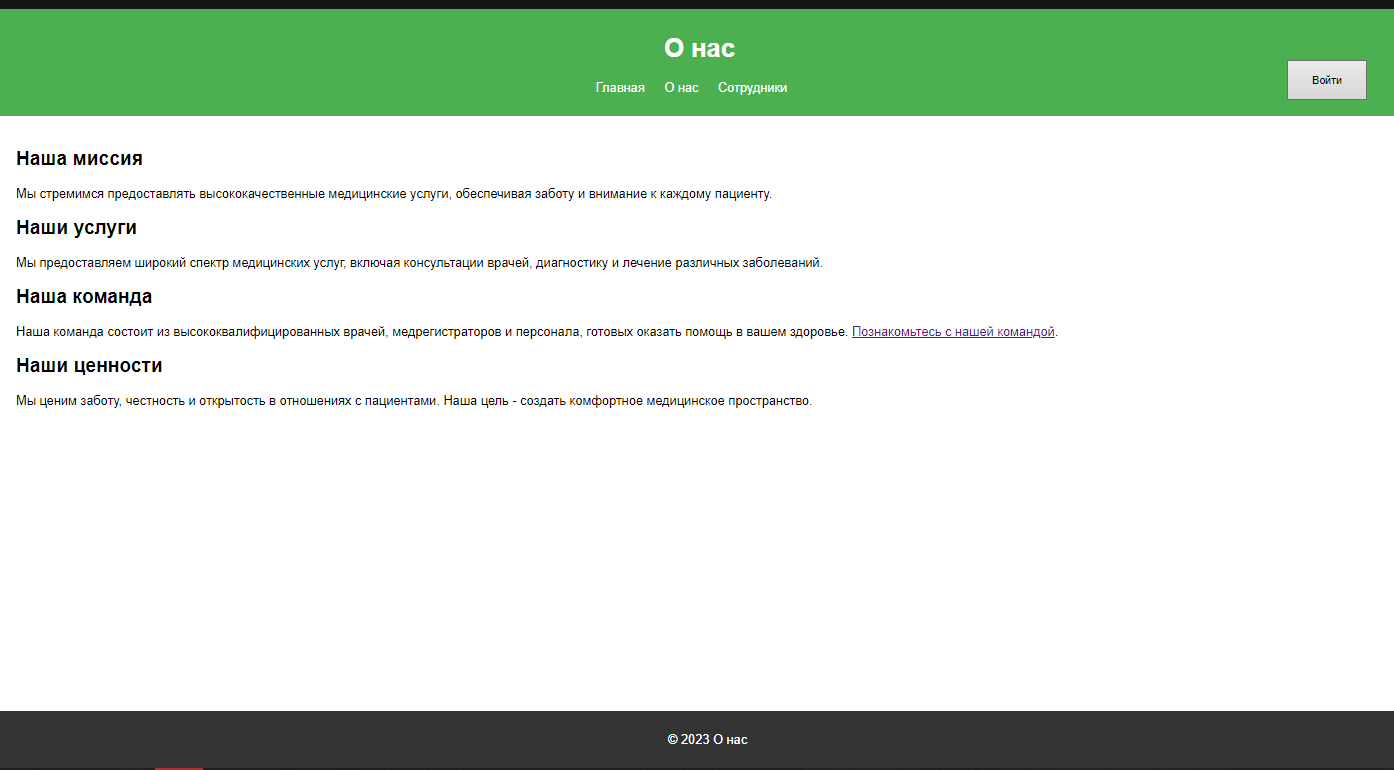


Рисунок 6 – Страница «О нас»

Также из главной страницы можно перейти на страницу авторизации (рис. 7)

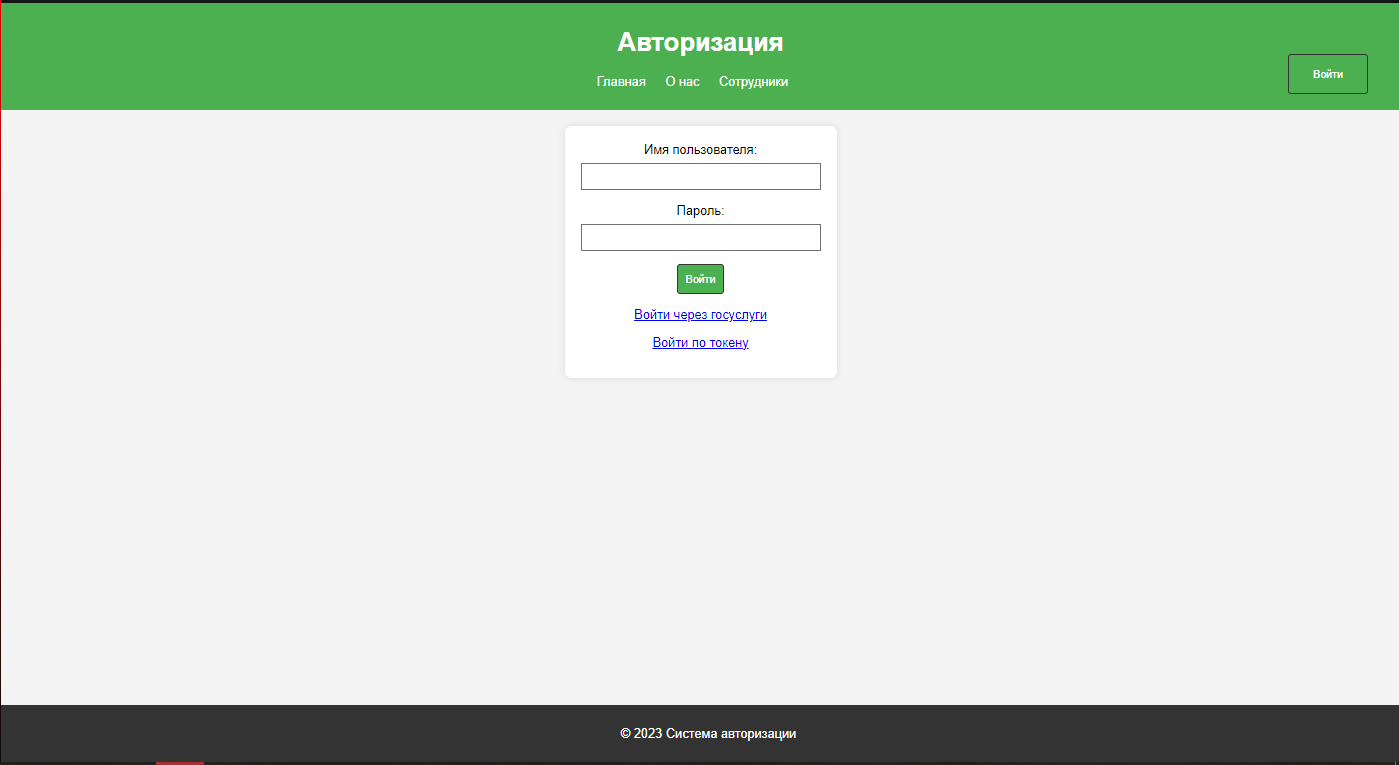


Рисунок 7- Страница авторизации

После успешной авторизации открывается личный кабинет. Для каждой роли отображается разная информация (рис. 8-11):

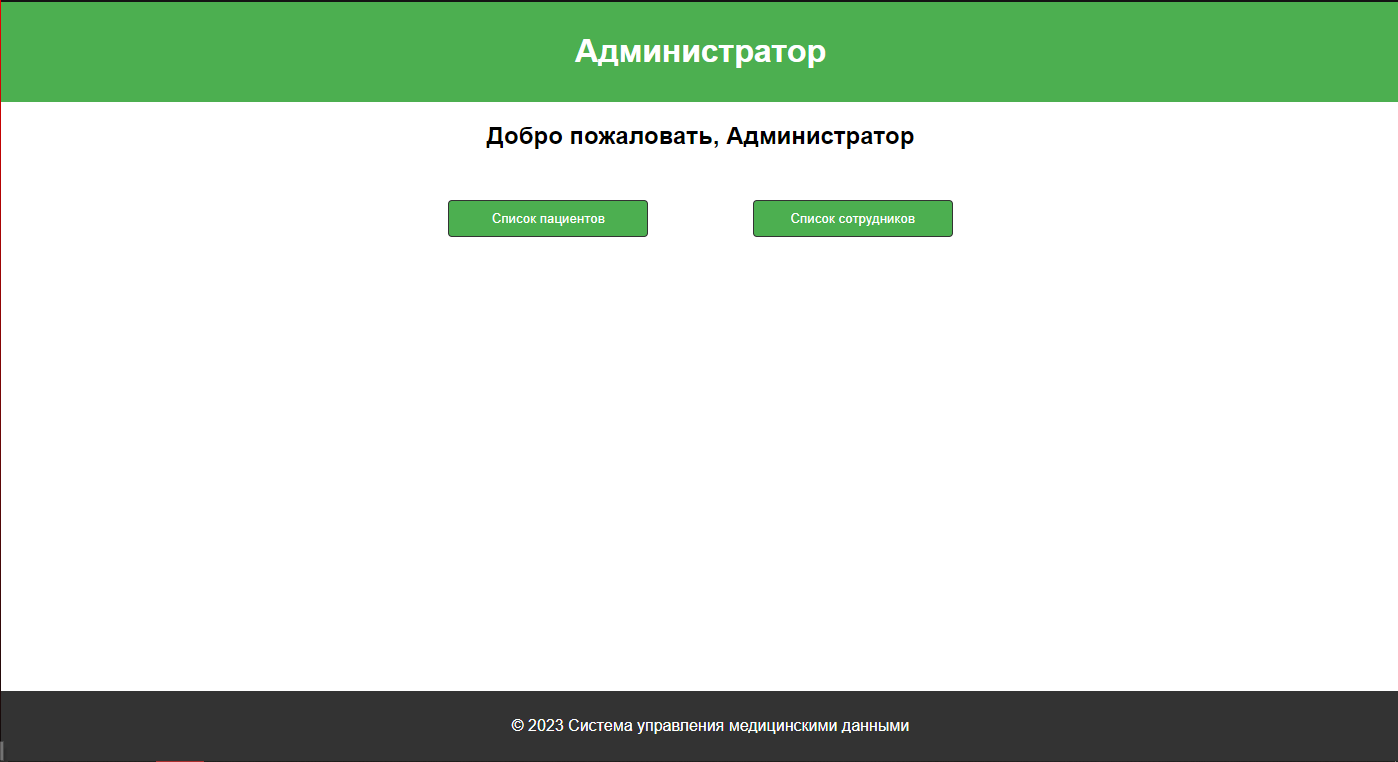


Рисунок 8

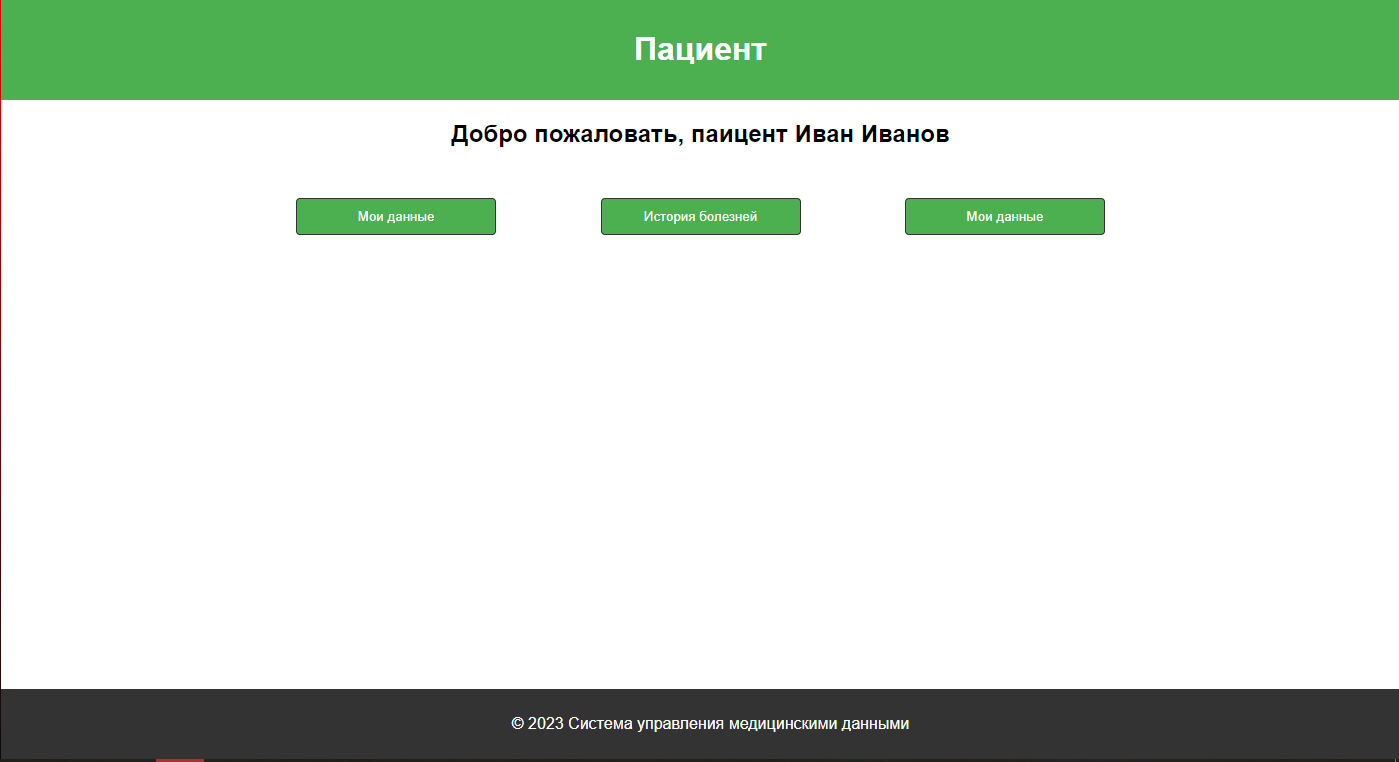


Рисунок 9

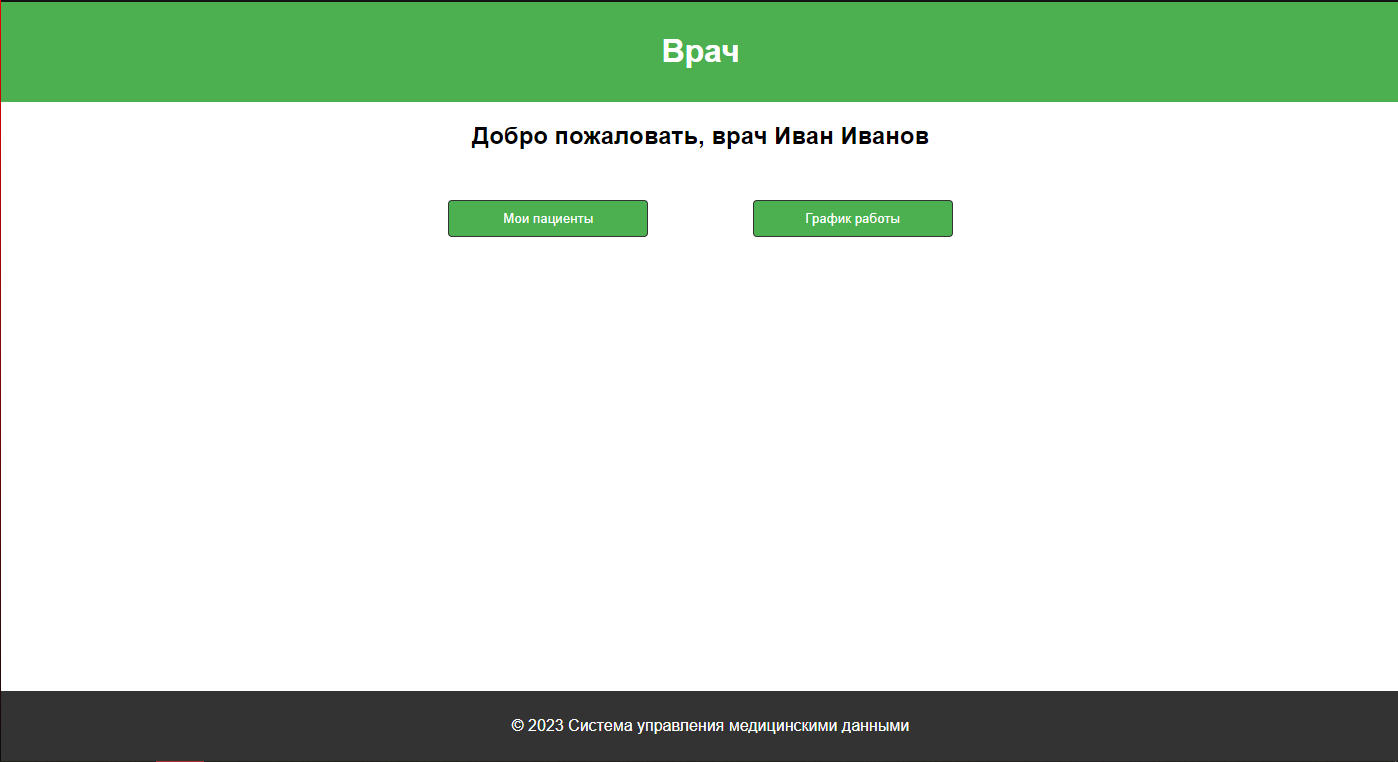


Рисунок 10

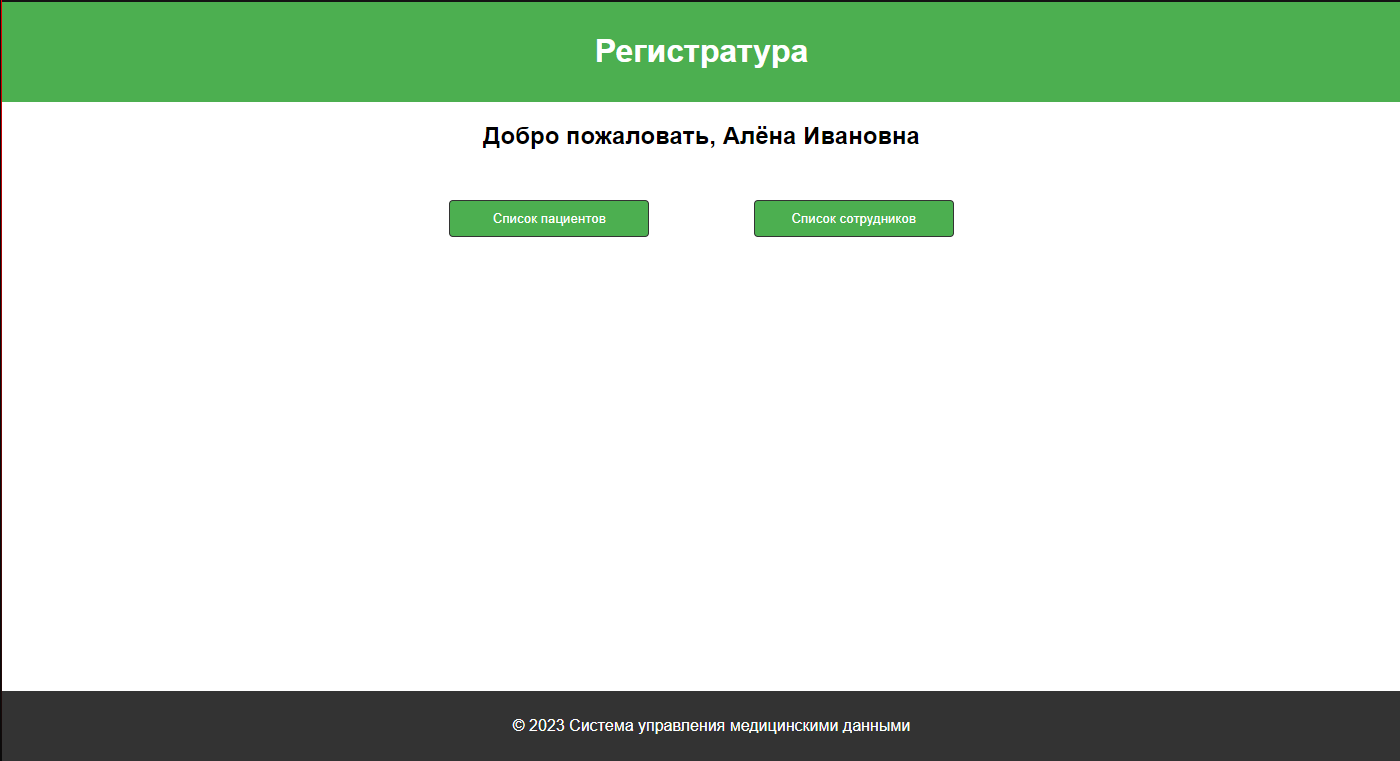


Рисунок 11

Далее будут представлены макеты страниц отдельных функций системы (рис. 12-20):

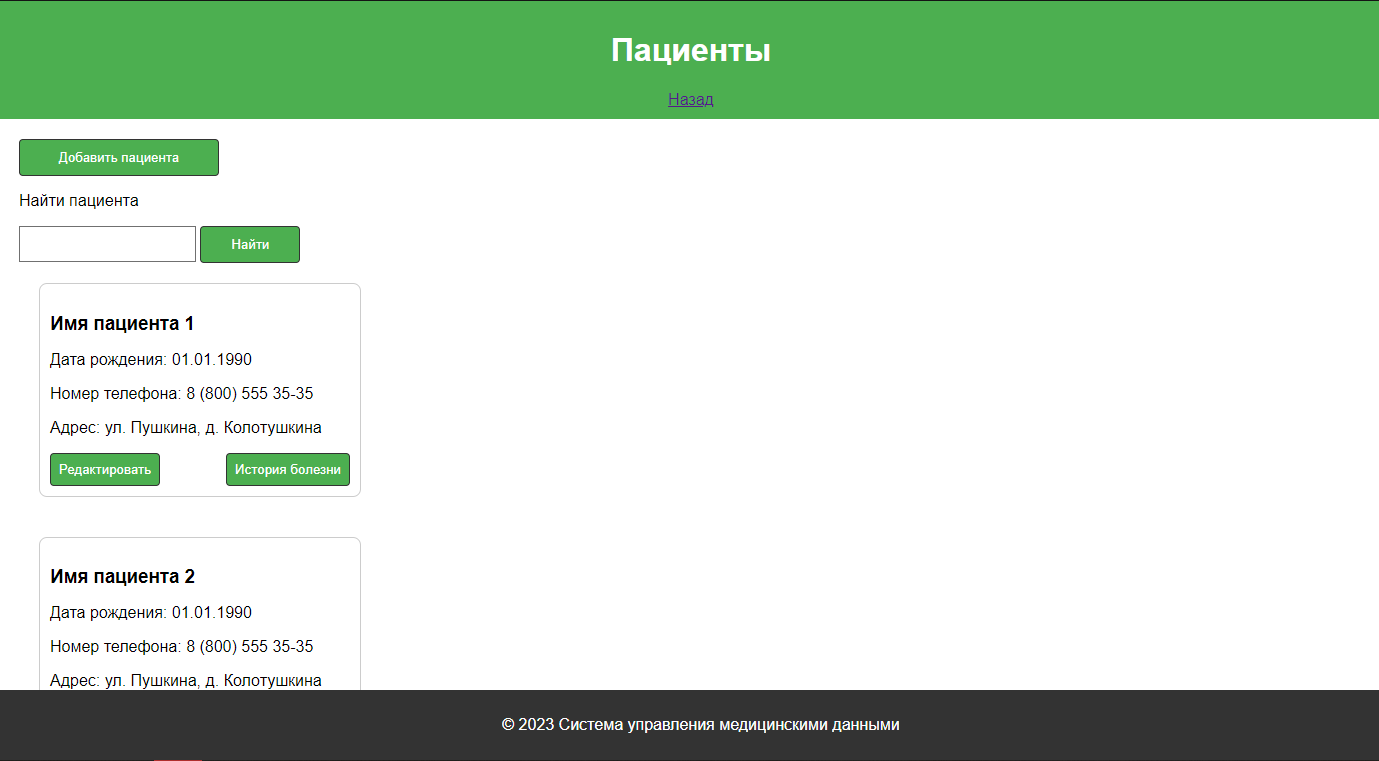


Рисунок 12 – Страница «Список пациентов»

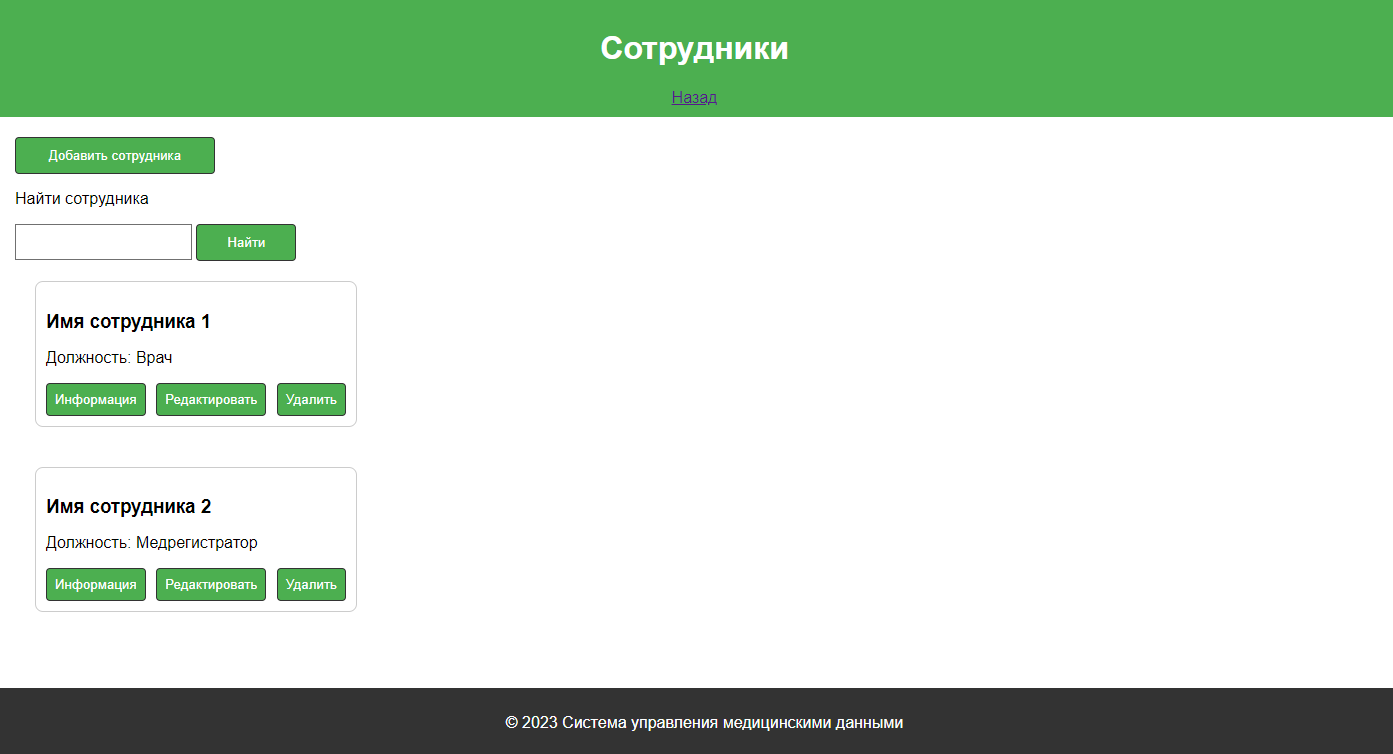


Рисунок 13 – Страница «Список сотрудников»

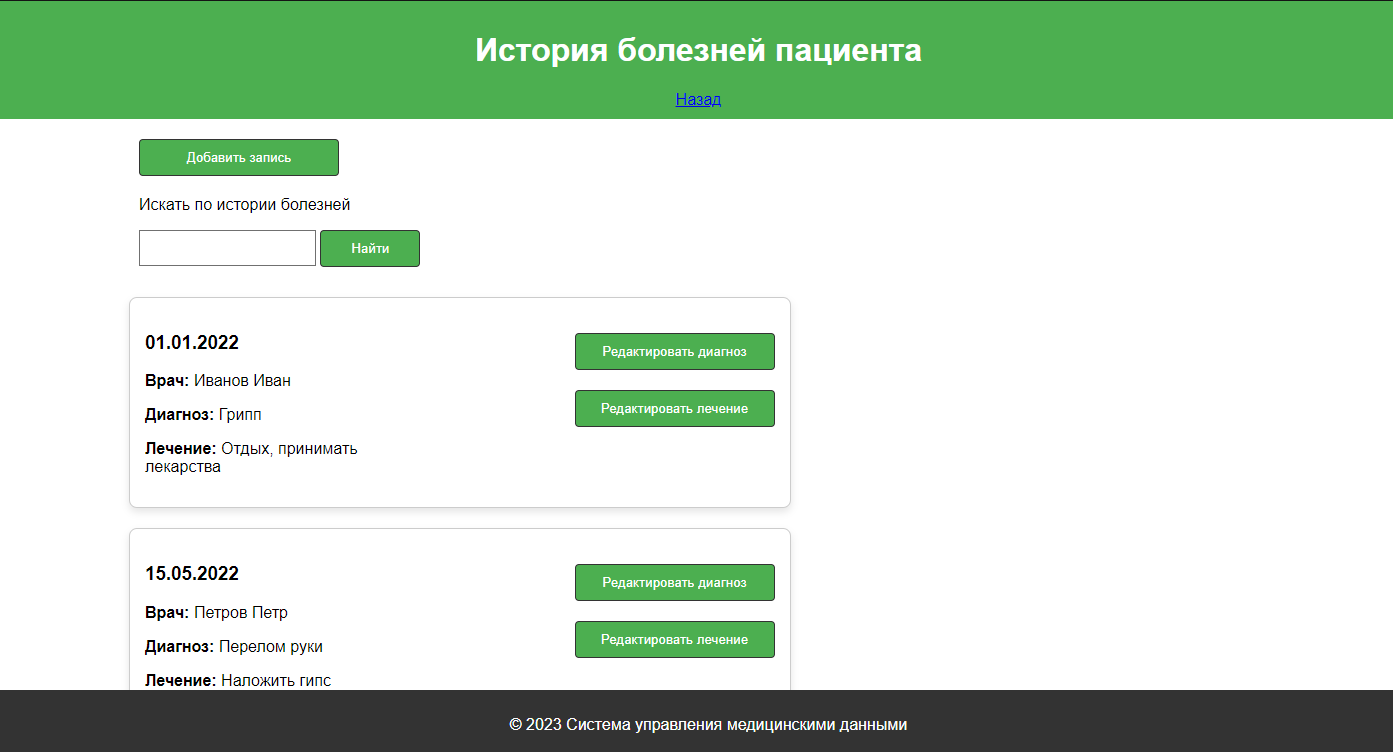


Рисунок 14 – Страница «История болезней»

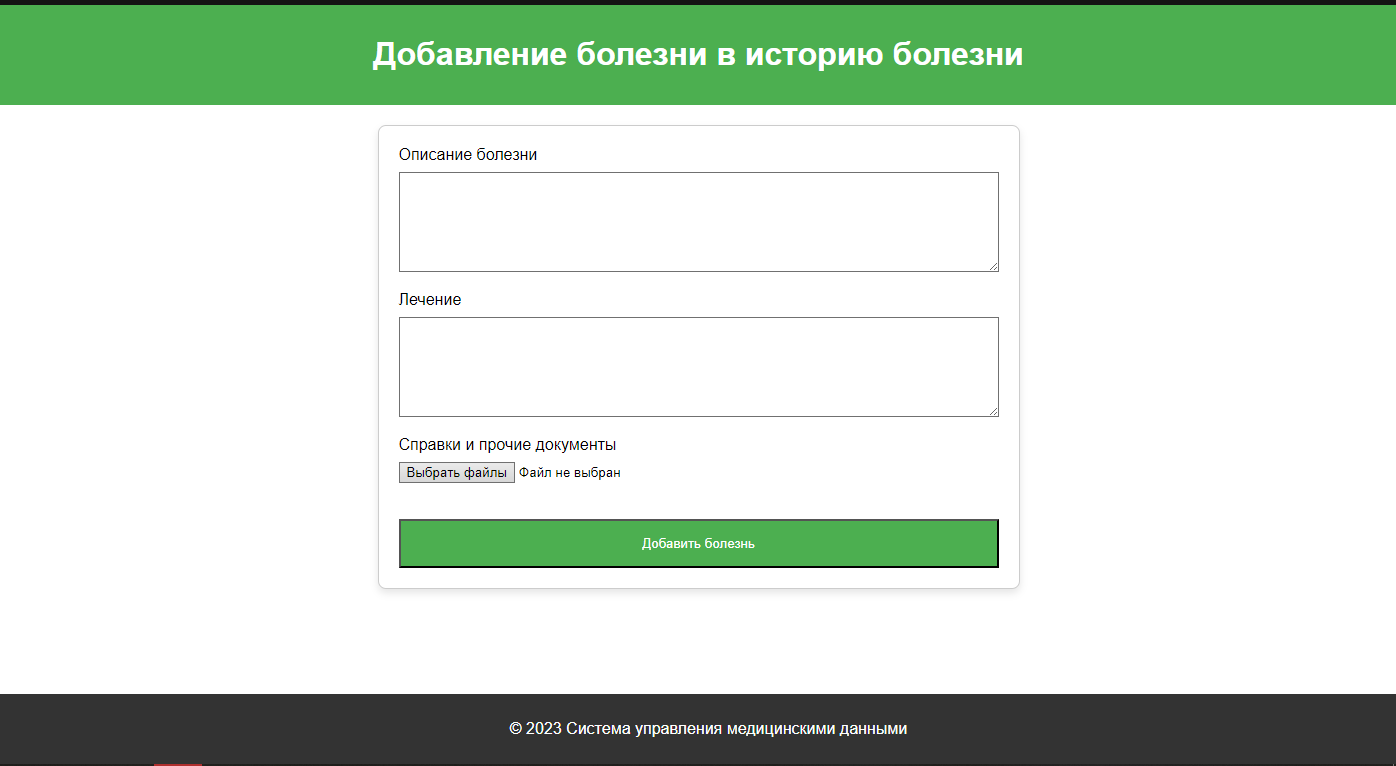


Рисунок 15 – Страница «Добавление записи в историю болезней пациента»

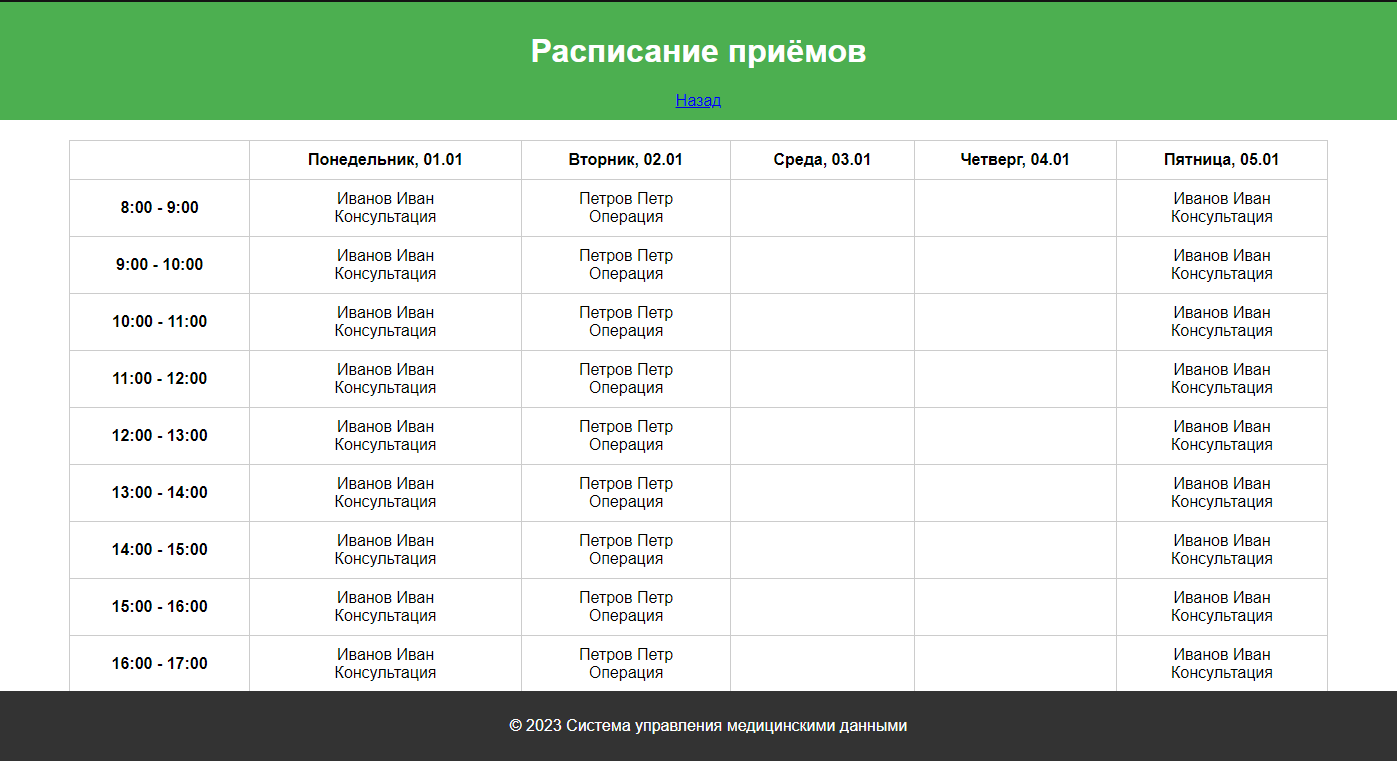


Рисунок 16 – Страница «Расписание приемов пациентов»

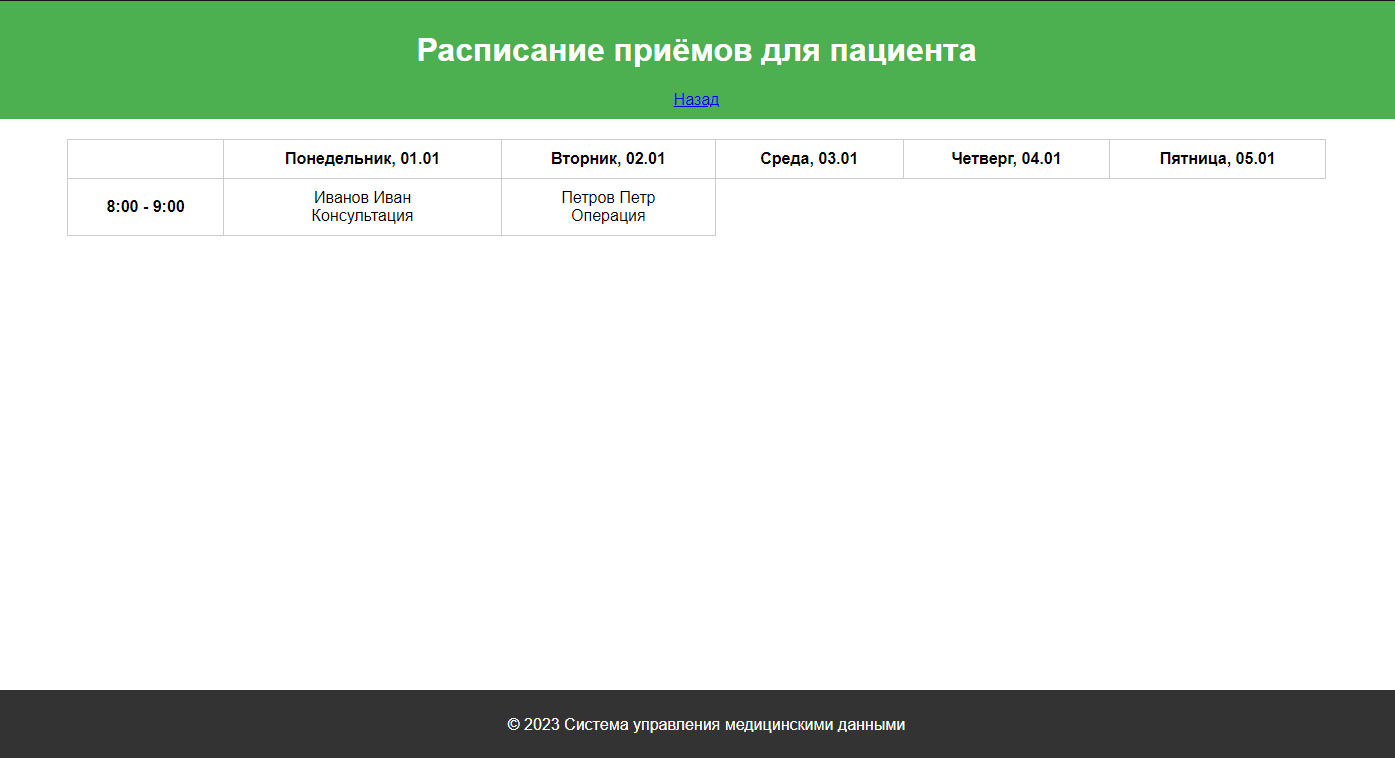


Рисунок 17 – Страница «Расписание записей к врачу»

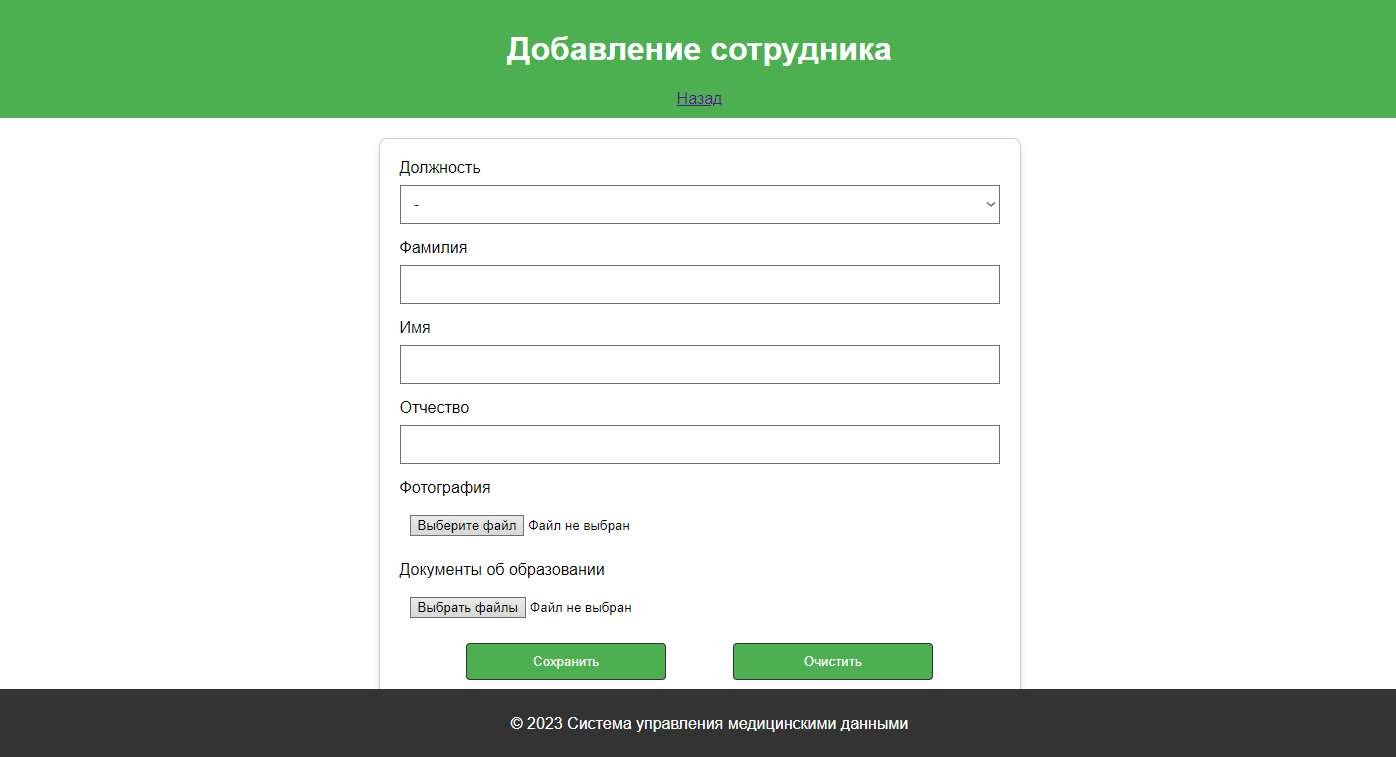


Рисунок 18 – Страница «Добавление нового сотрудника»

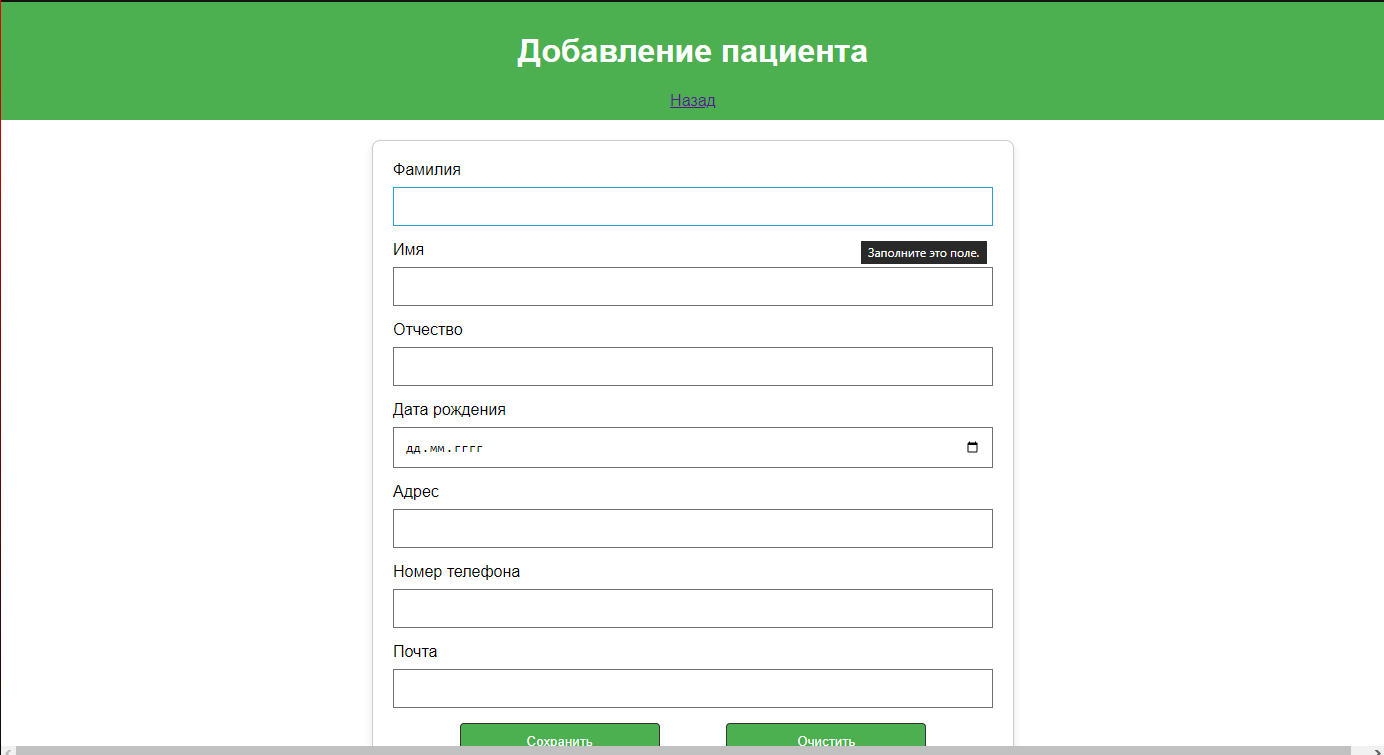


Рисунок 19 – Страница «Добавление пациента»

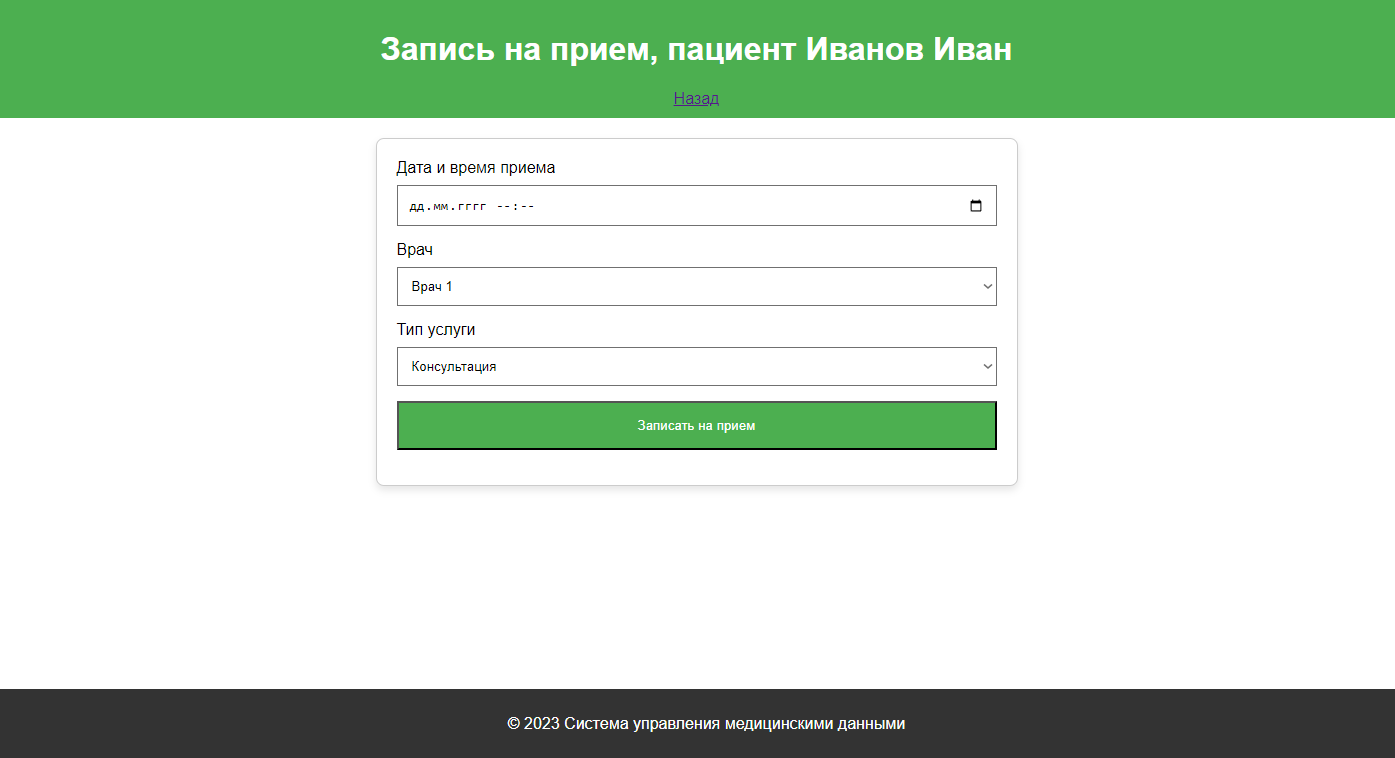


Рисунок 20 – Страница «Запись пациента на прием»

# 3 Анализ инструментов разработки

## 3.1 Технологии разработки клиентской части

Так как система будет использоваться различными медицинскими учреждениями и ссылка на нее всегда будет доступна с других ресурсов, то сайту системы понадобится индексация, поэтому будут применяться технологии разработки, использующие рендер страниц на стороне клиента. Для этого отлично подходит React. React – это библиотека JavaScript, которая используется для создания пользовательских интерфейсов. Она разработана и поддерживается компанией Facebook. Вот некоторые ключевые особенности и преимущества React:

* Компонентный подход: React основан на концепции компонентов, которые представляют собой независимые и переиспользуемые блоки интерфейса. Каждый компонент может содержать свою собственную логику и стиль, что упрощает разработку и поддержку приложений.
* Виртуальный DOM: React использует виртуальное дерево DOM для эффективного обновления пользовательского интерфейса. При изменении данных React сначала обновляет виртуальное дерево DOM, а затем сравнивает его с реальным DOM и применяет только необходимые изменения, что повышает производительность приложения.
* Однонаправленный поток данных: React пропагандирует однонаправленный поток данных, что означает, что данные передаются сверху вниз через иерархию компонентов. Это способствует более простой и предсказуемой разработке приложений.
* JSX синтаксис: React использует JSX - расширение синтаксиса JavaScript, которое позволяет объединять HTML и JavaScript в одном файле. Это делает код более читаемым и понятным, особенно для разработчиков, привыкших к работе с HTML.

Для повышения надёжности и качества кода также будет использован TypeScript и Redux.

TypeScript – это язык программирования, который представляет собой надстройку над JavaScript, добавляющую статическую типизацию. Вот несколько ключевых особенностей TypeScript:

* Статическая типизация: TypeScript позволяет определять типы данных для переменных, параметров функций, возвращаемых значений и других элементов кода. Это улучшает надежность кода, облегчает его понимание и уменьшает количество ошибок на этапе разработки.
* Расширенная поддержка IDE: Интегрированные среды разработки (IDE), такие как WebStorm, обеспечивают более продвинутую поддержку для TypeScript, включая автодополнение, статический анализ кода, быструю навигацию и рефакторинг.
* Улучшенная читаемость кода: Использование типов данных делает код более понятным и самодокументируемым. Это особенно полезно при работе с большими проектами и командами разработчиков.
* Совместимость с JavaScript: TypeScript совместим с существующим JavaScript кодом, поэтому вы можете постепенно внедрять его в существующие проекты без необходимости полной переписывания.

Redux – это библиотека управления состоянием приложения для JavaScript и его фреймворков, таких как React и Angular. Вот основные концепции Redux:

* Централизованное хранилище: Redux хранит состояние всего приложения в одном централизованном хранилище, что делает управление состоянием более прозрачным и предсказуемым.
* Предсказуемость и неизменяемость: Состояние в Redux не может быть изменено напрямую. Вместо этого для изменения состояния используются чистые функции, называемые редюсерами, которые принимают текущее состояние и действие и возвращают новое состояние.
* Однонаправленный поток данных: Redux следует принципу однонаправленного потока данных, в котором действия генерируются компонентами и передаются в хранилище через редюсеры, а затем обновленное состояние передается обратно в компоненты.
* Расширяемость и удобство отладки: Redux обладает мощным инструментарием для отладки, включая инструменты разработчика Redux DevTools, которые позволяют просматривать и отслеживать изменения состояния в реальном времени.

Использование Redux совместно с TypeScript позволяет создавать масштабируемые, надежные и легко поддерживаемые приложения, особенно в контексте больших проектов и команд разработчиков.

Для того чтобы система была поддерживаемой и расширяемой будут написаны разного вида тесты: юнит тесты, e2e тесты, интеграционные тесты. Для написания юнит-тестов, E2E (end-to-end) тестов и интеграционных тестов для веб-сайта существует множество библиотек и инструментов. Вот несколько популярных вариантов для каждого типа тестирования:

* Jest: Jest – это популярная библиотека для тестирования JavaScript-кода. Она обладает простым в использовании синтаксисом, поддерживает автоматическое обнаружение тестов и предоставляет множество встроенных утилит для тестирования, таких как моки и утверждения.
* Mocha: Mocha – это гибкий фреймворк для тестирования JavaScript с широким набором функций. Он позволяет писать тесты в стиле BDD (Behavior-Driven Development) или TDD (Test-Driven Development) и легко интегрируется с различными утилитами и библиотеками.
* Enzyme: Enzyme – это удобная библиотека для тестирования React-компонентов. Она предоставляет удобные методы для манипулирования и тестирования компонентов, а также интегрируется с различными средствами тестирования, такими как Jest и Mocha.
* Cypress: Cypress – это современный инструмент для написания E2E тестов. Он предоставляет простой и понятный синтаксис, автоматическую установку и запуск браузера, возможность записи и воспроизведения тестов, а также обширные инструменты для отладки и анализа результатов тестирования.
* Selenium WebDriver: Selenium WebDriver – это набор инструментов для автоматизации веб-приложений. Он позволяет писать тесты на различных языках программирования (в том числе JavaScript) и запускать их в различных браузерах.
* TestCafe: TestCafe – это другой инструмент для автоматизации E2E тестирования. Он позволяет писать тесты на чистом JavaScript, обеспечивает автоматическое управление браузерами и простую настройку тестовых сценариев.
* Supertest: Supertest – это библиотека для тестирования веб-приложений Node.js. Она предоставляет простой и понятный интерфейс для отправки HTTP-запросов и проверки ответов сервера.

## 3.2 Технологии разработки серверной части

Nest.js – это фреймворк для разработки серверных приложений на языке TypeScript (или JavaScript) с использованием паттерна проектирования MVC (Model-View-Controller), инъекции зависимостей и других современных подходов. PostgreSQL – это мощная и расширяемая реляционная база данных с открытым исходным кодом. Комбинация Nest.js и PostgreSQL предоставляет разработчикам удобное и эффективное средство для создания серверных приложений с базой данных. Вот несколько ключевых особенностей этой комбинации:

Интеграция с TypeORM: Nest.js поддерживает интеграцию с различными ORM (Object-Relational Mapping) библиотеками, в том числе с TypeORM. TypeORM - это ORM для TypeScript и JavaScript, который позволяет взаимодействовать с базой данных PostgreSQL с использованием объектно-ориентированных моделей данных. С помощью TypeORM и Nest.js вы можете определять сущности и их отношения, выполнять запросы к базе данных и обрабатывать результаты запросов в приложении Nest.js.

* Модульность и структурирование: Nest.js поощряет модульность и структурирование приложений. Вы можете разделить ваше приложение на модули, каждый из которых отвечает за определенную функциональность. Это позволяет легко масштабировать и поддерживать приложение. Вы можете создать модуль для работы с базой данных PostgreSQL и внедрить его в основное приложение Nest.js с помощью механизма инъекции зависимостей.
* Использование декораторов: Nest.js использует декораторы для определения контроллеров, провайдеров, модулей и других компонентов приложения. Это делает код более читаемым и выразительным. Например, с помощью декораторов вы можете определить контроллер для обработки HTTP-запросов к вашему API и провайдеры для взаимодействия с базой данных PostgreSQL.
* Асинхронные операции: Nest.js поддерживает асинхронные операции, что позволяет эффективно обрабатывать запросы к базе данных PostgreSQL без блокировки основного потока выполнения. Вы можете использовать асинхронные функции и операторы await/async для выполнения запросов и обработки результатов.
* Обработка ошибок и исключений: Nest.js предоставляет удобные механизмы для обработки ошибок и исключений, возникающих при взаимодействии с базой данных PostgreSQL. Вы можете определить глобальные обработчики ошибок и использовать middleware для централизованной обработки ошибок в вашем приложении.

Комбинация Nest.js и PostgreSQL предоставляет мощный и гибкий инструмент для разработки серверных приложений с базой данных. Это позволяет создавать масштабируемые, надежные и эффективные приложения, удовлетворяющие требованиям современного веб-разработки.

# ЗАКЛЮЧЕНИЕ

Проектирование медицинской информационной системы «Медицинская карта» включает в себя несколько ключевых этапов, начиная с определения целей системы и заканчивая разработкой ее функционала. Основные цели системы заключаются в обеспечении надежного хранения и управления медицинскими данными, предоставлении пациентам доступа к их медицинской информации и поддержке врачей в процессе диагностики и лечения.

Важным аспектом проектирования является учет различных ролей пользователей, таких как администраторы, врачи, пациенты и медрегистраторы. Каждая роль требует определенного набора функций и данных. Например, врачи должны иметь доступ к информации о пациентах и возможности назначения лечения, в то время как пациенты должны иметь возможность просматривать свою медицинскую историю и записываться на прием.

Техническая реализация включает создание структурированных баз данных и интерфейсов, которые обеспечивают интуитивный доступ к необходимой информации. Использование инструментов, таких как Figma, позволяет разрабатывать детализированные макеты интерфейсов, которые учитывают потребности всех пользователей системы. В конечном итоге, интеграция системы с внешними сервисами и обеспечение безопасности данных помогут повысить эффективность медицинского обслуживания и удовлетворенность пациентов.

# Список используемых источников

1. React Documentation. Retrieved from  
   <https://ru.legacy.reactjs.org/docs/getting-started.html>
2. Redux Documentation. Retrieved from <https://redux.js.org/tutorials/essentials/part-1-overview-concepts>
3. TensorFlow Documentation. Retrieved from <https://www.tensorflow.org/resources/learn-ml?hl=ru>
4. Nest. Retrieved from <https://docs.nestjs.com/>
5. TypeScript Documentation. Retrieved from <https://www.typescriptlang.org/>
6. PostgreSQL Documentation. Retrieved from <https://www.postgresql.org/>
7. Гамма, Э., Хелм, Р., Джонсон, Р., & Влиссидес, Д. (2002). Приемы объектно-ориентированного проектирования. Паттерны проектирования. Питер.
8. Макконнелл, С. (2007). Быстрое разработка программного  
   обеспечения. Питер.
9. Gamma, E., Helm, R., Johnson, R., & Vlissides, D. (2002). Design Patterns: Elements of Reusable Object-Oriented Software. Addison-Wesley.
10. Fowler, M. (2002). Patterns of Enterprise Application Architecture. Addison-Wesley.
11. Martin, R. C. (2008). Clean Code: A Handbook of Agile Software Craftsmanship. Prentice Hall.
12. C. Larman. (2004). Applying UML and Patterns: An Introduction to Object-Oriented Analysis and Design. Prentice Hall.
13. McConnell, S. (2004). Code Complete: A Practical Handbook of Software Construction. Microsoft Press.
14. Fowler, M. (2018). Refactoring: Improving the Design of Existing Code. Addison-Wesley.
15. Booch, G., Rumbaugh, J., & Jacobson, I. (2005). The Unified Modeling Language User Guide (2nd ed.). Addison-Wesley.
16. Sommerville, I. (2011). Software Engineering (9th ed.). Addison-Wesley.
17. Ambler, S. W. (2004). Introduction to UML 2 Activity Diagrams. Retrieved from <http://www.agilemodeling.com/artifacts/activityDiagram.htm>
18. Зеллер, А. (2018). Электронные медицинские карты: Практическое руководство по использованию. Питер.