Федеральное государственное бюджетное образовательное учреждение высшего образования

«Владимирский государственный университет

имени Александра Григорьевича и Николая Григорьевича Столетовых»

Кафедра информационных систем и программной инженерии

**ПОЯСНИТЕЛЬНАЯ ЗАПИСКА**

**к курсовому проекту по дисциплине**

**«Распределенные программные системы»**

**на тему**

Единая система автоматизации поликлиник «ЕСАП»

Выполнил: студент гр. ИСТ-120

Карабанов А.А.

Принял: доц. Тимофеев А.А.

Владимир, 2022

**Аннотация**

Курсовой проект посвящён проектированию и разработке прототипа программной системы - «Единая система автоматизации поликлиник (ЕСАП)», которая предназначена для автоматизации и оптимизации работы поликлиник, позволяя эффективно управлять пациентами, записями, расписаниями и медицинскими данными.

В рамках проекта был проведен анализ и изучение существующих проблем и недостатков в работе поликлиник, таких как длительное ожидание, неэффективное распределение ресурсов и проблемы с доступностью медицинской информации. Программная часть системы реализована на основе платформы Java EE c использованием фреймворка Spring Boot и библиотеки React TS для реализации клиентской части.

Подсистема хранения данных курсового проекта разработана на основе СУБД PostgreSQL.

Курсовой проект представлен на 76 страницах, рисунков – 29, таблиц – 8, использованных источников – 8, приложений – 2.

The course project is devoted to the design and development of a prototype software system – «Unified Polyclinic Automation System (UPAS)», which is designed to automate and optimize the work of polyclinics, allowing you to effectively manage patients, records, schedules and medical data.

As part of the project, an analysis and study of existing problems and shortcomings in the work of polyclinics, such as long waiting times, inefficient allocation of resources and problems with the availability of medical information, was carried out. The software part of the system is implemented on the basis of the Java EE platform using the Spring Boot framework and the React TS library to implement the client part.

The data storage subsystem of the course project was developed on the basis of the PostgreSQL DBMS.

The course project is presented on 76 pages, drawings - 29, tables - 8, sources used - 8, applications - 2.

**СОДЕРЖАНИЕ**

[1 ВВЕДЕНИЕ 5](#_Toc136475397)

[2 ПОСТАНОВКА ЗАДАЧИ 6](#_Toc136475398)

[2.1 Описание предметной области 6](#_Toc136475399)

[2.2 Основные понятия предметной области 8](#_Toc136475400)

[2.3 Функциональные требования к системе 9](#_Toc136475401)

[2.4 Нефункциональные требования к системе 10](#_Toc136475402)

[2.5 Сравнительный анализ аналогов 11](#_Toc136475403)

[2.6 Обзор актуальных технологий 18](#_Toc136475404)

[3 АНАЛИЗ ЗАДАЧИ И ФУНКЦИОНАЛЬНАЯ ДЕКОМПОЗИЦИЯ СИСТЕМЫ 23](#_Toc136475405)

[3.1 Анализ бизнес-процесса «Деятельность поликлиники» 23](#_Toc136475406)

[3.2 Сценарий взаимодействия пользователя с системой 29](#_Toc136475407)

[3.3 Осуществление прием пациента 31](#_Toc136475408)

[3.4 Внести данные о больнице (регистрация больницы в системе) 32](#_Toc136475409)

[3.5 Добавить персонал 32](#_Toc136475410)

[3.5 Запись на прием 33](#_Toc136475411)

[3.6 Вывод 34](#_Toc136475412)

[4 СТРУКТУРНАЯ ОРГАНИЗАЦИЯ СИСТЕМЫ 35](#_Toc136475413)

[4.1 Общие принципы организации системы 35](#_Toc136475414)

[4.3 Организация доступа к данным 35](#_Toc136475415)

[4.4 Организация бизнес-логики 36](#_Toc136475416)

[4.5 Организация веб-интерфейса 37](#_Toc136475417)

[4.6 Взаимодействие компонентов системы для процесса «Поиск карты». 40](#_Toc136475418)

[4.7 Взаимодействие компонентов системы для процесса «Осуществить прием пациента». 41](#_Toc136475419)

[4.8 Организация обработки ошибок 42](#_Toc136475420)

[4.9 Организация управления доступом 44](#_Toc136475421)

[4.10 Вывод 45](#_Toc136475422)

[5 РЕАЛИЗАЦИЯ КОМПОНЕНТОВ СИСТЕМЫ 46](#_Toc136475423)

[5.1 Реализация серверной части 46](#_Toc136475424)

[5.1.1 Реализация слоя доступа к данным 46](#_Toc136475425)

[5.1.2 Реализация слоя бизнес-логики 48](#_Toc136475426)

[5.1.3 Реализация слоя контроллеров 51](#_Toc136475427)

[5.2 Реализация клиентской части 53](#_Toc136475428)

[5.2.1 Реализация маршрутизации 53](#_Toc136475429)

[5.2.2 Реализация базовых компонентов 54](#_Toc136475430)

[5.2.3 Реализация пагинации у клиента 55](#_Toc136475431)

[5.2.4 Реализация валидации форм 57](#_Toc136475432)

[5.3 Физическая структура баз данных 59](#_Toc136475433)

[5.4 Обеспечение целостности данных 59](#_Toc136475434)

[5.4.1 Ограничения безопасности 59](#_Toc136475435)

[5.4.2 Ограничения целостности 60](#_Toc136475436)

[5.4.3 Организация распределенных транзакций 61](#_Toc136475437)

[5.5 Развертывание системы 62](#_Toc136475438)

[5.6 Вывод 62](#_Toc136475439)

[6 ИНСТРУКЦИЯ ПО УСТАНОВКЕ ПРИЛОЖЕНИЯ 63](#_Toc136475440)

[7 ОРГАНИЗАЦИЯ УПРАВЛЕНИЯ ПРОЕКТОМ 66](#_Toc136475441)

[7.1 Распределение ролей в команде/зоны ответственности 66](#_Toc136475442)

[7.3 Календарный план работ 66](#_Toc136475443)

[8 НАГРУЗОЧНОЕ ТЕСТИРОВАНИЕ 68](#_Toc136475444)

[9 ЗАКЛЮЧЕНИЕ 72](#_Toc136475445)

[ПРИЛОЖЕНИЕ А ИСХОДНЫЙ КОД ПРИЛОЖЕНИЯ 74](#_Toc136475446)

[ПРИЛОЖЕНИЕ Б СКРИПТ СОЗДАНИЯ БАЗЫ ДАННЫХ 75](#_Toc136475447)

# **1 ВВЕДЕНИЕ**

Актуальность автоматизации процессов здравоохранения обусловлена ростом объемов обрабатываемой информации, а также необходимостью повышать качество предоставляемых услуг в области здравоохранения. Кроме того, актуальность связана с ростом внедрений информационных систем в области здравоохранения, что обусловлено Постановлением Правительства РФ №140 от 09.02.2022 "О единой государственной информационной системе в сфере здравоохранения".

Единая система автоматизации поликлиник — это комплексное программное решение, разработанное для улучшения организации работы поликлиник и обеспечения более качественного и эффективного медицинского обслуживания.

В современных условиях, когда здравоохранение является одной из важнейших сфер государственной деятельности, создание единой системы автоматизации поликлиник становится все более актуальным. Она позволяет автоматизировать все процессы, связанные с работой поликлиник, в том числе учет пациентов, их медицинской истории, расписание врачей, выписку рецептов и другие задачи.

Подобная система может быть использована не только для улучшения качества медицинского обслуживания, но и для оптимизации работы персонала поликлиники, повышения эффективности управления и уменьшения затрат на бумажную документацию.

Кроме того, единая система автоматизации поликлиник позволяет сократить время на обработку данных, увеличить точность ведения медицинских записей, улучшить координацию работы медицинского персонала и предоставить пациентам возможность более удобно и оперативно получать медицинскую помощь.

# **2 ПОСТАНОВКА ЗАДАЧИ**

## 2.1 Описание предметной области

В качестве предметной области рассматривается медицинское учреждение, имеющее несколько отделений, которые возглавляют ведущие специалисты поликлиники. Каждый сотрудник прикреплен за каким-то одним отделением. На каждого впервые обратившегося за услугами в поликлинику гражданина заводится амбулаторная карта, которая создается в электронном виде и хранится на сервере. Также пациенту выдается талон, даже в случае поступления больного на «скорой помощи». В случае необходимости стационарного лечения с пациентом заключается договор. Договор — это бумажная книжка, в которой фиксируются все события, связанные с лечением больного. После выздоровления пациента, договор прикрепляется к амбулаторной карте. Каждое медицинское оборудование закреплено за каким-либо кабинетом или хранится на складе.

После проведения анализа предметной области были выявлены следующие проблемы:

* с увеличением количества пациентов поликлиники, увеличивается количество информационных потоков, что приводит к снижению управляемости из-за несвоевременного получения информации, ошибочной информации или вовсе из-за её отсутствия. Это является недопустимым, так как может затянуть процесс лечения, привести к неверной или несвоевременной постановке диагноза;
* затраты большого количества временных и человеческих ресурсов на обработку документов и необходимость увеличивать эти ресурсы с ростом количества клиентов;
* тратится много времени на подготовку и поиск необходимых историй болезней, личных карточек, результатов обследований;
* в связи с большим бумажным документооборотом возрастает нагрузка на персонал, в связи с чем растет количество ошибок;
* при обращении пациента в различные поликлиники могут произойти ошибки при передаче сведений;
* из-за необходимости получать документы и данные из других подразделений возникают временные задержки, связанные с их доставкой.

Этот проект представляет собой современный взгляд на классическую систему здравоохранения, которая предоставляет медицинские услуги для удовлетворения потребностей населения в области здравоохранения.

Основная цель проекта – это разработать централизованную систему здравоохранения, позволяющую усовершенствовать информационную и программную поддержку основных процессов поликлиник.

Для достижения поставленной цели были сформулированы следующие задачи:

* исследовать медицинские информационные системы;
* разработать полную модель деятельности разрабатываемой ис;
* выполнить анализ существующих процессов, построить модель idef0;
* выработать требования и критерии к ис;
* выполнить расчет экономической эффективности;
* информационная поддержка принятия управленческих решений в медицинской организации;
* мониторинг и управление потоками пациентов (электронная регистратура);
* ведение электронной медицинской карты пациента (ЭМК);
* оказание медицинской помощи с применением телемедицинских технологий;
* организация профилактики заболеваний, включая проведение диспансеризации, профилактических медицинских осмотров;
* организация иммунопрофилактики инфекционных болезней;
* иные функциональные возможности по решению оператора информационной системы.

## 2.2 Основные понятия предметной области

*Поликлиника* — медицинское учреждение для оказания амбулаторной медицинской помощи больным на приёме и на дому.

*Медицинская карта* — медицинский документ, в котором лечащими врачами ведётся запись истории болезни пациента и назначаемого ему лечения.

*Врач* — человек, использующий свои навыки, знания и опыт в предупреждении и лечении заболеваний, поддержании нормальной жизнедеятельности организма человека.

*Главный врач —* врач, занимающий руководящую должность в поликлинике.

*Дежурный врач —* врач, который ведет прием пациентов непосредственно в день обращения в поликлинику.

*Регистратор —* сотрудник поликлиники, работающий в регистратуре. Он встречает пациентов, осуществляет распределение клиентов по специалистам и запись на прием, а также ведет картотеку организации.

Концептуальная диаграмма классов представлена на рисунке 1.

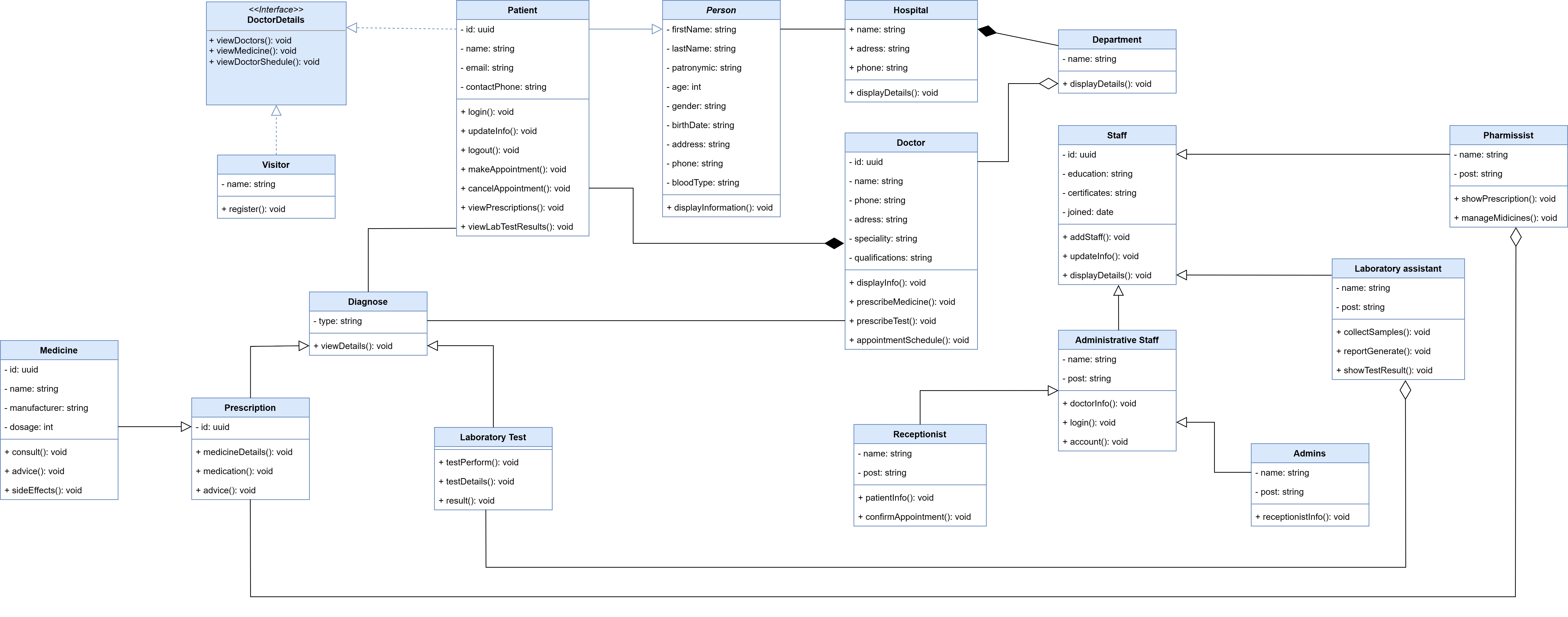


Рисунок 1 – Концептуальная диаграмма классов.

## 2.3 Функциональные требования к системе

Согласно приказу Министерства Здравоохранения РФ от 24.12.2018 № 911н «Об утверждении Требований к государственным информационным системам в сфере здравоохранения субъектов Российской Федерации, медицинским информационным системам медицинских организаций и информационным системам фармацевтических организаций» посредством медицинской информационной системы должны обеспечиваться:

* Система должна обеспечивать возможность регистрацию / удаление больниц в системе;
* Система должна обеспечивать возможность регистрации / авторизации / аутентификации пользователей;
* Система должна обеспечивать возможность установления ролей для медперсонала при регистрации (врач, лаборант, регистратор);
* Система должна обеспечивать возможность мониторинга и управления потоками пациентов (создание электронной очереди);
* Система должна обеспечивать возможность ведение электронной медицинской карты пациента (ЭМК);
* Система должна обеспечивать возможность создание версии для печати ЭМК;
* Система должна обеспечивать возможность печати ЭМК;
* Система должна обеспечивать возможность прикрепления пациента к больнице;
* Система должна обеспечивать возможность врачу получать данные о пациенте;
* Система должна обеспечивать возможность получение данных об назначенных обследованиях для лаборантов.

## 2.4 Нефункциональные требования к системе

**Расчет нагрузки**

Так как нет точных данных о численности медперсонала, была проведена примерная оценка загруженности сервера. На рисунке 2 представлено количество медицинских работников на 100000 населения, что составляет примерно 476000 работников на 2014 год.

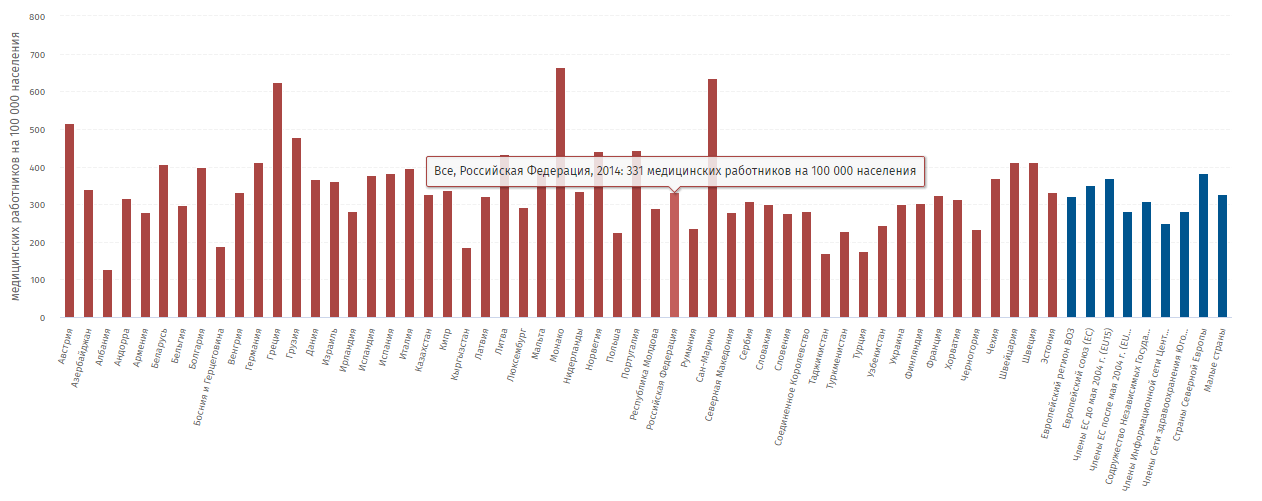


Рисунок 2 – Количество работников на 100000 населения.

Также были проанализированы расчетные данные на 2024 год, где примерное количество медработников будет составлять 1 396 000 (рисунок 3).

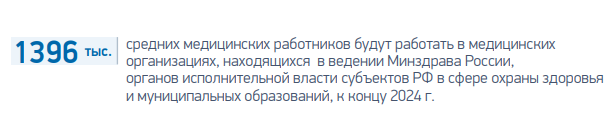


Рисунок 3 – Расчетные данные к 2024 год.

С учетом выше приведенных цифр максимальное количество пользователей будет 1 400 000 человек, а средняя 500 000 человек.

С учетом исследуемой статики выше были выделены следующие нефункциональные требования к системе:

* Система должна иметь понятный русскоязычный интерфейс;
* Система должна открываться в течение не больше 15 секунд;
* Отклик системы для типовых действий не должен превышать 10 секунд для средней нагрузки (300000 - 500000 пользователей);
* Система должна обеспечивать безопасность хранения пользовательских данных, путём их хеширования;
* Система должна обеспечивать стабильную работу при одновременном использовании 1500000 пользователями;
* Клиент должен взаимодействовать с системой посредством протокола HTTP.

## 2.5 Сравнительный анализ аналогов

Существует множество систем автоматизации работы медицинских учреждений, каждая из которых имеет свои преимущества и недостатки. При выборе подходящей системы необходимо учитывать множество факторов, таких как размер медицинского учреждения, его специализация, требования к безопасности и т.д.

Рассмотрим некоторые из них:

* МИС «БАРС.Здравоохранение» — это информационная система, разработанная для автоматизации работы больниц и поликлиник. Основной функционал системы включает в себя учет и анализ медицинских данных, планирование и учет услуг, управление персоналом, учет финансов и материальных запасов, анализ работы медицинского учреждения. Система позволяет организовать эффективную работу медицинского учреждения и повысить качество обслуживания пациентов.
* МИС «Инфоклиника» — это система автоматизации работы медицинских учреждений. Основной функционал системы включает в себя учет медицинских карт, планирование и учет услуг, управление персоналом, учет финансов и материальных запасов, анализ работы медицинского учреждения. Одним из преимуществ системы является интеграция с электронным реестром и страховыми компаниями, что позволяет быстро и эффективно вести учет услуг и оплаты.
* МИС «Медиалог» — это комплексная система автоматизации работы медицинских учреждений. Основной функционал системы включает в себя учет и анализ медицинских данных, планирование и учет услуг, управление персоналом, учет финансов и материальных запасов, анализ работы медицинского учреждения. Система позволяет эффективно управлять медицинским учреждением, повысить качество обслуживания пациентов и увеличить прибыльность.
* МИС «1С:Медицина» — это программа для автоматизации работы медицинских учреждений. Основной функционал системы включает в себя учет и анализ медицинских данных, планирование и учет услуг, управление персоналом, учет финансов и материальных запасов, анализ работы медицинского учреждения. В программе реализованы модули для ведения учета электронных медицинских карт, реестра услуг и оплаты, статистического учета, управления документооборотом и другие.

Необходимо провести анализ на основе методики с использованием весовых коэффициентов, для определения наилучшей. Для получения наиболее обоснованного результата будет использована методика определения обобщённой степени соответствия альтернатив.

Для оценки альтернатив нужно выделить ряд критериев, по которым будет производится оценка аналогов:

* **Стоимость** – оценивает стоимость приобретения и поддержки системы, включая начальные инвестиции и операционные расходы.

Значения:

* 8 – 10 низкая стоимость, система доступна для большинства клиник и может быть легко поддерживаема;
* 5 – 7 средняя стоимость, система доступна для большинства клиник, но ее поддержка может потребовать дополнительных ресурсов;
* 0 – 4 высокая стоимость, система может быть недоступна для большинства клиник.
* **Масштабируемость –** определяет, насколько легко можно масштабировать систему для удовлетворения потребностей растущей клиники.

Значения:

* 8 – 10 система может быть легко масштабирована, чтобы удовлетворить потребности растущей клиники;
* 5 – 7 система может быть масштабирована, но могут потребоваться дополнительные ресурсы;
* 0 – 4 система не может быть легко масштабирована.
* **Интерфейс пользователя –** оценивает, насколько интуитивно понятен и прост в использовании интерфейс системы для пользователей.

Значения:

* 8 – 10 интерфейс системы прост в использовании, интуитивно понятен и не вызывает затруднений у пользователей;
* 5 – 7 интерфейс системы может быть немного сложным или не интуитивным, что требует некоторого обучения пользователей;
* 0 – 4 интерфейс системы очень сложен и неудобен в использовании.
* **Возможность экспорта/импорта данных –** определяет наличие возможности импорта и экспорта данных, что позволяет упростить процесс миграции данных и обмена информацией с другими системами.

Значения:

* 8 – 10 система поддерживает импорт и экспорт данных в различных форматах, что облегчает обмен информацией с другими системами;
* 5 – 7 система поддерживает импорт и экспорт данных в некоторых форматах, что может потребовать дополнительных ресурсов;
* 0 – 4 система не поддерживает импорт и экспорт данных.
* **Функционал –** это набор возможностей, которые предоставляет система пользователю. В данном случае, оценивается наличие и разнообразие функций, которые включают в себя запись на прием, ведение медицинской карты, выписывание рецептов, проведение медицинских осмотров, формирование отчетности и другие.

Значения:

* 8 – 10 система имеет широкий функционал, позволяющий полностью автоматизировать работу поликлиники;
* 5 – 7 функционал системы охватывает основные процессы работы поликлиники, но может быть ограничен в некоторых моментах;
* 0 – 4 система имеет ограниченный функционал, не позволяющий полностью автоматизировать работу поликлиники.
* **Удобность навигации –** это критерий, оценивающий удобство использования системы, ее логическую структуру, удобство доступа к различным функциям и информации.

Значения для данного критерия могут быть определены следующим образом:

* 8 – 10 навигация в системе легка и интуитивно понятна, все функции быстро доступны пользователю;
* 5 – 7 навигация в системе может быть несколько сложной, некоторые функции могут быть недоступны на первый взгляд;
* 0 – 4 навигация в системе крайне неудобна, функции плохо организованы и могут вызывать затруднения у пользователя.

Результат оценки аналогов по каждому критерию представлен в таблице 1.

Таблица 1 – Оценка систем по ряду критериев:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Критерии | Барс | Инфоклиника | Медиалог | 1C.Медицина |
| Стоимость | 6 | 8 | 7 | 6 |
| Масштабируемость | 9 | 9 | 8 | 9 |
| Интерфейс пользователя | 5 | 8 | 7 | 9 |
| Экспорт/импорт данных | 9 | 9 | 8 | 9 |
| Функционал | 9 | 9 | 9 | 9 |
| Удобность навигации | 7 | 8 | 7 | 9 |

Критерии и их веса представлены в таблице 2.

Вес критерия – определяет, на сколько сильное влияние оказывает тот или иной параметр на итоговое значение.

Взвесим критерии, т.е. определим, значение каких из них наиболее влияют на выбор системы.

Таблица 2 – Критерии и весовые коэффициенты.

|  |  |
| --- | --- |
| Критерии | Вес |
| Стоимость | 0,15 |
| Масштабируемость | 0,2 |
| Интерфейс пользователя | 0,15 |
| Экспорт/импорт данных | 0,2 |
| Функционал | 0,2 |
| Удобность навигации | 0,1 |

В данном случае были выбраны следующие весовые коэффициенты на основе предположения, что наиболее важными критериями являются масштабируемость, функционал и экспорт/импорт данных. Остальные критерии также являются важными, но их значимость ниже, поэтому им были присвоены меньшие веса.

Выполним оценку систем, подсчитав взвешенные оценки. Для этого для каждого аналога суммируем оценки по критериям, умноженным на веса этих критериев.

Таблица с результатами подсчёта взвешенных оценок представлена в таблице 3.

Таблица 3 – Результат подсчёта взвешенных оценок.

|  |  |
| --- | --- |
| Аналог | Взвешенные оценки |
| БАРС | 7,75 |
| Инфоклиника | 8,2 |
| Медиалог | 7,8 |
| 1C.Медицина | 8,55 |

На основе этой оценки, наиболее подходящей системой является Инфоклиника и 1C.Медицина, так как они набрали наибольшее количество баллов по всем критериям.

Необходимо провести сравнительный обзор по функциональности продуктов, с целью раскрыть основные функции систем.

Обзор функции систем представлен в таблице 4.

В качестве «+» подразумевается наличие данного функционала, «-»  
означает его отсутствие.

Таблица 4 – Функциональный обзор рассматриваемых систем.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Функции | БАРС | Инфоклиника | 1C.Медицина | Медиалог |
| Расписание врачей/кабинетов, запись на прием. | + | + | + | + |
| Учет пациентов, оформление  документов. | + | + | + | + |
| Ведение электронной медицинской  карты (ЭМК). | + | + | + | + |
| Планы лечения и стандарты лечения. | + | - | - | + |
| Работа с больничными листами | + | + | + | + |
| Учет направлений/рекомендаций от внешних и внутренних врачей (мед. учреждений). | + | - | - | + |
| Отчетность и аналитика (финансовая, медицинская, маркетинговая). | + | + | + | + |
| Учет финансовой информации –  регистрация оказания услуг, оплата услуг (нал./карта/безнал). | + | - | + | - |
| Интеграция с другими информационными системами. | - | - | + | - |

Вывод: По результатам данного анализа, можно сделать следующий вывод: система «1С:Медицина.Поликлиника» является одним из наиболее перспективных программных продуктов среди представленных. Она обладает преимуществами, которые делают ее привлекательным выбором для многих организаций в области здравоохранения. Главным преимуществом данной системы является ее широкий функционал, позволяющий автоматизировать и управлять различными аспектами работы поликлиники. Она предлагает интегрированные модули для учета пациентов, ведения медицинских карт, планирования приемов, финансового учета и других важных процессов. Поэтому разрабатываемая система также будет стремиться предоставить широкий функционал, сопоставимый с функционалом системы «1С:Медицина.Поликлиника».

## 2.6 Обзор актуальных технологий

* MVC

MVC (Model-View-Controller) – это популярная архитектурная модель, которая является универсальным шаблоном проектирования для создания веб-приложений, основанных на языках программирования.

MVC удобнее, так как он более простой и понятный для понимания любым программистом. Кроме того, паттерн используется во многих фреймворках и приложениях, что позволяет быстрее разрабатывать и поддерживать код. В MVC также проще реализовывать взаимодействие между сущностями и обрабатывать ошибки.

* JPA

JPA (Java Persistence API) — это высокоуровневый Java-интерфейс, который упрощает доступ к базам данных, обеспечивая абстракцию от конкретных реализаций и улучшая возможности для создания, чтения, обновления и удаления объектов Java в базе данных.

JPA можно сравнить с использованием слоя DAO. Реализовывая слой DAO, можно добиться большей производительности так как система не тратит время на генерирование SQL-кода, а JPA во много раз укоряет создание системы, поэтому он и был выбран.

* SPA

SPA (Single Page Application) — это веб-приложение, которое обновляет содержимое страницы без перезагрузки всей страницы. Вместо этого одна страница загружается при первом открытии и в дальнейшем модифицируется динамически при помощи AJAX-запросов к серверу. Существует несколько аналогов SPA, таких как Angular, React и Vue.js. Они являются фреймворками для разработки SPA, имеют свои особенности и выбор конкретного фреймворка зависит от конкретных задач и требований проекта.

* Быстрота загрузки: однажды загрузив весь необходимый код для приложения, пользователь может перемещаться между разными разделами, не ожидая загрузки каждой отдельной страницы;
* Удобство использования: приложение SPA позволяет работать более быстро и эффективно. Пользователь может получать более оперативный доступ к информации;
* Улучшенный пользовательский интерфейс: приложения SPA могут работать с более сложными и интерактивными пользовательскими интерфейсами, такими как анимации, расширенные формы;
* Удобное масштабирование: SPA можно легко масштабировать. Приложение можно заменять и обновлять поэтапно, не останавливая работу на сервере;
* Удобная поддержка: SPA обеспечивает более удобную и эффективную поддержку приложения. Загрузка всего приложения происходит в начале, поэтому обращения к серверу в различных зонах приложения становятся проще и эффективнее.
* React.js

React.js является библиотекой JavaScript, предназначенной для создания пользовательских интерфейсов. React был создан компанией Facebook и вышел в свет в 2013 году. Он используется для создания масштабируемых и быстрых веб-приложений.

Сравнивая React.js, Vue и Angular все три JavaScript-фреймворка являются мощными инструментами для создания современных веб-приложений. Однако, каждый из них имеет свои особенности:

React.js:

* Эффективный: React.js имеет множество возможностей и удобство, которые позволяют ему быть быстрым и масштабируемым для разработки больших приложений.
* Гибкий: React.js позволяет разработчикам использовать синтаксический сахар JSX, который делает написание кода чище, более понятным и производительным. Также фреймворк не является монолитным, и дает возможность разработчикам добавлять в него собственные библиотеки и инструменты.

Vue:

* Лёгкий: Vue.js использует меньше ресурсов, чем Angular и React, при этом имеет множество функций и удобство разработки.
* Простой: Vue.js имеет понятный синтаксис и удобный интерфейс, что делает его легче для освоения новичками.

Angular:

* Полный: Angular предоставляет множество возможностей в одном коробке, включая управление состоянием, валидацию форм, роутинг, тестирование и многое другое.
* Официальный: Angular разрабатывается и поддерживается компанией Google, интеграция с другими инструментами Google обеспечивает безопасность и стабильность работы.

Возвращаясь к вопросу, почему React.js лучше, чем Vue и Angular, следует отметить, что он предоставляет гибкость и эффективность в разработке, а также активно используется во многих крупных проектах, что подтверждает его надежность и эффективность.

* TypeScript

JavaScript и TypeScript являются двумя популярными языками программирования, широко используемыми для разработки веб-приложений. JavaScript является стандартным языком сценариев веб-страниц и обладает простым синтаксисом, что делает его доступным для начинающих разработчиков. TypeScript, с другой стороны, является надмножеством JavaScript и предоставляет статическую типизацию, расширяя возможности JavaScript и повышая безопасность и надежность кода. Оба языка являются мощными инструментами для создания интерактивных и масштабируемых веб-приложений.

JavaScript:

* Гибкий, динамичный и кроссплатформенный;
* Используется как на стороне клиента, так и на стороне сервера;
* Легкая интерпретация;
* Поддерживается всеми браузерами;
* Слабо типизированный;
* Jit компиляция;

TypeScript:

* Простота обслуживания и повышение продуктивности проекта;
* Возможна статическая типизация и аннотации;
* Поддерживает объектно-ориентированные функции, такие как интерфейс, наследование и классы;
* Легкая отладка и раннее обнаружение ошибок;

Когда код становится огромным, сложным в обработке и более подверженным ошибкам, лучше, если некоторые ошибки будут обнаружены во время самой компиляции. Вот здесь и помогает TypeScript. Поэтому он будет использоваться в проекте.

**3 АНАЛИЗ ЗАДАЧИ И ФУНКЦИОНАЛЬНАЯ ДЕКОМПОЗИЦИЯ СИСТЕМЫ**

**3.1 Анализ бизнес-процесса «Деятельность поликлиники»**

**Участники процесса**

Пользователь — это работник медицинской организации, физическое лицо, которое имеет медицинское или иное образование, работает в медицинской организации и в трудовые обязанности которого входит осуществление медицинской деятельности.

Основная целевая аудитория — взрослые мужчины и женщины, проживающие в конкретном районе в возрасте от 22 до 65 лет, имеющие высшее медицинское образование и работающие по трудовому договору.

Второстепенная целевая аудитория — это занятые молодые пары, которые считают, что время — это ценный и ограниченный ресурс.

Пример портрета целевой аудитории:

«Соколова Наталья Филипповна работает в должности медсестры терапевтического отделения ОГБУЗ «Седановская городская больница №1» с мая 2011 года. Оформлена переводом с аналогичной должности из ГБ-2 города Братска. Общий стаж работы в лечебно-профилактических медицинских учреждениях 6 лет.

Окончила в 2008 году государственное медицинское училище г. Братска. В 2012 году прошла курсы повышения квалификации в Иркутском государственном медицинском университете по специальности «Организация сестринского дел».

В процессе работы Соколова Н.Ф. осуществляла уход за пациентами и наблюдение за их состоянием, обеспечивала соблюдение лечебно-охранительного и санитарно-эпидемиологического режимов в отделении, осуществляла получение лекарственных средств, обеспечивала их учёт и хранение. Обязанности выполнялись в полном объёме, врачебные назначения выполнялись своевременно и точно.

В коллективе Соколова Н.Ф. зарекомендовала себя с положительной стороны: принимает активное участие в общественной жизни коллектива, пользуется уважением коллег, имеет положительные отзывы о пациентов. Наиболее заметными качествами в личном плане являются аккуратность, ответственность, доброжелательное отношение к людям, осознанное стремление всемерно повышать свой профессиональный уровень».

**Перечень ролей**

В рамках Единой системы автоматизации поликлиник (ЕСАП) предусмотрены следующие роли пользователей:

* Врач:
* Может просматривать расписание приема пациентов;
* Может создавать, изменять и удалять записи на прием;
* Может просматривать медицинскую историю пациента;
* Может добавлять записи;
* Может вносить диагнозы и рекомендации для пациентов;
* Регистратор:
* Может создавать новых пациентов в системе;
* Может записывать пациентов на прием к врачам;
* Может просматривать и редактировать информацию о пациентах;
* Может проверять доступность и свободное время в расписании врачей;
* Главный врач:
* Обладает всеми правами врача и регистратора;
* Может управлять расписанием приема врачей;
* Может назначать и удалять врачей;
* Может просматривать общую статистику работы поликлиники;
* Может просматривать и управлять медицинскими данными и историей пациентов;
* Администратор:
* Обладает всеми правами в системе;
* Может управлять пользователями и их ролями;
* Может просматривать и управлять доступом к системе;

**Диаграммы бизнес-процессов**

Процесс «Деятельность поликлиники» до автоматизации (AS-IS) был декомпозирован на следующие процессы:

* Регистрирование больных;
* Проведение анализов;
* Прием анализов;

Для моделирования процесса до автоматизации была выбрана методология IDEF0, также использовалась нотация BPMN. В результате создания IDEF0 моделей было получено более детальное и точное представление бизнес-процесса, что в свою очередь дает возможность для анализа и оптимизации подпроцессов, а также для их автоматизации. (IDEF0 модели представлены в приложении B). IDEF0-модель декомпозиции процесса «Деятельность поликлиники» представлена на рисунке 4.

Разработав BPMN-диаграммы, повысилось понимание того, как процесс будет работать в будущем. Также данная диаграмма помогла определить требования к IT-системе для поддержки данного процесса.



Рисунок 4 – Декомпозиция «Деятельность поликлиники».

Рассмотрим подробнее работы, включающие основные процессы клиники. На этот раз воспользуемся нотацией BPMN.

При звонке пациента с целью записи на прием к врачу, администратор просматривает журнал записи с целью поиска свободного времени для записи и при этом удобного пациенту. После согласования с пациентом времени приема регистратор формирует запись, фиксируя данные пациента в журнал.

Процесс оформления документов в регистратуре перед приемом представлен на рисунке 5.

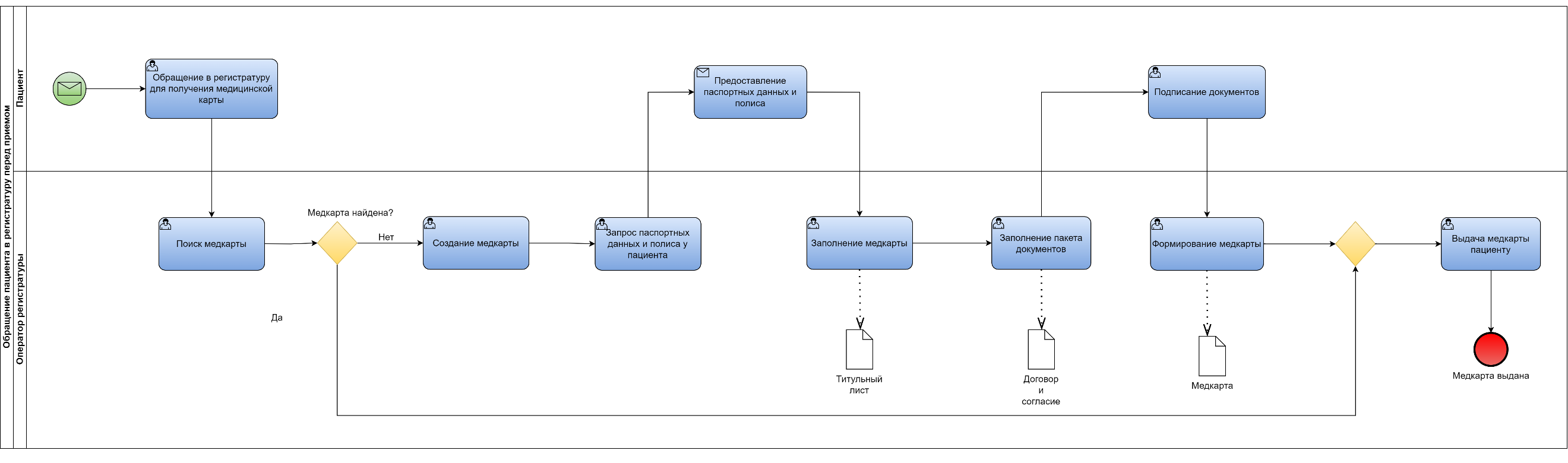


Рисунок 5 – Обращение пациента в регистратуру перед приемом.

Перед приемом пациент обращается в регистратуру для оформления документов. Все документы заполняются регистратором вручную.

Прием пациента врачом представлен на рисунке 6.

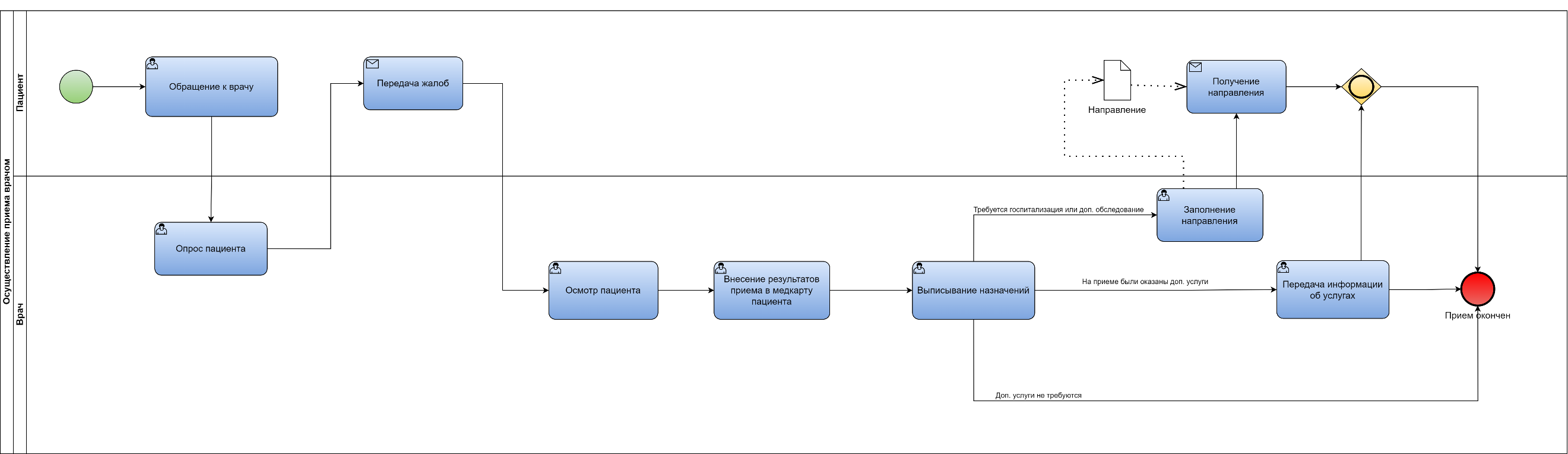


Рисунок 6 – Осуществление приема пациента врачом.

Врач на приеме после осмотра и консультации пациента вносит результаты приема в карту пациента, выписывает направления и рецепт вручную.

TO-BE диаграмма основных процессов клиники. Описание процессов также представлено в нотации BPMN. Представленная модель показывает то, как изменится процесс при внедрении МИС. Основное изменение в том, что появиться централизованная информационная база.

При записи регистратор проверяет, имеется ли пациент в МИС электронную медицинскую карту (далее ЭМК), после чего заводит карту пациента (если ее нет).

Процесс обращения пациента в регистратуру для получения медкарты представлен на рисунке 7.

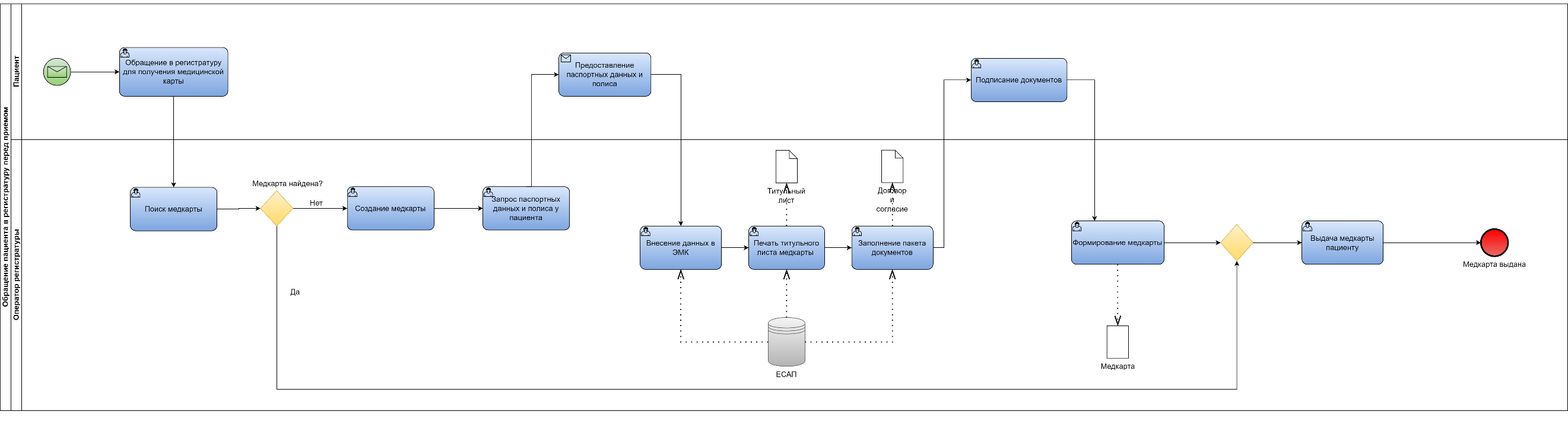


Рисунок 7 – Процесс обращения пациента в регистратуру для получения медкарты.

В данном бизнес-процессе показан процесс автоматизации формирования документов. Предполагается, что регистратор ничего не будет заполнять вручную, все должно печататься из МИС на основании информации, внесенной в ЭМК.

Врач на приеме также использует МИС. Протокол приема формируется на основании шаблона, куда врач вносит информацию по приему пациента. После чего все документы распечатываются. Теперь врач, ведущий осмотр, видит в одном окне, всю динамику пациента.

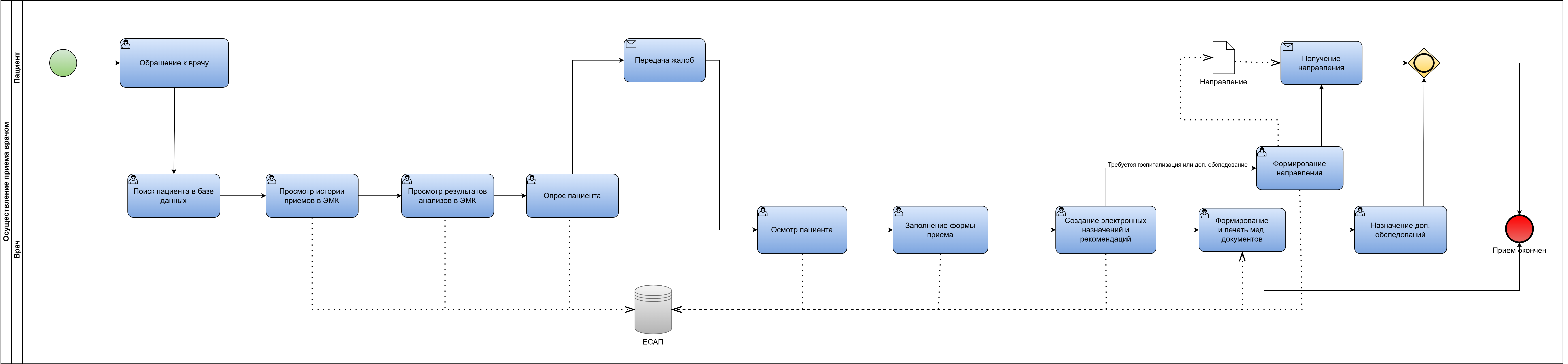


Рисунок 8 – Процесс приема пациента врачом.

**3.2 Сценарий взаимодействия пользователя с системой**

Диаграмма прецедентов разрабатываемой информационной системы представлена на рисунке 9.

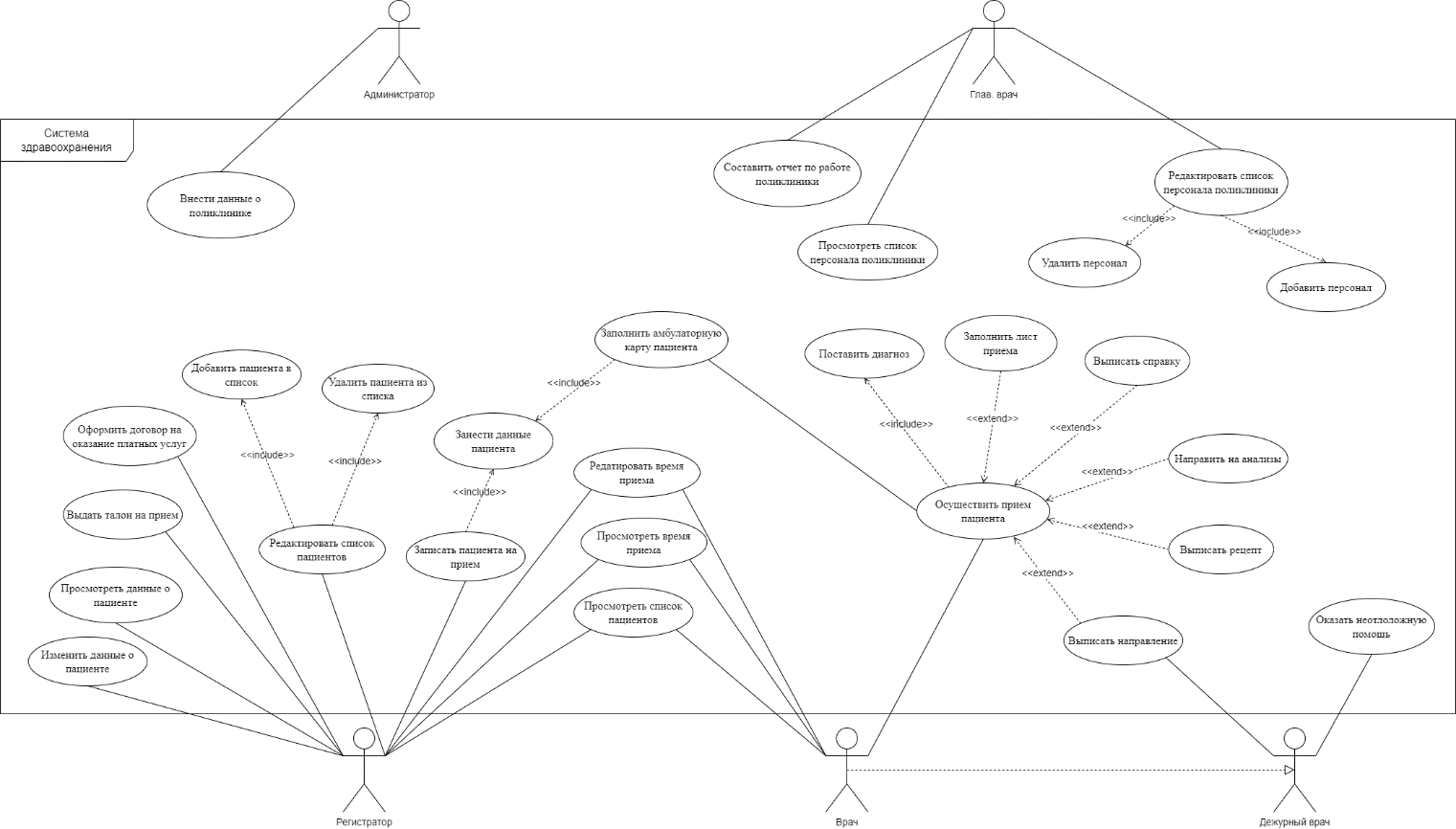


Рисунок 9 – Диаграмма прецедентов.

Данная диаграмма демонстрирует возможные сценарии действия, поддерживаемые разрабатываемым прототипом программной системы.

Диаграммы состояния документов «Пациент» и «Врач», демонстрирующие состояния и условия изменения данных состояний у документов представлены на рисунках 10-11.

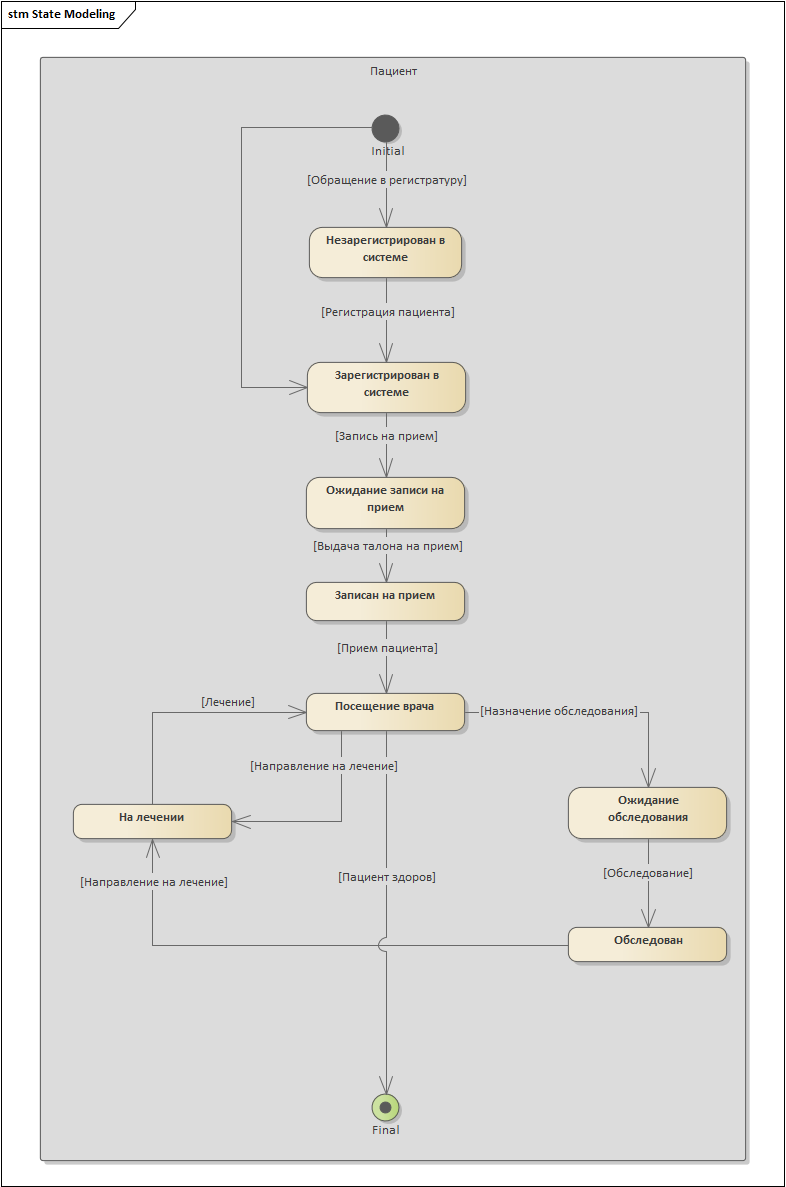


Рисунок 10 – Диаграмма состояний документа «Пациент».

Диаграмма состояний документа «Врач» представлена на рисунке 11.

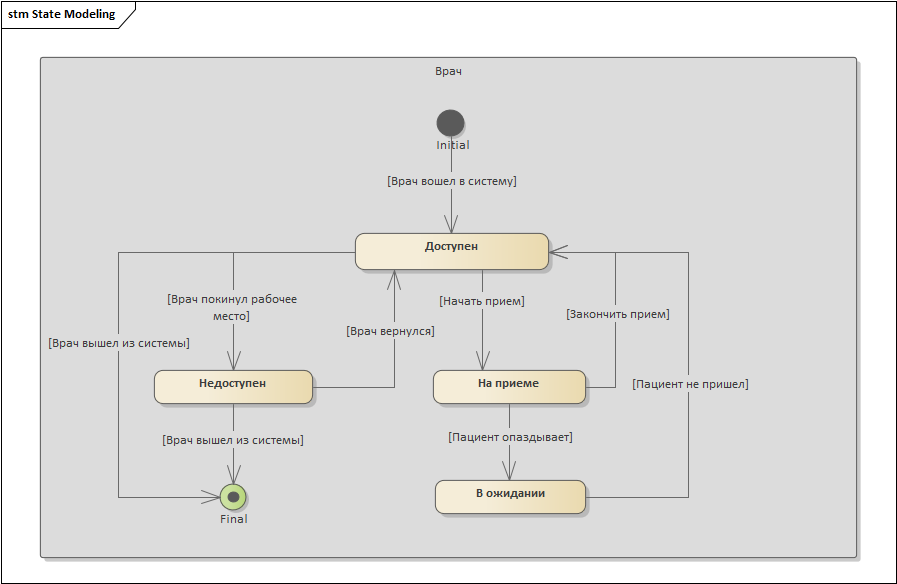


Рисунок 11 – Диаграмма состояний документа «Врач».

**3.3 Осуществление прием пациента**

Расширенное описание прецедента «Осуществить прием пациента» представлено на таблице 5.

Таблица 5 – Описание прецедента.

|  |  |
| --- | --- |
| **Название** | «Осуществление прием пациента» |
| **Предусловие** | Пациент записан на прием. |
| **Действующее лицо** | Врач |
| **Основной поток** | *Осуществление прием пациента*  Врач переходит на страницу очереди и вызывает следующего человека в ней. На странице очереди врач выбирает медицинскую карту пациента.  Врач проводит обследование пациента. После окончание обследования врач вносит данные в электронную форму, которая отправляется на сервер. |
| **Альтернативный поток** | Пациент не явился на прием. |

**3.4 Внести данные о больнице (регистрация больницы в системе)**

Расширенное описание прецедента «Внести данные о больнице» представлено на таблице 6.

Таблица 6 – Описание прецедента.

|  |  |
| --- | --- |
| **Название** | «Внести данные о больнице» |
| **Предусловие** | Главврач отправляет запрос на регистрацию медучреждения. |
| **Действующее лицо** | Администратор |
| **Основной поток** | *Внести данные о больнице*  Администратор заполняет форму данными, полученными от медучреждения, и сохраняет их в БД. |
| **Альтернативный поток** | Обязательные поля пустые. Ошибка валидации. |
| **Постусловие** | Если регистрация прошла успешно, системы сгенерирует имя пользователя и пароль для авторизации, которые администратор предоставит Главврачу. |

**3.5 Добавить персонал**

Расширенное описание прецедента «Добавить персонал» представлено на таблице 7.

Таблица 7 – Описание прецедента.

|  |  |
| --- | --- |
| **Название** | «Добавить персонал» |
| **Предусловие** | Работник устроился на работу в медучреждение. |
| **Действующее лицо** | Главврач. |
| **Основной поток** | *Добавить персонал*  Главврач заполняет форму регистрации сотрудника в медучреждении. |
| **Альтернативный поток** | Обязательные поля пустые. Ошибка валидации. |
| **Постусловие** | Работник зарегистрированы в медучреждении. |

**3.5 Запись на прием**

Расширенное описание прецедента «Запись на прием» представлено на таблице 8.

Таблица 8 – Описание прецедента.

|  |  |
| --- | --- |
| **Название** | «Запись на прием» |
| **Предусловие** | Пациент обратился в регистратуру. |
| **Действующее лицо** | Регистратор. |
| **Основной поток** | *Запись на прием*  Регистратор узнает причину обращения пациента.  Посредством поиска на странице ищет необходимого врача в поликлинике  Открывает расписания подходящего врача.  Выбирает время приема  С помощью страхового полиса производится поиск пациента.  Заносится запись в БД.  Регистратор сообщает время приема пациенту. |
| **Альтернативный поток** | Нет подходящих специалистов, нет свободного время приема, пациент не смог сообщить номер страхового полиса. |
| **Постусловие** | Пациент записан на прием. |

**3.6 Вывод**

В результате обзора предметной области были выявлены функциональные и нефункциональные требования к системе, произведен обзор аналогов, составлены различные диаграммы такие как диаграмма состояний, классов, бизнес-процесса, прецедентов и т.д. Выделены основные сущности, написан словарь предметной области и определены пользователи будущей подсистемы.

**4 СТРУКТУРНАЯ ОРГАНИЗАЦИЯ СИСТЕМЫ**

**4.1** **Общие принципы организации системы**

В основу структуры системы заложен принцип REST API (Representational State Transfer Application Programming Interface) — это архитектурный подход, который используется для создания и предоставления веб-сервисов. Он основан на протоколе HTTP и включает в себя набор методов запросов, таких как GET, POST, PUT и DELETE. Каждый метод имеет свое предназначение и позволяет взаимодействовать с ресурсами.

Альтернативой может быть организация веб-сервера, который отдает статические HTML-страницы в ответ на HTTP-запросы. Однако, в случае, когда проект разрабатывается командой, где одна часть занимается разработкой пользовательского интерфейса, а другая - веб-сервера, более удобным решением будет использование REST API. Это позволяет разделить систему на два независимых проекта. Такой подход повышает эффективность создания приложения, позволяет решить проблему конфликта общего кода и упрощает разделение задач между разработчиками.

**4.3 Организация доступа к данным**

В ходе разработки диаграмма классов существенно изменилась. Изменения в основном коснулись сущностей, которые не имело смысла создавать, таких как «Регистратор», «Лаборант», «Администратор». Эти сущности были объединены в одну «Доктор». Теперь функционал распределяется по ролям, что обеспечивает гибкость настройку безопасности системы. Обновленная диаграмма классов-сущностей представлена на рисунке 12.



Рисунок 12 – Диаграмма классов-сущностей.

Классами-сущностями разрабатываемого прототипа программной системы являются: роль, врач, поликлиника, пациент, медицинская карта и запись в ней, анализ, расписание врача и записи встреч.

**4.4 Организация бизнес-логики**

В процессе разработки прототипа программной системы была разработана диаграмма классов уровня бизнес-логики. Разработанная диаграмма представлена на рисунке 13.

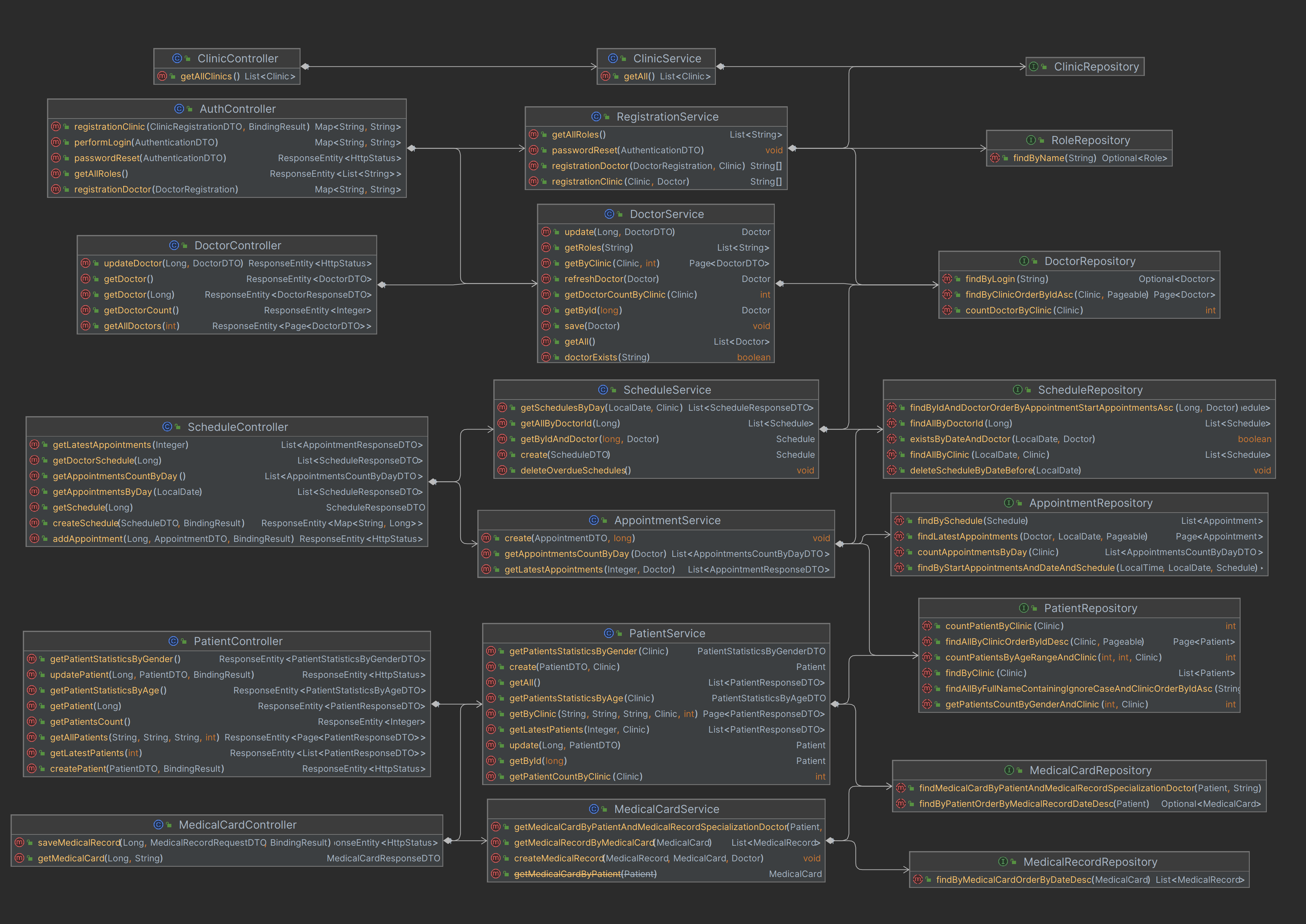


Рисунок 13 – Диаграмма классов уровня бизнес-логики.

**4.5 Организация веб-интерфейса**

В ходе проектирования системы была создана диаграмма переходов между страницами (рисунок 14).

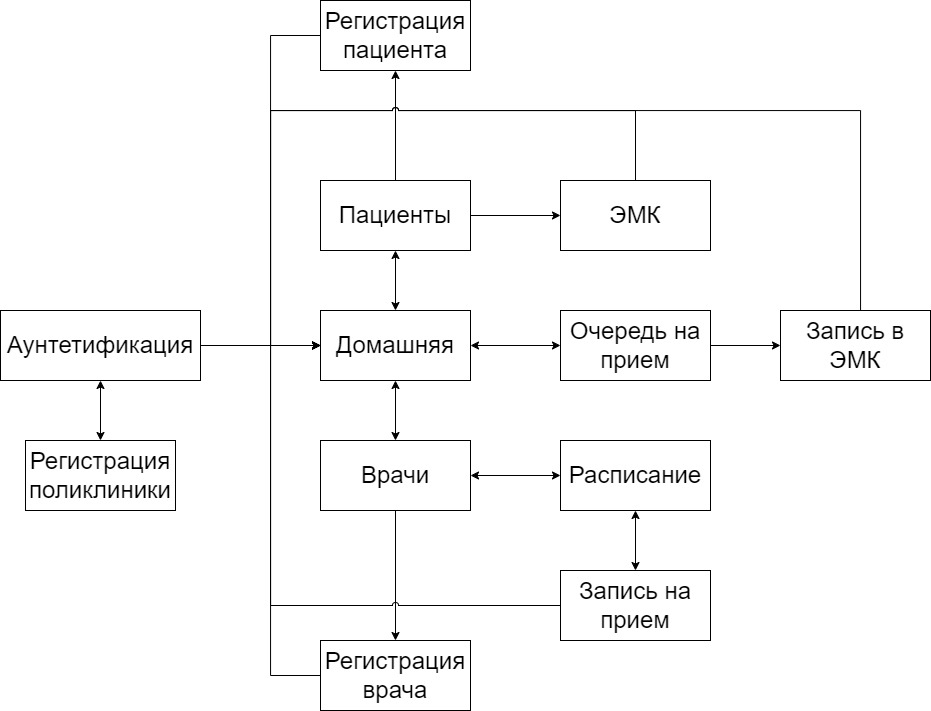


Рисунок 14 – Диаграмма переходов между страницами.

На данной диаграмме мы можем наблюдать 8 страниц:

* Домашняя страница;
* Страница врачей (список);
* Страница пациентов (список);
* Страница с расписанием врача;
* Страница с очередью на прием;
* Страница для записи на прием;
* Страница с информацией о мед. карте пациента;
* Страница для добавления записи в карту;
* Страница авторизации;
* Страница регистрации.

Макет был разработан в приложении Figma. (Ниже представлена часть макета). На рисунке 15 изображен макет домашней страницы. В левой части страницы расположено меню для перехода на страницы врачей и пациентов.

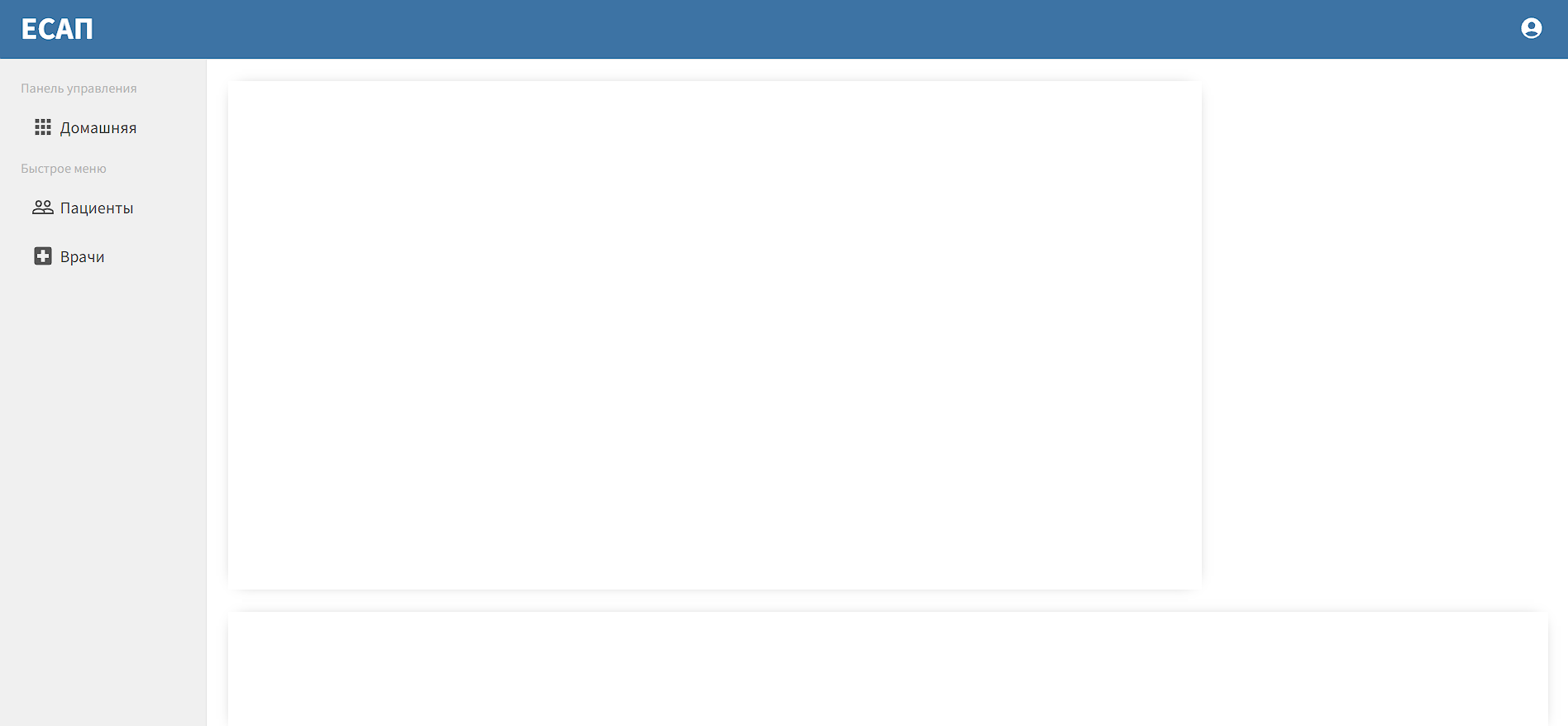


Рисунок 15 – Макет домашней страницы.

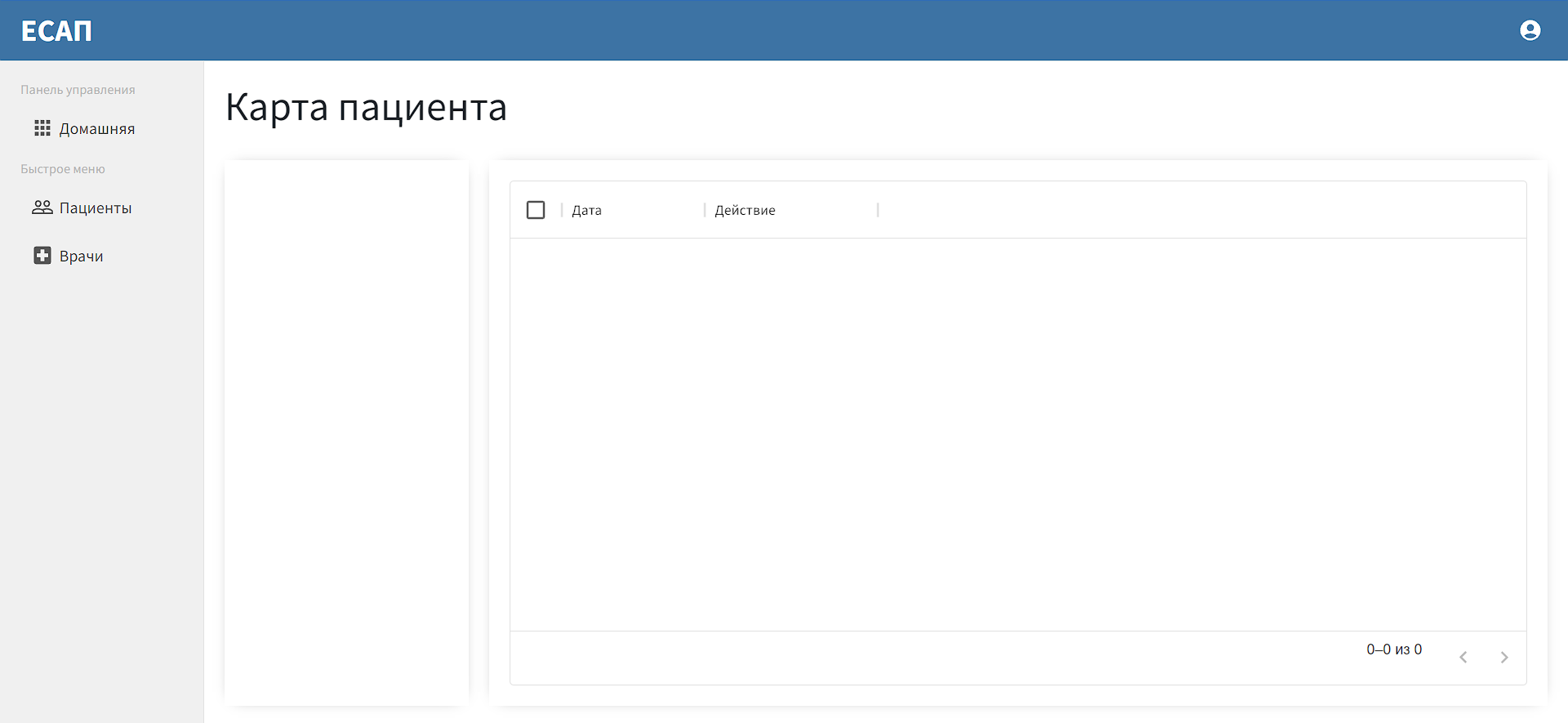


Рисунок 16 – Макет страницы с информацией о мед. карте пациента.

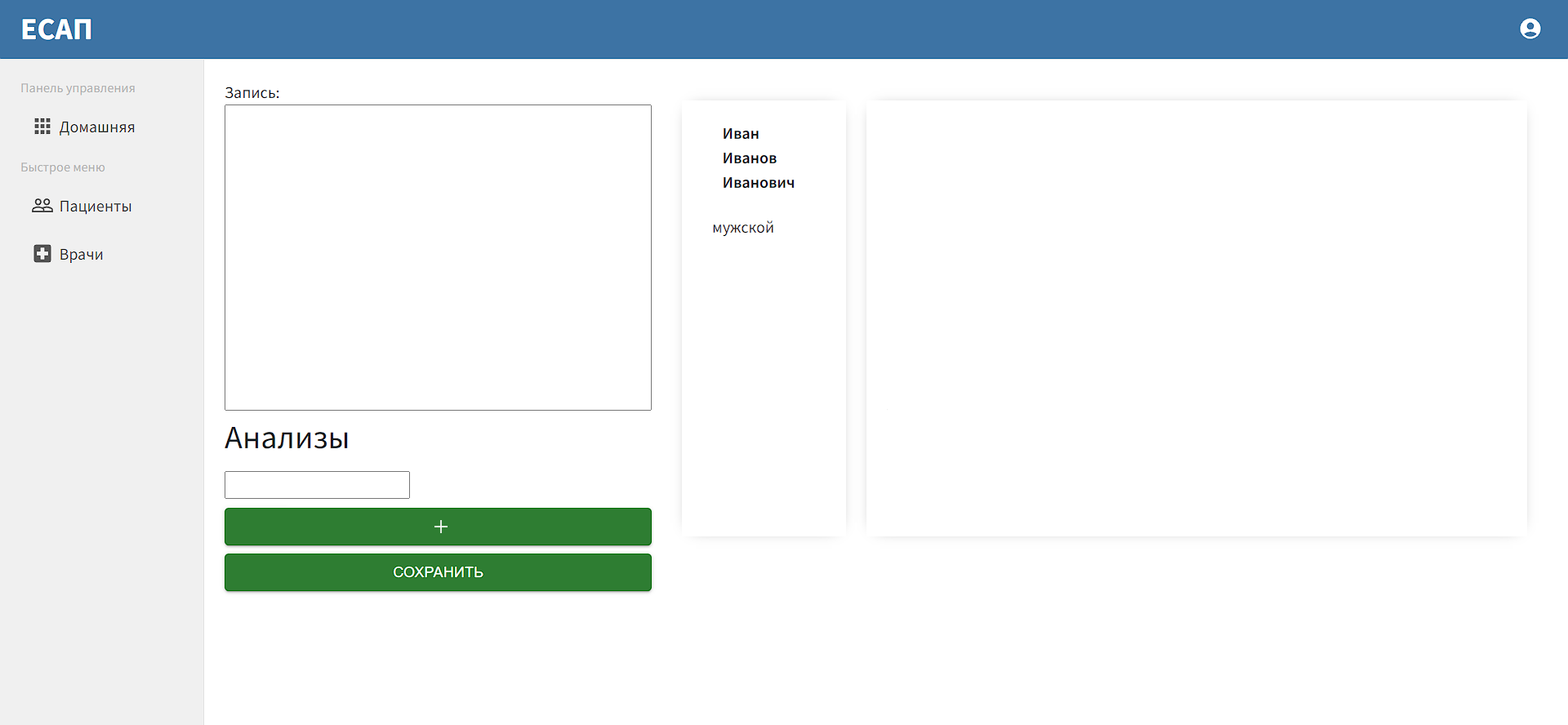


Рисунок 17 – Макет страницы для добавления записи в карту.

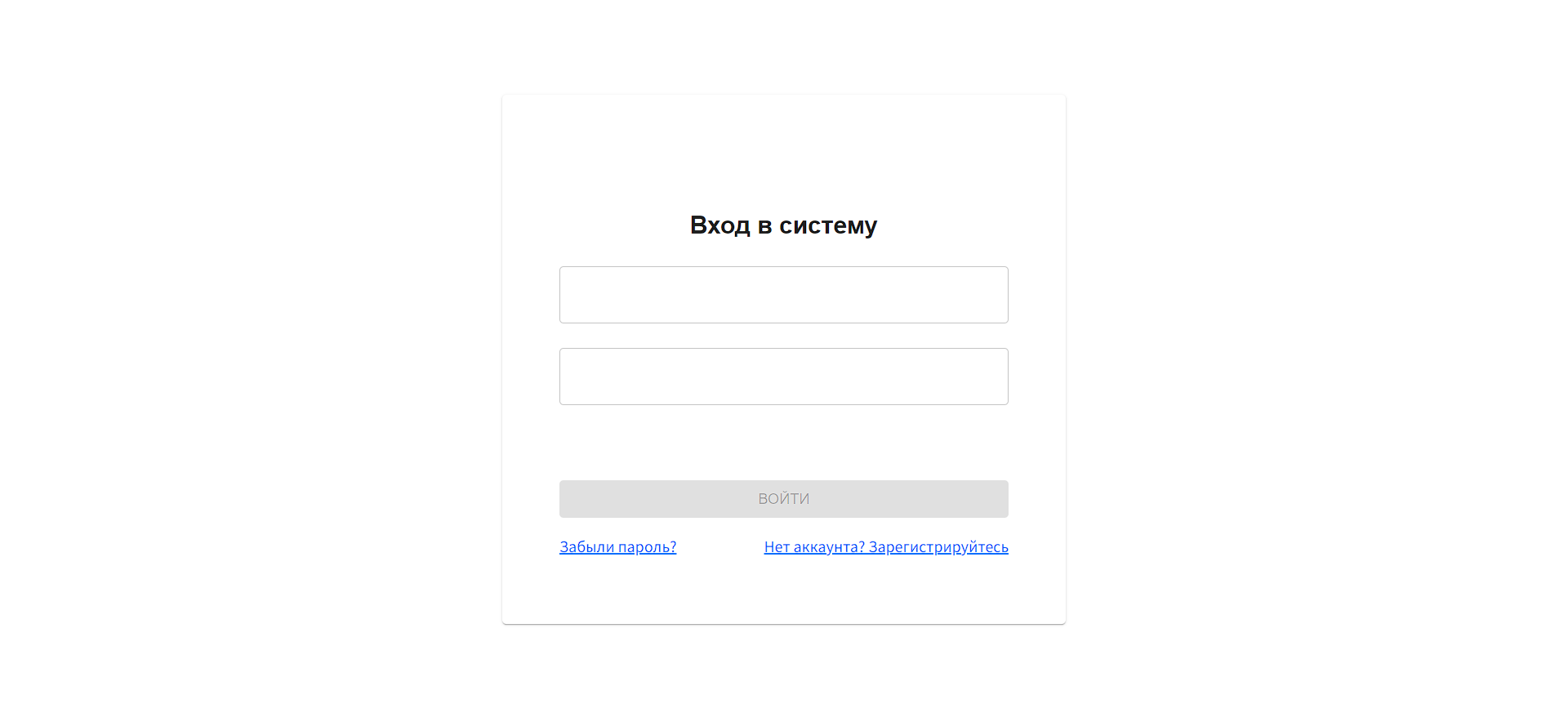


Рисунок 18 – Страница авторизации.

**4.6 Взаимодействие компонентов системы для процесса «Поиск карты».**

Для бизнес-процесса «Поиск карты» была построена диаграмма последовательности, представленная на рисунке 19.

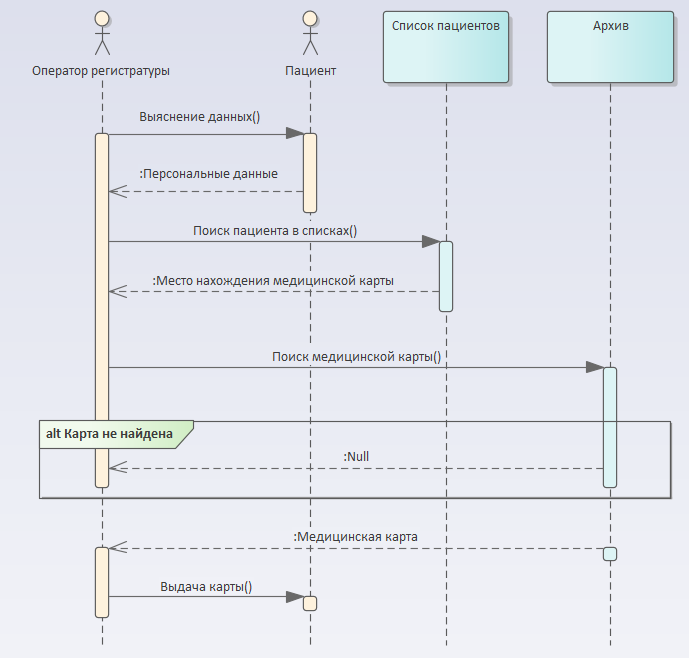


Рисунок 19 – Диаграмма последовательностей для бизнес-процесса «Поиск карты».

Данная диаграмма демонстрирует последовательность действий, происходимых в системе в процессе выполнения бизнес-процесса «Поиск карты».

**4.7 Взаимодействие компонентов системы для процесса «Осуществить прием пациента».**

Для бизнес-процесса «Осуществить прием пациента» была построена диаграмма последовательности, представленная на рисунке 20.

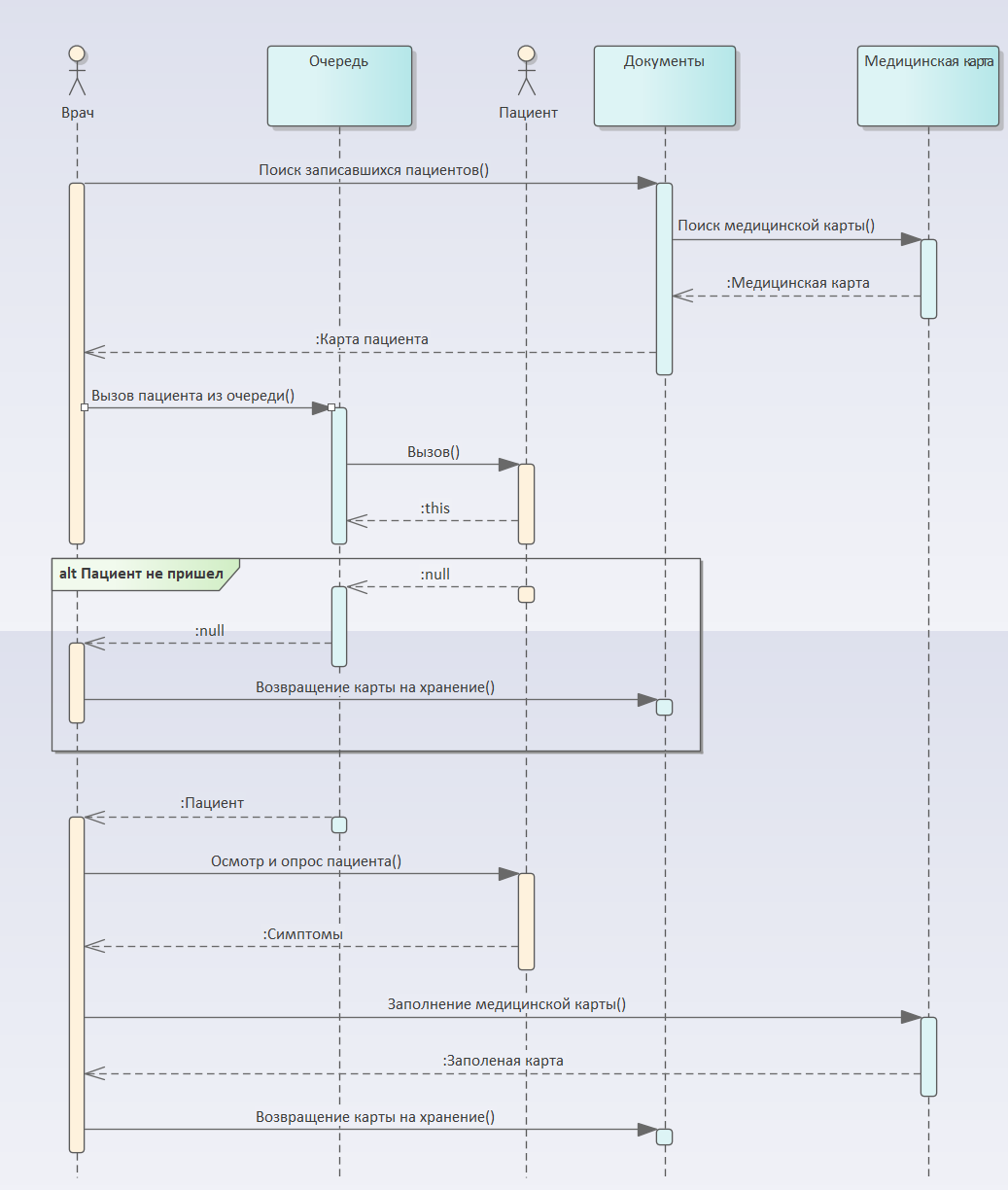


Рисунок 20 – Диаграмма последовательностей для бизнес-процесса «Осуществить прием пациента»».

Данная диаграмма демонстрирует последовательность действий, происходимых в системе в процессе выполнения бизнес-процесса «Осуществить прием пациента».

**4.8 Организация обработки ошибок**

В разрабатываемой системе будет передаваться огромное количество данных, которые необходимо проверить на наличие ошибок.

На стороне сервера обработка ошибок будет организована следующим образом:

* Виды ошибок:
* NotCreateException - исключение, которое обрабатывает ошибки, связанные с невозможностью создания объекта.
* NotFoundException - исключение, которое обрабатывает ошибки, связанные с не найденными объектами.
* AuthenticationException - исключение, которое обрабатывает ошибки аутентификации.
* Ошибки валидации, такие как ошибка недопустимых значений или диапазона данных, ошибка неверного формата данных, а также ошибка пользовательской валидации.
* Принципы обработки:
* ApiExceptionHandler, который является глобальным обработчиком исключений для API.
* Класс ApiError представляет объект, содержащий информацию об ошибке, включая код, статус и сообщение.
* Стандартный механизм обработки ошибок валидации с использованием аннотации @Valid, предоставляемый Spring Framework.
* Кастомный валидатор, который выполняет валидацию ошибок, связанных с временем записи на прием. В данном случае требуется проверка наличия промежутка в 30 минут между записями.

Также разрабатываемая система включает в себя различные формы для создания и изменения данных. После заполнения форм, данные отправляются на сервер. Поэтому для обеспечения целостности данных важно провалидировать все поля формы и проверить их на стороне клиента. Для этого будет использоваться библиотека React Hook Form.

Например, форма регистрации поликлиники содержит следующие поля и правила их заполнения:

* Название поликлиники:
* Поле обязательное;
* Тип вводимых данных: только строка;
* Максимальная длина: 30;
* Минимальная длина: 5.
* Адрес поликлиники:
* Поле обязательное;
* Тип вводимых данных: только строка;
* Максимальная длина: 40;
* Минимальная длина: 10.
* Контактный телефон:
* Поле обязательное;
* Тип вводимых данных: только числа;
* Максимальная длина: 15;
* Минимальная длина: 8.

**4.9 Организация управления доступом**

Организация управления доступом будет реализована следующим образом:

* На сервере:
* При помощи Spring Security определены основные роли пользователей и настройки доступа к различным методам контроллеров.
* На уровне контроллеров указаны требуемые права доступа для конкретных методов (@PreAuthorize).
* На фронтенде:
* После успешной аутентификации пользователя на сервере, сервер передаст информацию о роли пользователя вместе с токеном аутентификации.
* Эта информация сохранится в локальное хранилище браузера для последующего использования.
* При обращении к определенным функциям или компонентам на фронтенде, будет выполнена проверка роли пользователя из локального хранилища и в соответствии с этим отображены или скрыты соответствующие элементы интерфейса.

Такая организация управления доступом позволит определить права доступа на сервере с использованием Spring Security и передать информацию о роли пользователя на фронтенд для принятия соответствующих решений на уровне интерфейса пользователя. Это обеспечит контроль доступа и поддержит согласованность прав доступа между серверной и клиентской частями приложения.

**4.10 Вывод**

В данной главе производилось проектирование будущего веб-приложения. Были составлены полная диаграмма классов, диаграммы последовательности и компонентов. Производилось проектирование слоев моделей, сервисов, репозиториев и контроллеров. Были составлены макеты визуальной составляющей приложения, а также диаграмма переходов между страницами.

# **5 РЕАЛИЗАЦИЯ КОМПОНЕНТОВ СИСТЕМЫ**

## 5.1 Реализация серверной части

## 5.1.1 Реализация слоя доступа к данным

Слой репозиториев является абстракцией, отвечающей за предоставление данных из различных источников, в данном случае - из базы данных. В проекте для каждой сущности системы создан свой интерфейс, который расширяет интерфейс JpaRepository из фреймворка Spring, указывая класс сущности через обобщение, с которым будет работать репозиторий.

В каждом интерфейсе репозитория могут содержаться методы, которые будут автоматически реализованы во время выполнения программы. Реализация методов может быть определена на основе их названия и сигнатуры или с помощью аннотации @Query, которая позволяет написать пользовательский SQL-запрос.

Таким образом, репозитории предоставляют удобный способ взаимодействия с базой данных, скрывая детали работы с ней и предоставляя готовые методы для выполнения различных операций, таких как сохранение, получение, обновление и удаление объектов сущностей.

В ходе разработки системы были реализованы следующие компоненты слоя доступа к данным:

* AnalysisRepository;
* AppointmentRepository;
* ClinicRepository;
* DoctorRepository;
* MedicalCardRepository;
* MedicalRecordRepository;
* PatientRepository;
* RoleRepository;
* ScheduleRepository.

Листинг 1. Код интерфейса «PatientRepository».

public interface PatientRepository extends JpaRepository<Patient, Long> {

List<Patient> findByClinic(Clinic clinic);

int countPatientByClinic(Clinic clinic);

Page<Patient> findAllByClinicOrderByIdDesc(Clinic clinic, Pageable pageable);

@Query("SELECT p FROM Patient p WHERE LOWER(p.firstName) LIKE %:firstName% AND LOWER(p.patronymic) LIKE %:patronymic% AND LOWER(p.lastName) LIKE %:lastName% AND p.clinic = :clinic ORDER BY p.id ASC")

Page<Patient> findAllByFullNameContainingIgnoreCaseAndClinicOrderByIdAsc(@Param("firstName") String firstName, @Param("patronymic") String patronymic, @Param("lastName") String lastName, @Param("clinic") Clinic clinic, Pageable pageable);

@Query("SELECT COUNT(p) FROM Patient p WHERE p.gender = :gender AND p.clinic = :clinic")

int getPatientsCountByGenderAndClinic(@Param("gender") int gender, @Param("clinic") Clinic clinic);

@Query("SELECT COUNT(p) FROM Patient p WHERE FUNCTION('DATEDIFF', YEAR, p.birthDate, CURRENT\_DATE) >= :minAge AND FUNCTION('DATEDIFF', YEAR, p.birthDate, CURRENT\_DATE) <= :maxAge AND p.clinic = :clinic")

int countPatientsByAgeRangeAndClinic(@Param("minAge") int minAge, @Param("maxAge") int maxAge, @Param("clinic") Clinic clinic);

}

Код остальных репозиториев приведен в приложении С.

## 5.1.2 Реализация слоя бизнес-логики

Слой сервисов является абстракцией между слоем бизнес-логики и слоем представления (в данном проекте представлен отдельной подсистемой). В этом проекте слой сервисов содержит бизнес-логику системы. Он получает данные из репозитория, обрабатывает их и передает контроллерам для предоставления пользователю.

Слой сервисов предоставляет дополнительные уровни абстракции и инкапсулирует бизнес-логику приложения. Он может выполнять такие задачи, как валидация данных, обработка бизнес-правил, координация различных операций и транзакций, а также взаимодействие с другими внешними сервисами или компонентами системы.

В ходе разработки системы были реализованы следующие компоненты бизнес-слоя:

* AppointmentService;
* ClinicService;
* DoctorDetailsService;
* DoctorService;
* MedicalCardService;
* PatientService;
* RegistrationService;
* ScheduleService.

Все зависимости в проекте внедрялись с помощью механизма DI/IoC встроенного в Spring Core.

ScheduleService содержит логику работы с расписаниями врачей. Он обеспечивает функциональность создания, чтения, обновления и удаления расписания в системе.

Листинг 2. Метод для добавления нового расписания.

@Transactional  
public Schedule create(ScheduleDTO scheduleDTO) throws NotCreateException {  
 Doctor doctor = doctorRepository.findById(scheduleDTO.getDoctorId())  
 .orElseThrow(() -> new NotFoundException("Doctor not found"));  
 boolean scheduleExists = scheduleRepository.existsByDateAndDoctor(scheduleDTO.getDate(), doctor);  
 if (scheduleExists) {  
 throw new NotCreateException("Schedule already exists for the specified date and doctor");  
 }  
 Schedule schedule = scheduleMapper.toSchedule(scheduleDTO);  
 schedule.setDoctor(doctor);  
 long minutesBetweenStartAndEnd = schedule.getStartDoctorAppointment().until(schedule.getEndDoctorAppointment(), ChronoUnit.*MINUTES*);  
 if (minutesBetweenStartAndEnd <= 0 && minutesBetweenStartAndEnd % 30 != 0) {  
 throw new NotCreateException("Invalid schedule time");  
 }  
 schedule.setMaxPatientPerDay(((int) minutesBetweenStartAndEnd / 30) + 1);  
 return scheduleRepository.save(schedule);  
}

DoctorDetailsService используется для авторизации доктора в системе и наследуется от UserDetailsService.

loadUserByUsername – метод принимает строку username и возвращающий UserDetails из которого Spring получает роли, логин, пароль, и различные флаги. Также метод может выкинуть исключение NotFoundException("User not found") если пользователь не был найден в системе.

Листинг 3. Метод loadUserByUsername.

@Override

@Transactional(readOnly = true)

public UserDetails loadUserByUsername(String username) throws NotFoundException {

Optional<Doctor> doctor = doctorRepository.findByLogin(username);

if (doctor.isEmpty()) {

throw new NotFoundException("User not found");

}

return new DoctorDetails(doctor.get());

}

RegistrationService обрабатывает всё что связано с регистрацией в системе. Рассмотрим методы подробнее:

* registrationClinic – метод принимает на вход «Clinic» и «Doctor». В методе генерируется пароль и логин для главврача поликлиники и назначается роль «CHIEF\_DOCTOR». Из метода возвращается логин и пароль в виде массива строк.

Листинг 4. Метод registrationClinic.

@Transactional  
public String[] registrationClinic(Clinic clinic, Doctor doctor) throws NotFoundException {  
 String login = lpg.generateLogin();  
 String password = lpg.generatePassword();  
 doctor.setLogin(login);  
 doctor.setPassword(passwordEncoder.encode(password));  
 doctor.setClinic(clinic);  
 doctor.setRole(new HashSet<>());  
 doctor.getRole().add(roleRepository.findByName("ROLE\_CHIEF\_DOCTOR")  
 .orElseThrow(() -> new NotFoundException("Role not found")));  
 clinic.setDoctors(Collections.singletonList(doctor));  
 clinicRepository.save(clinic);  
 return new String[]{login, password};  
}

* registrationDoctor – метод принимает на вход «DoctorRegistration» и «Clinic». В методе генерируется пароль и логин для доктора, а также назначается роль, пришедшая с формы. Метод возвращает логин и пароль для зарегистрированного доктора.

Листинг 5. Метод registrationDoctor.

@Transactional  
public String[] registrationDoctor(DoctorRegistration doctorDTO, Clinic clinic) throws NotFoundException {  
 String login = lpg.generateLogin();  
 String password = lpg.generatePassword();  
 Doctor doctor = doctorMapper.toDoctor(doctorDTO);  
 doctor.setLogin(login);  
 doctor.setPassword(passwordEncoder.encode(password));  
 doctor.setClinic(clinic);  
 doctor.setRole(new HashSet<>());  
 doctor.getRole().add(roleRepository.findByName("ROLE\_" + doctorDTO.getRole())  
 .orElseThrow(() -> new NotFoundException("Role not found")));  
 doctorRepository.save(doctor);  
 return new String[] {login, password};  
}

Код остальных сервисов приведен в приложении С.

**5.1.3 Реализация слоя контроллеров**

Слой контроллеров в приложении отвечает за обработку запросов от пользователей и управление бизнес-логикой.

В данном проекте используются аннотации из фреймворка Spring для определения путей и обработчиков запросов, а также для управления доступом к определенным путям для пользователей с различными ролями. Контроллеры вызывают соответствующие методы сервисов, которые обрабатывают запросы.

В ходе разработки системы были реализованы следующие компоненты слоя контроллеров:

* ClinicController;
* AuthController;
* DoctorController;
* MedicalCardController;
* PatientController;
* ScheduleController.

Листинг 6. Класс «AuthController» (фрагмент).

@PostMapping("/registration/clinic")  
public Map<String, String> registrationClinic(@RequestBody @Valid ClinicRegistrationDTO clinicRegistrationDTO,  
 BindingResult bindingResult) {  
 if (bindingResult.hasErrors()) {  
 throw new NotCreateException(ResponseMessageError.createErrorMsg(bindingResult.getFieldErrors()));  
 }  
 String[] loginPassword = registrationService.registrationClinic(clinicMapper.toClinic(clinicRegistrationDTO.getClinic()),  
 doctorMapper.toDoctor(clinicRegistrationDTO.getDoctor()));  
 return Map.of("login", loginPassword[0], "password", loginPassword[1]);  
}  
  
@PostMapping("/login")  
public Map<String, String> performLogin(@RequestBody AuthenticationDTO authenticationDTO) {  
 UsernamePasswordAuthenticationToken authenticationToken =  
 new UsernamePasswordAuthenticationToken(authenticationDTO.getLogin(), authenticationDTO.getPassword());  
 authenticationManager.authenticate(authenticationToken);  
 String token = jwtUtil.generateToken(authenticationDTO.getLogin());  
 return Map.of("jwt", token, "roles", String.join(";", doctorService.getRoles(authenticationDTO.getLogin())));  
}

Код остальных контроллеров приведен в приложении С.

## 5.2 Реализация клиентской части

Клиентская часть реализована при помощи библиотеки React по макету дизайна.

## 5.2.1 Реализация маршрутизации

Для реализации маршрутизации в приложении использовалась библиотека React Router DOM. Эта библиотека предоставляет набор инструментов для управления маршрутами в React-приложениях.

С помощью React Router DOM можно определить маршруты для различных страниц и компонентов приложения. Она также позволяет управлять историей браузера, передавать параметры маршрута и обрабатывать навигацию пользователя.

Листинг 7. Основные маршруты приложения (фрагмент).

const App: React.FC = () => {

const tokenStorageService = new TokenStorageService();

const isAuthenticated = Boolean(tokenStorageService.getToken());

return (

<Routes>

{isAuthenticated ? (

<Route path="/" element={<MainLayout />}>

<Route path="/" element={<Home />} />

<Route path="/doctors" element={<DoctorList />} />

<Route path="/doctor/:doctorId" element={<DoctorPage />} />

<Route path="/patients" element={<PatientList />} />

<Route path="/patient/:patientId" element={<PatientPage />} />

<Route path="/medicalCard/:patientId" element={<MedicalCard />} />

<Route path="/editingMedicalCard/:patientId" element={<EditingMedicalRecord />} />

<Route path="/patient/new" element={<NewPatient />} />

<Route path="/queue/:scheduleId" element={<Queue />} />

<Route path="/register/doctor" element={<DoctorRegistrationForm />} />

</Route>

) : (

<Route path="\*" element={<Navigate to="/" />} />

)}

<Route path="/" element={<LoginForm />} />

<Route path="/register" element={<RegistrationPage />} />

<Route path="/password/reset" element={<ResetPasswordForm />} />

</Routes>

);

};

## 5.2.2 Реализация базовых компонентов

При разработке системы были созданы различные переиспользуемые компоненты, которые служат основой для построения страниц приложения. В React для управления состоянием компонентов используются хуки, такие как useState, useEffect, useContext и другие. Эти хуки позволяют создавать более динамичные и интерактивные компоненты, которые могут обрабатывать пользовательский ввод и отображать соответствующую информацию.

В проекте широко используется хук useEffect для работы с запросами. Он позволяет инициализировать состояние компонента при его первой загрузке и обновлять его в нужный момент. Например, при загрузке страницы со списком пациентов отправляется запрос на сервер для получения списка пациентов.

Листинг 8. Компонент виджета для отображения последних пациентов.

const WidgetSm: React.FC = () => {

const [patients, setPatients] = useState<Patient[]>([]);

useEffect(() => {

HttpService.getLatestPatients()

.then((response) => {

setPatients(response);

});

}, []);

return (

<div className='widgetSm'>

<span className='title'>Новые пациенты</span>

{patients.length === 0 ? (

<p>Новые пациенты отсутствуют</p>

) : (

<List dense={true}>

{patients.map((patient) => (

<ListItem key={patient.id}>

<ListItemButton>

<ListItemAvatar>

<Avatar alt={`${patient.firstName} ${patient.lastName}`} />

</ListItemAvatar>

<ListItemText

primary={`${patient.lastName} ${patient.firstName} ${patient.patronymic}`}

secondary={`${patient.gender === 1 ? 'Мужской' : 'Женский'}, ${patient.birthDate}`}

/>

</ListItemButton>

</ListItem>

))}

</List>

)}

</div>

);

};

## 5.2.3 Реализация пагинации у клиента

Также в приложении была реализована пагинация на стороне клиента. Пагинация — это процесс разделения большого объема контента на несколько страниц для удобства навигации и улучшения пользовательского опыта. Она часто используется на сайтах с большим количеством записей, таких как блоги, интернет-магазины или новостные порталы. Каждая страница содержит определенное количество записей, а пользователь может перемещаться между страницами с помощью кнопок навигации или номеров страниц.

Листинг 9. Страница со списком пациентов с пагинацией.

const PatientList: React.FC = () => {

const [page, setPage] = useState(0);

const [totalPages, setTotalPages] = useState(0);

const [data, setData] = useState<Patient[]>([]);

const [searchTerm, setSearchTerm] = useState<string>('');

useEffect(() => {

HttpService.getPatientList(page)

.then(response => {

setData(response.content);

setTotalPages(response.totalPages);

})

.catch(error => console.error(error));

}, [page]);

const handleSearch = () => {

const [name, lastName, patronymic] = searchTerm.trim().split(/\s+/);

HttpService.searchPatientList(name, lastName, patronymic)

.then(response => {

setData(response.content);

setTotalPages(response.totalPages);

setPage(0);

})

.catch(error => console.error(error));

};

return (

<div className='patientListPage'>

<Box sx={{display: 'flex', alignItems: 'center', justifyContent: 'space-between', my: '9px'}}>

<TextField

value={searchTerm}

onChange={e => setSearchTerm(e.target.value)}

variant='outlined'

fullWidth

size='small'

placeholder='Поиск пациента'

InputProps={{

startAdornment: (

<InputAdornment position='start'>

<Search onClick={handleSearch} />

</InputAdornment>

),

endAdornment: (

<IconButton onClick={() => setSearchTerm('')}>

<Clear />

</IconButton>

),

}}

InputLabelProps={{ shrink: true }}

sx={{ width: '800px' }}

/>

<Link to="/patient/new">

<Button variant="contained" color="primary" aria-label="add patient" component="label" sx={{ marginRight: '10px' }}>

<AddIcon />

</Button>

</Link>

</Box>

<DataGrid

localeText={ruRU.components.MuiDataGrid.defaultProps.localeText}

rows={data}

disableSelectionOnClick

columns={columns}

pagination

page={page}

pageSize={10}

rowCount={totalPages \* 10}

rowsPerPageOptions={[13]}

checkboxSelection

sx={{border: "none"}}

onPageChange={(newPage) => setPage(newPage)}

paginationMode="server"

/>

</div>

);

};

## 5.2.4 Реализация валидации форм

В приложении все формы были провалидированы и обработаны с помощью библиотеки React Hook Form. React Hook Form представляет собой инструмент для управления состоянием и валидацией форм в React. Он основан на использовании хуков React, что обеспечивает простоту и эффективность работы с формами.

Пример валидации представлен в листинге 10.

Листинг 10. Форма регистрации с валидацией (фрагмент).

const RegistrationForm: React.FC<RegistrationFormProps> = ({ onSubmit }) => {

const {

handleSubmit,

control,

formState: { errors },

} = useForm();

const onSubmitForm = (data: DoctorRegistrationDTO) => {

onSubmit(data);

};

return (

<Container maxWidth="sm">

<Box sx={{ display: 'flex', justifyContent: 'center', alignItems: 'center', height: '100vh' }}>

<Card>

<CardContent>

<Box sx={{ display: 'flex', flexDirection: 'column', alignItems: 'center', padding: '40px' }}>

<Box component="form" noValidate onSubmit={handleSubmit(onSubmitForm)} sx={{ mt: 3 }}>

<Grid item xs={12} sm={6}>

<Controller

name="firstName"

control={control}

defaultValue=""

rules={{ required: 'Обязательное поле' }}

render={({ field }) => (

<TextField

autoComplete="given-name"

fullWidth

id="firstName"

label="Имя"

autoFocus

{...field}

error={!!errors.firstName}

helperText={errors.firstName?.message}

/>

)}

/>

</Grid>

<Grid item xs={12} sm={6}>

<Controller

name="patronymic"

control={control}

defaultValue=""

rules={{ required: 'Обязательное поле' }}

render={({ field }) => (

<TextField

autoComplete="patronymic"

fullWidth

id="patronymic"

label="Отчество"

{...field}

error={!!errors.patronymic}

helperText={errors.patronymic?.message}

/>

)}

/>

</Grid>

<Grid item xs={12}>

<Controller

name="lastName"

control={control}

defaultValue=""

rules={{ required: 'Обязательное поле' }}

render={({ field }) => (

<TextField

autoComplete="family-name"

fullWidth

id="lastName"

label="Фамилия"

{...field}

error={!!errors.lastName}

helperText={errors.lastName?.message}

/>

)}

/>

</Grid>

<Grid item xs={12}>

...

</Grid>

<Button

type="submit"

fullWidth

variant="contained"

sx={{ mt: 3, mb: 2 }}

disabled={Object.keys(errors).length > 0}

>

Зарегистрироваться

</Button>

...

</Box>

</Box>

</CardContent>

</Card>

</Box>

</Container>

);

};

## 5.3 Физическая структура баз данных

В процессе разработки прототипа программной системы «Единая система автоматизации поликлиник» использовалась СУБД PostgreSQL. После разработки уровня доступа к данным средствами DBeaver была получена ER диаграмма созданной БД.

Полученная диаграмма представлена на рисунке 21.

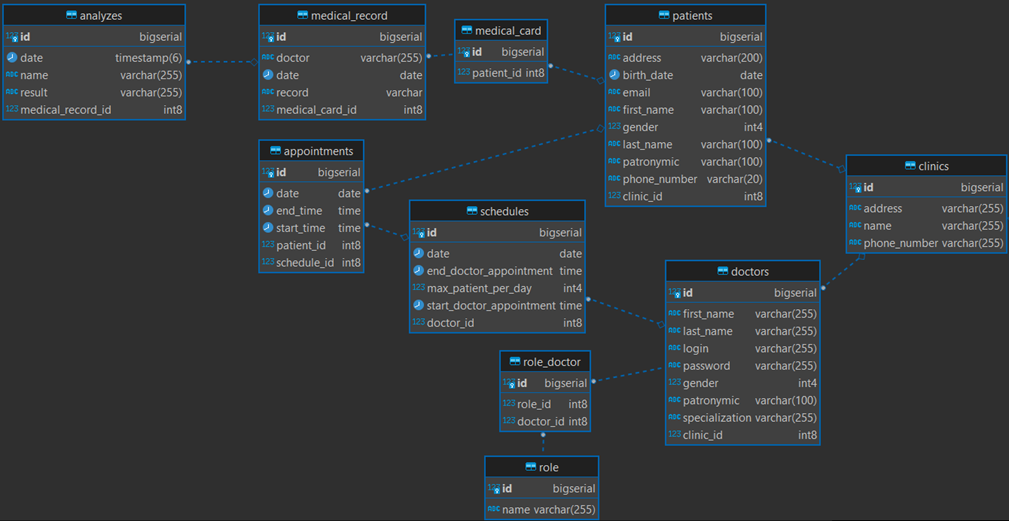


Рисунок 21 – Физическая структура БД.

## 5.4 Обеспечение целостности данных

## 5.4.1 Ограничения безопасности

В приложении используется авторизация на основе JWT-токенов. JWT (JSON Web Token) — это стандарт для безопасной передачи информации в зашифрованном виде между backend и frontend. Шифрование и подпись токенов выполняется с использованием алгоритма HS256. Логика шифрования и создания токенов обрабатывается в классе JwtUtil.

Листинг 11. Класс «JwtUtil».

@Component

public class JWTUtil {

@Value("${jwt\_secret}")

private String secret;

public String generateToken(String login) {

Date expirationDate = Date.from(ZonedDateTime.now().plusMonths(1).toInstant());

return JWT.create()

.withSubject("Person details")

.withClaim("login", login)

.withIssuedAt(new Date())

.withIssuer("ru.javavlsu.kb.esap")

.withExpiresAt(expirationDate)

.sign(Algorithm.HMAC256(secret));

}

public String validateToken(String token) throws JWTVerificationException {

JWTVerifier verifier = JWT.require(Algorithm.HMAC256(secret))

.withSubject("Person details")

.withIssuer("ru.javavlsu.kb.esap")

.build();

DecodedJWT jwt = verifier.verify(token);

return jwt.getClaim("login").asString();

}

}

## 5.4.2 Ограничения целостности

В процессе разработки были приняты меры и установлены ограничения целостности данных на нескольких уровнях: уровне базы данных и уровне приложения.

Ограничения на уровне базы данных:

* Уникальность: для обеспечения уникальности данных в базе данных были применены ограничения уникальности на определенные столбцы таблиц.
* Внешние ключи: использовались внешние ключи для связи данных между таблицами и обеспечения ссылочной целостности.

Ограничения на уровне приложения:

* Валидация данных: при обработке входящих запросов приложение выполняет валидацию данных, чтобы убедиться, что они соответствуют определенным правилам и форматам.
* Бизнес-правила: реализованы бизнес-правила, которые определяют допустимые операции и состояния данных в системе.

## 5.4.3 Организация распределенных транзакций

В проекте используется обработка распределенных транзакций с помощью механизма транзакций Spring. Аннотация @Transactional применяется к методам или классам и указывает, что данные операции должны быть выполнены в рамках транзакции.

В некоторых случаях, когда операция только читает данные из базы данных и не изменяет их, можно использовать аннотацию @Transactional с параметром readOnly=true. Это указывает, что транзакция является только для чтения и не будет выполнять никаких записей или изменений в базе данных. Установка readOnly=true может повысить производительность, так как база данных может оптимизировать запросы, зная, что они только для чтения.

Листинг 12. Пример обработки транзакции.

@Transactional(readOnly = true)

public MedicalCard getMedicalCardByPatient(Patient patient, String specializationDoctor) throws NotFoundException {

if(specializationDoctor != null && !specializationDoctor.isBlank()){

return medicalCardRepository

.findMedicalCardByPatientAndMedicalRecordSpecializationDoctor(patient, specializationDoctor)

.orElseThrow(() -> new NotFoundException("Medical Card not found"));

}

return medicalCardRepository.findByPatientOrderByMedicalRecordDateDesc(patient)

.orElseThrow(() -> new NotFoundException("Medical Card not found"));

}

## 5.5 Развертывание системы

Также была разработана диаграмма развертывания системы с целью наглядного представления компонентов и аппаратного обеспечения в системе. Данная диаграмма отображает распределение компонентов между серверами и другими устройствами, а также их взаимодействие друг с другом. Диаграмма представлена на рисунке 22.

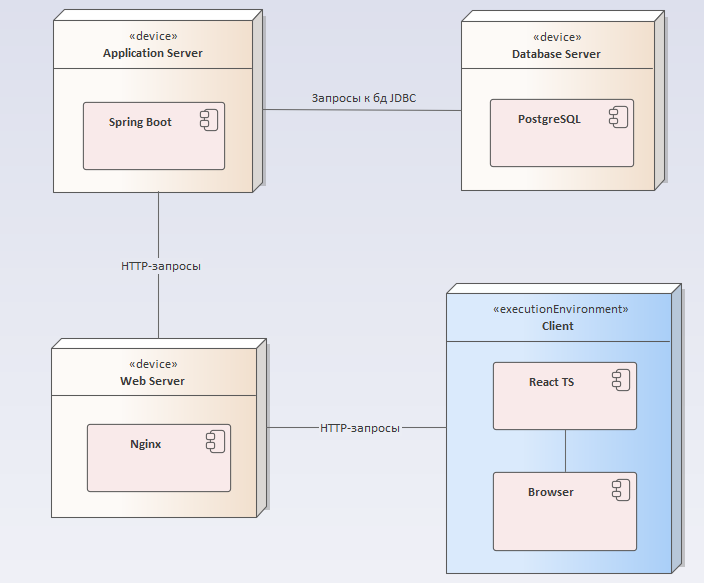


Рисунок 22 – Диаграмма развертывания.

## 5.6 Вывод

На данном этапе была выполнена реализация системы. По итогу была получена полностью готовая система, удовлетворяющая всем ранее описанным требованиям.

# **6 ИНСТРУКЦИЯ ПО УСТАНОВКЕ ПРИЛОЖЕНИЯ**

Для установки и запуска проекта требуются следующие зависимости:

Backend:

* Java 17
* PostgreSQL

Frontend:

* Node.js
* npm

Процесс установки и запуска проекта, следующий:

1. Склонируйте или скачайте исходный код проекта из репозитория: <https://github.com/javavlsu/course-work-team-1917-karabanov-bulanov>
2. Перед настройкой подключения к базе данных, создайте базу данных в вашей установленной СУБД. В файле "application.properties", который находится в каталоге "/backend/esap/src/main/resources/", укажите созданную базу данных, логин и пароль для PostgreSQL. В файле указаны значения по умолчанию. Пример настроек подключения к базе данных:

* spring.datasource.url=jdbc:postgresql://localhost:5432/esap
* spring.datasource.username=postgres
* spring.datasource.password=postgres

1. Соберите проект с помощью Maven. Откройте консоль в каталоге "/backend/esap/" и выполните команду "mvn package".
2. В каталоге "/backend/esap/target/" появится файл "esap-0.0.1-SNAPSHOT.jar". Запустите его с помощью команды "java -jar esap-0.0.1-SNAPSHOT.jar".

Теперь backend часть проекта запущена. Далее перейдем к запуску frontend части:

1. В каталоге "/frontend/esap-admin" откройте консоль и выполните команду "npm install".
2. После завершения установки выполните команду "npm run start" в том же каталоге.

После выполнения этих шагов вы сможете получить доступ к проекту по адресу "<http://localhost:3000/>".

Также проект имеет поддержку docker-compose. Для развертывания приложения через Docker Compose выполните следующие шаги:

1. Создайте файл с именем "docker-compose.yml" и скопируйте в него приведенный выше код.
2. Убедитесь, что у вас установлен Docker и Docker Compose.
3. В командной строке перейдите в каталог, содержащий файл "docker-compose.yml".
4. Выполните команду "docker-compose up -d" для запуска контейнеров приложения в фоновом режиме. Docker Compose автоматически загрузит и создаст все необходимые образы и контейнеры.
5. Подождите, пока контейнеры загрузятся и запустятся. Вы можете проверить их статус с помощью команды "docker-compose ps". Убедитесь, что все контейнеры имеют статус "Up".
6. После успешного запуска приложение будет доступно по следующим адресам:

* Frontend: <http://localhost:3000>
* Backend: <http://localhost:8080>
* PostgreSQL (db): http://localhost:15432
* pgAdmin: <http://localhost:5050>

Теперь вы можете использовать приложение, запущенное в контейнерах Docker, для разработки или тестирования.

# **7 ОРГАНИЗАЦИЯ УПРАВЛЕНИЯ ПРОЕКТОМ**

## 7.1 Распределение ролей в команде/зоны ответственности

* Руководитель команды. Full-stack разработчик – Карабанов Алексей Анатольевич.
* Заместитель руководителя команды. Full-stack разработчик – Буланов Владислав Александрович.

## 7.3 Календарный план работ

Был создан календарный план работ в виде диаграммы Ганта с указанием задач и временным периодом их выполнения. Календарный план представлен на рисунке 23.

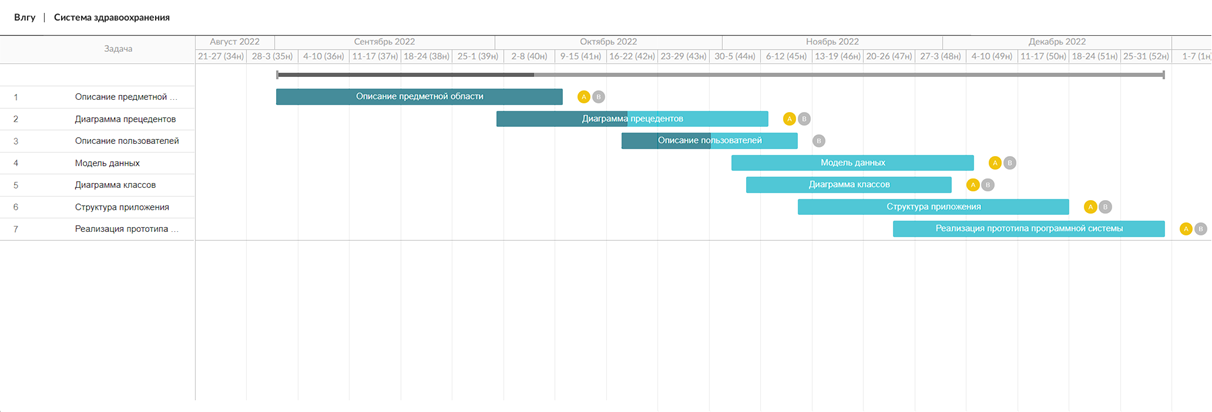


Рисунок 23 – Диаграмма Ганта.

В процессе организации управления проектом были выполнены следующие задачи: определение целей и требований проекта, планирование и распределение ресурсов, назначение ролей и определение ответственностей в команде, разработка графика работ и мониторинг прогресса проекта.

В результате этих усилий были достигнуты следующие ключевые результаты в организации управления проектом:

* Определение конкретных целей и требований проекта, что помогло команде разработчиков иметь ясное представление о его области и целях.
* Разработка детального плана проекта, включающего этапы работ, сроки и ресурсы. Это обеспечило структурированность и системность в ходе выполнения проекта.

В результате этих мероприятий команда смогла более эффективно управлять проектом, иметь ясное представление о его целях и требованиях, а также следить за прогрессом выполнения работ и ресурсами, необходимыми для достижения поставленных целей.

# **8 НАГРУЗОЧНОЕ ТЕСТИРОВАНИЕ**

Нагрузочное тестирование выполняется для оценки производительности сервера и выявления потенциальных проблем. При расчете ожидаемой нагрузки системы было принято решение обеспечить среднюю нагрузку в размере 300 000 пользователей. Для тестирования были выбраны основные бизнес-процессы, включая создание расписания для врача, запись пациента к врачу и прием пациента. Все эти процессы были объединены в одной группе тестирования.

Для выполнения нагрузочного тестирования система была разделена на три компьютера. На первом компьютере запускался сервер с приложением, на втором - база данных, а на третьем - инструмент JMeter с настроенным тест-планом. Тест-план, описывающий процессы тестирования, представлен на рисунке 24.

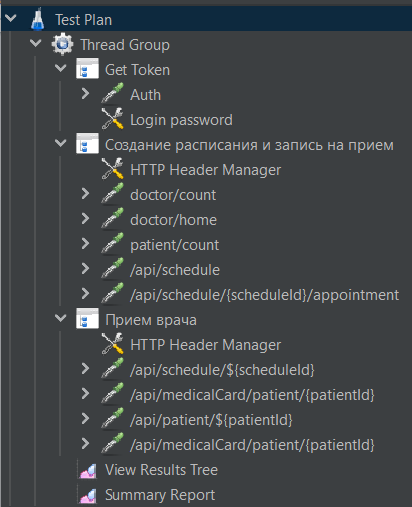


Рисунок 24 – Тест план бизнес-процесса.

Первый тест проводился на 5000 пользователей. Система работала не стабильно, некоторые запросы не обрабатывались вовсе. Результаты первого теста представлены на рисунках 25 – 28.

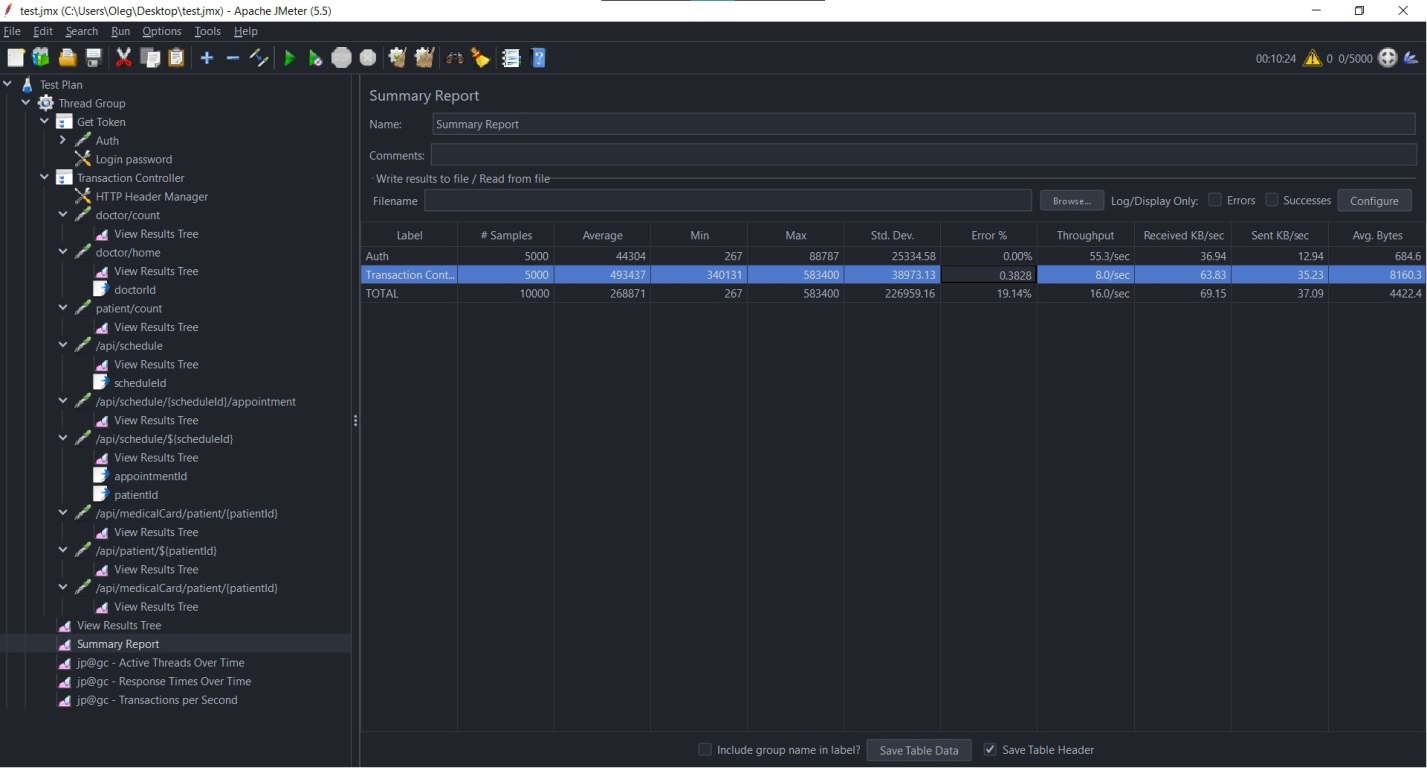


Рисунок 25 – Результат первого теста на 5000 пользователей.

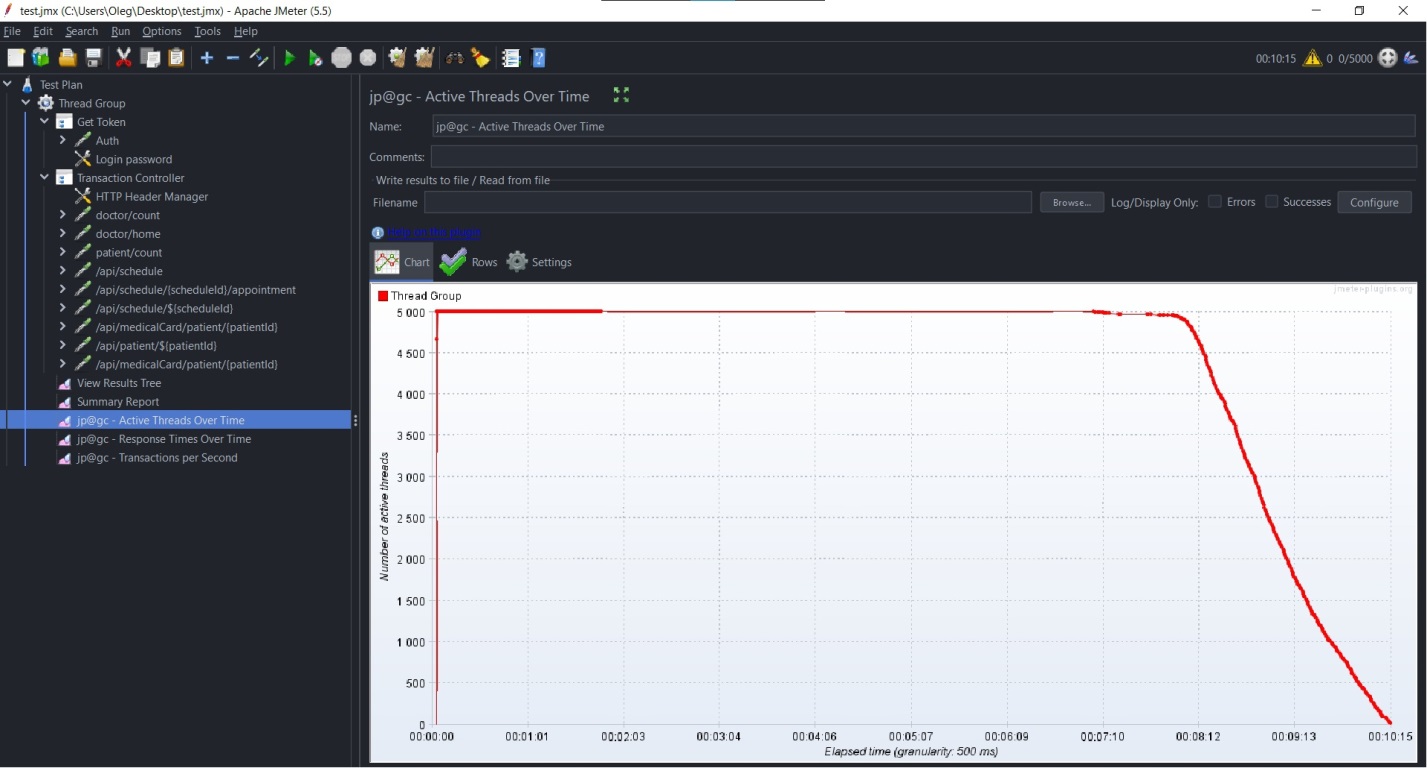


Рисунок 26 – Активные потоки.

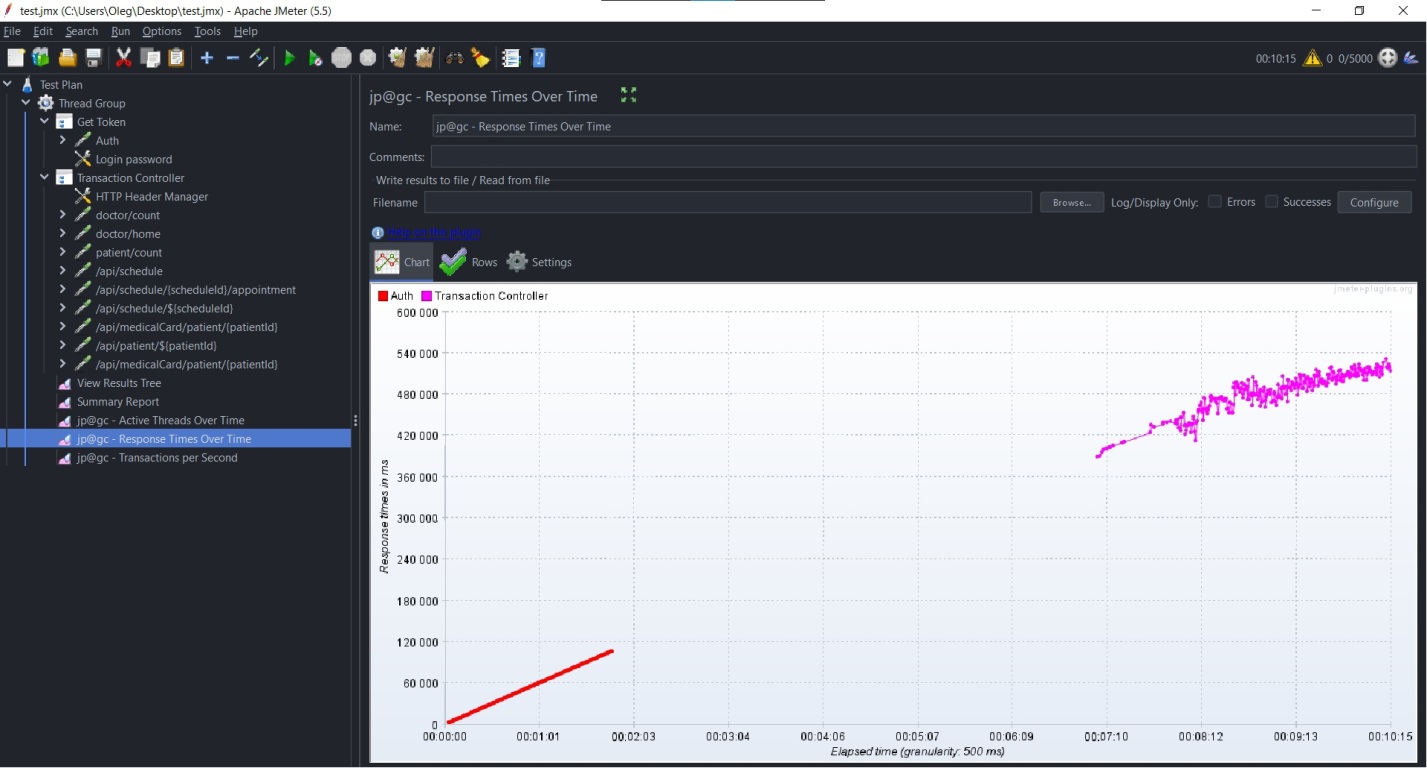


Рисунок 27 – Время ответа.

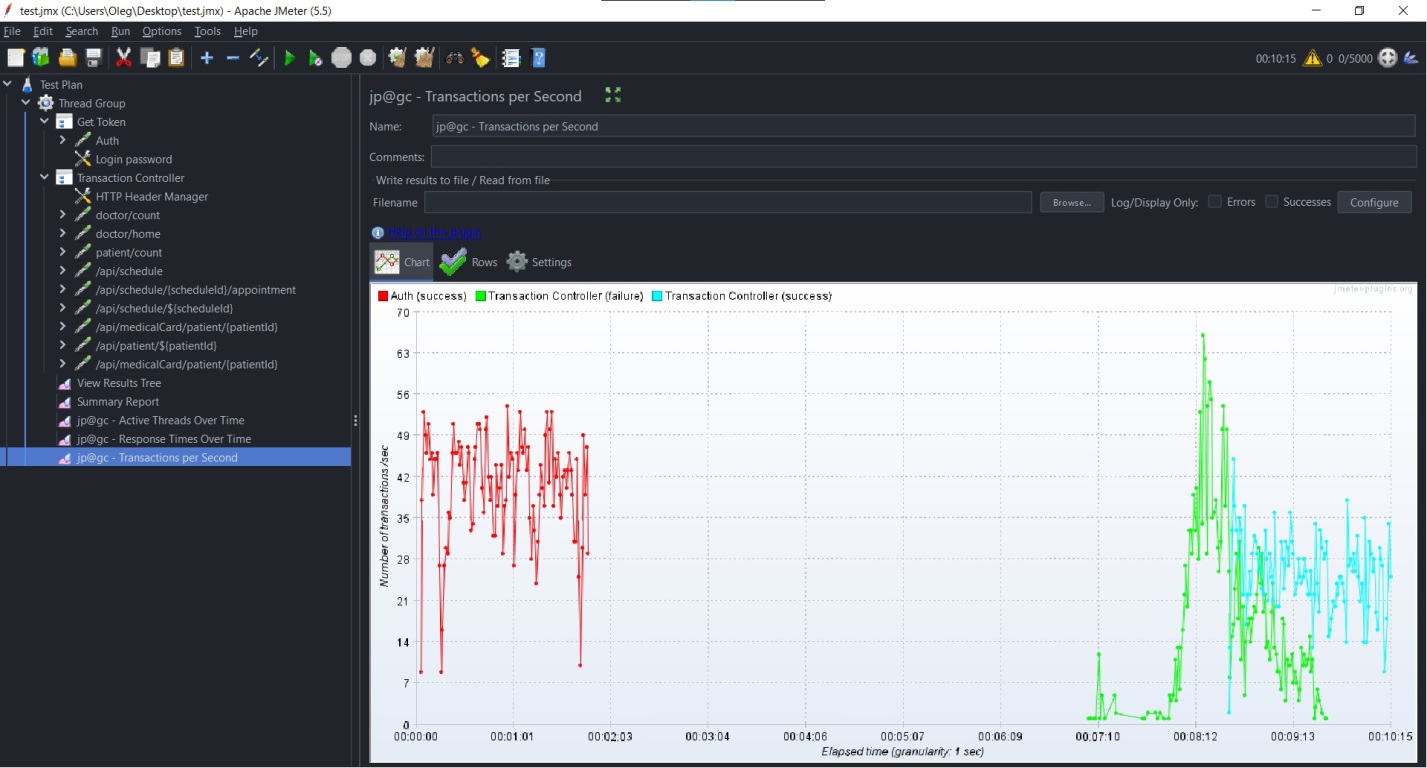


Рисунок 28 – Транзакций в секунду.

Система показала стабильную работу при 4000 пользователей. При 4500 пользователях была обнаружена ошибка «Connection time out» представленная на рисунке 29.

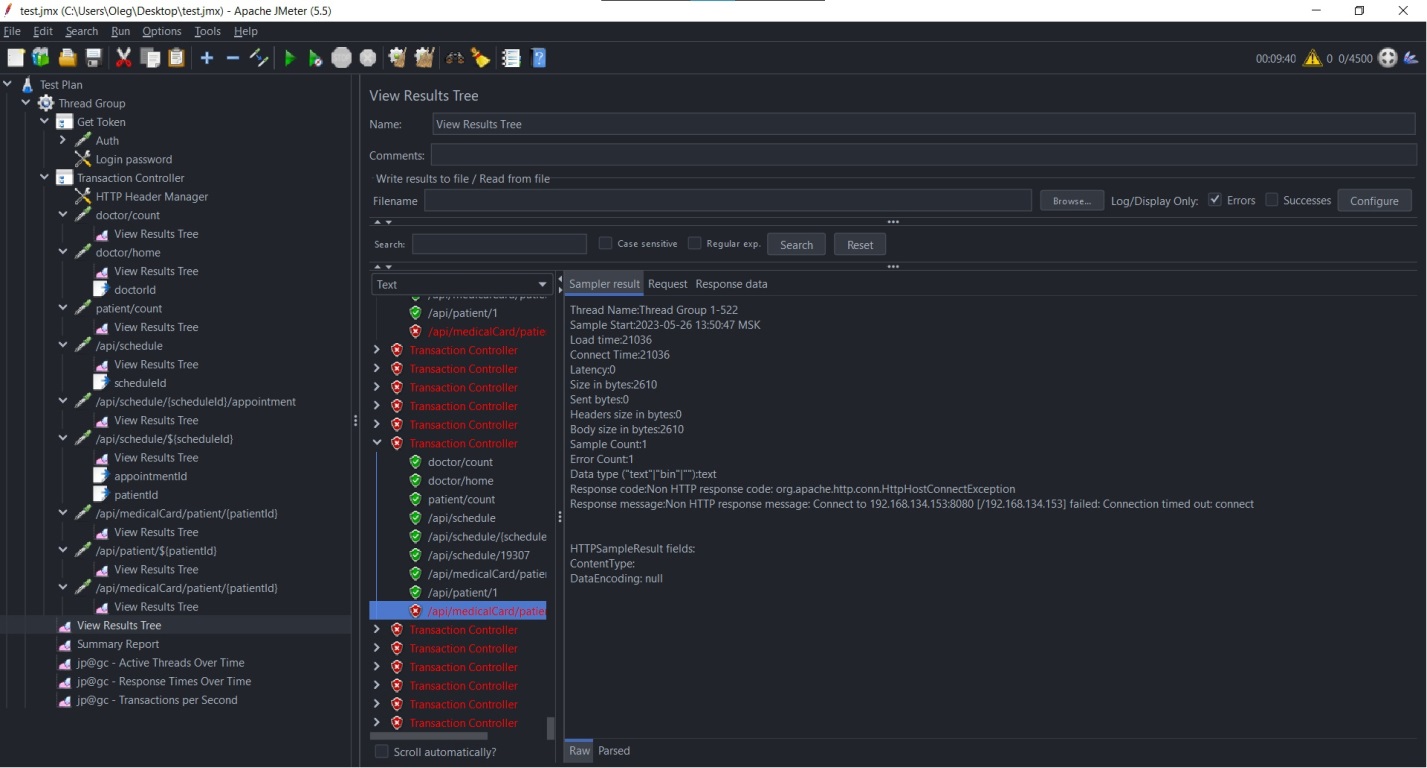


Рисунок 29 – Ошибка «Connection time out».

В процессе работы была обнаружена проблема, связанная с "N+1" запросами, которая приводит к долгой обработке запросов. Для решения этой проблемы были разработаны собственные SQL-запросы, исключившие использование ORM.

Однако, в ходе тестирования не удалось полностью достичь поставленных требований, особенно в отношении количества пользователей. Для достижения необходимого числа пользователей в размере 300 000, необходимо провести оптимизацию SQL-запросов и создать несколько экземпляров бэкенд-сервера для распределения нагрузки, например, с помощью прокси-сервера Nginx.

Дальнейшие усилия должны быть направлены на оптимизацию запросов и масштабирование системы, чтобы обеспечить требуемую производительность и удовлетворить ожидания пользователей.

# **9 ЗАКЛЮЧЕНИЕ**

В рамках данного проекта был разработан прототип программной системы «Единая система автоматизации поликлиник (ЕСАП)» на платформе Java EE c использованием фреймворка Spring Boot и библиотеки React TS для реализации клиентской части. Целью данного проекта было создание прототипа системы, которая позволяет эффективно управлять пациентами, записями, расписаниями и медицинскими данными.

В процессе разработки были определены требования к системе, проведен анализ существующих аналогов и выбраны технологии, наиболее подходящие для реализации поставленных задач. Были применены современные подходы и принципы разработки, такие как модульность, масштабируемость и безопасность, что позволило создать гибкую и надежную систему.

В ходе разработки были реализованы основные функциональные возможности системы.

Полученный прототип системы был протестирован, и в ходе тестирования выявлены и исправлены некоторые ошибки и недочеты. Были проведены тесты на производительность, которые показали хорошую отзывчивость системы даже при большом количестве одновременных запросов.

В заключение можно сказать, что разработанная система «Единая система автоматизации поликлиник» полностью соответствует поставленным целям и требованиям. Прототип системы готов к дальнейшей разработке и расширению функциональности, и может служить основой для создания полноценного коммерческого приложения.

**10 СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ**

1. Постановление Правительства РФ «О единой государственной информационной системе в сфере здравоохранения» от 09.02.2022 № 140 // Docs.cntd.ru — электронный фонд нормативно-технической и нормативно-правовой информации Консорциума «Кодекс» - 11.02.202 URL: <https://docs.cntd.ru/document/728122057> (дата обращения: 25.02.2023).
2. Приказ Министерства Здравоохранения РФ «Об утверждении Требований к государственным информационным системам в сфере здравоохранения субъектов Российской Федерации, медицинским информационным системам медицинских организаций и информационным системам фармацевтических организаций» от 24.12.2018 № 911н // Docs.cntd.ru — электронный фонд нормативно-технической и нормативно-правовой информации Консорциума «Кодекс» - 19.06.2019 URL: <https://docs.cntd.ru/document/552196746>
3. БАРС.Здравоохранение - МИС // Барс Груп URL: <https://bars.group/health/meditsinskaya-informatsionnaya-sistema>
4. ИНФОКЛИНИКА // «Смарт Дельта Системс» URL: <https://sdsys.ru/products/infoclinica>
5. МИС МЕДИАЛОГ // Компания «Пост Модерн Текнолоджи» (ПМТ) URL: <https://medialog.ru>
6. Learn Spring Boot [Электронный ресурс]: www.baeldung.com. – Режим доступа: <https://www.baeldung.com/spring-boot/>
7. Spring Framework - официальная документация [Электронный ресурс]: docs.spring.io. – Режим доступа <https://docs.spring.io/spring-framework/docs/current/reference/html/index.html>
8. JDK 19 Documentation [Электронный ресурс]: docs.oracle.com. – Режим доступа: <https://docs.oracle.com/en/java/javase/19/index.html>

# **ПРИЛОЖЕНИЕ А ИСХОДНЫЙ КОД ПРИЛОЖЕНИЯ**

Весь исходный код доступен по ссылке: <https://github.com/javavlsu/course-work-team-1917-karabanov-bulanov>

# **ПРИЛОЖЕНИЕ Б СКРИПТ СОЗДАНИЯ БАЗЫ ДАННЫХ**

create table clinics  
(  
 id bigserial not null,  
 address varchar(255) not null,  
 name varchar(255) not null,  
 phone\_number varchar(255) not null,  
 primary key (id)  
);  
  
create table doctors  
(  
 id bigserial not null,  
 first\_name varchar(255) not null,  
 last\_name varchar(255) not null,  
 login varchar(255),  
 password varchar(255),  
 gender integer not null check (gender<=2 AND gender>=1),  
 patronymic varchar(100) not null,  
 specialization varchar(255) not null,  
 clinic\_id bigint, primary key (id),  
 FOREIGN KEY (clinic\_id) REFERENCES clinics (id)  
);  
  
create table patients  
(  
 id bigserial not null,  
 address varchar(200),  
 birth\_date date not null,  
 email varchar(100),  
 first\_name varchar(100),  
 gender integer not null check (gender<=2 AND gender>=1),  
 last\_name varchar(100),  
 patronymic varchar(100),  
 phone\_number varchar(20),  
 clinic\_id bigint,  
 primary key (id),  
 FOREIGN KEY (clinic\_id) REFERENCES clinics (id)  
);  
  
create table medical\_card  
(  
 id bigserial not null,  
 patient\_id bigint,  
 primary key (id),  
 FOREIGN KEY (patient\_id) REFERENCES patients (id)  
);  
  
create table medical\_record  
(  
 id bigserial not null,  
 doctor varchar(255) not null,  
 date date not null,  
 record varchar(255),  
 medical\_card\_id bigint not null,  
 primary key (id),  
 FOREIGN KEY (medical\_card\_id) REFERENCES medical\_card (id)  
);  
  
create table analyzes  
(  
 id bigserial not null,  
 date timestamp(6),  
 name varchar(255),  
 result varchar(255),  
 medical\_record\_id bigint,  
 primary key (id),  
 FOREIGN KEY (medical\_record\_id) REFERENCES medical\_record (id)  
);  
  
create table schedules  
(  
 id bigserial not null,  
 date date,  
 end\_doctor\_appointment time,  
 max\_patient\_per\_day integer not null,  
 start\_doctor\_appointment time,  
 doctor\_id bigint,  
 primary key (id),  
 FOREIGN KEY (doctor\_id) REFERENCES doctors (id)  
);  
  
create table appointments  
(  
 id bigserial not null,  
 date date,  
 end\_time time,  
 start\_time time,  
 patient\_id bigint,  
 schedule\_id bigint,  
 primary key (id),  
 FOREIGN KEY (patient\_id) REFERENCES patients (id),  
 FOREIGN KEY (schedule\_id) REFERENCES schedules (id)  
);  
  
create table role (  
 id bigserial not null,  
 name varchar(255) not null,  
 primary key (id)  
);  
  
create table role\_doctor (  
 id bigserial not null,  
 role\_id bigint not null,  
 doctor\_id bigint not null,  
 FOREIGN KEY (role\_id) REFERENCES role (id),  
 FOREIGN KEY (doctor\_id) REFERENCES doctors (id),  
 primary key (id)  
);