

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РФ
Федеральное государственное бюджетное образовательное учреждение
высшего образования
«ПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»
ПОЛИТЕХНИЧЕСКИЙ ИНСТИТУТ

Факультет
вычислительной техники

Кафедра
МОиПЭВМ

Направление подготовки 09.03.04 «Программная инженерия»

Направленность Программная инженерия

БАКАЛАВРСКАЯ РАБОТА

на тему

Разработка автоматизированной
информационной системы сопровождения
лечебно-диагностического процесса частного
медицинского кабинета

Студент

(подпись, дата)

Гуро Николай Валерьевич

(ФИО полностью)

Руководитель

(подпись, дата)

Самуйлов С.В.

(фамилия, инициалы)

Нормоконтролёр

(подпись, дата)

Такташкин Д.В.

(фамилия, инициалы)

Работа допущена к защите (протокол заседания кафедры от 03.06.2026г. № 12)

Заведующий кафедрой

(подпись)

Козлов А.Ю.

(фамилия, инициалы)

Работа защищена с отметкой _____ (протокол заседания ГЭК от _____ № _____)

Секретарь ГЭК

(подпись)

Попова Н.А.

(фамилия, инициалы)

Пенза, 2026

Федеральное государственное бюджетное образовательное учреждение
высшего образования
ПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
ПОЛИТЕХНИЧЕСКИЙ ИНСТИТУТ

Факультет
Вычислительной техники

Кафедра
МО и ПЭВМ

«Утверждаю»
Заведующий кафедрой
А.Ю. Козлов
«26» декабря 2025 г

**ЗАДАНИЕ
НА ВЫПУСКНУЮ КВАЛИФИКАЦИОННУЮ РАБОТУ
БАКАЛАВРА**

1. Студент Гуро Николай Валерьевич гр. 22ВП2 факультета ВТ
<Фамилия, имя, отчество полностью>

2. Тема работы Разработка автоматизированной информационной системы
сопровождения лечебно-диагностического процесса частного медицинского кабинета

Тема утверждена приказом ПГУ № 1583/о от "30" декабря 2025 г.

3. Руководитель работы к.т.н., доцент, доцент кафедры «МО и ПЭВМ» Самуйлов С.В.

4. Задание на работу (назначение разработки, исходные данные и т.п.)

1) Назначение: разработка клиент-серверной веб-ориентированной
информационной системы онлайн-записи пациентов в частный медицинский кабинет.

2) Среда выполнения: Windows 11.

Языковые средства разработки: Go, Javascript, SQL.

Инструментальные средства: Visual Studio Code, pgAdmin 4.

Технологии разработки: клиент-серверная архитектура, REST API.

СУБД: PostgreSQL.

3) Основные функции приложения:

а) Регистрация и авторизация клиентов;

б) Просмотр списка врачей и их расписания;

в) Создание записи на приём с выбором врача, даты и времени;

г) Отмена записи на приём;

д) Логирование действий пользователей.

5. Перечень подлежащих разработке вопросов:

<i>1) Анализ предметной области и постановка задачи;</i>
<i>2) Проектирование информационной системы;</i>
<i>3) Реализация программных средств;</i>
<i>4) Тестирование и контроль качества программного обеспечения;</i>
<i>5) Оформление пояснительной записки.</i>

6. Календарный график выполнения работы

№ п/п	Наименование этапов работы	Объем работы	Срок выполнения	Подпись руководителя
1	Анализ предметной области и требований к системе	10%	07.04.2026 – 13.04.2026	
2	Проектирование программных средств	20%	14.04.2026 – 25.04.2026	
3	Реализация программных средств	30%	27.04.2026 – 11.05.2026	
4	Тестирование программных средств	20%	12.05.2026 – 21.05.2026	
5	Оформление пояснительной записки	20%	22.05.2026 – 02.06.2026	

Дата выдачи задания «26» декабря 2025г.

Руководитель бакалаврской работы _____ 26.12.2025 г.
(подпись, дата)

Задание к исполнению принял студент _____ 26.12.2025 г.
(подпись, дата)

Работу к защите допустить
Декан ФВТ _____ Л.Р. Фионова
(подпись, дата)

Реферат

Пояснительная записка содержит 106 листов, 18 рисунков, 11 таблиц, 22 использованных источника и 4 приложения.

КЛИЕНТ-СЕРВЕР, ВЕБ-ПРИЛОЖЕНИЕ, GOLANG, JWT, POSTGRESQL, JAVASCRIPT, ИНФОРМАЦИОННАЯ СИСТЕМА, ОНЛАЙН-ЗАПИСЬ, АВТОРИЗАЦИЯ, UML, PLANTUML

Предмет работы: разработка веб-ориентированной информационной системы онлайн-записи пациентов в частный медицинский кабинет.

Объект работы: автоматизация процесса записи на прием и управления расписанием врачей в медицинской организации.

Цель работы: разработка клиент-серверной информационной системы сопровождения лечебно-диагностического процесса частного медицинского кабинета с использованием современных веб-технологий.

Технология разработки: серверная часть на языке Go (фреймворк Gin), клиентская часть на JavaScript (нативный JS), база данных PostgreSQL, авторизация через JWT-токены.

Результаты работы: спроектирована и разработана информационная система, позволяющая пациентам регистрироваться, просматривать список врачей, выбирать свободное время и создавать записи на прием.

Область применения: частные медицинские клиники, специализированные центры лечения боли, стоматологии.

					ПГУ 09.03.04 – 08BP222.06 ПЗ			
Изм.	Лист	№ докум.	Подпись	Дата				
Разраб.		Гуро Н.В.			Разработка автоматизированной информационной системы сопровождения лечебно-диагностического процесса частного медицинского кабинета Пояснительная записка	Лит.	Лист	Листов
Провер.		Самуйлов С.В.					4	106
						Группа 22ВП2		
Н. Контр		Такташкин Д.В.						
Утверд.		Козлов А.Ю.						

Содержание

Введение.....	7
1 Постановка задачи и анализ аналогов разрабатываемой системы	9
1.1 Постановка задачи.....	9
1.2 Анализ информационных систем аналогичного назначения	11
1.2.1 DocPlanner.....	11
1.2.2 Yclients	12
1.2.3 Medesk	12
1.2.4 Инфоклиника	13
1.3 Сравнение разрабатываемой информационной системы и систем аналогичного назначения	14
1.4 Выбор средств для разработки и проектирования информационной системы.....	15
2 Проектирование информационной системы	18
2.1 Проектирование инфологической модели базы данных.....	18
2.2 Проектирование программного обеспечения.....	21
2.2.1 Диаграмма вариантов использования	21
2.2.2 Диаграмма деятельности	22
2.2.3 Диаграмма последовательности	25
2.2.4 Диаграмма состояний	26
3 Реализация информационной системы	28
3.1 Реализация структуры данных (PostgreSQL)	28
3.2 Реализация серверной части на Go	29
3.3 Реализация пользовательского интерфейса (Frontend)	32
3.4 Диаграмма развертывания.....	33
4 Тестирование информационной системы.....	35
4.1 Функциональное тестирование.....	35
4.2 Тестирование метрикой Холстеда.....	36
4.3 Системное тестирование	37

5 Планирование и оценка стоимости проекта.....	40
Заключение	45
Список использованных источников	49
Приложение А (справочное) Глоссарий	52
Приложение Б (обязательное) Код программы.....	54
Приложение В (рекомендуемое) Результаты тестирования.....	100
Приложение Г (рекомендуемое) Акт о внедрении результатов работы	105

Введение

Сегодня информационные системы играют важную роль в управлении медицинскими организациями. Цель разработки информационной системы для частного медицинского кабинета заключается в оптимизации записи пациентов, повышении доступности услуг и упрощении работы администраторов.

Актуальность темы обусловлена тем, что существующие на рынке коммерческие решения ориентированы преимущественно на крупные сети клиник и обладают избыточным функционалом и высокой стоимостью подписки.

Разработка информационной системы онлайн-записи предназначена для развития эффективности управления клиникой. Она позволяет упростить работу персонала, ускоряет процесс записи пациентов и предоставляет пациентам удобный инструмент для выбора врача и времени приема.

Предметом выполнения работы является разработка информационной системы онлайн-записи в частный медицинский кабинет.

Для разработки информационной системы были поставлены следующие цели и задачи:

- определить задачи и требования на разработку информационной системы онлайн-записи;
- исследовать аналогичные информационные системы и определить их преимущества и недостатки;
- выполнить проектирование информационной системы (база данных, UML-диаграммы [1,2]);
- разработать серверную часть на языке Go и клиентскую часть на JavaScript;
- провести тестирование программы и проанализировать качество кода;
- выполнить планирование проекта и оценку его стоимости.

Выпускная квалификационная работа состоит из введения, пяти разделов, заключения, списка использованных источников и четырёх приложений. В первом разделе рассматривается постановка задачи и проводится анализ информационных систем аналогичного назначения. Второй раздел посвящён проектированию информационной системы: построению инфологической модели базы данных и UML-диаграмм. В третьем разделе описана реализация серверной и клиентской частей системы. Четвёртый раздел содержит результаты функционального и системного тестирования, а также оценку качества программного кода методом Холстеда. В пятом разделе выполнено планирование проекта, оценка его стоимости и анализ рисков.

1 Постановка задачи и анализ аналогов разрабатываемой системы

1.1 Постановка задачи

Частный медицинский кабинет является организацией, предоставляющей услуги по диагностике и лечению болевых синдромов различного происхождения. Пациенты могут обратиться в клинику с жалобами на головные боли, боли в спине, суставах или другие виды хронической и острой боли. В клинике работают квалифицированные специалисты: неврологи, ортопеды, физиотерапевты, реабилитологи.

В данной предметной области необходимо учитывать следующие аспекты:

- пациенты – люди, которые обращаются в клинику для получения медицинской помощи, имеют различные болевые синдромы и нуждаются в квалифицированном лечении;
- врачи – квалифицированные специалисты, которые проводят диагностику, лечение и реабилитацию пациентов;
- услуги – различные виды диагностики (МРТ, КТ, УЗИ), консультаций, процедур (блокады, физиотерапия) и лечения, которые предоставляет клиника;
- запись на прием: пациент может выбрать врача, удобную дату и время через веб-интерфейс;
- учет записей – система должна фиксировать все записи, хранить историю обращений пациента, позволять отменять запись;
- авторизация и безопасность – доступ к функционалу записи должен быть только у авторизованных пользователей.

Для учета выше приведенных аспектов требуется разработать информационную систему онлайн-записи. Задачи на разработку представлены ниже:

- регистрация и авторизация пациентов: безопасный вход по номеру телефона и паролю с использованием JWT-токенов, при регистрации – указание

ФИО, номера телефона и пароля, соответствующего требованиям безопасности (минимум 8 символов, цифра и специальный символ);

- учет врачей – ведение базы данных врачей с указанием ФИО, специализации, фотографии и опыта работы, с возможностью для администратора клиники добавлять, редактировать и удалять записи о врачах;

- просмотр расписания и свободных слотов: отображение списка врачей и доступных временных слотов (с 7:00 до 19:00 с шагом 30 минут) при выборе врача и даты, с исключением уже занятых интервалов;

- создание записи на прием: выбор врача, даты, времени и указание причины обращения с проверкой занятости выбранного времени перед сохранением и уведомлением пациента при успешном создании записи;

- просмотр и удаление своих записей: возможность для пациента просматривать все свои активные записи (дата, время, врач) и отменять их при необходимости с освобождением временного слота;

- логирование действий: фиксация в серверных логах всех важных действий в системе (регистрация, вход, создание записи, удаление записи) с указанием времени, пользователя и результата операции;

- современный и адаптивный интерфейс: интуитивно понятный и адаптивный для различных устройств (ПК, планшеты, смартфоны) пользовательский интерфейс с привлекательным дизайном;

- отчетность – предоставление администратору возможности просматривать статистику по записям (количество записей по врачам, загруженность по дням) через логи или дополнительные SQL-запросы.

Разработка информационной системы онлайн-записи для частного медицинского кабинета позволит оптимизировать его работу, ускорить процесс записи, уменьшить нагрузку на администраторов и повысить удовлетворенность пациентов. Использование традиционных способов записи на прием, таких как телефонные звонки или ведение бумажных журналов, увеличивает вероятность ошибок и конфликтов в расписании, в частности двойного бронирования одного и того же временного слота.

1.2 Анализ информационных систем аналогичного назначения

Для разработки информационной системы нужно провести анализ аналогичных систем, чтобы определить их сильные и слабые стороны.

На сегодняшний день разработано и внедрено множество информационных систем для записи в медицинские учреждения, отличающихся функционалом, стоимостью, технологической платформой и бизнес-процессами. Для анализа были выбраны четыре наиболее популярные системы: DocPlanner, Yclients и Medesk, Инфоклиника.

1.2.1 DocPlanner

DocPlanner – это международная платформа для записи пациентов к врачам, которая объединяет в себе элементы медицинской информационной системы, CRM и маркетплейса. Платформа позволяет пациентам находить врачей по рейтингу, отзывам и специализации, записываться на прием онлайн, получать напоминания.

Особенностями данной информационной системы являются:

- наличие личного кабинета пациента, где можно просмотреть историю посещений, получить электронные рецепты;
- возможность оставлять отзывы о врачах и оценивать качество обслуживания;
- интеграция с электронными медицинскими картами;
- автоматические напоминания о записи через SMS и push-уведомления;
- рейтинговая система врачей, которая помогает пациентам выбирать специалиста.

Стоимость внедрения системы для клиники начинается от 3000 рублей в месяц. Помимо основного функционала, DocPlanner предоставляет врачам инструменты для управления расписанием и снижения количества пропущенных визитов, что позволяет более эффективно использовать рабочее время специалиста. Для клиник также доступна базовая аналитика по количеству

записей и источникам трафика пациентов, что упрощает оценку эффективности использования платформы.

1.2.2 Yclients

Yclients – это российская CRM-система для медицинских центров, салонов красоты и других сфер услуг. Система предоставляет широкий функционал для управления бизнесом: запись клиентов, финансы, маркетинговые рассылки, бонусную программу.

Возможностями данной информационной системы являются:

- онлайн-запись через сайт, мобильное приложение, социальные сети и мессенджеры;
- автоматическое построение расписания специалистов;
- ведение клиентской базы с историей посещений и медицинскими картами;
- интеграция с IP-телефонией и чат-ботами;
- наличие мобильного приложения для администраторов и клиентов.

Стоимость внедрения начинается от 2500 рублей в месяц и зависит от количества сотрудников.

1.2.3 Medesk

Medesk – это веб-ориентированная медицинская информационная система, которая помогает автоматизировать административные и лечебно-диагностические процессы. Система имеет версии для медицинских центров, стоматологий, клиник и частных врачей.

Особенностями данной информационной системы являются:

- наличие личного кабинета пациента для записи на прием, оплаты услуг и просмотра истории лечения;
- электронная медицинская карта с возможностью прикрепления снимков и документов;

- автоматическая отправка результатов обследований на электронную почту пациента;
- формирование ежедневных отчетов о деятельности клиники;
- интеграция с лабораторными информационными системами.

Стоимость использования начинается от 890 рублей в месяц.

Однако у данной информационной системы отсутствует полноценное мобильное приложение и интерфейс может быть излишне сложным для небольших клиник.

1.2.4 Инфоклиника

Инфоклиника – это российская медицинская информационная система, разрабатываемая компанией «Смарт Дельта Системс» и предназначенная для автоматизации работы поликлиник, больниц, медицинских центров и сетей частных клиник.

Особенностями данной информационной системы являются:

- ведение электронной картотеки и медицинской карты пациента с полной историей посещений и назначений;
- интеграция с Госуслугами, внешними лабораториями и страховыми организациями;
- наличие личного кабинета пациента и врача (веб, iPhone, Android);
- поддержка системы управления сетью филиалов с распределёнными базами данных;
- автоматические SMS- и push-уведомления о записи и изменениях расписания.

Система входит в Единый реестр российских программ для ЭВМ и БД и в реестр медицинских изделий Росздравнадзора, что подтверждает соответствие требованиям законодательства РФ в области обработки медицинских данных.

Вместе с тем Инфоклиника ориентирована на крупные медицинские организации и сети клиник: внедрение системы требует значительных

финансовых затрат и обучения персонала по каждому модулю, что делает её избыточной для частного медицинского кабинета с одним-двумя врачами.

1.3 Сравнение разрабатываемой информационной системы и систем аналогичного назначения

Для проведения сравнительного анализа информационных систем были выбраны следующие критерии:

- функциональные возможности (набор ключевых функций для записи и учета);
- технологическая платформа;
- наличие системы лояльности и бонусов;
- стоимость внедрения и использования;
- возможность кастомизации под конкретную клинику.

Эти критерии помогают определить наиболее уязвимые места в существующих решениях и обосновать необходимость разработки собственной системы. Результаты сравнения представлены в таблице 1.

Сравнение аналогичных информационных систем показало, что большинство имеют высокую стоимость, содержат избыточный функционал (складской учет, сложная CRM, бонусные программы), который не требуется для небольшой специализированной клиники. Также многие системы сложны в настройке и требуют длительного внедрения.

Для разрабатываемой информационной системы произведена примерная стоимость использования (расчет приведен в разделе 5)

Таблица 1 – Сравнение информационных систем

Система	Функциональные возможности	Платформа	Система лояльности	Стоимость
DocPlanner	Личный кабинет, отзывы, рейтинг врачей, интеграция с медкартами, напоминания	Веб, мобильное приложение	Нет	от 3000 руб/мес

Продолжение таблицы 1

Система	Функциональные возможности	Платформа	Система лояльности	Стоимость
Yclients	Онлайн-запись, CRM, складской учет, рассылки, бонусная программа, мобильное приложение	Веб, мобильное приложение	Да	от 2500 руб/мес
Medesk	Личный кабинет, электронная медкарта, отправка результатов, ежедневные отчеты	Веб-приложение	Нет	от 890 руб/мес
Инфоклиника	Электронная медкарта, интеграция с Госуслугами и лабораториями, личный кабинет пациента/врача, управление сетью филиалов	Веб, мобильные приложения (iOS/Android)	Да	Высокая, индивидуальный расчёт
Разрабатываемая система	Регистрация, авторизация, просмотр врачей, выбор свободного времени, создание/удаление записей, логирование	Веб-приложение (Go + JS)	Нет	от 1100 руб/мес (оценка)

Результаты сравнения, приведённые в таблице 1, подтверждают целесообразность разработки собственной информационной системы, ориентированной на нужды небольшого частного медицинского кабинета.

1.4 Выбор средств для разработки и проектирования информационной системы

Для разработки информационной системы были выбраны следующие программные средства.

Для серверной части выбран язык программирования Go (Golang) и фреймворк Gin.

Go – это современный компилируемый язык программирования, разработанный компанией Google. Преимуществами Go являются высокая производительность (сопоставима с C++), встроенная поддержка конкурентности (горутины), простота синтаксиса, быстрая компиляция, богатая стандартная библиотека для работы с сетью и HTTP.

Gin – это легковесный веб-фреймворк для Go, предоставляющий удобную маршрутизацию, middleware для логирования и CORS, а также высокую производительность (один из самых быстрых фреймворков для Go).

Для клиентской части выбран JavaScript (нативный) и HTML5/CSS3.

JavaScript является стандартом для создания динамических веб-интерфейсов[3]. Использование нативного JavaScript без тяжелых фреймворков (React, Vue, Angular) позволяет снизить объем загружаемого кода, упростить отладку и обеспечить быструю загрузку страниц.

CSS3 используется для стилизации и адаптивного дизайна [4 - 7].

Для базы данных выбрана PostgreSQL.

PostgreSQL – это мощная объектно-реляционная СУБД с открытым исходным кодом. Преимуществами являются поддержка сложных запросов, целостность данных через внешние ключи, надежность, масштабируемость, а также встроенная поддержка JSON-данных [8 - 10].

Для аутентификации используется JWT (JSON Web Token).

JWT позволяет реализовать безсессионную аутентификацию: сервер выпускает подписанный токен, который клиент предъявляет при каждом запросе. Токен содержит идентификатор пользователя и время истечения срока действия [11].

Для проектирования информационной системы и построения UML-диаграмм использовались следующие средства: PlantUML (текстовое описание диаграмм с последующей генерацией изображений), Erwin Data Modeler (для инфологической модели базы данных), ProjectLibre (для диаграммы Ганта и планирования проекта).

PlantUML – это инструмент с открытым исходным кодом, позволяющий описывать UML-диаграммы на простом текстовом языке и автоматически генерировать изображения. Этот подход удобен для версионирования диаграмм в системе контроля версий (например, Git) и для коллективной работы [12].

ProjectLibre – программное средство для управления проектами, являющееся бесплатным аналогом Microsoft Project. Оно позволяет создавать проекты, составлять иерархический список задач, назначать ресурсы, строить диаграмму Ганта и формировать отчеты по затратам [13, 14].

Для разработки кода использовались среда разработки Visual Studio Code (для фронтенда) и GoLand (для бэкенда). Обе среды предоставляют удобные инструменты для написания кода, отладки, интеграции с Git и управления зависимостями [15 - 17].

2 Проектирование информационной системы

2.1 Проектирование инфологической модели базы данных

Инфологической моделью данных называют обобщенное неформальное описание создаваемой базы данных, выполненное с использованием естественного языка, таблиц, графиков и других средств, понятных всем участникам процесса проектирования [18].

Целью такого моделирования является усовершенствование и повышение эффективности поиска, сбора и отображения информации, которая хранится в базе данных.

Инфологическая модель, представленная на рисунке 1, была спроектирована в среде Erwin Data Modeler.

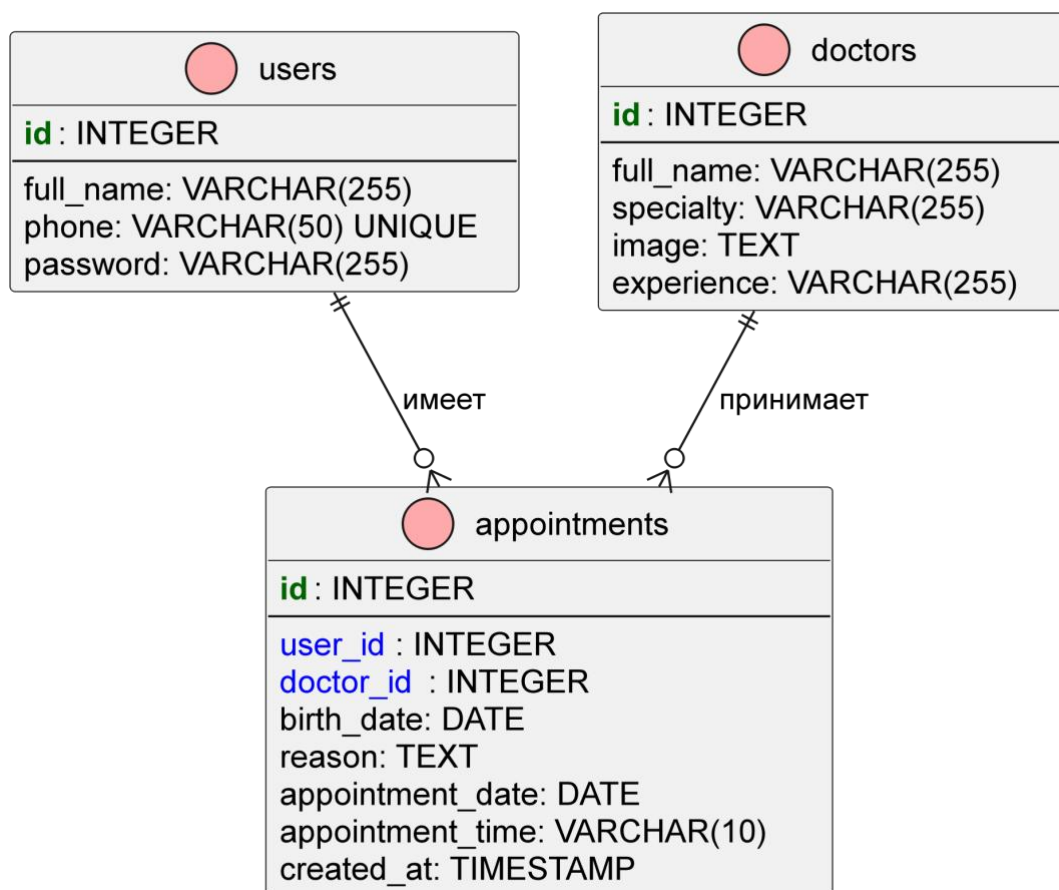


Рисунок 1 – Инфологическая модель базы данных

Данная модель содержит три основные сущности, каждая из которых связана между собой логическими связями.

Описание сущностей, их атрибутов и связей в логической модели представлены в таблице 2.

Таблица 2 – Описание сущностей и атрибутов базы данных

Сущность	Атрибут	Роль атрибута	Описание атрибута
Users	id	PK	Уникальный идентификатор пользователя
Users	full_name	—	Фамилия, имя и отчество пациента
Users	phone	UNIQUE	Номер мобильного телефона (используется для входа)
Users	password	—	Хеш пароля (хранится в зашифрованном виде)
Doctors	id	PK	Уникальный идентификатор врача
Doctors	full_name	—	Фамилия, имя и отчество врача
Doctors	specialty	—	Специализация врача (например, «Невролог», «Ортопед»)
Doctors	image	—	Путь к файлу фотографии врача
Doctors	experience	—	Строка с описанием опыта работы
Appointments	id	PK	Уникальный идентификатор записи
Doctors	experience	—	Строка с описанием опыта работы
Appointments	id	PK	Уникальный идентификатор записи
Appointments	user_id	FK ($\rightarrow$ Users.id)	Ссылка на пациента, который записался

Продолжение таблицы 2

Сущность	Атрибут	Роль атрибута	Описание атрибута
Appointments	doctor_id	FK ($\rightarrow$ Doctors.id)	Ссылка на врача, к которому записан пациент
Appointments	birth_date	–	Дата рождения пациента (запрашивается при записи)
Appointments	reason	–	Причина обращения (текст, вводимый пациентом)
Appointments	appointment_date	–	Дата приема (в формате ГГГГ-ММ-ДД)
Appointments	appointment_time	–	Время приема (в формате ЧЧ:ММ)
Appointments	created_at	–	Дата и время создания записи в системе

Связи между сущностями:

– Users и Appointments связаны отношением «один ко многим» – один пациент может иметь много записей, но каждая запись принадлежит одному пациенту;

– Doctors и Appointments также связаны отношением «один ко многим» – один врач может принимать много пациентов, но каждая запись относится к одному врачу.

После построения инфологической модели базы данных необходимо провести нормализацию. Это нужно для того, чтобы устранить избыточность данных и аномалии.

Проанализировав данную схему базы данных, можно определить соответствие следующим условиям:

- наличие простых (атомарных) атрибутов – все атрибуты неделимы;
- каждый не ключевой атрибут полностью зависит от первичного ключа (отсутствует частичная зависимость);
- каждый не ключевой атрибут не зависит от других не ключевых атрибутов (отсутствуют транзитивные зависимости).

В результате анализа было определено, что данная модель является нормализованной и находится в третьей нормальной форме (3НФ).

2.2 Проектирование программного обеспечения

2.2.1 Диаграмма вариантов использования

Диаграмма вариантов использования (use case diagram) графически изображает функциональные возможности системы, показывает взаимодействие между пользователями системы и ее функциями. Данная диаграмма состоит из следующих элементов:

- актеры – пользователи системы, которые исполняют определенные роли (пациент, администратор);
- прецеденты (варианты использования) – описывают определенные действия актеров.

Данная диаграмма предназначена для определения основных действий системы, понимания взаимодействия пользователей и функций, определения области ответственности каждого актера, определения границ возможностей системы.

Диаграмма прецедентов для информационной системы онлайн-записи в «Частный медицинский кабинет» представлена на рисунке 2.



Рисунок 2 – Диаграмма вариантов использования

Описание диаграммы прецедентов приведено в таблице 3.

Таблица 3 – Описание вариантов использования

Действующее лицо	Вариант использования	Описание
Пациент (неавторизованный)	Зарегистрироваться	Пациент вводит номер телефона, ФИО и пароль для создания учетной записи
Пациент (авторизованный)	Авторизоваться	Пациент вводит номер телефона и пароль для входа в систему
Пациент (авторизованный)	Просмотреть список врачей	Пациент может увидеть всех врачей клиники с их специализацией и опытом
Пациент (авторизованный)	Выбрать свободное время	Пациент выбирает врача и дату, система показывает доступные слоты
Пациент (авторизованный)	Создать запись на прием	Пациент указывает причину обращения и выбранное время, система создает запись
Пациент (авторизованный)	Просмотреть свои записи	Пациент может увидеть все свои активные записи (дата, время, врач)
Пациент (авторизованный)	Удалить запись	Пациент может отменить существующую запись, освободив время
Администратор	Управлять врачами (CRUD)	Администратор добавляет, редактирует и удаляет врачей в системе
Администратор	Просматривать логи	Администратор анализирует логи для контроля работы системы
Система	Отправлять уведомления (в перспективе)	Система информирует пациента о создании/отмене записи

Представленные в таблице 3 варианты использования охватывают основные сценарии взаимодействия пользователей с системой и служат основой для построения диаграммы деятельности, рассматриваемой в следующем разделе.

2.2.2 Диаграмма деятельности

Диаграмма деятельности (activity diagram) описывает последовательное выполнение действий или операций, показывает изменения в поведении

системы. На данной диаграмме отображаются решения, потоки управления, порядок выполнения определенных действий и взаимосвязи между ними.

Диаграмма деятельности для прецедента «Создание записи на прием» представлена на рисунке 3.

Порядок действий выглядит следующим образом:

- пациент авторизуется в системе (если еще не авторизован, система перенаправляет на экран входа);
- пациент переходит к форме записи (раздел «Онлайн запись» на главной странице);
- пациент выбирает врача из выпадающего списка;
- пациент выбирает дату приема;
- система отправляет асинхронный запрос на сервер для получения списка занятых временных слотов для выбранного врача и даты;
- сервер выполняет запрос к базе данных, возвращает JSON-массив занятых слотов;
- система отправляет асинхронный запрос на сервер для получения списка занятых временных слотов для выбранного врача и даты;
- сервер выполняет запрос к базе данных, возвращает JSON-массив занятых слотов;
- клиентская часть генерирует сетку временных слотов (с 7:00 до 19:00 с шагом 30 минут). Занятые слоты отображаются серым цветом и недоступны для выбора;
- пациент выбирает свободный слот (кликает на него). Выбранный слот подсвечивается;
- пациент вводит причину обращения в текстовое поле;
- кнопка «Записаться» становится активной;
- пациент нажимает кнопку «Записаться»;
- клиент отправляет POST-запрос на сервер с данными: `doctor_id`, `appointment_date`, `appointment_time`, `reason`;

- сервер проверяет валидность JWT-токена, извлекает user_id;
- сервер выполняет повторную проверку – не занято ли выбранное время другим пациентом (защита от ситуации, когда два пациента одновременно выбрали одно и то же время);
- если слот свободен, сервер вставляет новую запись в таблицу appointments;
- сервер фиксирует действие в логге (логгер записывает: кто, к какому врачу, на какое время записался);
- сервер возвращает успешный ответ (status 200) с сообщением «Запись успешно создана»;
- клиент показывает пациенту всплывающее уведомление об успешной записи;
- система обновляет сетку временных слотов (занятый слот теперь отображается как занятый).

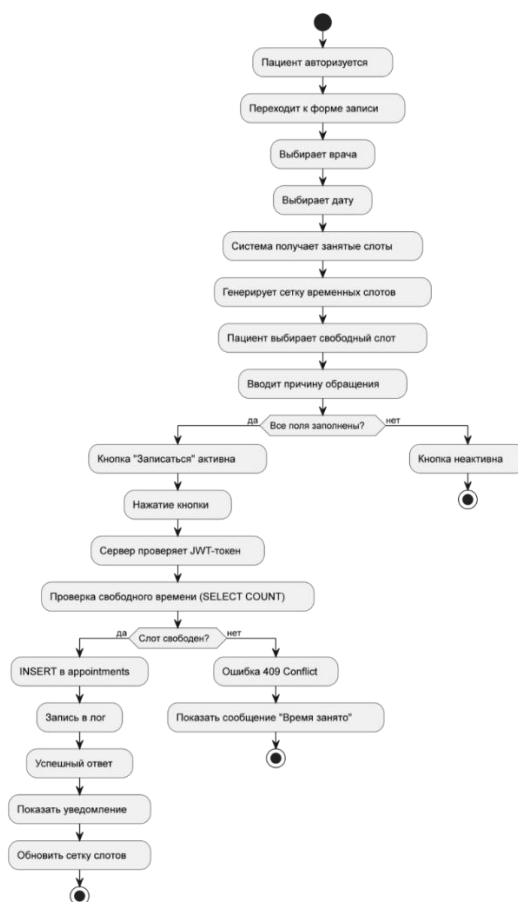


Рисунок 3 – Диаграмма деятельности для прецедента «Создание записи»

2.2.3 Диаграмма последовательности

Диаграмма последовательности (sequence diagram) подробно показывает взаимодействие объектов между собой в системе. Для этого демонстрируется порядок сообщений, которые объекты передают друг другу за определенный период времени.

Диаграмма последовательности для прецедента «Создание записи на прием» представлена на рисунке 4.

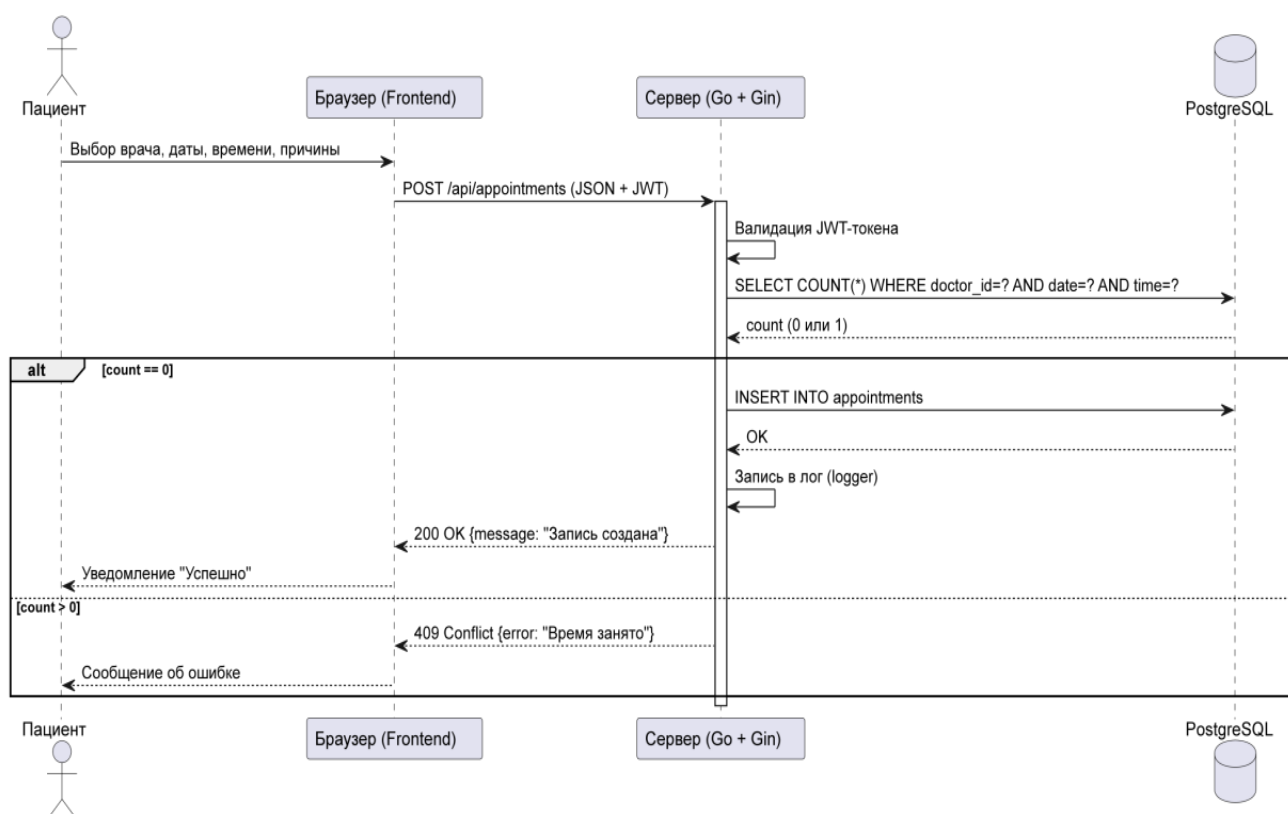


Рисунок 4 – Диаграмма последовательности

На диаграмме последовательности показаны следующие объекты:

- «Пациент» – пользователь системы;
- «Браузер» (Frontend) – клиентская часть (JavaScript);
- «Сервер» (Backend) – сервер на Go;
- «База данных» – PostgreSQL.

Порядок сообщений:

- «Пациент → Браузер»: выбирает врача, дату, время, вводит причину;

- «Браузер → Сервер»: POST /api/appointments (содержит doctor_id, appointment_date, appointment_time, reason, JWT-токен);
- «Сервер → Сервер»: валидация JWT-токена (проверка подписи и срока действия).
- «Сервер → База данных»: SELECT COUNT(*) FROM appointments WHERE doctor_id = ? AND appointment_date = ? AND appointment_time = ?;
- «База данных → Сервер»: возвращает количество занятых слотов (0 или 1);
- «Сервер → База данных»: INSERT INTO appointments (user_id, doctor_id, birth_date, reason, appointment_date, appointment_time);
- «База данных → Сервер»: подтверждение успешной вставки;
- «Сервер → Лог»: запись в файл server.log;
- «Сервер → Браузер»: HTTP 200 OK, JSON с сообщением;
- «Браузер → Пациент»: вывод уведомления «Запись успешно создана».

2.2.4 Диаграмма состояний

Диаграмма состояний (state machine diagram) предназначена для визуализации и анализа поведения объекта в различных состояниях. Она описывает все возможные состояния системы или отдельного объекта, а также переходы между этими состояниями.

Диаграмма состояний для объекта «Запись на прием» (сущность Appointment) представлена на рисунке 5.

На данной диаграмме представлены следующие состояния объекта «Запись на прием»:

- «Создана» – запись создана на клиенте, но еще не отправлена на сервер;
- «Проверка свободного времени» – сервер проверяет, не занят ли выбранный слот;

- «Активна» – запись успешно сохранена в базе данных, пациент может ее просматривать и отменять;
- «Отменена» – запись удалена пациентом (или администратором);
- «Завершена» – запись переходит в это состояние после наступления даты и времени приема (можно реализовать через фоновую задачу или маркировать «прошедшие» записи при запросе).

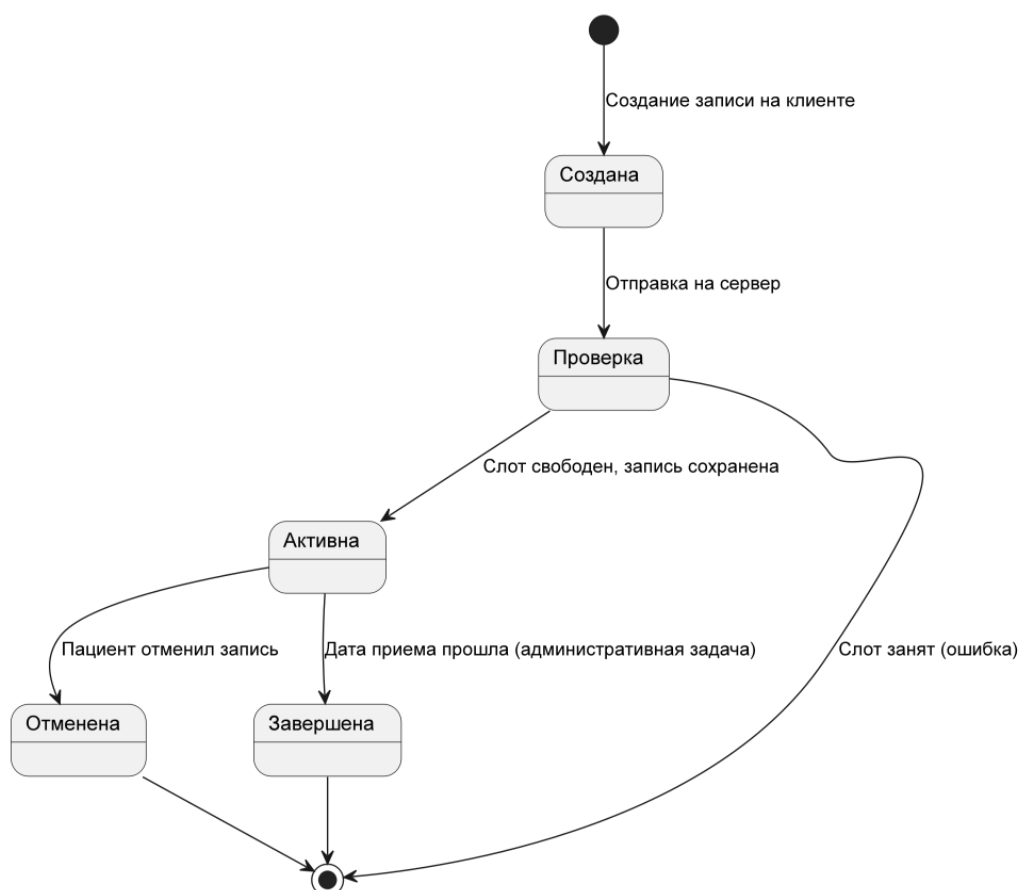


Рисунок 5 – Диаграмма состояний записи на прием

Начальное состояние – «Создана». При успешном сохранении в БД после проверки свободного времени запись переходит в состояние «Активна». Если при попытке сохранения выясняется, что слот уже занят, запись переходит в состояние «Ошибка» (или просто не сохраняется). Из состояния «Активна» пациент может перевести запись в состояние «Отменена» (через DELETE-запрос). При наступлении даты и времени приема (или по прошествии одного дня после даты) запись может быть автоматически переведена в состояние «Завершена» (административной задачей) [19, 20].

3 Реализация информационной системы

3.1 Реализация структуры данных (PostgreSQL)

В PostgreSQL настройка базы данных начинается с создания схемы и таблиц. Таблицы описывают текущие процессы и сущности, характерные для предметной области.

Для разработки информационной системы онлайн-записи были созданы следующие таблицы:

- users (пациенты);
- doctors (врачи);
- appointments (записи на прием).

Проиллюстрировать компоненты, из которых состоит база данных, можно с помощью диаграммы компонентов (рисунок 6).

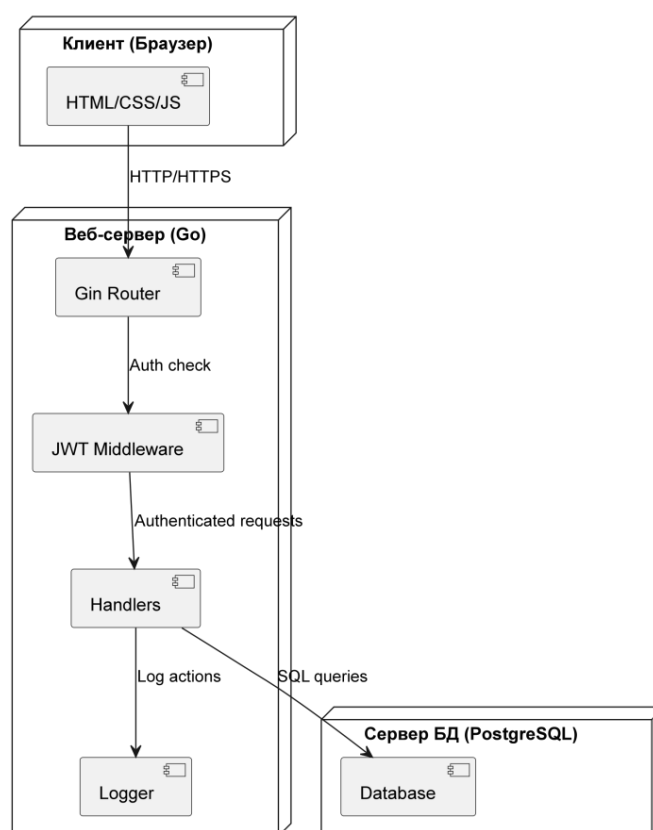


Рисунок 6 – Диаграмма компонентов базы данных

На диаграмме компонентов отображены три основных компонента базы данных, соответствующие таблицам предметной области: Users, Doctors и

Appointments, а также связи между ними, реализованные через внешние ключи `user_id` и `doctor_id`. Компонент `Users` хранит учётные данные пациентов, включая хешированный пароль; компонент `Doctors` – справочные данные о врачах клиники; компонент `Appointments` – записи на приём, связывающие конкретного пациента с конкретным врачом, датой и временем приёма. Такое разделение соответствует принципу единственной ответственности и облегчает сопровождение базы данных при дальнейшем расширении системы.

3.2 Реализация серверной части на Go

Серверная часть реализована на языке Go с использованием фреймворка Gin. Архитектура сервера построена по принципу разделения ответственности: маршруты (routes), обработчики (handlers), модели (models), конфигурация (config), утилиты (utils), middleware для аутентификации [21].

Структура проекта приведена на рисунке 7.

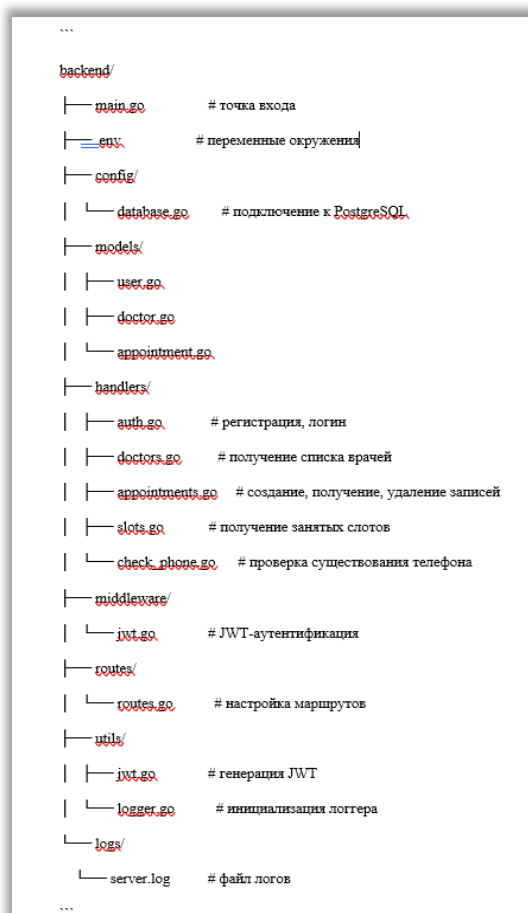


Рисунок 7 – Структура проекта

Архитектура сервера организована по слоям с чётким разделением обязанностей, что отображено в таблице 4.

Таблица 4 – Слои серверной части приложения

Слой	Файлы / пакет	Назначение
config	config/database.go	Установка соединения с базой данных PostgreSQL (database/sql, lib/pq)
models	models/user.go, doctor.go, appointment.go	Структуры данных (User, Doctor, Appointment) с JSON-тегами для сериализации запросов и ответов
handlers	handlers/*.go	Обработчики HTTP-запросов и бизнес-логика (регистрация, авторизация, работа с врачами, слотами и записями)
middleware	middleware/jwt.go	Проверка JWT-токена для защищённых маршрутов
routes	routes/routes.go	Регистрация маршрутов и их группировка по уровню доступа
utils	utils/jwt.go, utils/logger.go	Генерация JWT-токенов и логирование событий сервера

Маршруты регистрируются в файле routes/routes.go и делятся на две группы: открытые, доступные без авторизации, и защищённые, требующие действительный JWT-токен. Полный перечень маршрутов приведён в таблице 5.

Таблица 5 – Маршруты REST API серверной части

Метод	Маршрут	Обработчик	Доступ	Назначение
POST	/api/register	Register	Открытый	регистрация нового пациента
POST	/api/login	Login	Открытый	авторизация, выдача JWT-токена
GET	/api/doctors	GetDoctors	Открытый	получение списка врачей клиники
GET	/api/check-phone/:phone	CheckPhone	Открытый	проверка, зарегистрирован ли номер телефона
GET	/api/occupied-slots	GetOccupiedSlots	Открытый	получение занятых временных слотов врача на дату
POST	/api/appointments	CreateAppointment	Защищённый (JWT)	создание записи на приём

Продолжение таблицы 5

Метод	Маршрут	Обработчик	Доступ	Назначение
GET	/api/appointments	GetMyAppointments	Защищённый (JWT)	получение записей текущего пациента
DELETE	/api/appointments/:id	DeleteAppointment	Защищённый (JWT)	удаление записи на приём

Регистрация и авторизация. При регистрации (обработчик Register) пароль пользователя не хранится в открытом виде: используется хеширование с помощью библиотеки bcrypt (golang.org/x/crypto/bcrypt) со стандартной стоимостью хеширования (bcrypt.DefaultCost), хеш сохраняется в столбец password таблицы users. При авторизации (обработчик Login) введённый пароль сравнивается с сохранённым хешем функцией bcrypt.CompareHashAndPassword, что исключает возможность восстановления исходного пароля из базы данных даже при её компрометации.

После успешной проверки пароля сервер формирует JWT-токен (функция GenerateJWT, пакет github.com/golang-jwt/jwt/v5) по алгоритму HS256. В тело токена (claims) помещаются идентификатор пользователя user_id и время истечения exp (72 часа с момента выдачи). Токен подписывается секретным ключом, заданным переменной окружения JWT_SECRET, и возвращается клиенту в теле ответа.

Для защищённых маршрутов реализовано middleware AuthMiddleware (файл middleware/jwt.go). Оно извлекает заголовок Authorization, проверяет наличие префикса Bearer, разбирает и валидирует токен по тому же секретному ключу, а при успешной проверке помещает идентификатор пользователя в контекст запроса (c.Set(«user_id», ...)), откуда он впоследствии считывается обработчиками. Если токен отсутствует или недействителен, middleware прерывает обработку запроса и возвращает код ответа 401 Unauthorized.

Защита от двойной записи на одно и то же время. Ключевым бизнес-правилом системы является запрет на создание двух записей к одному врачу на одну дату и время. В обработчике CreateAppointment перед вставкой новой записи выполняется запрос SELECT COUNT(*) FROM appointments WHERE

`doctor_id = ? AND appointment_date = ? AND appointment_time = ?`. Если найдена хотя бы одна совпадающая запись, сервер возвращает код ответа 409 Conflict с сообщением «Это время уже занято», а попытка фиксируется в журнале событий с указанием пациента, врача, даты и времени.

Логирование. Все ключевые события системы (регистрация, авторизация, создание и удаление записи, ошибки валидации и обращения к базе данных) записываются в журнал через пакет `utils/logger.go`. Логгер инициализируется один раз при старте сервера (`InitLogger`) и одновременно выводит сообщения в стандартный поток вывода и в файл `logs/server.log` с помощью `io.MultiWriter`, что позволяет анализировать работу системы как в процессе разработки, так и постфактум (файл журнала).

3.3 Реализация пользовательского интерфейса (Frontend)

Клиентская часть реализована на нативном JavaScript, HTML5 и CSS3. Интерфейс имеет современный дизайн с фоновым слайдером, плавными анимациями и адаптивной версткой.

Основные компоненты интерфейса:

- «Экран авторизации» – вход по номеру телефона и паролю, регистрация нового пользователя. Реализована пошаговая форма: сначала проверяется номер телефона (существует ли пользователь), затем предлагается войти или зарегистрироваться;
- «Главный экран» – после входа отображается приветствие, фоновый слайдер с изображениями клиники, навигационное меню;
- «Форма записи» – выбор врача (`select`), даты (`date picker`), причины обращения (`textarea`), динамическая генерация временных слотов;
- «Сетка врачей» – карточки с фотографией, специализацией и опытом работы;
- «Блок контактов» – адрес, телефон, время работы клиники.
- Листинг программы реализации пользовательского интерфейса (Frontend) приведен в Приложении Б.

3.4 Диаграмма развертывания

Диаграмма развертывания (deployment diagram) – это тип UML-диаграммы, который показывает физическую архитектуру системы, включая такие узлы, как аппаратные или программные средства исполнения, а также промежуточное программное обеспечение, соединяющее их.

Данная диаграмма предназначена для визуализации модулей системы, помогает спланировать процессы развертывания, показывает производительность и масштабируемость системы.

Диаграмма развертывания информационной системы онлайн-записи представлена на рисунке 8.

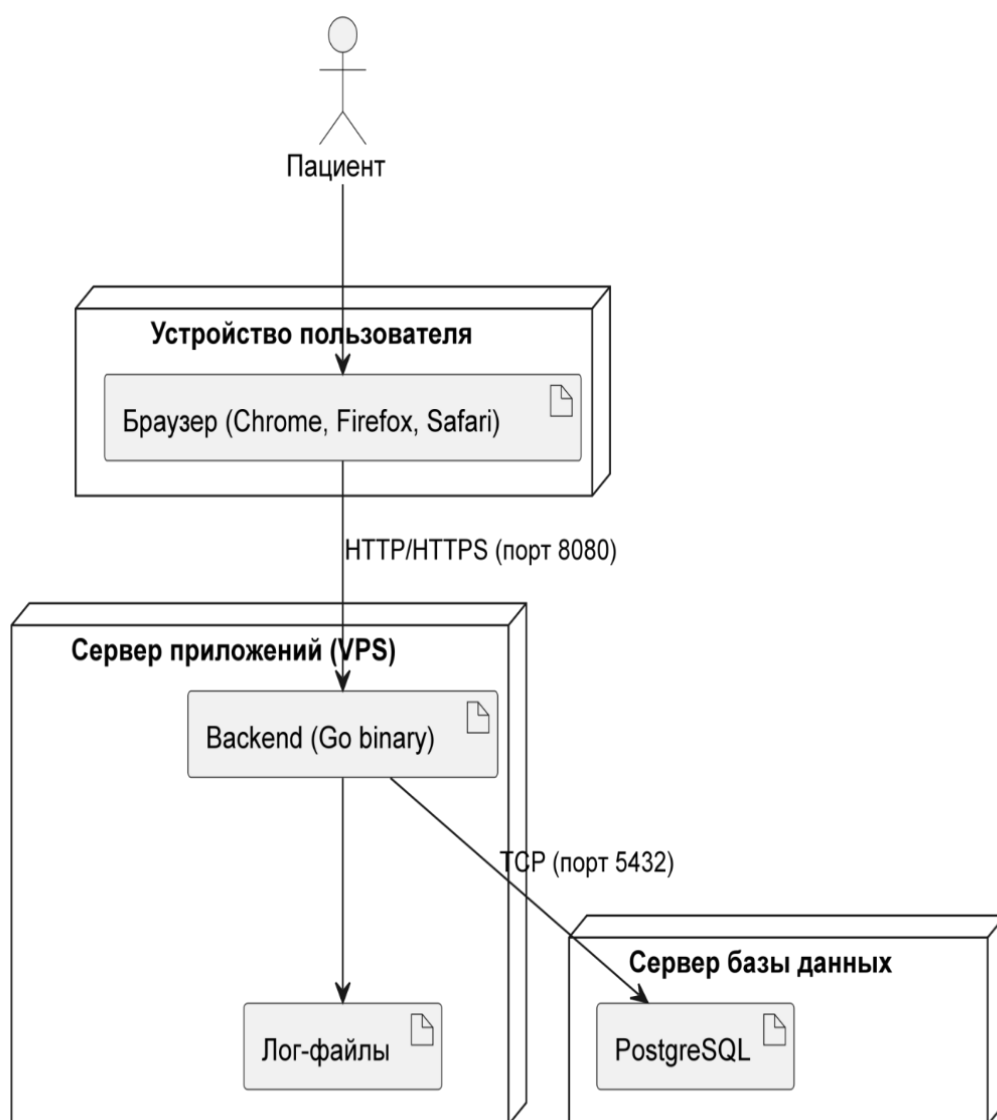


Рисунок 8 – Диаграмма развертывания

На диаграмме развертывания показано, что информационная система состоит из следующих узлов:

- клиент (Frontend) – браузер пользователя (Chrome, Firefox, Safari) на устройстве (ПК, ноутбук, смартфон, планшет). Клиент загружает HTML/CSS/JS с веб-сервера;
- веб-сервер (Backend) – сервер, на котором запущен исполняемый файл, скомпилированный из Go-кода, слушает порт 8080, обрабатывает API-запросы, выполняет бизнес-логику;
- сервер базы данных – сервер с PostgreSQL, хранящий данные о пользователях, врачах и записях;
- файловая система – хранилище логов (logs/server.log).

Взаимодействие между узлами происходит по протоколам HTTP/HTTPS (между клиентом и веб-сервером) и TCP (между веб-сервером и СУБД).

4 Тестирование информационной системы

4.1 Функциональное тестирование

Функциональное тестирование является составной частью системного тестирования и предназначено для проверки системы на ее работоспособность. При осуществлении данного метода тестирования дается оценка работе функционала программы.

Функциональное тестирование и тест-кейсы описаны в таблице 6.

Таблица 6 – Функциональное тестирование

Название теста	Описание теста	Полученный результат
Регистрация нового пользователя	Ввод номера телефона (11 цифр), ФИО, пароля (с цифрой и спецсимволом), подтверждение пароля. Нажатие «Создать аккаунт».	Тест пройден успешно. Пользователь создан, выполнен автоматический вход (рисунок В.1).
Авторизация существующего пользователя	Ввод номера телефона, пароля. Нажатие «Войти».	Тест пройден успешно. Получен JWT-токен, скрыта форма авторизации, показан основной интерфейс (рисунок В.2).
Проверка существующего телефона	Ввод номера телефона, нажатие «Продолжить».	Тест пройден успешно. Система определяет, зарегистрирован ли телефон, и показывает соответствующую форму (вход или регистрация) (рисунок В.3).
Получение списка врачей	После авторизации отправляется GET-запрос на /api/doctors.	Тест пройден успешно. Список врачей загружен и отображен в виде карточек и выпадающего списка (рисунок В.4).
Генерация временных слотов	Выбор врача и даты.	Тест пройден успешно. Сгенерированы слоты с 7:00 до 19:00 с шагом 30 минут. Занятые слоты отмечены серым цветом (рисунок В.5).
Создание записи на свободное время	Выбор свободного слота, ввод причины, нажатие «Записаться».	Тест пройден успешно. Запись создана в БД, занятый слот стал недоступен, в лог записано действие (рисунок В.6).

Продолжение таблицы 6

Название теста	Описание теста	Полученный результат
Запрет записи на занятое время	Попытка повторной записи (или одновременной) на занятый слот.	Сервер вернул ошибку 409 Conflict, сообщение «Это время уже занято» (рисунок В.7).
Получение своих записей	После создания записи отправляется GET-запрос на /api/appointments.	Тест пройден успешно. Возвращен массив с информацией о записи (дата, время, врач) (Рисунок В.8).
Логирование действий	Проверка файла logs/server.log после регистрации, входа, создания и удаления записи.	Тест пройден успешно. Все действия зафиксированы с указанием времени, пользователя и деталей (рисунок В.9).

Функциональное тестирование показало, что информационная система работает корректно, без ошибок.

4.2 Тестирование метрикой Холстеда

Для оценки сложности кода была использована метрика Холстеда. Метрика Холстеда позволяет провести оценку качества кода на основании числа уникальных операторов и операндов, их общего количества.

Для проведения оценки был выбран исходный код обработчика создания записи (файл handlers/appointments.go). В основу метрики Холстеда входят следующие характеристики:

- $n1$ – число уникальных операторов (ключевые слова Go, операторы, имена функций);
- $n2$ – число уникальных операндов (переменные, константы, строковые литералы);
- $N1$ – общее количество операторов;
- $N2$ – общее количество операндов;
- Vocabulary – словарь программы ($n1 + n2$);
- Length – длина программы ($N1 + N2$);
- Volume – объем программы ($V = \text{Length} * \log_2(\text{Vocabulary})$);
- Difficulty – сложность программы ($D = (n1/2) * (N2/n2)$);

- Effort – количество усилий для разработки программы ($E = D * V$);
- Time – предполагаемое время разработки (в секундах, $E / 18$);
- Bugs – оценка количества возможных ошибок ($B = V / 3000$).

Результаты оценки качества кода методом Холстеда представлены в таблице 7.

Таблица 7 – Результат оценки сложности кода методом Холстеда

Показатель	Значение
n1 (уникальные операторы)	24
n2 (уникальные операнды)	31
N1 (всего операторов)	142
N2 (всего операндов)	167
Vocabulary (n1 + n2)	55
Length (N1 + N2)	309
Объем программы V	$309 * \log_2(55) \approx 309 * 5.78 \approx 1786$ бит
Сложность D	$(24/2) * (167/31) = 12 * 5.39 \approx 64.7$
Усилия E	$64.7 * 1786 \approx 115\,600$
Время разработки T	$115600 / 18 \approx 6420$ сек ≈ 107 мин ≈ 1.8 часа
Ошибки B	$1786 / 3000 \approx 0.6$

Анализ кода метрикой Холстеда показал, что сложность программы находится в норме, а прогнозируемое количество ошибок крайне мало (менее одной ошибки на весь модуль). Это свидетельствует о высоком качестве и читаемости кода.

4.3 Системное тестирование

Системное тестирование выполняется для проверки работы информационной системы в целом, в условиях, приближенных к реальной эксплуатации, и включает проверку совместимости с различными операционными системами и веб-браузерами на стороне клиента[22].

Поскольку клиентская часть системы реализована в виде веб-приложения на нативном JavaScript без использования сторонних фреймворков, особое внимание было уделено корректности отображения интерфейса и работы динамических элементов (генерация сетки временных слотов, валидация форм, всплывающие уведомления) в разных браузерах.

Системное тестирование проводилось на комбинациях операционных систем и браузеров, отображенных в таблице 8.

Таблица 8 – Результаты системного тестирования

ОС	Браузер	Версия	Результат
Windows 11	Google Chrome	Актуальная	Корректно
Windows 10	Mozilla Firefox	Актуальная	Корректно
Windows 11	Microsoft Edge	Актуальная	Корректно
macOS	Safari	Актуальная	Корректно
Ubuntu Linux	Google Chrome	Актуальная	Корректно

В ходе тестирования проверялись: корректность отображения вёрстки и адаптивность интерфейса при изменении размера окна браузера; работоспособность форм регистрации и авторизации; корректность генерации и обновления сетки свободных и занятых временных слотов; работа всплывающих уведомлений об успешных и ошибочных операциях; сохранение работоспособности при повторном запросе данных после обновления страницы.

Помимо проверки совместимости с операционными системами и браузерами, в рамках системного тестирования была отдельно проверена устойчивость информационной системы к некорректным и граничным входным данным, а также к нестандартным сценариям использования, неизбежно возникающим при реальной эксплуатации.

При попытке создания записи на уже занятый временной слот, в том числе при имитации быстрой повторной отправки запроса непосредственно после первой успешной записи, сервер во всех случаях возвращал код ответа 409 Conflict с сообщением «Это время уже занято», не допуская дублирования записей в расписании врача. Попытки обращения к защищённым маршрутам (/api/appointments, /api/appointments/:id) без передачи заголовка Authorization, а также с истёкшим или искажённым JWT-токеном последовательно завершались ошибкой 401 Unauthorized, что подтверждает корректную работу middleware-проверки авторизации, описанного в разделе 3.2.

Дополнительно проверялась реакция системы на некорректные данные при регистрации: ввод номера телефона, не соответствующего маске из 11 цифр, ввод пароля без цифр или специальных символов, а также несовпадение пароля

и его подтверждения. В каждом из перечисленных случаев клиентская часть блокировала отправку формы и отображала соответствующее предупреждение, а при намеренном обходе клиентской проверки (отправка запроса напрямую на `/api/register` с некорректными данными) серверная валидация также отклоняла запрос с кодом ответа 400 Bad Request.

Так как клиентский интерфейс реализован с применением адаптивной вёрстки, отдельно была проверена корректность отображения интерфейса при различных разрешениях экрана с использованием инструментов разработчика браузера: 320×568 (мобильный телефон), 768×1024 (планшет), 1366×768 (ноутбук) и 1920×1080 (настольный монитор). Во всех указанных разрешениях форма авторизации, сетка временных слотов и карточки врачей корректно перестраивались под доступную ширину экрана без появления горизонтальной прокрутки и без потери читаемости текста.

Во всех проверенных комбинациях операционных систем и браузеров информационная система функционировала корректно, без визуальных искажений интерфейса и потери функциональности. Незначительные отличия затронули только скорость анимации фоновых слайдера на устройствах с ограниченными аппаратными ресурсами, что не повлияло на работоспособность системы.

5 Планирование и оценка стоимости проекта

Управление проектами включает в себя определение задач и целей, расчет бюджета, разработку плана действий, утверждение ресурсов, оценку рисков и контроль за исполнением проекта.

Планирование и оценка стоимости информационной системы онлайн-записи выполнялось в ProjectLibre. Для работы над проектом были выделены необходимые ресурсы, приведенные в таблице 9.

Таблица 9 – Ресурсы и затраты проекта

Название ресурса	Тип ресурса	Стандартная ставка	Начисление
Backend-разработчик (Go)	Трудовой	400,00 Р/ч	Пропорционально
Frontend-разработчик (JS)	Трудовой	350,00 Р/ч	Пропорционально
Аналитик	Трудовой	300,00 Р/ч	Пропорционально
Тестировщик	Трудовой	280,00 Р/ч	Пропорционально
Технический писатель	Трудовой	250,00 Р/ч	Пропорционально
Сервер (аренда VPS)	Материальный	800,00 Р/мес	В конце
Персональный компьютер (амортизация)	Материальный	500,00 Р/мес	В начале

В результате разработки проекта было выделено 20 задач, которые приведены в таблице 10.

Таблица 10 – Задачи проекта

Название задачи	Длительность	Начало	Окончание
Начало проекта	0 дней	08.04.2026	08.04.2026
Анализ предметной области	3 дня	08.04.2026	10.04.2026
Анализ требований и постановка задач	2 дня	08.04.2026	09.04.2026
Выбор средств разработки	1 день	12.04.2026	12.04.2026
Завершение анализа	0 дней	13.04.2026	13.04.2026
Проектирование базы данных	3 дня	13.04.2026	15.04.2026

Продолжение таблицы 10

Название задачи	Длительность	Начало	Окончание
Проектирование UML-диаграмм	4 дня	16.04.2026	19.04.2026
Завершение проектирования	0 дней	20.04.2026	20.04.2026
Разработка серверной части (Go)	6 дней	20.04.2026	25.04.2026
Разработка клиентской части (HTML/CSS/JS)	5 дней	26.04.2026	30.04.2026
Интеграция бэкенда и фронтенда	2 дня	01.05.2026	02.05.2026
Завершение разработки	0 дней	03.05.2026	03.05.2026
Функциональное тестирование	3 дня	03.05.2026	05.05.2026
Анализ качества кода (Холстед)	1 день	06.05.2026	06.05.2026
Системное тестирование	2 дня	07.05.2026	08.05.2026
Завершение тестирования	0 дней	09.05.2026	09.05.2026
Написание документации	4 дня	09.05.2026	12.05.2026
Завершение проекта	0 дней	13.05.2026	13.05.2026

В результате анализа расчета ресурсов для реализации проекта потребуется 26 рабочих дней (с учетом календаря без выходных), а стоимость выполнения проекта составит 98 400 рублей. Определение стоимости проекта нужно для того, чтобы установить необходимый бюджет на разработку, определить приемлемый способ использования денежных средств и контроля затрат на протяжении всей работы проекта.

Рассмотрим детализацию затрат на реализацию проекта:

- заработная плата разработчиков:
 - а) бэкенд: $6 \text{ дней} \times 8 \text{ часов} \times 400 \text{ руб/ч} = 19\,200 \text{ руб.};$
 - б) фронтенд: $5 \text{ дней} \times 8 \text{ часов} \times 350 \text{ руб/ч} = 14\,000 \text{ руб.};$
 - в) аналитик: $5 \text{ дней} \times 8 \text{ часов} \times 300 \text{ руб/ч} = 12\,000 \text{ руб.};$

- г) тестировщик: 6 дней $\times$ 8 часов $\times$ 280 руб/ч = 13 440 руб.;
- д) технический писатель: 4 дня $\times$ 8 часов $\times$ 250 руб/ч = 8 000 руб.;
- е) итого по трудовым ресурсам: $\approx$ 66 640 руб.;
- аренда VPS на 1 месяц: 800 руб.;
- амортизация ПК и прочие расходы: 1000 руб.;
- итого прямые затраты: $\approx$ 68 440 руб.;
- накладные расходы (40%): 27 376 руб.;
- резерв на непредвиденные работы (10%): 6 844 руб.;
- НДС (20%): 13 688 руб.

Общая стоимость проекта: 98 348 руб. (округлено 98 400 руб.).

При планировании разработки информационной системы был проведён анализ возможных рисков, способных повлиять на сроки, стоимость или качество выполнения проекта. Идентифицированные риски, их характеристика и меры по снижению приведены в таблице 11.

Таблица 11 – Анализ рисков проекта

Риск	Вероятность	Влияние	Меры по снижению
Изменение требований заказчика в ходе разработки	Средняя	Среднее	Фиксация требований на этапе постановки задачи (раздел 1.1), согласование изменений до начала реализации модуля
Технические сложности при интеграции серверной и клиентской частей	Низкая	Среднее	Использование REST API с чётко определённым форматом обмена данными (JSON), раннее прототипирование эндпоинтов
Ошибка проверки занятости временного слота при одновременной записи двух пациентов	Средняя	Высокое	Повторная проверка занятости слота на сервере непосредственно перед вставкой записи в базу данных

Продолжение таблицы 11

Риск	Вероятность	Влияние	Меры по снижению
Превышение запланированных сроков разработки отдельных модулей	Средняя	Низкое	Резерв времени в календарном плане декомпозиция задач на более мелкие этапы
Недостаточная производительность при росте числа пользователей	Низкая	Среднее	Использование индексов в базе данных PostgreSQL, возможность горизонтального масштабирования серверной части
Утечка персональных данных пациентов	Низкая	Высокое	Хеширование паролей (bcrypt), передача данных по HTTPS, аутентификация по JWT-токену с ограниченным сроком действия

Наибольшую угрозу для проекта представляют риски с высоким влиянием – нарушение целостности данных при одновременной записи на одно и то же время и утечка персональных данных пациентов. Для обоих рисков в системе предусмотрены технические меры защиты, описанные в разделах 3.1 и 3.2 настоящей работы, что позволяет считать данные риски в значительной степени митигированными уже на этапе разработки. Риски, связанные с изменением требований и сроками разработки, носят организационный характер и контролируются за счёт календарного планирования.

Анализ календарного плана в ProjectLibre показал, что критический путь проекта проходит через задачи анализа предметной области, проектирования базы данных, разработки серверной части, интеграции серверной и клиентской частей и тестирования программных средств. Задержка любой из перечисленных задач приведёт к сдвигу даты завершения проекта в целом, тогда как написание документации обладает собственным резервом времени и может выполняться параллельно с завершающими этапами тестирования без влияния на общий срок выполнения работы.

Помимо прямых затрат на разработку, целесообразно оценить ориентировочный экономический эффект от внедрения системы для медицинской организации. В настоящее время запись пациентов осуществляется администратором по телефону, что занимает в среднем 3 – 5 минут на одного пациента с учётом уточнения свободного времени и данных пациента. При среднем потоке 15 – 20 обращений в день автоматизация процесса записи позволяет сократить затраты рабочего времени администратора приблизительно на 1 – 1,5 часа ежедневно, что при ставке администратора порядка 250 руб/ч эквивалентно ориентировочной экономии 3 000 – 4 500 рублей в месяц. С учётом стоимости разработки (98 400 рублей) срок окупаемости системы для одной клиники по приближённой оценке составляет около 22 – 32 месяцев, после чего эксплуатация системы становится для клиники чистой экономией по сравнению с содержанием административного персонала в прежнем объёме.

Контроль выявленных рисков осуществлялся на протяжении всего жизненного цикла проекта: фактический ход выполнения задач регулярно сопоставлялся с плановым календарным графиком, что позволяло своевременно выявлять отклонения и пересматривать приоритеты оставшихся задач. Дополнительно после завершения каждого крупного модуля (регистрация и авторизация, работа с врачами и временными слотами, создание и отмена записей) проводилось краткое ревью реализованного кода, что снижало вероятность накопления технического долга и упрощало последующие этапы функционального и системного тестирования.

Заключение

В ходе выполнения выпускной квалификационной работы была разработана информационная система онлайн-записи в частный медицинский кабинет.

Основные результаты работы:

- проанализирована предметная область автоматизации записи пациентов в медицинской организации;
- выявлены требования к системе: регистрация, авторизация, просмотр врачей, выбор свободного времени, создание и отмена записей;
- проведён анализ существующих систем (DocPlanner, Yclients, Medesk, Инфоклиника);
- определены их преимущества (широкий функционал, интеграции) и недостатки (высокая стоимость, избыточность, сложность настройки);
- обоснована необходимость разработки собственной системы;
- выполнено проектирование информационной системы:
 - а) спроектирована инфологическая модель базы данных (3 сущности, 3НФ);
 - б) построены UML-диаграммы: вариантов использования, деятельности, последовательности, состояний, компонентов, развёртывания;
- разработана серверная часть на языке Go с использованием фреймворка Gin, PostgreSQL и JWT-аутентификации;
- реализованы REST API эндпоинты для регистрации, логина, получения врачей, занятых слотов, создания и удаления записей;
- разработана клиентская часть на нативном JavaScript, HTML5 и CSS3 с адаптивным дизайном, динамической генерацией временных слотов и плавными анимациями;
- проведено тестирование программного обеспечения:
 - а) функциональное тестирование (10 тест-кейсов, все успешно);
 - б) оценка качества кода методом Холстеда (показатели в норме,

- в) прогнозируемое количество ошибок < 1);
- г) системное тестирование на пяти ОС и браузерах (все работает
- д) корректно);

– выполнено планирование проекта. Рассчитана стоимость разработки: 98 400 рублей, длительность: 26 рабочих дней.

Все цели и задачи, поставленные в начале работы, были успешно достигнуты. Информационная система готова к внедрению в различных медицинских организациях для оптимизации процесса записи пациентов, снижения нагрузки на администраторов и повышения качества обслуживания. В частности, при некоторой доработке, разработанная система может быть внедрена в Пензенском медицинском центре «Клиника лечения боли». Клиника зарегистрирована в Министерстве здравоохранения Пензенской области. При небольшом количестве штатных врачей и обслуживающего персонала в клинике оказывается широкий спектр медицинских услуг. Возможность внедрения результатов данной работы была обсуждена и одобрена руководством центра «Клиника лечения боли», что подтверждается соответствующим актом. Акт о внедрения приведен в приложении В.

Разработанная информационная система имеет ряд перспективных направлений для дальнейшего развития. В числе наиболее значимых из них можно выделить следующие:

- разработка мобильного приложения для пациентов и врачей с push-уведомлениями о приближающемся приёме, что повысит удобство использования системы и снизит количество пропущенных визитов;
- интеграция с электронными медицинскими картами (ЕМИАС, региональные МИС) для автоматического обмена данными о пациентах и результатами обследований;
- расширение рейтинговой системы врачей с учётом отзывов пациентов, что позволит повысить прозрачность работы клиники и доверие к выбору специалиста;

– масштабирование архитектуры системы для поддержки нескольких медицинских организаций одновременно (мультиотенантность), что позволит предлагать систему как SaaS-решение для широкого круга частных клиник.

Реализация перечисленных направлений позволит повысить конкурентоспособность разработанной системы на рынке программных решений для медицинских организаций и расширить область её практического применения.

Таким образом, разработанная информационная система может рассматриваться не только как готовое к внедрению прикладное решение для конкретного частного медицинского кабинета, но и как основа для дальнейшего развития в направлении более универсального продукта, востребованного широким кругом небольших медицинских организаций. Полученный в ходе выполнения выпускной квалификационной работы опыт проектирования и реализации системы, а также проведённое тестирование и анализ качества кода подтверждают практическую применимость выбранных технологических решений (Go, Gin, PostgreSQL, JWT-аутентификация, нативный JavaScript) для задач данного класса. Полученные результаты могут быть использованы в качестве методической основы при разработке аналогичных информационных систем для других предприятий сферы услуг.

Проведённое в рамках работы функциональное и системное тестирование показало, что разработанная информационная система устойчиво обрабатывает основные сценарии использования: регистрацию и авторизацию пациентов, формирование списка свободных временных слотов с учётом графика конкретного врача, создание и отмену записи на приём, а также корректное отображение информации о клинике и контактных данных. Отдельное внимание было уделено проверке корректности работы системы при граничных условиях – попытках создания записи на уже занятое время, вводе некорректных данных при регистрации и обращении к защищённым эндпоинтам без действительного JWT-токена. Во всех перечисленных случаях система возвращала ожидаемые

сообщения об ошибках, что подтверждает корректность реализованной серверной логики.

Применённая при разработке архитектура с разделением серверной и клиентской частей позволяет в дальнейшем независимо развивать каждый из компонентов системы: дополнять серверную часть новыми эндпоинтами без изменения клиентского кода, а также заменять или дорабатывать пользовательский интерфейс без затрагивания бизнес-логики. Такой подход соответствует современным практикам разработки веб-приложений и облегчает дальнейшее сопровождение и масштабирование системы силами разработчика или сторонней команды.

Список использованных источников

1. Резиг, Дж. JavaScript для профессионалов : пер. с англ. / Дж. Резиг, Р. Фергюсон, Дж. Пакстон. – 3-е изд. – М. : ООО «И.Д. Вильямс», 2024. – 240 с. : ил.
2. Документация PostgreSQL [Электронный ресурс]. – URL: <https://www.postgresql.org/docs/current/> (дата обращения: 29.04.2026).
3. Кантор, И. Современный учебник JavaScript [Электронный ресурс]. – URL: <https://learn.javascript.ru> (дата обращения: 15.04.2026).
4. Роббинс, Дж. HTML5, CSS3 и JavaScript. Исчерпывающее руководство : пер. с англ. / Дж. Роббинс ; [пер. М. А. Райтман]. – 4-е изд. – М. : Эксмо, 2014. – 528 с.
5. Хоган, Б. HTML5 и CSS3. Веб-разработка по стандартам нового поколения / Б. Хоган. – 2-е изд. – СПб. : Питер, 2014. – 320 с. : ил. – ISBN 978-5-496-00979-9.
6. Фрейн, Б. HTML5 и CSS3. Разработка сайтов для любых браузеров и устройств / Б. Фрейн. – СПб. : Питер, 2014. – 304 с. : ил.
7. Хрусталеv, А. А. HTML5 + CSS3. Основы современного WEB-дизайна / А. А. Хрусталеv, А. В. Кириченко. – СПб. : Наука и Техника, 2018. – 352 с. – ISBN 978-5-94387-750-6.
8. Моргунов, Е. П. PostgreSQL. Основы языка SQL : учеб. пособие / Е. П. Моргунов ; под ред. Е. В. Рогова, П. В. Лузанова. – СПб. : БХВ-Петербург, 2018. – 336 с. : ил.
9. Маркин, А. В. Базы данных. PostgreSQL : учебник для среднего профессионального образования / А. В. Маркин. – М. : Издательство Юрайт, 2026. – 828 с.
10. Документация MySQL [Электронный ресурс]. – URL: <https://dev.mysql.com/doc/> (дата обращения: 16.04.2026).
11. Документация JSON Web Token (JWT) [Электронный ресурс]. – URL: <https://jwt.io/introduction> (дата обращения: 20.04.2026).

12. PlantUML Language Reference Guide [Электронный ресурс]. – URL: <https://plantuml.com/guide> (дата обращения: 06.06.2024).
13. Катышева, М. А. Основы проектной деятельности : лабораторный практикум / М. А. Катышева, Ю. Г. Кирюхин, В. А. Маслов. – Пенза : Изд-во ПГУ, 2022. – Ч. 2. – 52 с.
14. Руденко, Л. И. Основы управления проектами : практический курс : учебно-методическое пособие / Л. И. Руденко. – Симферополь : ФГАОУ ВО «Крымский федеральный университет имени В. И. Вернадского», 2018. – 70 с.
15. Visual Studio [Электронный ресурс] : официальный сайт IDE. – URL: <https://visualstudio.microsoft.com/> (дата обращения: 22.04.2025).
16. Visual Studio Code [Электронный ресурс] : официальный сайт редактора кода. – URL: <https://code.visualstudio.com/> (дата обращения: 22.04.2025).
17. Документация по Visual Studio [Электронный ресурс]. – URL: <https://learn.microsoft.com/ru-ru/visualstudio/windows/> (дата обращения: 15.05.2026).
18. Власов, С. А. Использование инфологической и даталогической модели системы при проектировании базы данных // Техническое регулирование в едином экономическом пространстве : сборник статей VIII Всероссийской научно-практической конференции. – Екатеринбург : РГППУ, 2021. – С. 240–248.
19. Буч, Г. Язык UML. Руководство пользователя / Г. Буч, Д. Рамбо, А. Якобсон. – 2-е изд. – М. : ДМК Пресс, 2021. – 496 с.
20. Фаулер, М. UML. Основы. Краткое руководство по стандартному языку объектного моделирования : пер. с англ. / М. Фаулер ; пер. А. Панина. – 3-е изд. – СПб. : Символ-Плюс, 2019. – 192 с.
21. Донован, А. Язык программирования Go : пер. с англ. / А. Донован, Б. Керниган. – М. : ООО «И.Д. Вильямс», 2016. – 432 с. : ил.

22. Что такое системное тестирование [Электронный ресурс]. – URL: <https://sky.pro/media/cto-takoe-sistemnoe-testirovanie> (дата обращения: 09.06.2024).

Приложение А
(справочное)
Глоссарий

Информационная система (ИС) – система, предназначенная для хранения, поиска и обработки информации, включающая в себя программные и аппаратные средства.

База данных (БД) – совокупность структурированных данных, хранящихся в памяти, отображающая состояние и взаимодействие объектов предметной области.

UML (Unified Modeling Language) – язык графического моделирования, предназначенный для разработки программного обеспечения и отображения бизнес-процессов.

REST API – архитектурный стиль взаимодействия компонентов распределенного приложения в сети.

JWT (JSON Web Token) – компактный URL-безопасный способ представления утверждений между двумя сторонами.

PostgreSQL – объектно-реляционная система управления базами данных с открытым исходным кодом.

Go (Golang) – компилируемый статически типизированный язык программирования, разработанный в Google.

Фреймворк Gin – высокопроизводительный HTTP-веб-фреймворк для Go.

CRUD (Create, Read, Update, Delete) – аббревиатура для базовых операций с данными.

Логирование – процесс записи событий, происходящих в программной системе.

Холстеда метрика – способ оценки сложности программного кода на основе количества операторов и операндов.

Приложение Б
(обязательное)
Код программы

Листинг Б.1 – Файл database.go

```
package config

import (
    "database/sql"
    "fmt"
    "log"
    "os"

    _ "github.com/lib/pq"
)

var DB *sql.DB

func ConnectDB() {
    connStr := fmt.Sprintf(
        "host=%s port=%s user=%s password=%s dbname=%s\n" +
        "sslmode=disable",
        os.Getenv("DB_HOST"),
        os.Getenv("DB_PORT"),
        os.Getenv("DB_USER"),
        os.Getenv("DB_PASSWORD"),
        os.Getenv("DB_NAME"),
    )

    db, err := sql.Open("postgres", connStr)
    if err != nil {
        log.Fatal(err)
    }

    if err = db.Ping(); err != nil {
        log.Fatal(err)
    }

    DB = db

    log.Println("Database connected")
}
```

Листинг Б.2 – Файл appointments.go

```
package handlers

import (
    "net/http"

    "guropnz/config"
    "guropnz/models"
    "guropnz/utils"

    "github.com/gin-gonic/gin"
)
```

```

func CreateAppointment(c *gin.Context) {

    var appointment models.Appointment

    err := c.BindJSON(&appointment)

    if err != nil {

        c.JSON(http.StatusBadRequest, gin.H{
            "error": "Неверные данные",
        })

        return
    }

    userID := c.GetInt("user_id")

    var userName string

    config.DB.QueryRow(
        "SELECT full_name FROM users WHERE id=$1",
        userID,
    ).Scan(&userName)

    var doctorName string

    config.DB.QueryRow(
        "SELECT full_name FROM doctors WHERE id=$1",
        appointment.DoctorID,
    ).Scan(&doctorName)

    var count int

    config.DB.QueryRow(
        `
        SELECT COUNT(*)
        FROM appointments
        WHERE doctor_id=$1
        AND appointment_date=$2
        AND appointment_time=$3
        `,
        appointment.DoctorID,
        appointment.AppointmentDate,
        appointment.AppointmentTime,
    ).Scan(&count)

    if count > 0 {

        utils.Logger.Println(
            "[ЗАНЯТОЕ ВРЕМЯ]",
            "Пациент:",
            userName,
            "| Врач:",

```

```

        doctorName,
        "| Дата:",
        appointment.AppointmentDate,
        "| Время:",
        appointment.AppointmentTime,
    )

    c.JSON(http.StatusConflict, gin.H{
        "error": "Это время уже занято",
    })

    return
}

_, err = config.DB.Exec(
    `
    INSERT INTO appointments
    (
        user_id,
        doctor_id,
        birth_date,
        reason,
        appointment_date,
        appointment_time
    )
    VALUES ($1, $2, $3, $4, $5, $6)
    `,
    userID,
    appointment.DoctorID,
    appointment.BirthDate,
    appointment.Reason,
    appointment.AppointmentDate,
    appointment.AppointmentTime,
)

if err != nil {

    utils.Logger.Println(
        "[ОШИБКА СОЗДАНИЯ ЗАПИСИ]",
        err.Error(),
    )

    c.JSON(http.StatusInternalServerError, gin.H{
        "error": err.Error(),
    })

    return
}

utils.Logger.Println(
    "[СОЗДАНА ЗАПИСЬ]",
    "Пациент:",
    userName,

```

```

        "| Врач:",
        doctorName,
        "| Причина:",
        appointment.Reason,
        "| Дата:",
        appointment.AppointmentDate,
        "| Время:",
        appointment.AppointmentTime,
    )

    c.JSON(http.StatusOK, gin.H{
        "message": "Запись успешно создана",
    })
}

func GetMyAppointments(c *gin.Context) {

    userID := c.GetInt("user_id")

    rows, err := config.DB.Query(
        SELECT
            appointments.id,
            doctors.full_name,
            appointment_date,
            appointment_time
        FROM appointments
        JOIN doctors
        ON doctors.id = appointments.doctor_id
        WHERE user_id=$1
        ,
        userID,
    )

    if err != nil {

        c.JSON(http.StatusInternalServerError, gin.H{
            "error": err.Error(),
        })

        return
    }

    defer rows.Close()

    var appointments []gin.H

    for rows.Next() {

        var id int
        var doctor string
        var date string
        var time string
    }

```

```

        rows.Scan(
            &id,
            &doctor,
            &date,
            &time,
        )

        appointments = append(
            appointments,
            gin.H{
                "id":      id,
                "doctor":  doctor,
                "date":    date,
                "time":    time,
            },
        )
    }

    c.JSON(http.StatusOK, appointments)
}

func DeleteAppointment(c *gin.Context) {

    id := c.Param("id")

    _, err := config.DB.Exec(
        "DELETE FROM appointments WHERE id=$1",
        id,
    )

    if err != nil {

        c.JSON(http.StatusInternalServerError, gin.H{
            "error": err.Error(),
        })

        return
    }
}

```

Листинг Б.3 – Файл auth.go

```

package handlers

import (
    "database/sql"
    "net/http"

    "guropnz/config"
    "guropnz/models"
    "guropnz/utils"

    "github.com/gin-gonic/gin"

```

```

        "golang.org/x/crypto/bcrypt"
    )

    func Register(c *gin.Context) {
        var user models.User

        if err := c.BindJSON(&user); err != nil {
            c.JSON(http.StatusBadRequest, gin.H{"error": "Invalid
data"})
            return
        }

        hashedPassword, _ :=
bcrypt.GenerateFromPassword([]byte(user.Password),
bcrypt.DefaultCost)

        _, err := config.DB.Exec(
            "INSERT INTO users(full_name, phone, password) VALUES($1,
$2, $3)",
            user.FullName,
            user.Phone,
            string(hashedPassword),
        )

        if err != nil {
            c.JSON(http.StatusBadRequest, gin.H{"error": "User
already exists"})
            return
        }

        utils.Logger.Println("REGISTER:", user.Phone)

        c.JSON(http.StatusOK, gin.H{"message": "Registered"})
    }

    func Login(c *gin.Context) {
        var request models.User

        c.BindJSON(&request)

        var user models.User

        err := config.DB.QueryRow(
            "SELECT id, password FROM users WHERE phone=$1",
            request.Phone,
        ).Scan(&user.ID, &user.Password)

        if err == sql.ErrNoRows {
            c.JSON(http.StatusUnauthorized, gin.H{"error": "Invalid
credentials"})
            return
        }
    }

```

```

    err = bcrypt.CompareHashAndPassword([]byte(user.Password),
[]byte(request.Password))

    if err != nil {
        c.JSON(http.StatusUnauthorized, gin.H{"error": "Invalid
credentials"})
        return
    }

    token, _ := utils.GenerateJWT(user.ID)

    utils.Logger.Println("LOGIN:", request.Phone)

    c.JSON(http.StatusOK, gin.H{
        "token": token,
    })
}

```

Листинг Б.4 – Файл check_phone.go

```

package handlers

import (
    "database/sql"
    "net/http"

    "guropnz/config"

    "github.com/gin-gonic/gin"
)

func CheckPhone(c *gin.Context) {

    phone := c.Param("phone")

    var id int

    err := config.DB.QueryRow(
        "SELECT id FROM users WHERE phone=$1",
        phone,
    ).Scan(&id)

    if err == sql.ErrNoRows {

        c.JSON(http.StatusOK, gin.H{
            "exists": false,
        })

        return
    }

    if err != nil {

        c.JSON(http.StatusInternalServerError, gin.H{

```

```

        "error": err.Error(),
    })

    return
}

c.JSON(http.StatusOK, gin.H{
    "exists": true,
})
}

```

Листинг Б.5 – Файл doctors.go

```

package handlers

import (
    "net/http"

    "guropnz/config"
    "guropnz/models"

    "github.com/gin-gonic/gin"
)

func GetDoctors(c *gin.Context) {

    rows, err := config.DB.Query(`
        SELECT id, full_name, specialty, image, experience
        FROM doctors
    `)

    if err != nil {

        c.JSON(http.StatusInternalServerError, gin.H{
            "error": err.Error(),
        })

        return
    }

    defer rows.Close()

    var doctors []models.Doctor

    for rows.Next() {

        var doctor models.Doctor

        rows.Scan(
            &doctor.ID,
            &doctor.FullName,
            &doctor.Specialty,
            &doctor.Image,
            &doctor.Experience,

```

```

        )

        doctors = append(doctors, doctor)
    }

    c.JSON(http.StatusOK, doctors)
}

```

Листинг Б.6 – Файл slots.go

```

package handlers

import (
    "net/http"

    "guropnz/config"

    "github.com/gin-gonic/gin"
)

func GetOccupiedSlots(c *gin.Context) {

    doctorID := c.Query("doctor_id")
    date := c.Query("date")

    rows, err := config.DB.Query(`
        SELECT appointment_time
        FROM appointments
        WHERE doctor_id=$1
        AND appointment_date=$2
    `,
        doctorID,
        date,
    )

    if err != nil {

        c.JSON(http.StatusInternalServerError, gin.H{
            "error": err.Error(),
        })

        return
    }

    defer rows.Close()

    var slots []string

    for rows.Next() {

        var time string

        rows.Scan(&time)
    }
}

```

```

        slots = append(slots, time)
    }

    c.JSON(http.StatusOK, gin.H{
        "occupied": slots,
    })
}
backend/logs/server.log
2026/05/17 17:54:50 REGISTER: +7 (987) 072-17-10
2026/05/17 17:55:25 LOGIN: +7 (987) 072-17-10
2026/05/17 18:08:02 LOGIN: +7 (987) 072-17-10
2026/05/17 18:08:03 LOGIN: +7 (987) 072-17-10
2026/05/17 18:08:27 LOGIN: +7 (987) 072-17-10

```

Листинг Б.7 – Файл `jwt_middleware.go`

```

package middleware

import (
    "net/http"
    "os"
    "strings"

    "github.com/gin-gonic/gin"
    "github.com/golang-jwt/jwt/v5"
)

func AuthMiddleware() gin.HandlerFunc {
    return func(c *gin.Context) {
        authHeader := c.GetHeader("Authorization")

        if authHeader == "" {
            c.JSON(http.StatusUnauthorized, gin.H{"error": "No
token"})
            c.Abort()
            return
        }

        tokenString := strings.Split(authHeader, "Bearer ")[1]

        token, err := jwt.Parse(tokenString, func(token
*jwt.Token) (interface{}, error) {
            return []byte(os.Getenv("JWT_SECRET")), nil
        })

        if err != nil || !token.Valid {
            c.JSON(http.StatusUnauthorized, gin.H{"error":
"Invalid token"})
            c.Abort()
            return
        }

        claims := token.Claims.(jwt.MapClaims)
        c.Set("user_id", int(claims["user_id"].(float64)))
    }
}

```

```

        c.Next()
    }
}

```

Листинг Б.8 – Файл appointment.go

```

package models

type Appointment struct {
    ID            int    `json:"id"`
    UserID        int    `json:"user_id"`
    DoctorID      int    `json:"doctor_id"`
    BirthDate     string `json:"birth_date"`
    Reason        string `json:"reason"`
    AppointmentDate string `json:"appointment_date"`
    AppointmentTime string `json:"appointment_time"`
}

```

Листинг Б.9 – Файл doctor.go

```

package models

type Doctor struct {
    ID            int    `json:"id"`
    FullName      string `json:"full_name"`
    Specialty     string `json:"specialty"`
    Image         string `json:"image"`
    Experience    string `json:"experience"`
}

```

Листинг Б.10 – Файл user.go

```

package models

type User struct {
    ID            int    `json:"id"`
    FullName      string `json:"full_name"`
    Phone        string `json:"phone"`
    Password     string `json:"password"`
}

```

Листинг Б.11 – Файл routes.go

```

package routes

import (
    "guropnz/handlers"
    "guropnz/middleware"

    "github.com/gin-gonic/gin"
)

func SetupRoutes(r *gin.Engine) {

```

```

api := r.Group("/api")

api.POST("/register", handlers.Register)
api.POST("/login", handlers.Login)

api.GET("/doctors", handlers.GetDoctors)

api.GET(
    "/check-phone/:phone",
    handlers.CheckPhone,
)

api.GET(
    "/occupied-slots",
    handlers.GetOccupiedSlots,
)

auth := api.Group("/")

auth.Use(middleware.AuthMiddleware())

auth.POST(
    "/appointments",
    handlers.CreateAppointment,
)

auth.GET(
    "/appointments",
    handlers.GetMyAppointments,
)

auth.DELETE(
    "/appointments/:id",
    handlers.DeleteAppointment,
)
}

```

Листинг Б.12 – Файл jwt_utils.go

```

package utils

import (
    "os"
    "time"

    "github.com/golang-jwt/jwt/v5"
)

func GenerateJWT(userID int) (string, error) {
    claims := jwt.MapClaims{
        "user_id": userID,
        "exp":     time.Now().Add(time.Hour * 72).Unix(),
    }
}

```

```

    token := jwt.NewWithClaims(jwt.SigningMethodHS256, claims)

    return token.SignedString([]byte(os.Getenv("JWT_SECRET")))
}

```

Листинг Б.13 – Файл logger.go

```

package utils

import (
    "io"
    "log"
    "os"
)

var Logger *log.Logger

func InitLogger() {

    os.MkdirAll("logs", os.ModePerm)

    file, err := os.OpenFile(
        "logs/server.log",
        os.O_APPEND|os.O_CREATE|os.O_WRONLY,
        0666,
    )

    if err != nil {
        panic(err)
    }

    multiWriter := io.MultiWriter(
        os.Stdout,
        file,
    )

    Logger = log.New(
        multiWriter,
        "",
        log.LstdFlags,
    )
}

backend/.env
DB_HOST=localhost
DB_PORT=5432
DB_USER=postgres
DB_PASSWORD=1234
DB_NAME=guropnz
JWT_SECRET=supersecretkey

```

Листинг Б.14 – Файл main.go

```

package main

```

```

func main() {
    godotenv.Load()

    utils.InitLogger()
    config.ConnectDB()

    r := gin.Default()

    r.Use(cors.Default())

    routes.SetupRoutes(r)

    log.Println("Server started on :8080")

    r.Run(":8080")
}

```

Листинг Б.15 – Файл schema.sql

```

CREATE TABLE users (
    id SERIAL PRIMARY KEY,
    full_name VARCHAR(255) NOT NULL,
    phone VARCHAR(50) UNIQUE NOT NULL,
    password VARCHAR(255) NOT NULL
);

CREATE TABLE doctors (
    id SERIAL PRIMARY KEY,
    full_name VARCHAR(255) NOT NULL,
    specialty VARCHAR(255) NOT NULL,
    image TEXT
);

CREATE TABLE appointments (
    id SERIAL PRIMARY KEY,
    user_id INTEGER REFERENCES users(id) ON DELETE CASCADE,
    doctor_id INTEGER REFERENCES doctors(id) ON DELETE CASCADE,
    birth_date DATE NOT NULL,
    reason TEXT NOT NULL,
    appointment_date DATE NOT NULL,
    appointment_time VARCHAR(10) NOT NULL,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);

```

Листинг Б.16 – Файл api.js

```
const API_URL = 'http://localhost:8080/api';
```

Листинг Б.17 – Файл app.js

```

const slides = document.querySelectorAll('.bg-slide');

let currentSlide = 0;

/* ===== */

```

```

/* BACKGROUND SLIDER */
/* ===== */

setInterval(() => {

    slides[currentSlide].classList.remove('active');

    currentSlide++;

    if (currentSlide >= slides.length) {
        currentSlide = 0;
    }

    slides[currentSlide].classList.add('active');

}, 5000);

/* ===== */
/* SCROLL */
/* ===== */

function scrollToSection(id) {

    document
        .getElementById(id)
        .scrollIntoView({
            behavior: 'smooth'
        });
}

/* ===== */
/* LOAD DOCTORS */
/* ===== */

async function loadDoctors() {

    try {

        console.log('Загрузка врачей...');

        const response = await fetch(
            'http://localhost:8080/api/doctors'
        );

        if (!response.ok) {
            throw new Error('Ошибка API');
        }

        const doctors = await response.json();

        console.log('Врачи:', doctors);

        const select = document.getElementById(

```

```

        'doctor-select'
    );

    const doctorsGrid = document.getElementById(
        'doctors-grid'
    );

    if (!select) {
        console.error('doctor-select не найден');
        return;
    }

    if (!doctorsGrid) {
        console.error('doctors-grid не найден');
        return;
    }

    /* очищаем */

    select.innerHTML = `
        <option value="">
            Выберите врача
        </option>
    `;

    doctorsGrid.innerHTML = '';

    /* добавляем врачей */

    doctors.forEach((doctor) => {

        /* SELECT */

        const option = document.createElement('option');

        option.value = doctor.id;

        option.textContent =
            `${doctor.full_name} — ${doctor.specialty}`;

        select.appendChild(option);

        /* CARD */

        const card = document.createElement('div');

        card.className = 'doctor-card';

        card.innerHTML = `

                                <div class="doctor-info">

                                    <h3>
                                        ${doctor.full_name}
                                    </h3>

                                    <p class="speciality">
                                        ${doctor.specialty}
                                    </p>

                                    <p class="experience">
                                        ${doctor.experience}
                                    </p>

                                </div>

                                `;

                                doctorsGrid.appendChild(card);

                                });

                                console.log('Врачи успешно загружены');

                                } catch (error) {

                                    console.error(
                                        'Ошибка загрузки врачей:',
                                        error
                                    );

                                }

                                }

/* ===== */
/* INIT */
/* ===== */

window.onload = () => {

    const token = localStorage.getItem('token');

    if (token) {

        document
            .getElementById('auth-screen')
            .classList.add('hidden');

        document
            .getElementById('site')

```

```

        .classList.remove('hidden');

        loadDoctors();
    }
};

```

Листинг Б.18 – Файл appointment.js

```

let selectedTime = null;

async function generateTimeSlots() {

    const doctor =
        document.getElementById('doctor-select').value;

    const date =
        document.getElementById('appointment-date').value;

    const container =
        document.getElementById('time-slots');

    container.innerHTML = '';

    if (!doctor || !date) {
        return;
    }

    const response = await fetch(
        `http://localhost:8080/api/occupied-
slots?doctor_id=${doctor}&date=${date}`
    );

    const data = await response.json();

    const occupied =
        data.occupied || [];

    for (let hour = 7; hour < 19; hour++) {

        for (let minute = 0; minute < 60; minute += 30) {

            const time =
                `${String(hour).padStart(2,
'0')}:${String(minute).padStart(2, '0')}`;

            const slot =
                document.createElement('div');

            slot.className = 'time-slot';

            slot.innerText = time;

            if (occupied.includes(time)) {

```

```

        slot.classList.add('occupied');
    }
    else {

        slot.onclick = () => {

            document
                .querySelectorAll('.time-slot')
                .forEach(el => {
                    el.classList.remove('active');
                });

            slot.classList.add('active');

            selectedTime = time;

            validateAppointmentForm();
        };
    }

    container.appendChild(slot);
}
}
}

function validateAppointmentForm() {

    const doctor =
        document.getElementById('doctor-select').value;

    const reason =
        document.getElementById('reason').value.trim();

    const date =
        document.getElementById('appointment-date').value;

    const button =
        document.getElementById('appointment-btn');

    if (
        doctor &&
        reason &&
        date &&
        selectedTime
    ) {

        button.classList.remove('disabled-btn');
    }
    else {

        button.classList.add('disabled-btn');
    }
}

```

```

async function createAppointment() {

    const token =
        localStorage.getItem('token');

    const doctorId =
        document.getElementById('doctor-select').value;

    const reason =
        document.getElementById('reason').value;

    const appointmentDate =
        document.getElementById('appointment-date').value;

    const response = await fetch(
        'http://localhost:8080/api/appointments',
        {
            method: 'POST',

            headers: {
                'Content-Type': 'application/json',
                'Authorization': `Bearer ${token}`
            },

            body: JSON.stringify({

                doctor_id: Number(doctorId),

                birth_date: '2000-01-01',

                reason: reason,

                appointment_date: appointmentDate,

                appointment_time: selectedTime
            })
        }
    );

    const data = await response.json();

    if (!response.ok) {

        alert(data.error);

        return;
    }

    alert('Запись успешно создана');

    generateTimeSlots();
}

```

Листинг Б.19 – Файл auth.js

```

const API_URL = 'http://localhost:8080/api';

let currentPhone = '';

async function checkPhone() {

    const phoneInput =
        document.getElementById('phone-check');

    const phone =
        phoneInput.value.trim();

    if (!/^\d{11}$/.test(phone)) {
        alert('Введите 11 цифр номера');
        return;
    }

    currentPhone = phone;

    try {

        const response = await fetch(
            `${API_URL}/check-phone/${phone}`
        );

        const data = await response.json();

        document
            .getElementById('phone-step')
            .classList.add('hidden');

        if (data.exists) {

            document
                .getElementById('login-step')
                .classList.remove('hidden');

        }
        else {

            document
                .getElementById('register-step')
                .classList.remove('hidden');

        }

    } catch (error) {

        console.error(error);

        alert('Ошибка сервера');

    }

}

```

```

async function register() {

    const fullName =
        document.getElementById('full-name').value.trim();

    const password =
        document.getElementById('password').value;

    const confirmPassword =
        document.getElementById('confirm-password').value;

    const passwordRegex =
        /^(?=.*[0-9])(?=.*[!@#$%^&*]).{8,}$/;

    if (!fullName) {
        alert('Введите ФИО');
        return;
    }

    if (!passwordRegex.test(password)) {
        alert(
            'Пароль должен содержать минимум 8 символов, цифру и
спецсимвол'
        );

        return;
    }

    if (password !== confirmPassword) {
        alert('Пароли не совпадают');

        return;
    }

    const response = await fetch(
        `${API_URL}/register`,
        {
            method: 'POST',

            headers: {
                'Content-Type': 'application/json'
            },

            body: JSON.stringify({
                full_name: fullName,
                phone: currentPhone,
                password: password
            })
        }
    );

```

```

const data = await response.json();

if (!response.ok) {

    alert(data.error || 'Ошибка регистрации');

    return;
}

await login(password);
}

async function login(customPassword = null) {

    const password =
        customPassword ||
        document.getElementById('login-password').value;

    const response = await fetch(
        `${API_URL}/login`,
        {
            method: 'POST',

            headers: {
                'Content-Type': 'application/json'
            },

            body: JSON.stringify({
                phone: currentPhone,
                password: password
            })
        }
    );

    const data = await response.json();

    if (!response.ok) {

        alert(data.error || 'Ошибка входа');

        return;
    }

    localStorage.setItem('token', data.token);

    document
        .getElementById('auth-screen')
        .classList.add('hidden');

    document
        .getElementById('site')
        .classList.remove('hidden');

```

```

        loadDoctors();
    }

```

Листинг Б.20 – Файл index.html

```

<!DOCTYPE html>
<html lang="ru">

<head>

    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-
scale=1.0">

    <title>GUROPNZ</title>

    <link rel="stylesheet" href="styles.css">

</head>

<body>

    <div class="background-slider">

        <div class="bg-slide active" style="background-
image:url('assets/images/fon1.jpg')">
        </div>

        <div class="bg-slide" style="background-
image:url('assets/images/fon2.jpg')">
        </div>

        <div class="bg-slide" style="background-
image:url('assets/images/fon3.jpg')">
        </div>

    </div>

    <!-- AUTH -->

    <div id="auth-screen" class="auth-screen">

        <div class="auth-card">

            <div class="auth-top">
                <div class="auth-logo"></div>

                <span>
                    Медицинская клиника
                </span>
            </div>

            <div id="phone-step">

```

```

<h1>
    Добро пожаловать
</h1>

<p>
    Введите номер телефона для входа
    или регистрации в системе.
</p>

<div class="input-block">

    <label>
        Номер телефона
    </label>

    <input type="text" id="phone-check"
placeholder="79999999999">

</div>

<button onclick="checkPhone()">
    Продолжить
</button>

</div>

<div id="login-step" class="hidden">

    <h1>
        Введите пароль
    </h1>

    <p>
        Авторизация в личный кабинет
    </p>

    <div class="input-block">

        <label>
            Пароль
        </label>

        <input type="password" id="login-password"
placeholder="Введите пароль">

    </div>

    <button onclick="login()">
        Войти
    </button>

</div>

```

```

<div id="register-step" class="hidden">

    <h1>
        Создание аккаунта
    </h1>

    <p>
        Заполните данные для регистрации
    </p>

    <div class="input-block">

        <label>
            ФИО
        </label>

        <input type="text" id="full-name"
placeholder="Введите ФИО">

    </div>

    <div class="input-block">

        <label>
            Пароль
        </label>

        <input type="password" id="password"
placeholder="Введите пароль">

    </div>

    <div class="input-block">

        <label>
            Повторите пароль
        </label>

        <input type="password" id="confirm-
password" placeholder="Повторите пароль">

    </div>

    <button onclick="register()">
        Создать аккаунт
    </button>

</div>

</div>

</div>

```

```

<div id="site" class="hidden">

    <header class="topbar">

        <div class="logo">
            GUROPNZ
        </div>

        <nav>

            <button onclick="scrollToSection('hero')">
                Главная
            </button>

            <button
onclick="scrollToSection('appointment')">
                Онлайн запись
            </button>

            <button
onclick="scrollToSection('specialists')">
                Специалисты
            </button>

            <button onclick="scrollToSection('contacts')">
                Контакты
            </button>

        </nav>

    </header>

    <section id="hero" class="section hero-section">

        <div class="hero-content">

            <span class="hero-subtitle">
                Частная медицинская клиника
            </span>

            <h1>
                Современная
                <br>
                медицина
                <br>
                нового уровня
            </h1>

            <p>
                Профессиональные специалисты,
                современное оборудование
                и комфортный сервис
            </p>

```

```

        для пациентов.
    </p>

    <div class="hero-buttons">

        <button
onclick="scrollToSection('appointment')">
            Записаться онлайн
        </button>

        <button class="secondary-btn"
onclick="scrollToSection('contacts')">
            О клинике
        </button>

    </div>

</div>

<div class="hero-side-card">

    <div class="hero-card-top">
        <span>
            Работаем ежедневно
        </span>

        <h3>
            07:00 — 19:00
        </h3>
    </div>

    <div class="hero-card-bottom">

        <div>
            Современные кабинеты
        </div>

        <div>
            Онлайн запись
        </div>

        <div>
            Квалифицированные врачи
        </div>

    </div>

</div>

</section>

<section id="appointment" class="section">

```

```

<div class="appointment-container">

    <div class="section-header">

        <span>
            Онлайн запись
        </span>

        <h2>
            Запись на прием
        </h2>

        <p>
            Выберите врача,
            удобную дату и время посещения.
        </p>

    </div>

    <div class="appointment-grid">

        <div class="field-card">

            <label>
                Специалист
            </label>

            <select id="doctor-select">
                <option value="">
                    Выберите врача
                </option>
            </select>

        </div>

        <div class="field-card">

            <label>
                Причина обращения
            </label>

            <textarea id="reason"
placeholder="Опишите причину обращения"></textarea>

        </div>

        <div class="field-card">

            <label>
                Дата приема
            </label>

```

```

date">
        <input type="date" id="appointment-

        </div>

    </div>

    <div class="times-block">

        <div class="times-header">

            <h3>
                Доступное время
            </h3>

            <span>
                Красным отмечено занятое время
            </span>

        </div>

        <div id="time-slots" class="time-
slots"></div>

    </div>

    <button id="appointment-btn" class="disabled-
btn" onclick="createAppointment()">
        Записаться на прием
    </button>

</div>

</section>

<section id="specialists" class="section">

    <div class="section-header">

        <span>
            Специалисты
        </span>

        <h2>
            Наши врачи
        </h2>

    </div>

    <div id="doctors-grid" class="doctors-grid"></div>

</section>

```

```

<!-- CONTACTS -->

<section id="contacts" class="section contacts-section">

    <div class="contacts-info">

        <span>
            Контакты
        </span>

        <h2>
            Клиника доктора Гуро
        </h2>

        <p>
            Адрес: Пенза
        </p>

        <p>
            Телефон: +7 (000) 000-00-00
        </p>

        <p>
            Время работы: 07:00 – 19:00
        </p>

    </div>

    <div class="map-block">
        Карта клиники
    </div>

</section>

</div>

<script src="api.js"></script>
<script src="auth.js"></script>
<script src="app.js"></script>
<script src="appointments.js"></script>

</body>

</html>

```

Листинг Б.21 – Файл style.css

```

@import
url('https://fonts.googleapis.com/css2?family=Inter:wght@300;400;5
00;600;700;800;900&display=swap');

* {
    margin: 0;
    padding: 0;

```

```

        box-sizing: border-box;
    }

    html {
        scroll-behavior: smooth;
    }

    body {
        font-family: 'Inter', sans-serif;
        background: #071018;
        color: white;
        overflow-x: hidden;
    }

    .background-slider {
        position: fixed;
        inset: 0;
        z-index: -10;
    }

    .bg-slide {
        position: absolute;
        inset: 0;

        background-size: cover;
        background-position: center;

        opacity: 0;

        transition: opacity 1.6s ease;
    }

    .bg-slide.active {
        opacity: 1;
    }

    .background-slider::after {
        content: '';

        position: absolute;
        inset: 0;

        background:
            linear-gradient(to bottom,
                rgba(4, 10, 18, .72),
                rgba(4, 10, 18, .94));

        backdrop-filter: blur(6px);
    }

    .hidden {
        display: none !important;
    }

```

```

.section {
  min-height: 100vh;

  padding:
    140px 80px;

  position: relative;
}

.auth-screen {
  width: 100%;
  height: 100vh;

  display: flex;
  justify-content: center;
  align-items: center;

  padding: 40px;
}

.auth-card {
  width: 520px;

  background:
    rgba(255, 255, 255, .05);

  border:
    1px solid rgba(255, 255, 255, .08);

  backdrop-filter: blur(30px);

  border-radius: 34px;

  padding: 48px;

  box-shadow:
    0 25px 80px rgba(0, 0, 0, .45);
}

.auth-top {
  display: flex;
  align-items: center;
  gap: 14px;

  margin-bottom: 40px;
}

.auth-logo {
  width: 16px;
  height: 16px;

  border-radius: 50%;

```

```

        background:
            linear-gradient(135deg,
                #28d1ff,
                #4776ff);

        box-shadow:
            0 0 24px rgba(40, 209, 255, .7);
    }

    .auth-top span {
        color: #8ce8ff;

        font-size: 13px;
        font-weight: 700;

        letter-spacing: 2px;
        text-transform: uppercase;
    }

    .auth-card h1 {
        font-size: 54px;
        line-height: 1;

        font-weight: 800;
    }

    .auth-card p {
        margin-top: 18px;
        margin-bottom: 42px;

        color: rgba(255, 255, 255, .65);

        line-height: 1.8;
    }

    .input-block {
        margin-bottom: 22px;
    }

    .input-block label {
        display: block;

        margin-bottom: 12px;

        color: #8ce8ff;

        font-size: 14px;
        font-weight: 600;
    }

    .input-block input {
        width: 100%;
    }

```

```

    height: 64px;

    border: none;
    outline: none;

    padding: 0 22px;

    border-radius: 18px;

    background:
        rgba(255, 255, 255, .06);

    border:
        1px solid rgba(255, 255, 255, .08);

    color: white;

    font-size: 15px;

    transition: .3s;
}

.auth-card button {
    width: 100%;
    height: 64px;

    margin-top: 10px;

    border: none;
    border-radius: 18px;

    cursor: pointer;

    background:
        linear-gradient(135deg,
            #28d1ff,
            #4776ff);

    color: white;

    font-size: 16px;
    font-weight: 700;

    transition: .35s;
}

.auth-card button:hover {
    transform: translateY(-2px);
}

.topbar {
    position: fixed;

```

```

top: 0;
left: 0;

width: 100%;

display: flex;
justify-content: space-between;
align-items: center;

padding:
    26px 70px;

z-index: 1000;

backdrop-filter: blur(20px);

background:
    rgba(4, 10, 18, .35);

border-bottom:
    1px solid rgba(255, 255, 255, .05);
}

.logo {
    font-size: 28px;
    font-weight: 900;
}

nav {
    display: flex;
    gap: 12px;
}

nav button {
    height: 48px;

    padding:
        0 22px;

    border: none;
    border-radius: 14px;

    cursor: pointer;

    background:
        rgba(255, 255, 255, .05);

    border:
        1px solid rgba(255, 255, 255, .07);

    color: white;

    transition: .3s;

```

```

}

nav button:hover {
    transform: translateY(-2px);

    background:
        rgba(255, 255, 255, .09);
}

.hero-section {
    display: flex;
    justify-content: space-between;
    align-items: center;

    gap: 60px;
}

.hero-content {
    max-width: 900px;
}

.hero-subtitle {
    display: inline-flex;

    padding:
        12px 22px;

    border-radius: 999px;

    background:
        rgba(115, 220, 255, .12);

    border:
        1px solid rgba(115, 220, 255, .14);

    color: #8ce8ff;

    font-size: 13px;
    font-weight: 700;
}

.hero-content h1 {
    margin-top: 34px;

    font-size: 108px;
    line-height: .92;

    font-weight: 900;

    letter-spacing: -5px;
}

```

```

.hero-content p {
  margin-top: 32px;

  max-width: 700px;

  color:
    rgba(255, 255, 255, .72);

  line-height: 1.9;

  font-size: 20px;
}

.hero-buttons {
  display: flex;
  gap: 18px;

  margin-top: 52px;
}

.hero-buttons button {
  height: 64px;

  padding:
    0 34px;

  border: none;
  border-radius: 18px;

  cursor: pointer;

  color: white;

  font-size: 15px;
  font-weight: 700;

  transition: .35s;
}

.hero-buttons button:first-child {
  background:
    linear-gradient(135deg,
      #28d1ff,
      #4776ff);
}

.secondary-btn {
  background:
    rgba(255, 255, 255, .06) !important;

  border:
    1px solid rgba(255, 255, 255, .08) !important;
}

```

```

.hero-side-card {
  width: 360px;

  padding: 36px;

  border-radius: 32px;

  background:
    rgba(255, 255, 255, .05);

  border:
    1px solid rgba(255, 255, 255, .08);

  backdrop-filter: blur(26px);
}

.hero-card-top span {
  color: #8ce8ff;

  font-size: 13px;
  font-weight: 700;
}

.hero-card-top h3 {
  margin-top: 18px;

  font-size: 54px;
}

.hero-card-bottom {
  margin-top: 40px;

  display: flex;
  flex-direction: column;
  gap: 16px;
}

.hero-card-bottom div {
  padding: 18px;

  border-radius: 16px;

  background:
    rgba(255, 255, 255, .04);

  border:
    1px solid rgba(255, 255, 255, .05);
}

.section-header span {
  color: #8ce8ff;
}

```

```

        letter-spacing: 2px;
        text-transform: uppercase;

        font-size: 13px;
        font-weight: 700;
    }

.section-header h2 {
    margin-top: 18px;

    font-size: 72px;
    line-height: 1;

    font-weight: 800;
}

.section-header p {
    margin-top: 24px;

    color:
        rgba(255, 255, 255, .7);

    line-height: 1.8;

    font-size: 18px;
}

.appointment-container {
    margin-top: 50px;

    padding: 54px;

    border-radius: 36px;

    background:
        rgba(255, 255, 255, .05);

    border:
        1px solid rgba(255, 255, 255, .08);

    backdrop-filter: blur(24px);
}

.appointment-grid {
    margin-top: 50px;

    display: grid;

    grid-template-columns:
        repeat(3, 1fr);

    gap: 24px;

```

```

        align-items: start;
    }

    .field-card {
        display: flex;
        flex-direction: column;
    }

    select option {
        background: #0c1720;
        color: white;
    }

    .times-block {
        margin-top: 60px;
    }

    .times-header {
        display: flex;
        justify-content: space-between;
        align-items: center;

        margin-bottom: 26px;
    }

    .times-header h3 {
        font-size: 28px;
    }

    .times-header span {
        color:
            rgba(255, 255, 255, .6);
    }

    .time-slots {
        display: flex;
        flex-wrap: wrap;

        gap: 14px;
    }

    .time-slot {
        width: 96px;
        height: 54px;

        display: flex;
        justify-content: center;
        align-items: center;

        border-radius: 16px;

        background:
            rgba(255, 255, 255, .05);
    }

```

```

border:
  1px solid rgba(255, 255, 255, .08);

cursor: pointer;

transition: .25s;
}

.time-slot:hover {
  transform: translateY(-2px);
}

.time-slot.active {
  background:
    linear-gradient(135deg,
      #28d1ff,
      #4776ff);
}

.time-slot.occupied {
  background:
    rgba(255, 80, 80, .18);

  border:
    1px solid rgba(255, 100, 100, .4);

  color: #ffb5b5;

  pointer-events: none;
}

#appointment-btn {
  width: 100%;
  height: 66px;

  margin-top: 52px;

  border: none;
  border-radius: 18px;

  cursor: pointer;

  background:
    linear-gradient(135deg,
      #28d1ff,
      #4776ff);

  color: white;

  font-size: 16px;
  font-weight: 700;

```

```

        transition: .35s;
    }

.disabled-btn {
    opacity: .45;
    pointer-events: none;
}

.doctors-grid {
    margin-top: 70px;

    display: grid;

    grid-template-columns:
        repeat(3, 1fr);

    gap: 30px;
}

.doctor-card {
    overflow: hidden;

    border-radius: 30px;

    background:
        rgba(255, 255, 255, .05);

    border:
        1px solid rgba(255, 255, 255, .08);

    backdrop-filter: blur(24px);

    transition: .35s;
}

.doctor-card:hover {
    transform: translateY(-8px);

    box-shadow:
        0 24px 50px rgba(0, 0, 0, .35);
}

.doctor-card img {
    width: 100%;
    height: 420px;

    object-fit: cover;

    display: block;
}

.doctor-info {
    padding: 28px;

```

```

}

.doctor-info h3 {
    font-size: 30px;
    line-height: 1.2;
}

.speciality {
    margin-top: 14px;

    color: #8ce8ff;

    font-size: 15px;
    font-weight: 600;
}

.experience {
    margin-top: 18px;

    color:
        rgba(255, 255, 255, .66);

    line-height: 1.8;
}

.contacts-section {
    display: grid;

    grid-template-columns:
        1fr 1fr;

    gap: 40px;

    align-items: center;
}

.contacts-info {
    padding: 48px;

    border-radius: 32px;

    background:
        rgba(255, 255, 255, .05);

    border:
        1px solid rgba(255, 255, 255, .08);

    backdrop-filter: blur(24px);
}

.contacts-info span {
    color: #8ce8ff;
}

```

```

    font-size: 13px;
    font-weight: 700;

    letter-spacing: 2px;
    text-transform: uppercase;
}

.contacts-info h2 {
    margin-top: 20px;

    font-size: 58px;
    line-height: 1;
}

.contacts-info p {
    margin-top: 26px;

    color:
        rgba(255, 255, 255, .72);

    line-height: 1.8;

    font-size: 18px;
}

.map-block {
    height: 540px;

    border-radius: 32px;

    background:
        rgba(255, 255, 255, .05);

    border:
        1px solid rgba(255, 255, 255, .08);

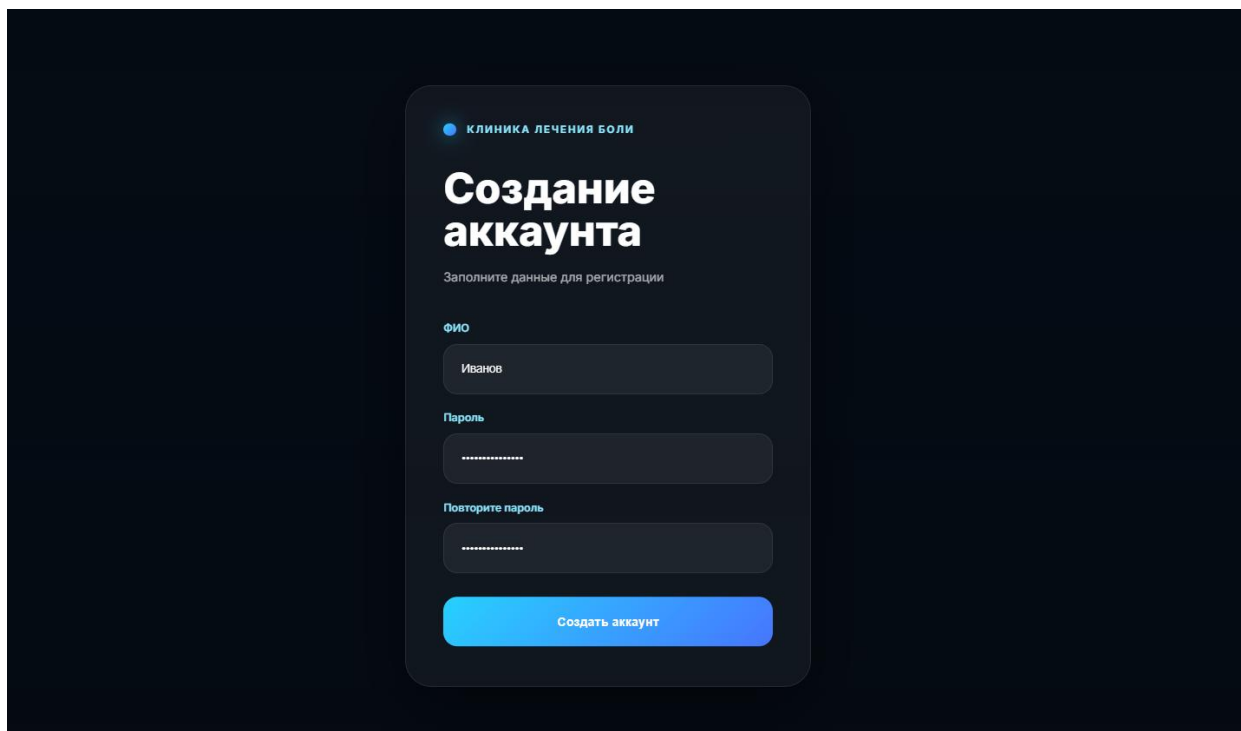
    display: flex;
    justify-content: center;
    align-items: center;

    font-size: 28px;
    font-weight: 700;
}

```

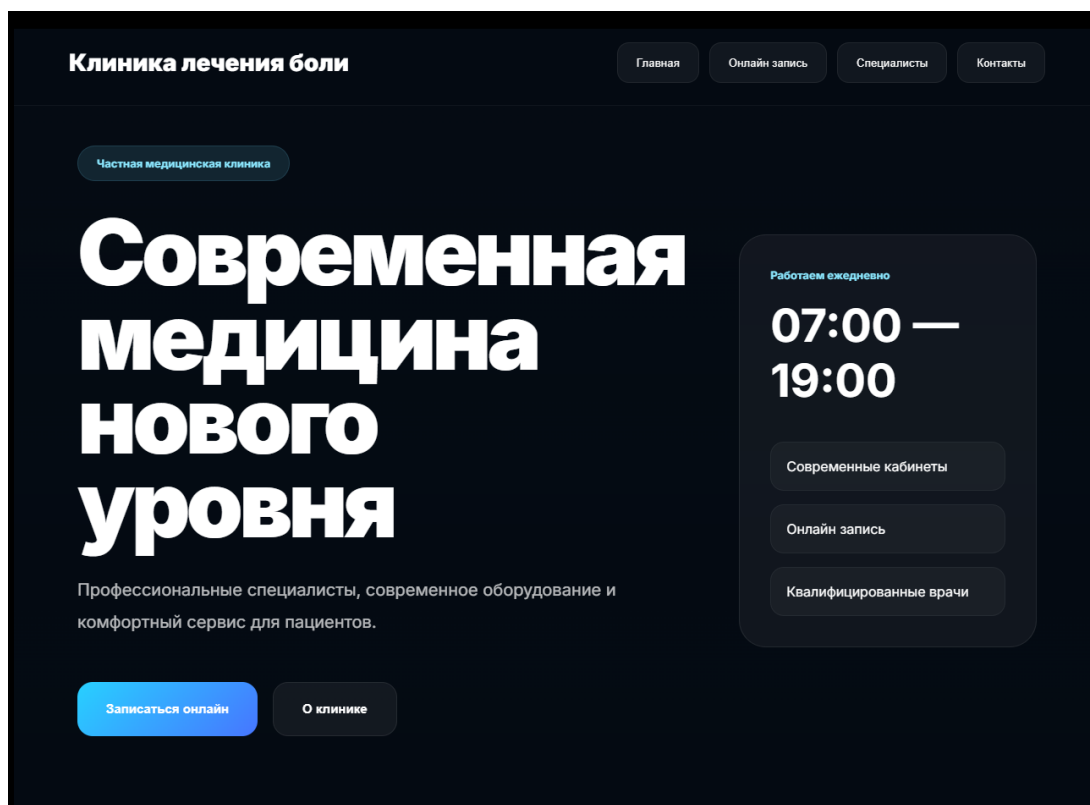
Приложение В
(рекомендуемое)
Результаты тестирования

Ниже, на рисунках представлены скриншоты экрана, подтверждающие правильность функционирования разработанного программного обеспечения.



The screenshot shows a registration form titled "Создание аккаунта" (Account Creation) for "КЛИНИКА ЛЕЧЕНИЯ БОЛИ". The form includes fields for "ФИО" (Name and Surname) with the value "Иванов", "Пароль" (Password), and "Повторите пароль" (Repeat password). A blue button labeled "Создать аккаунт" (Create account) is at the bottom.

Рисунок В.1 – Экран регистрация



The screenshot shows the main interface of the "Клиника лечения боли" (Pain Treatment Clinic) program. The header includes the clinic name and navigation buttons: "Главная" (Home), "Онлайн запись" (Online booking), "Специалисты" (Specialists), and "Контакты" (Contacts). The main content area features the text "Современная медицина нового уровня" (Modern medicine of a new level) and "Профессиональные специалисты, современное оборудование и комфортный сервис для пациентов." (Professional specialists, modern equipment and comfortable service for patients). A blue button "Записаться онлайн" (Book online) and a grey button "О клинике" (About the clinic) are at the bottom left. On the right, a box displays "Работаем ежедневно 07:00 — 19:00" (We work daily 07:00 — 19:00) and three buttons: "Современные кабинеты" (Modern cabinets), "Онлайн запись" (Online booking), and "Квалифицированные врачи" (Qualified doctors).

Рисунок В.2 – Главный экран программы

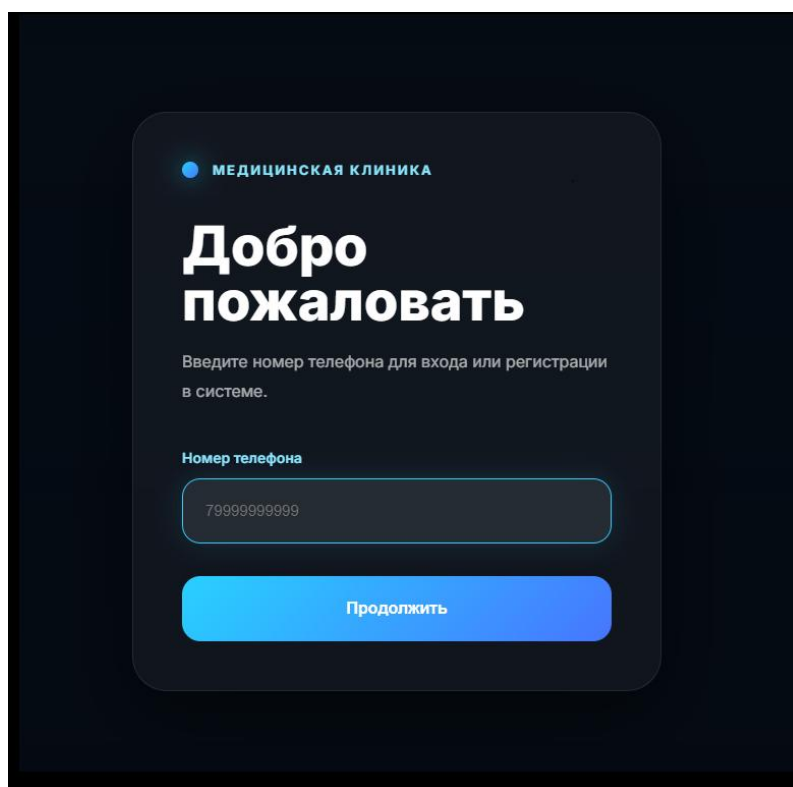


Рисунок В.3 – Экран авторизации

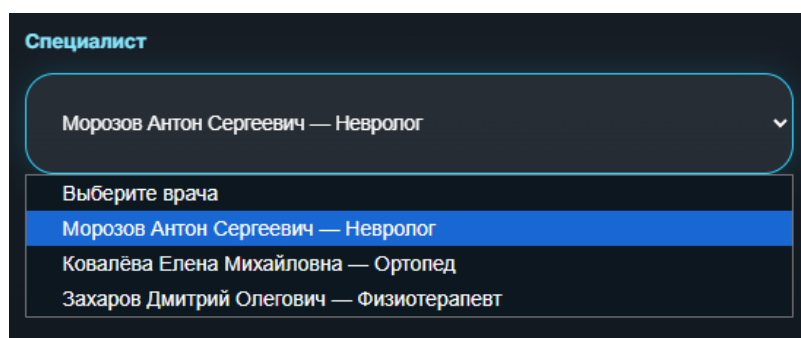


Рисунок В.4 – Выбор лечащего врача

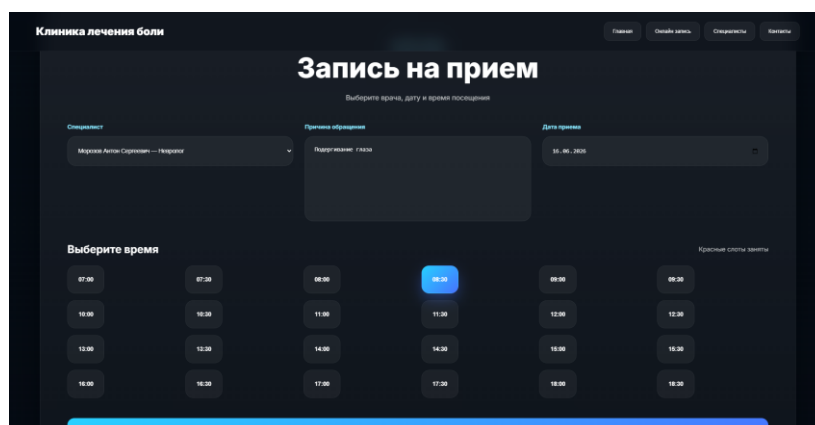


Рисунок В.5 – Общий вид экрана записи

Рисунок В.6– Выбор даты приема

Рисунок В.6– Выбор даты приема

Рисунок В.6– Выбор даты приема

Рисунок В.8 – Получение данных о записи на сервер



Рисунок В.9 – Логирование записи

The screenshot shows the GUROPNZ web application interface. At the top, there are navigation links: Главная, Онлайн запись, Специалисты, and Контакты. The main form is divided into three sections: Специалист (Specialist), Причина обращения (Reason for appeal), and Дата приема (Appointment date). The Specialist section has a dropdown menu showing 'Анна Иванова — Терапевт'. The Reason for appeal section has a text input field with the letter 'd'. The Appointment date section has a date picker showing '20.05.2026'. Below these sections is a section titled 'Выберите время' (Select time) with a grid of time slots. The 15:30 slot is highlighted in blue. To the right of the grid, it says 'Красные слоты заняты' (Red slots are occupied). At the bottom, there is a large blue button labeled 'Записаться на прием' (Book appointment).

Рисунок В.10 – Отсутствие возможности записи без указания достаточной причины обращения

Приложение Г

(рекомендуемое)

Акт о внедрении результатов работы

Российская Федерация г. Пенза

Общество с ограниченной ответственностью «Клиника лечения боли»

Юридический адрес: Кижеватова ул. , 27 «А» - 57, г. Пенза, 440031
Местонахождение: Набережная реки Пензы ул. , 3а. г. Пенза, 440010
Тел.: (8412) 39-39-16; (8412) 305-709.
Моб.: 89272893916;
E-mail: drguro@yandex.ru
ОГРН: 1035803509180
ОКПО: 71233104

ИНН 5837020989, КПП 583701001
р. счет 40702810015850001635
Филиал «Центральный» Банка ВТБ (ПАО) г. Москва
БИК: 044525411, к. счет №30101810145250000411
Лицензия: ЛО-58-01-001470 от 24.11.2014 г.
выдана МИНИСТЕРСТВОМ ЗДРАВООХРАНЕНИЯ
И СОЦИАЛЬНОГО РАЗВИТИЯ ПЕНЗЕНСКОЙ ОБЛАСТИ

11.06.2026

Акт о внедрении

Результаты теоретических и практических исследований, приведённые в выпускной квалификационной работе Гуро Николая Валерьевича, на тему «Разработка автоматизированной информационной системы сопровождения лечебно-диагностического процесса частного медицинского кабинета», внедрены в практическую деятельность ООО «Клиника лечения боли».

Главный врач

МО ООО «Клиника лечения боли»

Заместитель главного врача

МО ООО «Клиника лечения боли»

по административно-хозяйственной
деятельности

Гуро

В. В. Гуро

Колотилин

М. А. Колотилин

