Министерство образования и науки Республики Башкортостан

Государственное автономное профессиональное образовательное учреждение

Уфимский колледж статистики, информатики и вычислительной техники

|  |  |
| --- | --- |
|  | УТВЕРЖДАЮ  Заместитель директора  по учебной работе  З.З. Курмашева  « » 2025г. |

ПРОЕКТИРОВАНИЕ БАЗЫ ДАННЫХ И РАЗРАБОТКА ПРИЛОЖЕНИЯ

для автоматизации работы регистратуры поликлиники

Пояснительная записка к курсовому проекту

МДК.11.01 Технология разработки и защиты баз данных

|  |  |
| --- | --- |
|  | Руководитель проекта  Р. Ф. Каримова  «\_\_\_» \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ 2025 г. |
|  | Студент гр. 22П-1  Р. Р. Сафина  «\_\_\_» \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ 2025 г. |

2025

Министерство образования и науки Республики Башкортостан

Государственное автономное профессиональное образовательное учреждение

Уфимский колледж статистики, информатики и вычислительной техники

|  |  |
| --- | --- |
|  | УТВЕРЖДАЮ  Заместитель директора  по учебной работе  З.З. Курмашева  « » 2025 г. |

ЗАДАНИЕ

на курсовой проект студенту дневного отделения, группы 22П-1, специальности 09.02.07 Информационные системы и программирование.

Фамилия, имя, отчество: Сафина Регина Рифовна

Тема курсового проекта: «Проектирование базы данных и разработка приложения для автоматизации работы регистратуры поликлиники».

Текст задания:

при выполнении курсового проекта должны быть решены следующие задачи:

1. спроектирована база данных;
2. разработана и реализована структура программы;
3. реализованы функции записи пациентов на приём в удобное для них время, составления графиков работы врачей и их просмотр, регистрация и авторизация пользователей.

В результате выполнения курсового проекта должны быть представлены:

1. пояснительная записка, состоящая из следующих разделов:

Введение

1 Постановка задачи

2 Экспериментальный раздел

Заключение

Приложения

Список сокращений

Список использованных источников

1. электронный носитель, содержащий разработанный программный продукт;
2. презентация курсового проекта в электронном виде.

Список рекомендуемых источников:

1. ГОСТ 34.321-96 Информационные технологии. Система стандартов по базам данных. Эталонная модель управления данными : межгосударственный стандарт : дата введения 2001-07-01 - https://internet-law.ru/gosts/gost/6808/?ysclid=led2xw9tta320329480 - Текст : электронный.
2. Основы проектирования баз данных : учебник для студ. учреждений сред. проф. Образования Г. Н. Федорова. - М. : Издательский центр «Академия», 2017. - 224 с.
3. Гордеев, С. И. Организация баз данных в 2 ч. Часть 2: учебник для вузов / С. И. Гордеев, В. Н. Волошина. — 2-е изд., испр. и доп. — Москва: Издательство Юрайт, 2023. — 513 с. — (Высшее образование). — ISBN 978-5-534-04470-6. — Текст: электронный // Образовательная платформа Юрайт [сайт]. — URL: https://urait.ru/bcode/515097
4. Документация по MySQL. URL : <https://dev.mysql.com/doc/>
5. Metanit: Сайт о программировании. URL: <https://metanit.com/>
6. Nweb42. Основы Windows Presentation Foundation (WPF). URL: <https://nweb42.com/books/csharp/osnovy-windows-presentation-foundation-wpf/>

Задание к выполнению получил «29» января 2025 г.

Студент \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Сафина Регина Рифовна

Срок окончания «30» мая 2025 г.

Руководитель курсового проекта Р.Ф. Каримова

Задание рассмотрено на заседании цикловой комиссии информатики

протокол № 4 от «11» января 2025 г.

Председатель цикловой комиссии информатики О.В.Фатхулова

Министерство образования и науки Республики Башкортостан

Государственное автономное профессиональное образовательное учреждение

Уфимский колледж статистики, информатики и вычислительной техники

ЗАКЛЮЧЕНИЕ

на курсовой проект

Студент Сафина Регина Рифовна

Группа 22П-1

Специальность 09.02.07 Информационные системы и программирование

Тема Проектирование базы данных и разработка приложения для автоматизации работы регистратуры поликлиники

Объем курсового проекта:

количество листов пояснительной записки

количество листов графической части

Заключение о степени соответствия заданию на курсовое проектирование

Характеристика качеств, проявленных студентом при работе над проектом: самостоятельность, дисциплинированность, умение планировать работу и пользоваться литературным материалом и т.д.

Положительные стороны курсового проекта

Недостатки курсового проекта

Характеристика общетехнической и специальной подготовки студента

Заключение и предлагаемая оценка за курсовой проект

Руководитель курсового проекта \_\_\_\_\_\_\_\_\_Р.Ф. Каримова

«\_\_\_\_» 2025 г.

Подпись

АННОТАЦИЯ

Пояснительная записка к курсовому проекту содержит постановку и программу решения задачи «Проектирование базы данных и разработка приложения для автоматизации работы регистратуры поликлиники».

Программа Polyclinic.exe написана на языке C# в среде программирования Visual Studio 2022 с использованием системы управления базой данных MySQL, предназначена для работы в операционной системе MS Windows 10 и выше, отлажена на данных контрольного примера.

СОДЕРЖАНИЕ

|  |  |
| --- | --- |
|  | лист |
| Введение | 4 |
| 1 Постановка задачи | 6 |
| 1.1 Описание предметной области | 6 |
| 1.2 Описание входной информации | 8 |
| 1.3 Описание выходной информации | 8 |
| 1.4 Концептуальное моделирование | 9 |
| 1.5 Логическое моделирование | 10 |
| 1.6 Описание структуры базы данных | 12 |
| 1.7 Контрольный пример | 14 |
| 1.8 Общие требования к программному продукту | 14 |
| 2 Экспериментальный раздел | 16 |
| 2.1 Описание программы | 16 |
| 2.2 Протокол тестирования программного продукта | 25 |
| 2.3 Руководство пользователя | 34 |
| 2.4 Меры по обеспечению защиты информации | 44 |
| Заключение | 46 |
| Приложение А. Концептуальная модель базы данных  Приложение Б. Схема данных  Приложение В. Входная информация контрольного примера  Приложение Г. Выходные данные контрольного примера | 47  48  49  50 |
| Список использованных источников | 107 |

ВВЕДЕНИЕ

Медицинские учреждения, такие как поликлиники, играют важную роль в обеспечении здоровья всего населения. Одним из наболее важных аспектов работы поликлиники является эффективное управление потоками людей (пациентов), ведение медицинских карт и правильная организация записи на прием к врачам. В условиях современного мира, где время - один из самых ценных ресурсов, автоматизация процессов в регистратуре становится необходимостью. Это позволяет не только ускорить обслуживание пациентов, но и повышает точность и качество учета данных, предотвратив их от возможной потери.

Главная особенность работы регистратуры поликлиники заключается в необходимости оперативного управления большими объёмами информации: данными о пациентах, работающих специалистах, их расписании и записях на приём.

Актуальность данной работы обусловлена тем, что в настоящее время многие поликлиники до сих пор сталкиваются с проблемами организации и учета данных. Несмотря на развитие электронных сервисов, сохраняются недостатки, связанные с несовершенством существующих решений.

Традиционные методы ведения бумажных карточек и журналов учёта становятся неэффективны: они часто приводят к ошибкам, задержкам и неудобствам для медицинского персонала и для пациентов. Существующие программные решения часто являются либо слишком сложными для использования, либо не полностью адаптированными под специфику работы медицинских учреждений. Например, приложение “Единый медицинский портал” предоставляет возможность записи на прием к первичным специалистам себя и родственников, вызов врача на дом, доступа к электронной медицинской карте, но имеет ряд ограничений: запись лишь в 10 регионах России, сложный интерфейс, отображение **неактуального расписания. Система “Госуслуги (ЕМИАС)” имеет тот же функционал, но имеет проблемы с частыми зависаниями, неудобным интерфейсом, отсутствием полной информации о врачах**.

Данные факторы снижают качество обслуживания пациентов и затрудняют работу медицинского персонала. В связи с этим возникает необходимость в разработке специализированного и удобного приложения, которое позволит автоматизировать процессы регистратуры, обеспечить удобное и понятное взаимодействие между пациентами и медучреждением, повысить эффективность работы медицинского персонала. Ключевыми требованиями к разработке приложения являются: интуитивно понятный интерфейс, мгновенная синхронизация данных, надежность и стабильность работы системы.

Цель курсового проекта – разработка базы данных и приложения для автоматизации работы регистратуры поликлиники, позволяющее упростить процессы записи на прием, улучшить качество обслуживания пациентов и повысить общую эффективность работы поликлиники.

Задачами курсового проекта являются:

* описать предметную область;
* разработать структуру базы данных;
* разработать приложение;
* провести тестирование приложения.

1 Постановка задачи

* 1. Описание предметной области

Требуется разработать информационную систему для автоматизации работы регистратуры в поликлинике. Система предназначена для учета пациентов, записи на приём к различным врачам, управления графиком работы врачей и хранения медицинской документации.

Данная информационная система предполагает наличие трёх групп пользователей: незарегистрированный пользователь, зарегистрированный пользователь(пациент), администратор регистратуры.

Незарегистрирвоанный пользователь имеет возможность просматривать список врачей, их график работы и какую-либо дополнительную информацию, также производить поиск по специализациям врачей, просматривать доступные дни для записи.

Зарегистрированный пользователь(пациент) имеет те же возможности, что и незарегистрированный, дополнительно получая доступ к полноценной записи на приём к врачу, а также талон, подтверждающего запись.

Администратор обладает расширенными возможностями и имеет следующие возможности:

* ведение и редактирование справочников (врачи, пациенты);
* управление данными об специализациях врачей;
* контроль графика работы врачей;

В базе данных должны храниться следующие справочники: врачи, пациенты, записи на приём, график работы, специализации.

Администратор авторизовывается в системе с использованием уникального адреса электронной почты и кода подтверждения. Пользователь, регистрируясь в системе, получает уникальный ID номер, который является его номером медицинской карты.

В справочнике “Врачи” содержится следующая информация:

* идентификатор врача;
* идентификатор специализации;
* фамилия врача;
* имя врача;
* отчество врача;
* адрес эл. почты;
* номер телефона;
* общий медицинский стаж;
* номер кабинета;
* статус работы;
* изображение специалиста;

Врач также имеет свою узкую специализацию, благодаря которой пользователи могут осуществить поиск или совершить запись на приём к врачу определенного направления. Также врач может работать только по одному графику работы и при этом иметь несколько записей.

В справочнике “Специализация” содержится следующая информация:

* идентификатор специализации;
* название специализации;
* время приёма пациента (в минутах);
* изображение специализации;

У одного пациента может быть несколько записей на прием к разным врачам. В справочнике «Запись на приём» содержится следующая информация:

* идентификатор записи;
* идентификатор принимающего врача;
* идентификатор мед. карты пациента;
* дата приёма;
* время приёма;
* статус записи (завершен, в ожидании).

Для успешной записи на приём пациент должен быть зарегистрирован в системе и иметь заполненные необходимые данные в определенных полях. После заполнения полей, пользователь автоматически прикрепляется к поликлинике и при повторной записи данные, в виде медицинского полиса и паспорта, могут не вноситься Аналогично, администратору требуется авторизация для внесения изменений в остальных справочниках. В справочнике “Пациенты” содержится следующая информация:

* идентификатор медицинской карты;
* адрес эл. почты;
* фамилия пользователя;
* имя пользователя;
* дата рождения;
* номер мед. полиса;
* номер и серия паспорта;
* номер телефона;

Пациенты могут записываться к выбранному врачу только в доступные дни и в доступное время. Врач может работать несколько дней в неделю, по несколько часов, в зависимости от его графика работы. В справочнике “График работы” содержится следующая информация:

* идентификатор графика;
* идентификатор врача;
* день недели;
* время начала работы;
* время окончания работы.

Входными данными являются: данные об пациентах и врачах поликлиники, график работы врачей. Выходными данными является список доступных записей на прием для пациента, различные поисковые результаты.

1.2 Описание входной информации

Входной информацией для выполнения задачи являются данные из справочников: пользователи, врачи, пациенты, график работы, специализации.

1.3 Описание выходной информации

Выходная информация представлена в виде выходного документа. Описание выходных документов представлено в таблице 1.4.1.

Таблица 1.3.1 – Описание выходных документов.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Наименование документа (шифр) | Периодичность выдачи документа | Кол-во экз. | Куда передаются | Поля сортировки | Поля группировки | Итоги |
| 1 | 2 | 3 | 4 | 5 | 6 | 7 |
| Электронный журнал записей на приём | По мере необходимости | 1 | Администратору, пациенту | По номеру записи в журнале | - | - |
| Расписание записей врача | Ежедневно | 1 | Администратору | По имени врача | - | - |
| Запись на приём | По мере необходимости | 1 | Пациенту | По статусу | - | - |

Шаблоны выходных документов представлены в приложении А.

1.4. Концептуальное моделирование

Концептуальная модель базы данных – это некая наглядная диаграмма, нарисованная в принятых обозначениях и подробно показывающая связь между объектами и их характеристиками. Создается концептуальная модель для дальнейшего проектирования базы данных и перевод ее, например, в реляционную базу данных. На концептуальной модели в визуально удобном виде прописываются связи между объектами данных и их характеристиками.

В концептуальной модели есть принятые обозначения элементов. Сущность или объект обозначать прямоугольником, отношения обозначать ромбом, атрибуты объектов, обозначаются овалом. Если сущность связана с отношением, то их связь обозначается прямой линией со стрелкой. Необязательная связь обозначается пунктирной линией. Мощная связь обозначается двойной линией.

Концептуальная модель базы данных представлена в приложении Б.

1.5 Логическое моделирование

При логическом моделировании происходит окончательное определение структуры данных, определяются ограничения, накладываемые на эти данные, целью которых является обеспечить целостность данных. Наиболее распространенной моделью данных является реляционная модель. В этой модели данных каждая сущность представляется в виде таблицы.

Логическое моделирование заключается в переходе от концептуальной модели к взаимосвязанным таблицам. Этот переход состоит из следующих шагов:

1. Преобразование сущностей:

* каждая простая сущность становится таблицей.
* каждый атрибут становится столбцом таблицы.
* уникальный идентификатор сущности становится ключом таблицы.

2. Преобразование связи:

* сущности, связанные обязательной связью один к одному можно объединить в одну таблицу.
* связи типа один к одному возможные и связи типа один ко многим реализуются путем переноса ключевых атрибутов таблиц, соответствующих сущностей, стоящих со стороны один в таблице соответствующих сущностей, стоящих со стороны многие.
* связи типа многие ко многим реализуются при помощи промежуточных таблиц, содержащих ключевые атрибуты связываемых таблиц в качестве внешних ключей.

Схема данных – это структура базы данных, описанная на формальном языке, поддерживаемом СУБД (системой управления базы данных). В реляционных базах данных схема определяет таблицы, поля в каждой таблице и ограничения целостности, такие как первичный и внешний ключи.

Схема данных представлена в приложении В.

1.6. Описание структуры базы данных

Для разработки программы будет использоваться СУБД MySQL.

Описание структуры базы данных представлено в таблицах 1.6.1 – 1.6.5.

Таблица 1.6.1 – Users (Справочник пациентов)

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Имя поля | Описание поля | Тип данных | Размер поля | Тип ключа  (РК-первичный, FK- внешний) |
| id\_med\_card | Id медицинской карты | INT |  | PK |
| email\_users | Адрес электронной почты | VARCHAR | 100 |  |
| surname\_users | Фамилия пациента | VARCHAR | 50 |  |
| name\_users | Имя пациента | VARCHAR | 50 |  |
| date\_birth | Дата рождения | DATE |  |  |
| medical\_policy | Номер мед. полиса | VARCHAR | 16 |  |
| passport\_number | Серия и номер паспорта | VARCHAR | 10 |  |
| phone\_number | Номер телефона | VARCHAR | 11 |  |

Таблица 1.6.2 - Doctor (Справочник врачей)

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Имя поля | Описание поля | Тип данных | Размер поля | Тип ключа  (РК-первичный, FK- внешний) |
| id\_doctor | ID врача | INT |  | PK |
| id\_speciality | ID специализации | INT |  | FK |
| surname\_doctor | Фамилия | VARCHAR | 50 |  |
| name\_doctor | Имя | VARCHAR | 50 |  |
| patronymic\_doctor | Отчество | VARCHAR | 50 |  |
| e-mail\_doctor | Эл. почта | VARCHAR | 100 |  |
| phone\_number\_doctor | Номер телефона | VARCHAR | 11 |  |
| medical\_experience | Общий медицинский стаж | INT |  |  |
| cabinet\_number | Номер кабинета | VARCHAR | 6 |  |
| status\_work | Статус работы | VARCHAR | 45 |  |
| icon\_doctor | Изображение врача | VARCHAR | 50 |  |

Таблица 1.6.3 - Appointment (Запись на приём)

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Имя поля | Описание поля | Тип данных | Размер поля | Тип ключа  (РК-первичный, FK- внешний) |
| 1 | 2 | 3 | 4 | 5 |
| id\_appointment | ID записи | INT |  | PK |
| id\_doctor | ID врача | INT |  | FK |
| id\_medical\_card | ID мед. карты | INT |  | FK |
| date\_appointment | Дата записи | DATE |  |  |
| time\_appointment | Время записи | TIME |  |  |

Продолжение таблицы 1.6.3

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 1 | 2 | 3 | 4 | 5 |
| status\_appointment | Статус записи | ENUM(“В ожидании”, “Завершён”) |  |  |

Таблица 1.6.4 - Specialty (Специализации)

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Имя поля | Описание поля | Тип данных | Размер поля | Тип ключа  (РК-первичный, FK- внешний) |
| 1 | 2 | 3 | 4 | 5 |
| id\_specialty | ID специализации | INT |  | PK |
| name\_specialty | Название специализации | VARCHAR | 50 |  |
| time\_accept | Время приёма | INT |  |  |
| icon\_specialty | Изображение специализации | VARCHAR | 50 |  |

Таблица 1.6.5 - Shedule (График работы)

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Имя поля | Описание поля | Тип данных | Размер поля | Тип ключа  (РК-первичный, FK- внешний) |
| 1 | 2 | 3 | 4 | 5 |
| id\_shedule | ID графика | INT |  | PK |
| id\_doctor | ID врача | INT |  |  |

Продолжение таблицы 1.6.5

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 1 | 2 | 3 | 4 | 5 |
| day\_week | День недели | ENUM | “Пн”,”Вт”,”Ср”,”Чт”,”Пт”,”Сб”,”Вс” | PK |
| time\_start | Время начала работы | TIME |  |  |
| time\_end | Время окончания работы | TIME |  |  |

1.7. Контрольный пример

Контрольный пример является ручным подсчётом задачи. По составленной программе обрабатываются исходные данные контрольного примера. Полученные результаты сравниваются с известными результатами контрольного примера. При несовпадении результатов производится поиск, исправление ошибок, и снова производится выполнение программы.

Входная информация контрольных примеров представлена в приложении Г.

Выходные данные для контрольных примеров показаны в приложении Д.

1.8. Общие требования к программному продукту

Пользователи должны иметь базовые навыки пользования персональным компьютером.

Минимальные требования к техническому обеспечению программного продукта следующие:

* ОС:  Windows 7 Service Pack1/ Windows 7 64Bit Service Pack1/ Windows 8.1 64Bit / Windows 10 64Bit / Mac OS;
* процессор: 2.40 Ггц (четырехъядерный) / AMD Phenom 9850 (четырехъядерный) @ 2.5 Ггц;
* оперативная память: 512МБ (Win 7/Win8 и выше);
* видеокарта: NVIDIA 9800 GT c 512 МБ видеопамяти/ AMD HD 4870 с 1 Гб видеопамяти (DX 9, 10, 10.1);
* жесткий диск: 10 гигабайт свободного места;
* Microsoft DirectX версия 9.0c;

Функциональные возможности программного продукта:

* приложение должно формировать и отображать выходные данные пользователю;
* в приложении должен быть обеспечен просмотр таблиц (справочников) базы данных с возможность добавления, редактирования, удаления данных.

Требования к надежности:

* приложение должно обрабатывать ошибочные действия пользователя и сообщать ему об этом;
* приложение должно обеспечивать контроль входной и выходной информации.

Требования к информационной и программной совместимости: обеспечить работу приложения с таблицами СУБД MySQL Workbench.

2. Экспериментальный раздел

2.1 Описание программы

Программа имеет модульную структуру. При ее запуске выполняется проект на WPF Main\_Project.exe. Программа Main\_Project.exe написана на языке C# в среде программирования Visual Studio 2022 с использованием системы управления базой данных MS SQL Server 2022.

Схема взаимодействия модулей программы представлена на рисунке 2.1.1.

Описание модулей и методов представлено в таблице 2.1.1.

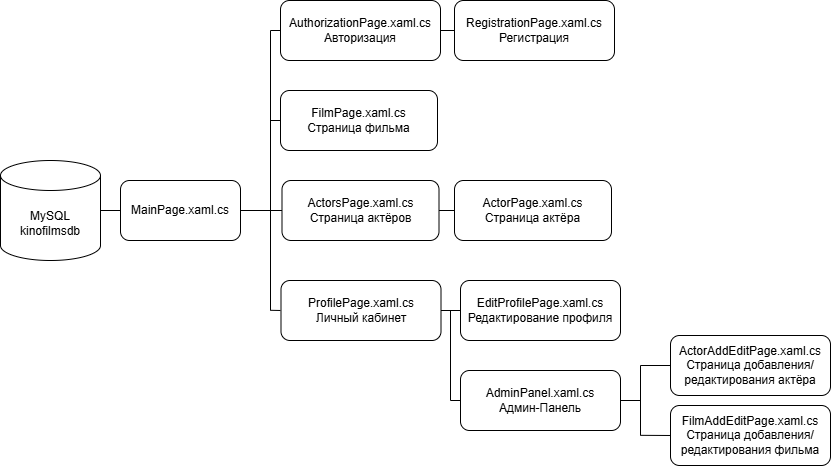


Рисунок 2.1.1 – Схема взаимодействия модулей

Таблица 2.1.1. - Описание модулей

|  |  |  |
| --- | --- | --- |
| Методы | Назначение | |
| 1 | 2 | |
| AboutClinik.xaml.cs | | |
| public AboutClinik() | | Устанавливает заголовок окна и инициализирует компоненты |
| AddRedactDoctors.xaml.cs | | |

|  |  |
| --- | --- |
| 1 | 2 |

|  |  |  |
| --- | --- | --- |
| public AddRedactDoctors(User user) | | Конструктор для добавления врача, инициализирует данные пользователя и загрузку специальностей |
| public AddRedactDoctors(Doctor doctor, User user) | | Конструктор для редактирования врача, инициализирует данные и заполняет поля |
| private void LoadData() | | Загружает список специальностей из БД и заполняет выпадающий список |
| private void FillDoctorData() | | Заполняет поля формы данными выбранного врача |
| private void btnCancel\_Click(object sender, RoutedEventArgs e) | | Обработчик нажатия на кнопку "Отмена", возвращает в панель администратора |
| private void btnSave\_Click(object sender, RoutedEventArgs e) | | Сохраняет изменения или добавляет нового врача в БД |
| AddRedactSched.xaml.cs | | |
| public AddRedactSched(User currentUser) | Конструктор для добавления графика, загружает врачей и дни недели | |
| public AddRedactSched(Schedule scheduleToEdit, User currentUser) | Конструктор для редактирования графика, заполняет данные графика | |
| private void InitializeDayWeekComboBox() | Заполняет выпадающий список днями недели | |

Продолжение таблицы 2.1.1

|  |  |
| --- | --- |
| 1 | 2 |

|  |  |
| --- | --- |
| private void LoadDoctors() | Загружает список врачей из БД для выбора |
| private void FillScheduleData() | Заполняет поля формы данными текущего графика |
| private bool CheckExistingSchedule(int doctorId, string dayOfWeek, int? excludeScheduleId = null) | Проверяет наличие другого графика у врача в этот день |
| private void btnCancel\_Click(object sender, RoutedEventArgs e) | Обработчик нажатия "Отмена", возвращает в админ-панель |
| private void btnSave\_Click(object sender, RoutedEventArgs e) | Сохраняет или обновляет график работы в БД |
| AddRedactSpecialities.xaml.cs | |
| public AddRedactSpecialities(User currentUser) | Конструктор для добавления новой специальности |
| public AddRedactSpecialities(Specialty specialtyToEdit, User currentUser) | Конструктор для редактирования существующей специальности |
| private void FillSpecialtyData() | Заполняет поля формы данными редактируемой специальности |
| private void btnCancel\_Click(object sender, RoutedEventArgs e) | Обработчик нажатия "Отмена", возвращает в админ-панель |

Продолжение таблицы 2.1.1

|  |  |
| --- | --- |
| 1 | 2 |
| private void btnSave\_Click(object sender, RoutedEventArgs e) | Сохраняет или обновляет данные специальности в БД |
| AddRedactUsers.xaml.cs | |
| public AddRedactUsers(User currentUser) | Конструктор для добавления нового пользователя |
| public AddRedactUsers(User userToEdit, User currentUser) | Конструктор для редактирования существующего пользователя |
| private void LoadData() | Загружает данные (роли и устанавливает значения по умолчанию) |
| private void FillUserData() | Заполняет поля формы данными редактируемого пользователя |
| private void btnCancel\_Click(object sender, RoutedEventArgs e) | Обработчик нажатия "Отмена", возвращает в админ-панель |
| private void btnSave\_Click(object sender, RoutedEventArgs e) | Сохраняет или обновляет данные пользователя в БД |
| private bool IsValidDate(string dateString) | Проверяет корректность формата даты рождения |
| Admin\_panel.xaml.cs | |
| public Admin\_panel(User user) | Инициализирует панель администратора, загружает данные |
| private void Datagrid\_SelectionChanged(object sender, SelectionChangedEventArgs e) | Обновляет видимость кнопки печати при выборе врача |
| private void getData() | Загружает все данные из БД: врачи, пользователи, специальности и график |
| public void LoadDoctors() | Загружает список врачей с пагинацией по 10 записей |
| public void LoadUsers() | Загружает список пользователей с пагинацией по 10 записей |
| private void LoadAppointment() | Загружает список записей на приём с пагинацией |
| private void LoadSpecialties() | Загружает список специальностей с пагинацией |

Продолжение таблицы 2.1.1

|  |  |
| --- | --- |
| 1 | 2 |
| private void LoadSchedules() | Загружает список графиков работы с пагинацией |
| public void LoadData() | Подготавливает и отображает данные в зависимости от выбранного режима |
| private void arrowLeft\_MouseLeftButtonDown(object sender, MouseButtonEventArgs e) | Переход на предыдущую страницу при навигации |
| private void arrowRight\_MouseLeftButtonDown(object sender, MouseButtonEventArgs e) | Переход на следующую страницу при навигации |
| private void cboxMode\_SelectionChanged(object sender, SelectionChangedEventArgs e) | Обновляет отображаемые данные при изменении режима |
| private void btnAdd\_Click(object sender, RoutedEventArgs e) | Открывает форму добавления в зависимости от текущего режима |
| private void btnEdit\_Click(object sender, RoutedEventArgs e) | Открывает форму редактирования выбранной записи |
| AdminAuth.xaml.cs |
| public AdminAuth() | Конструктор, инициализирует форму входа в админ-панель |
| private void code\_button\_Click(object sender, RoutedEventArgs e) | Отправляет код подтверждения на email администратора |
| private void enter\_button\_Click(object sender, RoutedEventArgs e) | Обработчик нажатия кнопки "Войти", открывает админ-панель |
| private bool Verify(string user\_code) | Проверяет корректность введённого кода подтверждения |
| AllSpecialtiesPage.xaml.cs |
| public AllSpecialtiesPage(Specialty selectedSpecialty) | Конструктор, инициализирует страницу выбранной специальностью |
| private void LoadDoctors() | Загружает список врачей по выбранной специальности |
| private void Back\_button\_Click(object sender, RoutedEventArgs e) | Обработчик нажатия кнопки "Назад", возвращает на предыдущую страницу |
| Admin\_panel.xaml.cs | |

Продолжение таблицы 2.1.1

|  |  |
| --- | --- |
| 1 | 2 |
| public Admin\_panel(User user) | Инициализирует панель администратора и загружает данные |
| private void Datagrid\_SelectionChanged(object sender, SelectionChangedEventArgs e) | Обновляет видимость кнопки печати при выборе врача |
| private void getData() | Загружает все данные из БД: врачи, пользователи, записи и т.д. |
| public void LoadDoctors() | Загружает список врачей с пагинацией по 10 записей |
| public void LoadUsers() | Загружает список пользователей с пагинацией по 10 записей |
| private void LoadAppointment() | Загружает список записей на приём с пагинацией |
| private void LoadSpecialties() | Загружает список специальностей с пагинацией |
| private void LoadSchedules() | Загружает список графиков работы с пагинацией |
| public void LoadData() | Подготавливает и отображает данные в зависимости от режима |
| private void arrowLeft\_MouseLeftButtonDown(object sender, MouseButtonEventArgs e) | Переход на предыдущую страницу при навигации |
| private void arrowRight\_MouseLeftButtonDown(object sender, MouseButtonEventArgs e) | Переход на следующую страницу при навигации |
| private void cboxMode\_SelectionChanged(object sender, SelectionChangedEventArgs e) | Обновляет отображаемые данные при изменении режима |
| private void btnAdd\_Click(object sender, RoutedEventArgs e) | Открывает форму добавления данных в зависимости от режима |
| private void btnEdit\_Click(object sender, RoutedEventArgs e) | Открывает форму редактирования выбранной записи |
| private void btnPrint\_Click(object sender, RoutedEventArgs e) | Метод-заглушка для печати информации о враче |
|  | |

Продолжение таблицы 2.1.1

|  |  |
| --- | --- |
| 1 | 2 |
| ClinikMainWindow.xaml.cs | |
| public ClinikMainWindow() | Конструктор, инициализирует главное окно и загружает начальную страницу |
| private void UpdateAppointmentStatuses() | Обновляет статусы прошедших записей на "Завершён" |
| private void Click\_aboutClinik\_MouseLeftButtonDown(object sender, MouseButtonEventArgs e) | Переход на страницу "О клинике" |
| private void Click\_contactsClinik\_MouseLeftButtonDown(object sender, MouseButtonEventArgs e) | Переход на страницу контактов клиники |
| private void Click\_appointmentClinik\_MouseLeftButtonDown(object sender, MouseButtonEventArgs e) | Переход на страницу выбора специальностей для записи |
| private void mainframe\_Navigated(object sender, System.Windows.Navigation.NavigationEventArgs e) | Обработчик события навигации по фрейму (заглушка) |
| private void Admin\_panel\_MouseLeftButtonDown(object sender, MouseButtonEventArgs e) | Переход на страницу авторизации администратора |
| ContactsClinik.xaml.cs |
| public ContactsClinik() | Конструктор, инициализирует страницу контактов клиники |
| EndAppointment |
| public EndAppointment(string date, string time, string specialty, string fio, string? cabinet) | Конструктор, инициализирует страницу с данными о записи |
| private static string GetShortName(string fullName) | Возвращает ФИО в сокращённом формате (с инициалами) |
|  |  |

Продолжение таблицы 2.1.1

|  |  |
| --- | --- |
| 1 | 2 |
| private void on\_main\_Click(object sender, RoutedEventArgs e) | Обработчик нажатия кнопки "На главную", возвращает на главную страницу |
| private void print\_talon\_Click(object sender, RoutedEventArgs e) | Генерирует PDF-талон на приём и сохраняет его |
| InformationDoctorPage.xaml.cs |
| public InformationDoctorPage(Specialty specialty, Doctor currDoctor) | Конструктор, инициализирует страницу информацией о враче и специальности |
| private void Back\_button\_Click(object sender, RoutedEventArgs e) | Обработчик нажатия кнопки "Назад", возвращает к списку специальностей |
| SpecialtiesPage.xaml.cs |
| public SpecialtiesPage() | Конструктор, инициализирует страницу выбора специальностей |
| private void LoadSpecialty() | Загружает список всех специальностей из БД |
| private void UpdateSpecialtyListView() | Обновляет отображение списка специальностей в интерфейсе |
| private void searchtxtbox\_TextChanged(object sender, TextChangedEventArgs e) | Фильтрует специальности по введенному в поле поиску |
| DoctorsControl.xaml.cs |
| public DoctorsControl(Doctor doctor, Specialty spec) | Конструктор, инициализирует элемент управления информацией о враче |
| private void toAppointment\_button\_Click(object sender, RoutedEventArgs e) | Обработчик нажатия кнопки "Записаться", открывает страницу записи к врачу |
| private void informationDoctor\_Click(object sender, RoutedEventArgs e) | Обработчик нажатия кнопки "Информация", открывает страницу с подробными сведениями о враче |
| SpecialtyControl.xaml.cs |
| public SpecialtyControl(Specialty specialty) | Конструктор, инициализирует элемент управления специальностью |
| private void UserControl\_Loaded(object sender, RoutedEventArgs e) | Обработчик события загрузки элемента управления (заглушка) |

Продолжение таблицы 2.1.1

|  |  |
| --- | --- |
| 1 | 2 |
| private void Click\_toAppointment(object sender, RoutedEventArgs e) | Обработчик нажатия кнопки, открывает страницу записи по выбранной специальности |

Код программы представлен в приложении Е.

2.2 Протокол тестирования программного продукта

В ходе тестирования программного продукта на корректных и некорректных данных не было обнаружено ошибок, которые влияли бы на работу самого программного продукта и всей системы.

Данный программный продукт удовлетворяет всем предъявленным требованиям, имеет комфортный интерфейс и интуитивно понятный функционал, исключает появления системных ошибок.

В таблицах 2.2.1 – 2.2.5 представлены протоколы тестирования.

Таблица 2.2.1 – Тестирование авторизации администратора с пустыми данными

|  |  |  |
| --- | --- | --- |
| Поле | Описание | |
| 1 | 2 | |
| Дата теста | 24.06.2025 | |
| Приоритет тестирования (*Низкий/Средний/Высокий*) | Высокий | |
| Заголовок/название теста | | Тестирование авторизации с пустыми данными |
| Резюме испытания | | Отображение корректного отображения сообщения об ошибки при попытке входа с пустым полем электронной почты. |
| Этапы теста | | 1 запуск приложения;  2 нажатие на кнопку «Админ»;  3 нажатие на кнопку «Войти»;  4 нажатие на кнопку «Войти». |
| Тестовые данные | | 1 электронная почта «»;  2 код подтверждения «»; |
| Ожидаемый результат | | Отображение сообщения «Введите корректный email адрес!» |
| Фактический результат | | Сообщение отобразилось, вход не произошёл |

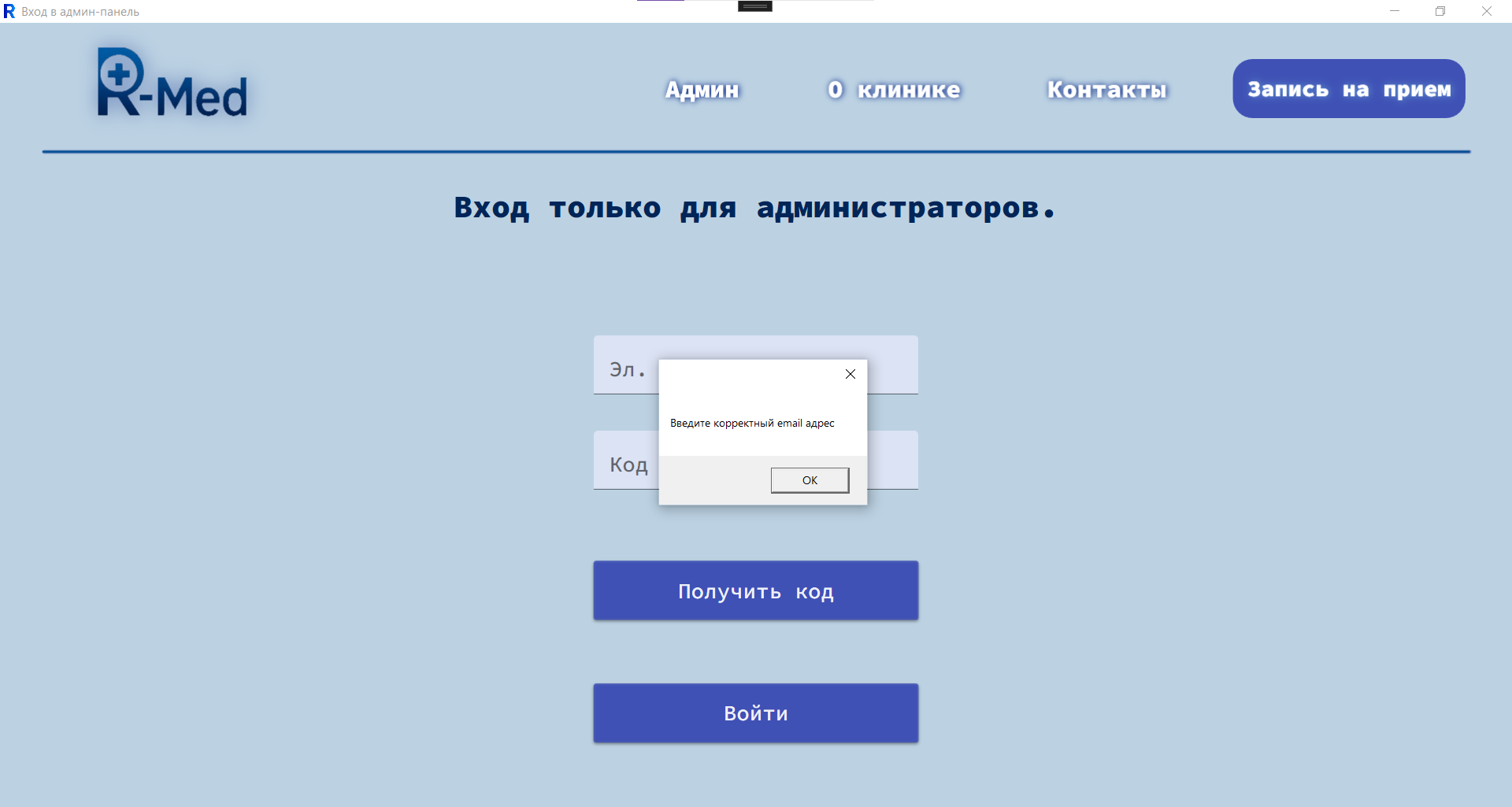


Рисунок 2.2.1 – Результат тестирования авторизации администратора с пустыми данными

Таблица 2.2.2 – Тестирование авторизации администратора с неккоректными данными

|  |  |
| --- | --- |
| Поле | Описание |
| 1 | 2 |
| Дата теста | 24.06.2025 |
| Приоритет тестирования (*Низкий/Средний/Высокий*) | Высокий |
| Заголовок/название теста | Тестирование авторизации с неккоректными данными |
| Резюме испытания | Отображение корректного отображения сообщения об ошибки при попытке входа без данных |
| Этапы теста | 1 запуск приложения;  2 нажатие на кнопку «Админ»;  3 ввод некорректной email адреса  3 нажатие на кнопку «Войти». |
| Тестовые данные | 1 email адрес «reginasaf@gmail.com»; |
| Ожидаемый результат | Отображение сообщения «У вас недостаточно прав для доступа к администраторской панели» |
| Фактический результат | Сообщение отобразилось, вход не произошёл |

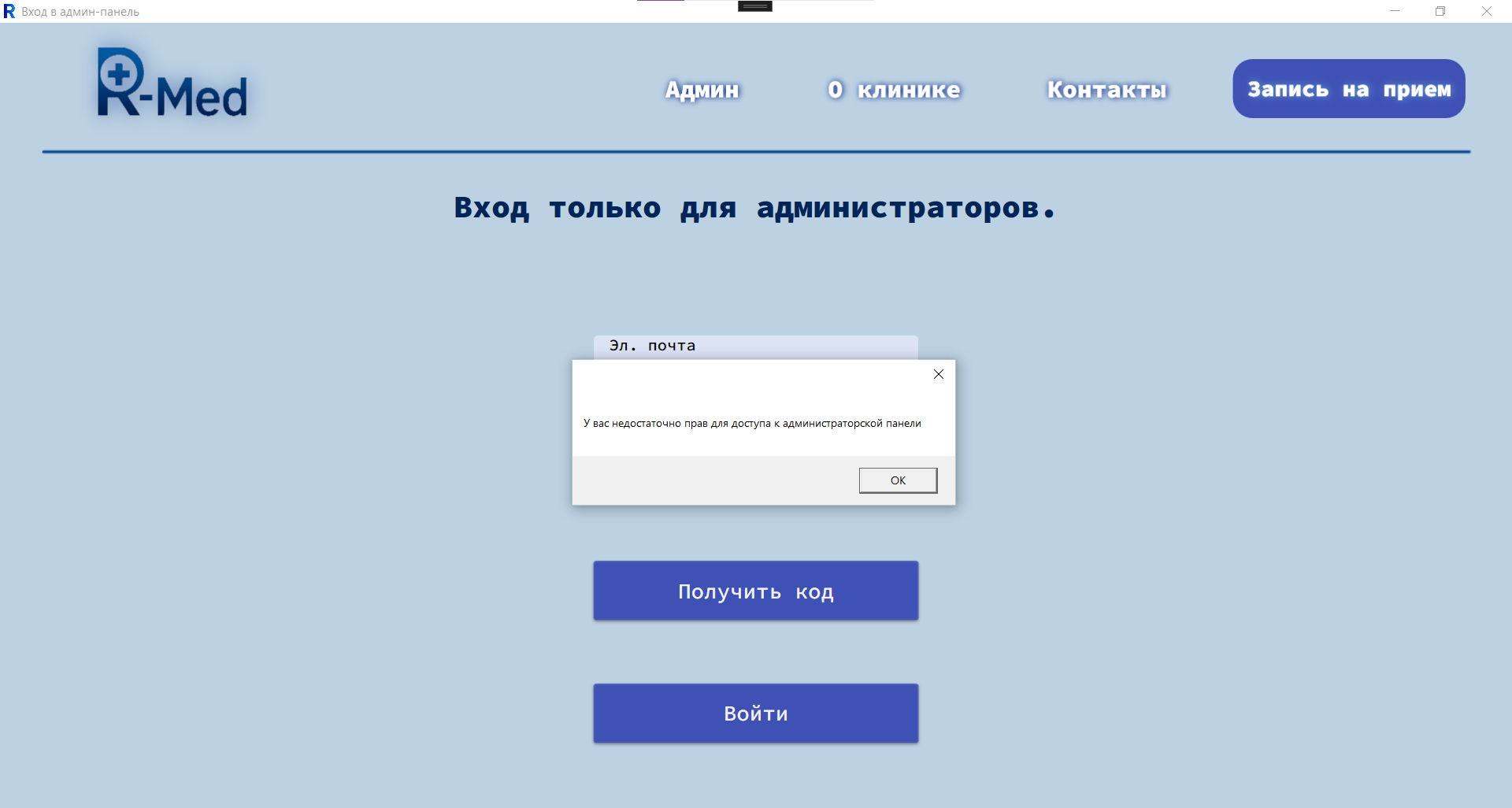


Рисунок 2.2.2 – Тестирование авторизации администратора с некорректными данными

Таблица 2.2.3 – Тестирование отправки кода подтверждения при авторизации администратора с корректными данными

|  |  |
| --- | --- |
| Поле | Описание |
| 1 | 2 |
| Дата теста | 24.06.2025 |
| Приоритет тестирования (*Низкий/Средний/Высокий*) | Высокий |
| Заголовок/название теста | Тестирование отправки кода подтверждения при авторизации с корректными данными |
| Резюме испытания | Отображение корректного отображения сообщения «Код подтверждения отправлен на почту reginasafina0227@gmail.com» |
| Этапы теста | 1 запуск приложения;  2 нажатие на кнопку «Админ»;  3 ввод эл. почты администратора;  4 нажатие на кнопку «Войти»; |
| Тестовые данные | 1. Электронная почта «reginasafina0227@gmail.com»; |
| Ожидаемый результат | Отображение сообщения «Код подтверждения отправлен на почту reginasafina0227@gmail.com» |
| Фактический результат | Сообщение отобразилось, код подтверждения пришёл на почту. |

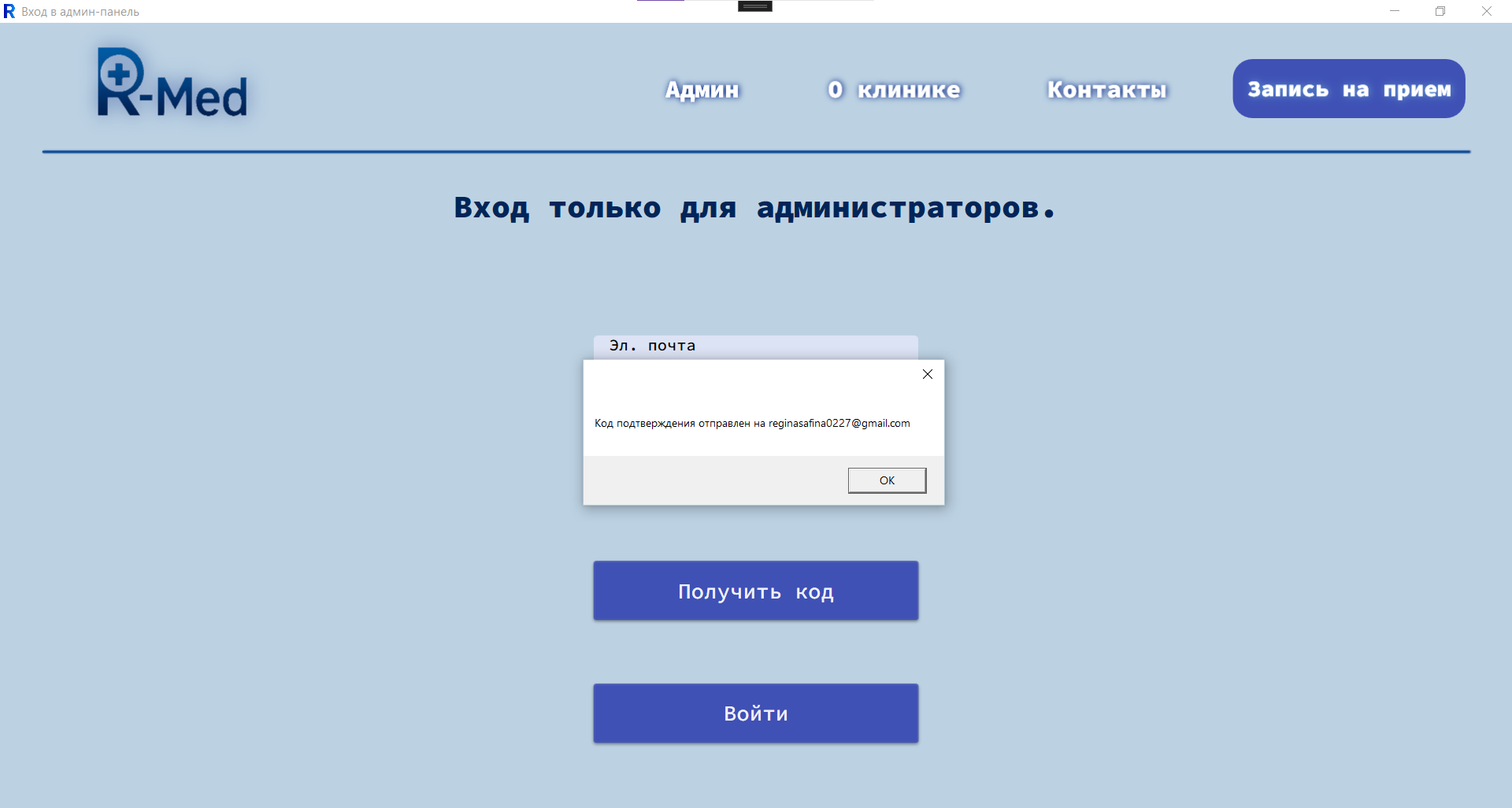


Рисунок 2.2.3 – Результат авторизации с корректными данными

Таблица 2.2.4 – Тестирование записи на приём с пустыми данными

|  |  |
| --- | --- |
| Поле | Описание |
| 1 | 2 |
| Дата теста | 24.06.2025 |
| Приоритет тестирования (*Низкий/Средний/Высокий*) | Высокий |
| Заголовок/название теста | Регистрация с пустыми данными |
| Резюме испытания | Отображение корректного отображения сообщения «Заполните все данные!» |
| Этапы теста | 1 запуск приложения;  2 нажатие на кнопку «Записаться»;  3 нажатие на кнопку «Получить код»; |
| Тестовые данные | Ввод пустых данных |
| Ожидаемый результат | Отображение сообщения «Заполните все данные!» |
| Фактический результат | Сообщение отобразилось, запись на приём не осуществилась |

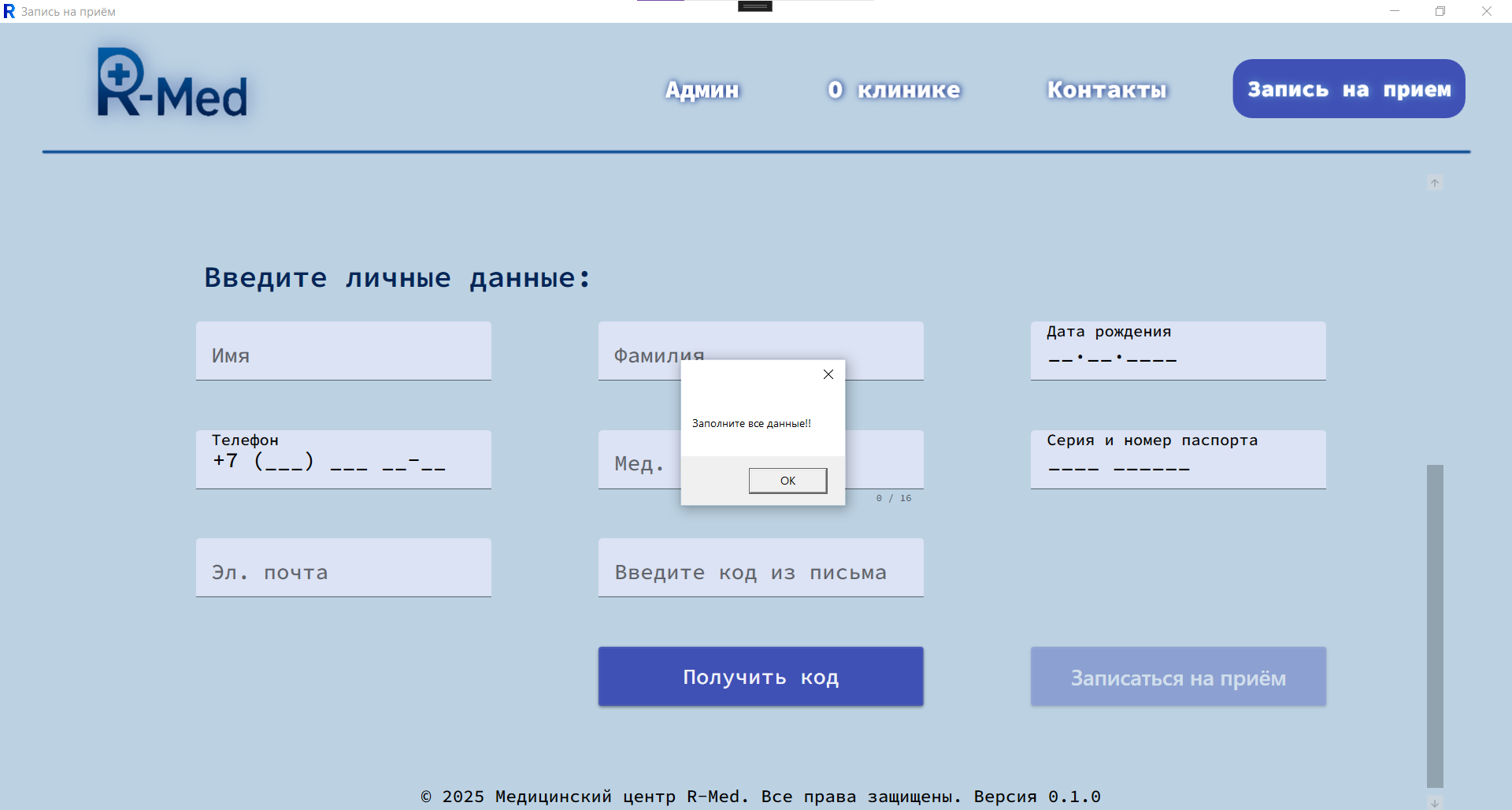


Рисунок 2.2.4 – Тестирование записи на прием с пустыми данными

Таблица 2.2.5 – Тестирование записи на прием с корректными данными

|  |  |
| --- | --- |
| Поле | Описание |
| 1 | 2 |
| Дата теста | 24.06.2025 |
| Приоритет тестирования (*Низкий/Средний/Высокий*) | Высокий |
| Заголовок/название теста | Тестирование записи на прием с корректными данными |
| Резюме испытания | Отображение сообщения «Регина, вы успешно записались на приём! Ваш номер мед. карты: 115»; |
| Этапы теста | 1 запуск приложения;  2 нажатие на кнопку «Записаться»;  3 нажатие на кнопку «Получить код»; |
| Тестовые данные | 1 имя «Регина»;   1. фамилия «Сафина»; 2. дата рождения «27.02.2006»; 3. телефон «79613601151»; 4. эл. почта «reginasafina0227@gmail.com»; |
| Ожидаемый результат | Отображение сообщения «Регина, вы успешно записались на приём! Ваш номер мед. карты: 115»; |
| Фактический результат | Сообщение отобразилось, запись совершилась |

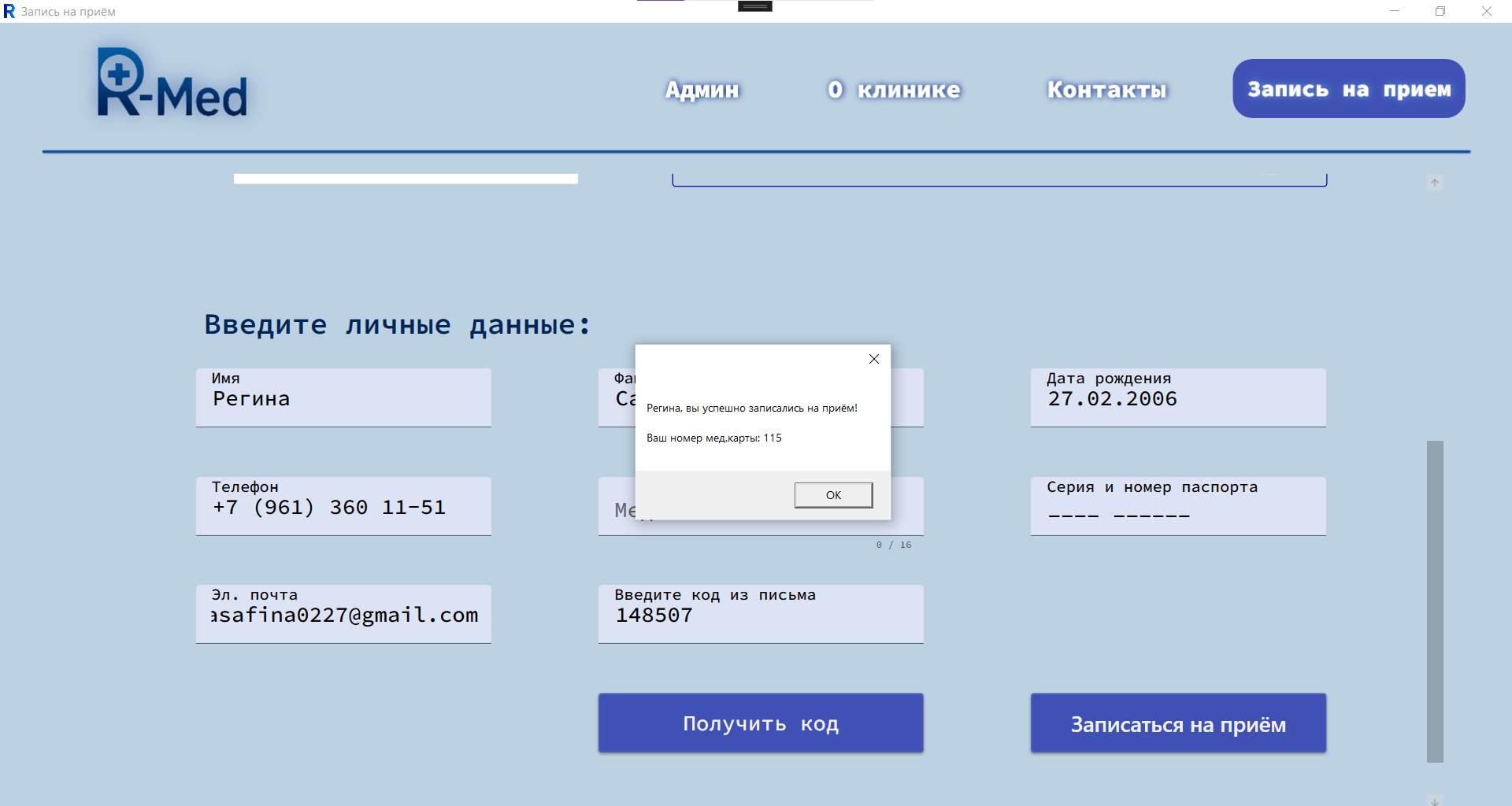


Рисунок 2.2.5 – Тестирование регистрации с корректными данными

2.3 Руководство пользователя

Назначение системы

Программное обеспечение «Медицинский центр R-Med» предназначена для автоматизации работы регистратуры медицинского учреждения, упрощения процессов записи пациентов на прием к врачам, а также повышения эффективности учета и управления медицинскими данными.

Основной целью данной информационной системы является обеспечение создание удобного и надежного инструмента, который: позволит пациентам быстро записываться на прием к врачам через интуитивно понятный интерфейс; устранит проблемы, связанные с бумажным документооборотом, снижая риск потери данных; повысит скорость обслуживания пациентов, сокращая время ожидания в очередях.

Условия применения системы

Программное обеспечение разрабатывается для персональной вычислительной техники со следующими характеристиками:

* Microsoft Windows 7 / 8 / 10;
* процессор 2 ГГц;
* 256 МБ ОЗУ;
* 100 МБ свободного пространства на диске;
* разрешение экрана монитора не менее 1920 × 1080;

Приложение предназначено как для простых пользователей, имеющих как минимум первоначальные навыки с графической операционной системой, на которой будет запускаться данная программа.

Подготовка системы к работе

Для запуска программы необходимо запустить приложение Main\_Project.exe. После запуска открывается главное окно приложения, содержащее каталог специальностей врачей (рисунок 2.3.1).

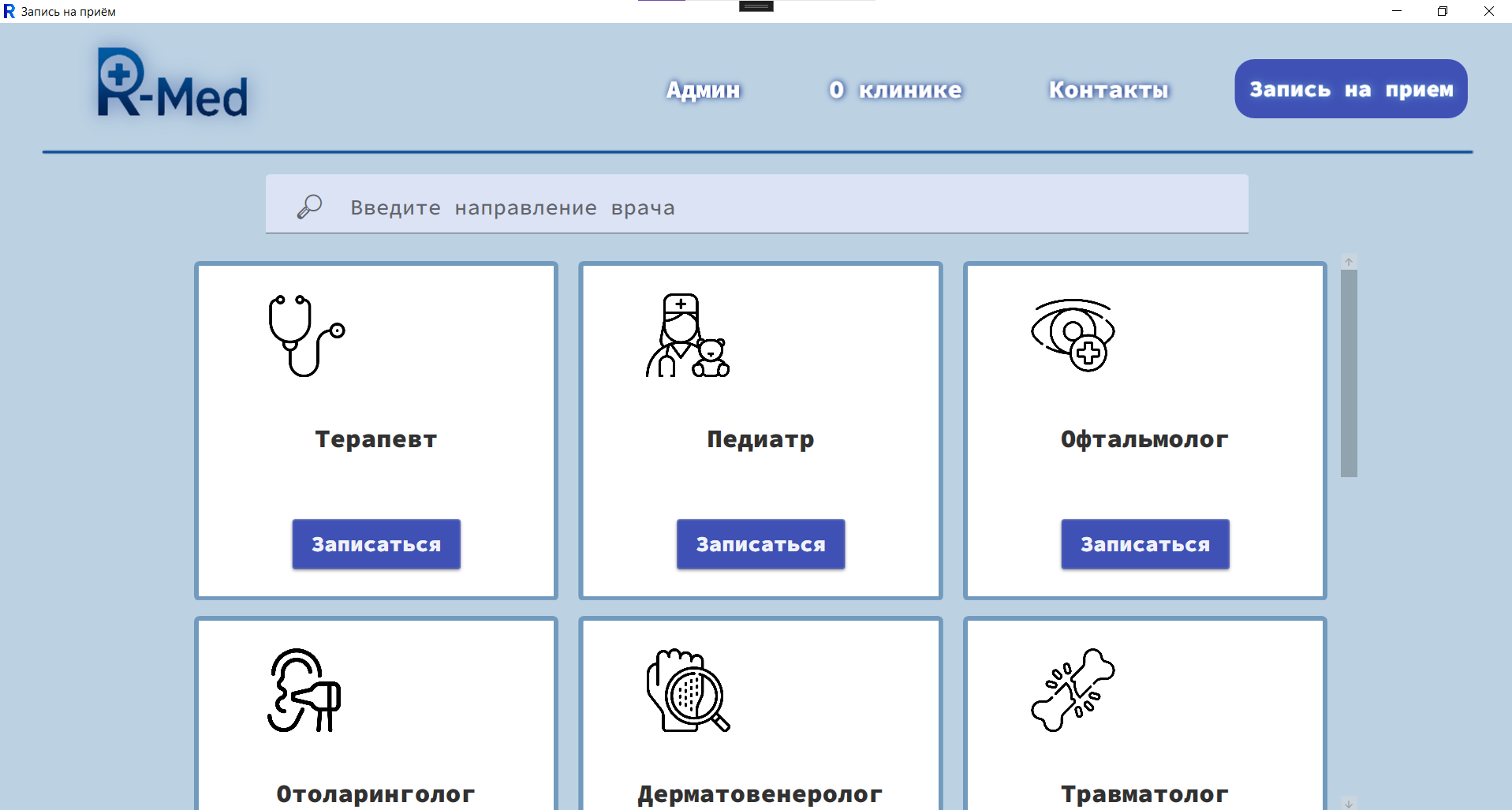


Рисунок 2.3.1 – Окно приложения

Описание операций

В верхней части окна видны 4 кнопки, при нажатии на которые, открываются соответствующие страницы. На главной странице представлены все специальности врачей и поле для поиска определенной специальности по ее названию. Нажав на кнопку “Контакты”, можно перейти на страницу, указывающей дополнительную информацию об медицинском центре (адрес, телефон, режим работы, email) (рисунок 2.3.2).

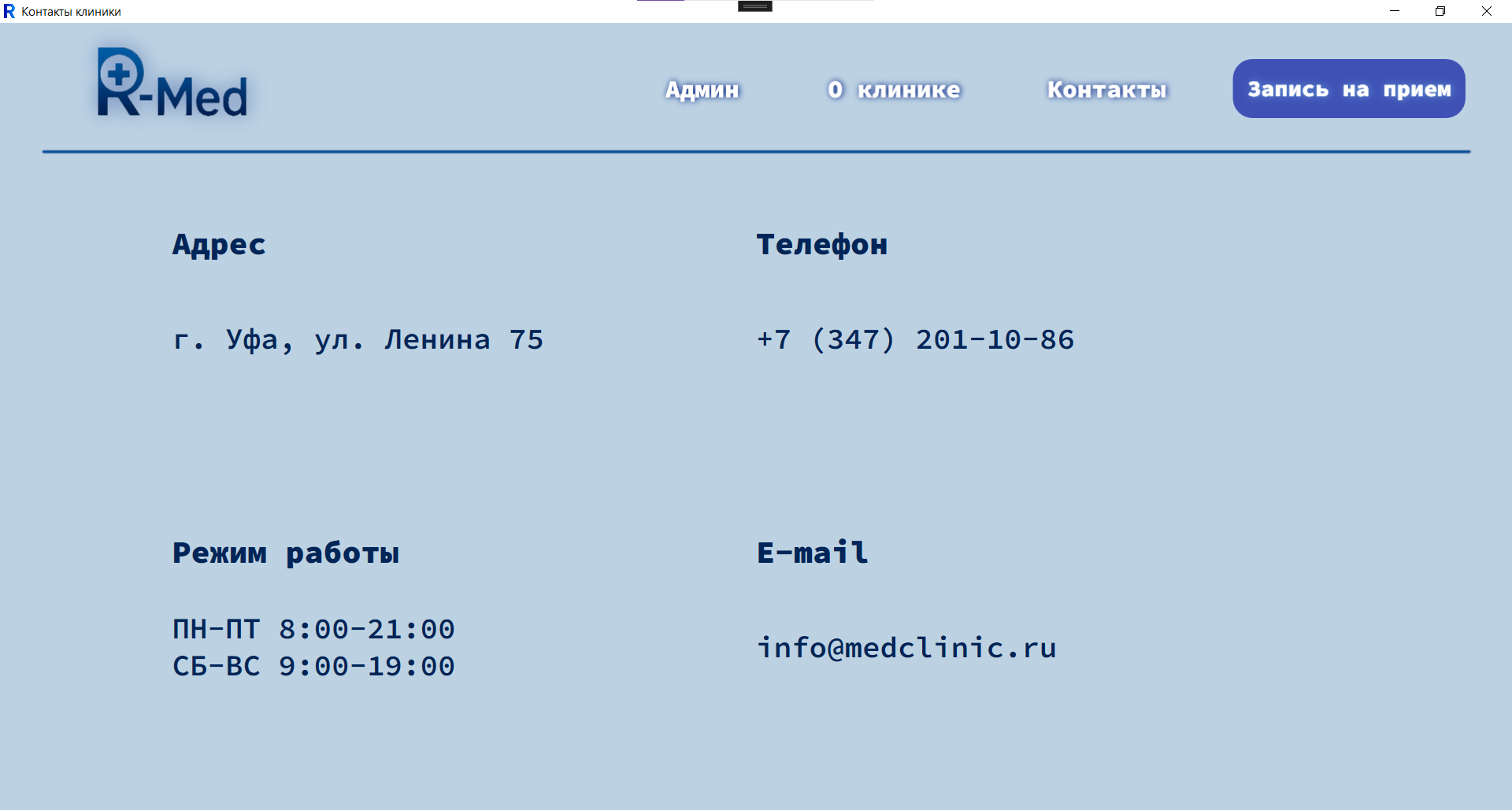


Рисунок 2.3.2 – Информация об контактах медицинского центра.

Нажав на кнопку “О клинике” в верхней части панели, система переходит на страницу, указывающей краткую информацию об медицинском центре (рисунок 2.3.3).

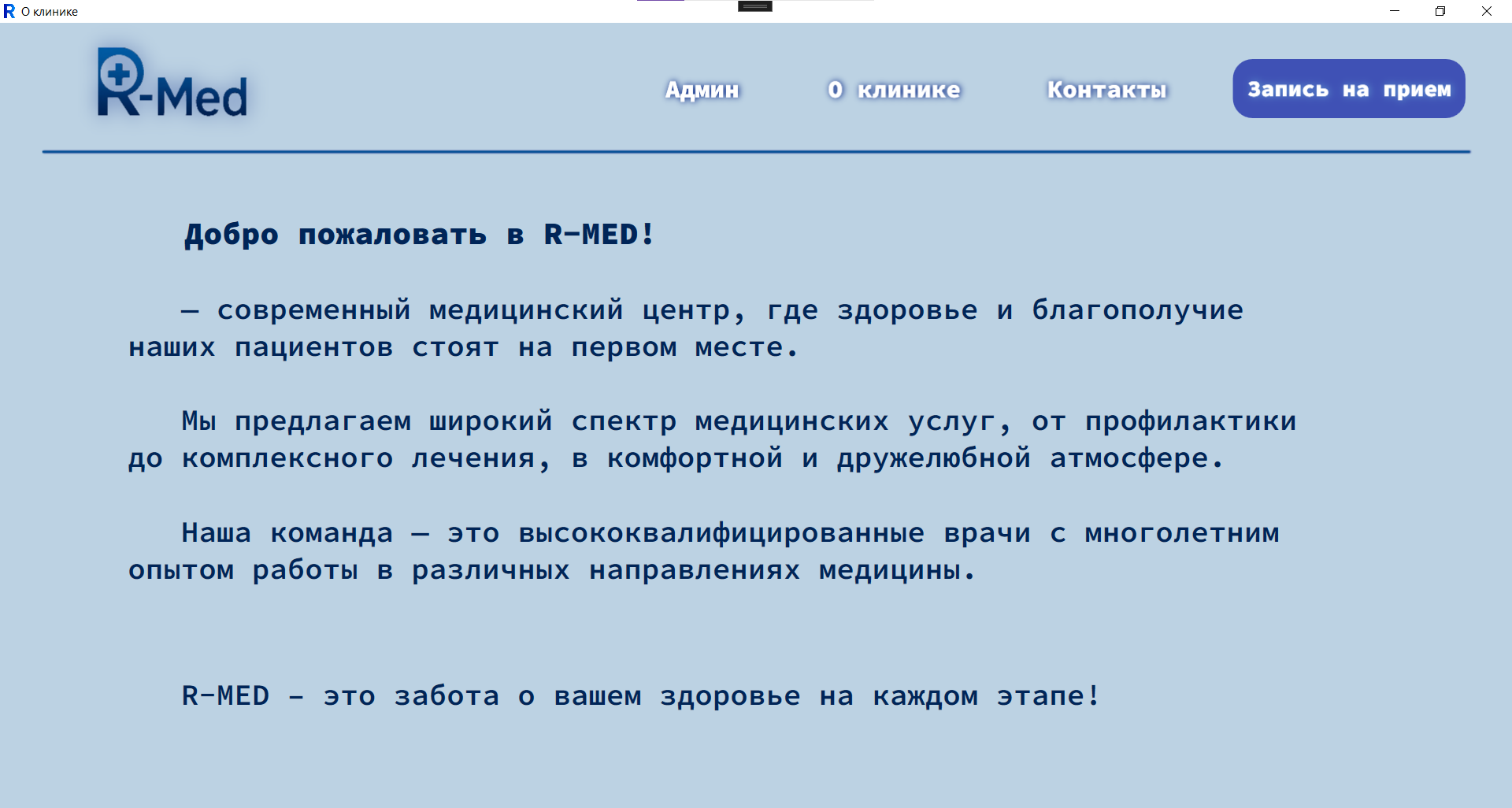


Рисунок 2.3.3 – Окно с краткой информацией об клинике.

В зависимости от роли пользователя, возможен вход на администраторскую панель, для перехода на которую, следует нажать на кнопку “Админ” в верхней части приложения.

Обычный пользователь не имеет прав для входа в администраторскую-панель, но администратор может внести данные своей электронной почты, получить соответствующий код подтверждения и войти в систему (рисунки 2.3.4 – 2.3.5).

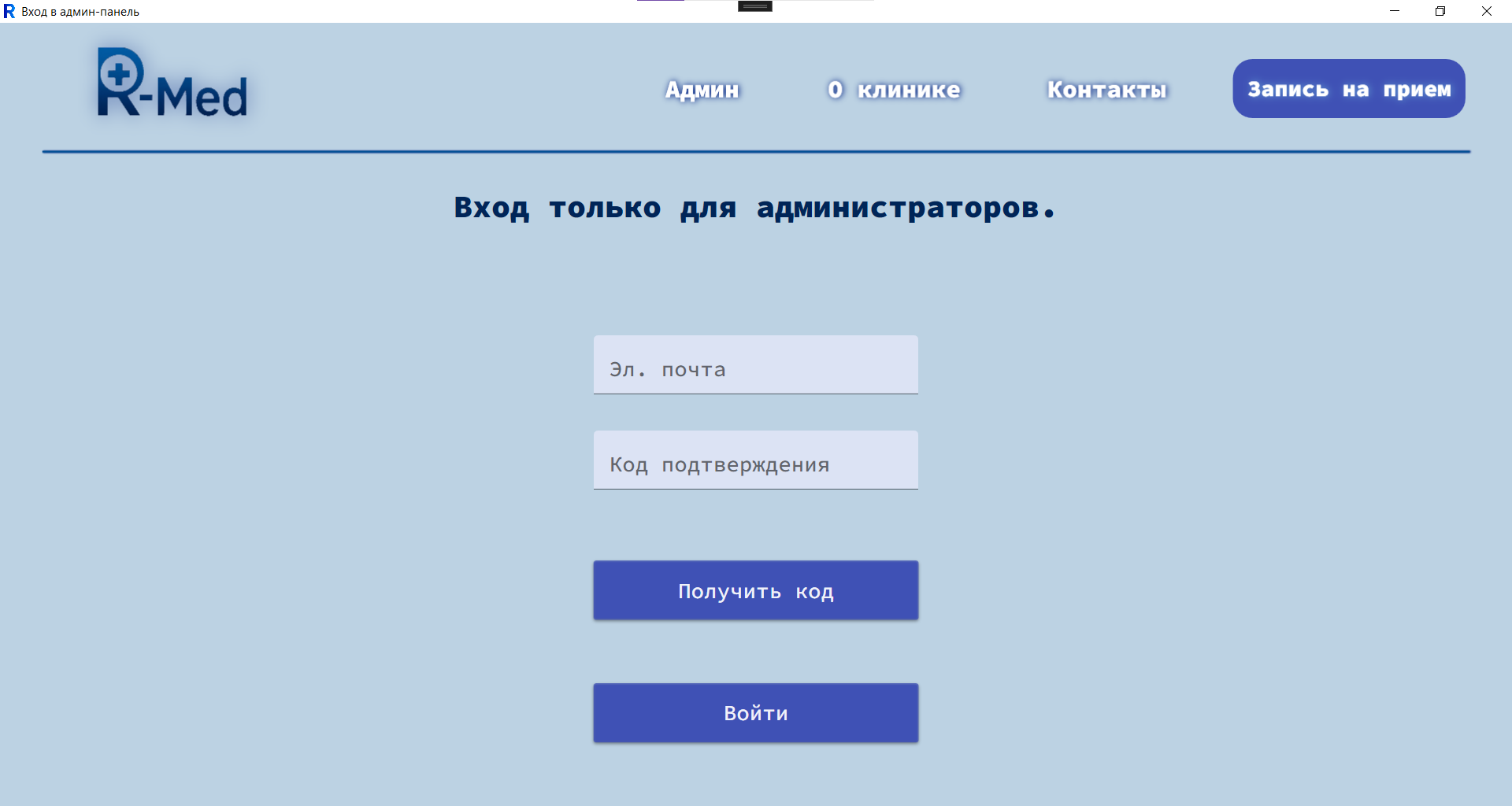


Рисунок 2.3.4 – Окно с авторизацией администраторов.

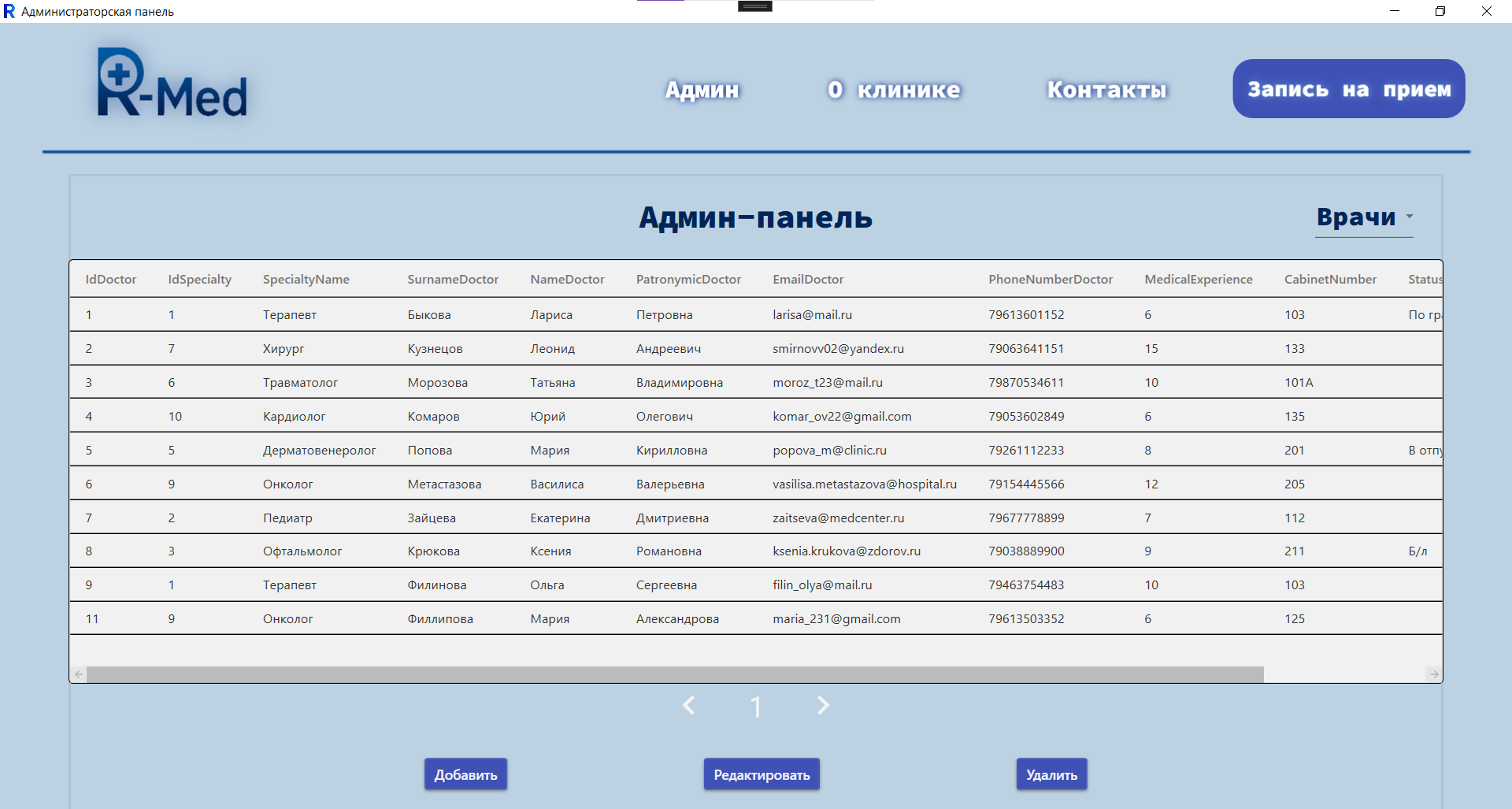


Рисунок 2.3.5 – Администраторская панель.

На главной странице со специальностями, пользователь может перейти на следующую страницу с врачами/специалистами для последующей записи, посредством нажатия на кнопку “Записаться” и увидеть ФИО найденных врачей с их определенной специальностью (рисунок 2.3.6).

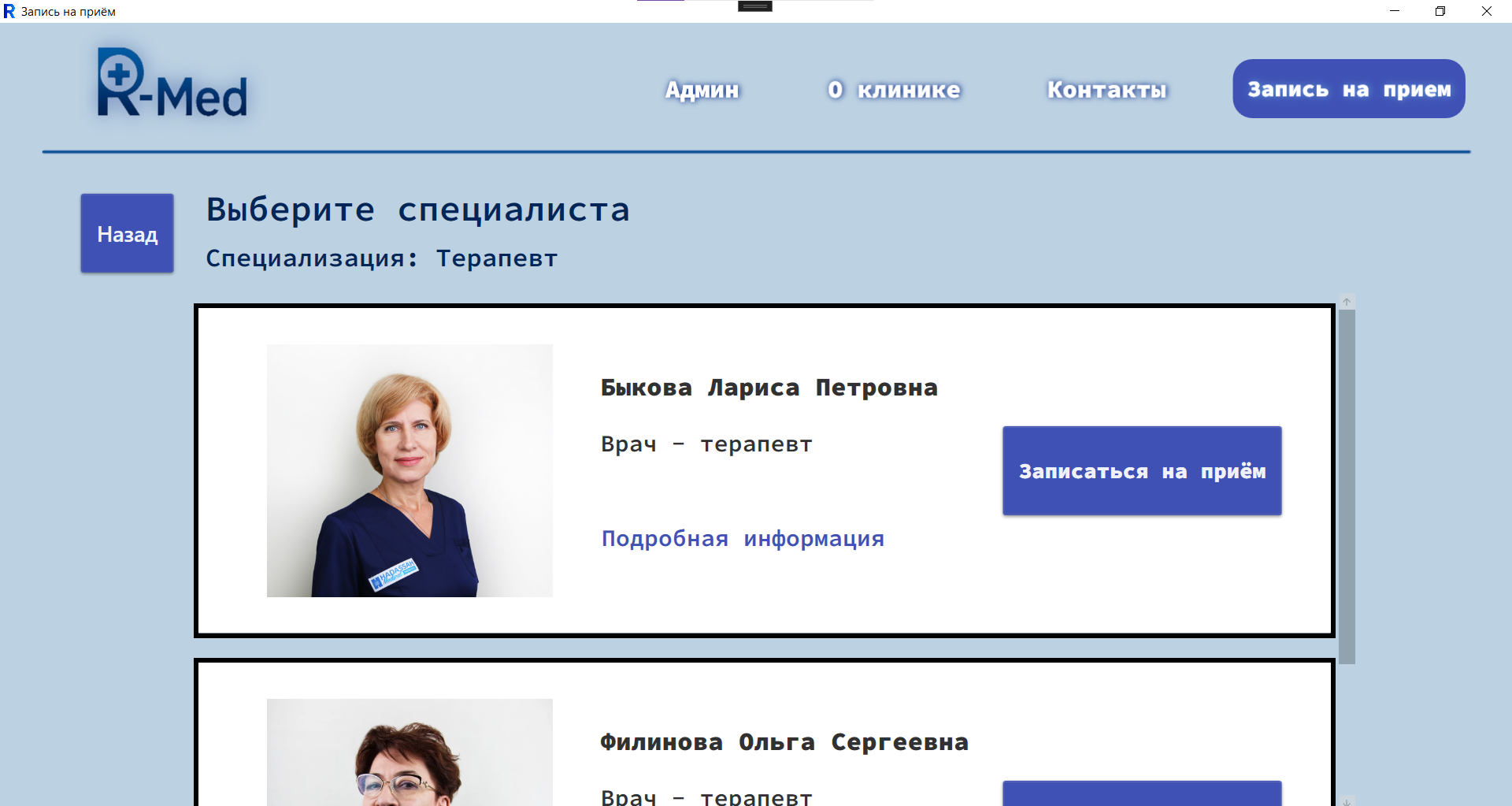


Рисунок 2.3.6 – Список специалистов.

При переходе на страницу со специалистами, есть возможность перейти на страницу с подробной информацией об конкретном специалисте и узнать его статус работы и медицинский стаж.

Для продолжении записи следует нажать на кнопку “Назад”, перейти на страницу с врачами определенной специальности и продолжить запись на странице нажав на кнопку “Записаться на приём” (рисунок 2.3.7).

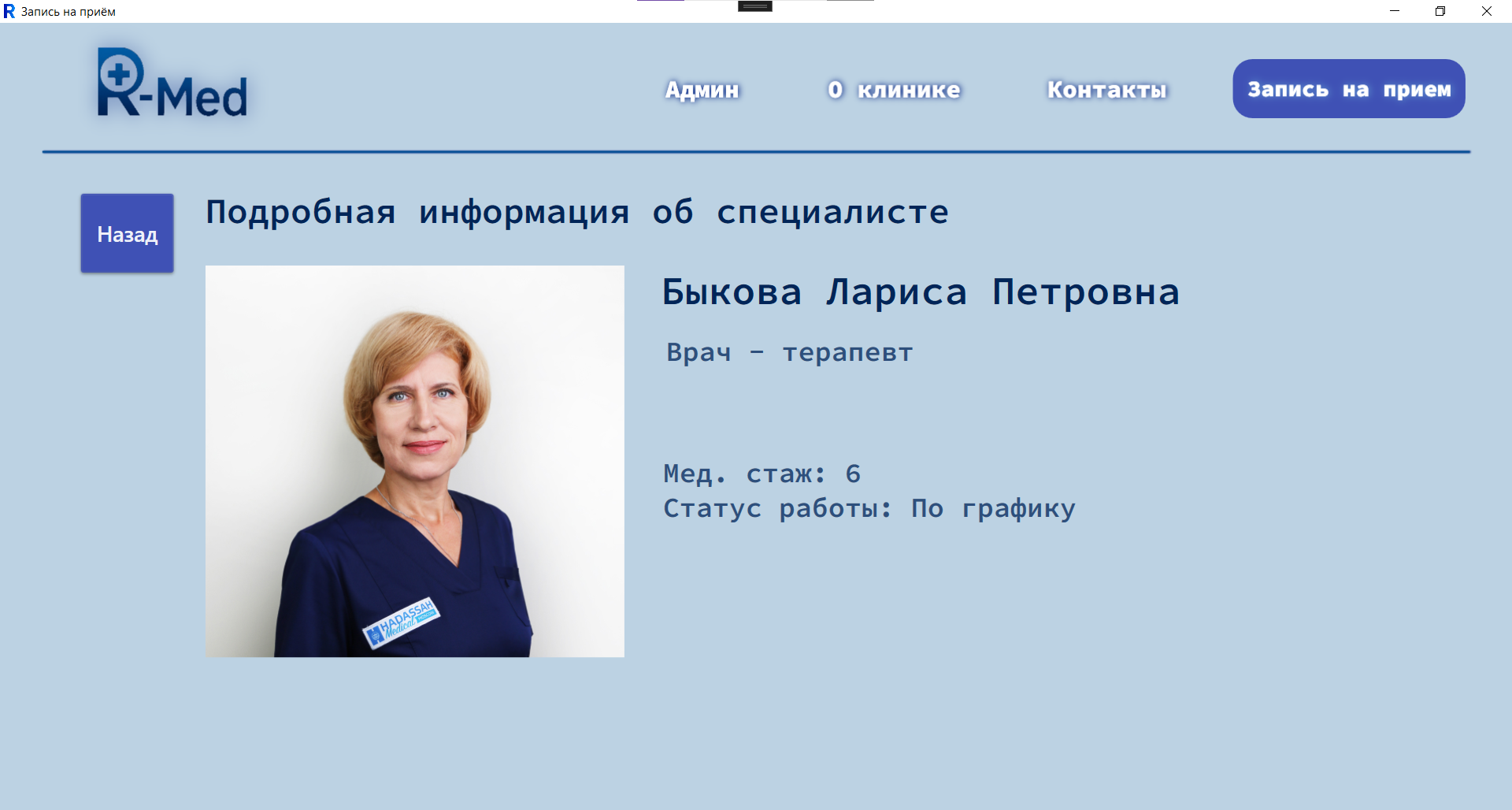


Рисунок 2.3.7 – Страница подробной информации специалиста.

При нажатии на кнопку “Записаться на приём” на странице с врачами, пользователя перенаправляет на страницу записи к врачу. На данной странице представлен календарь с ограниченными датами и надпись об выборе даты для показа свободных лотов времени. (рисунок 2.3.8).

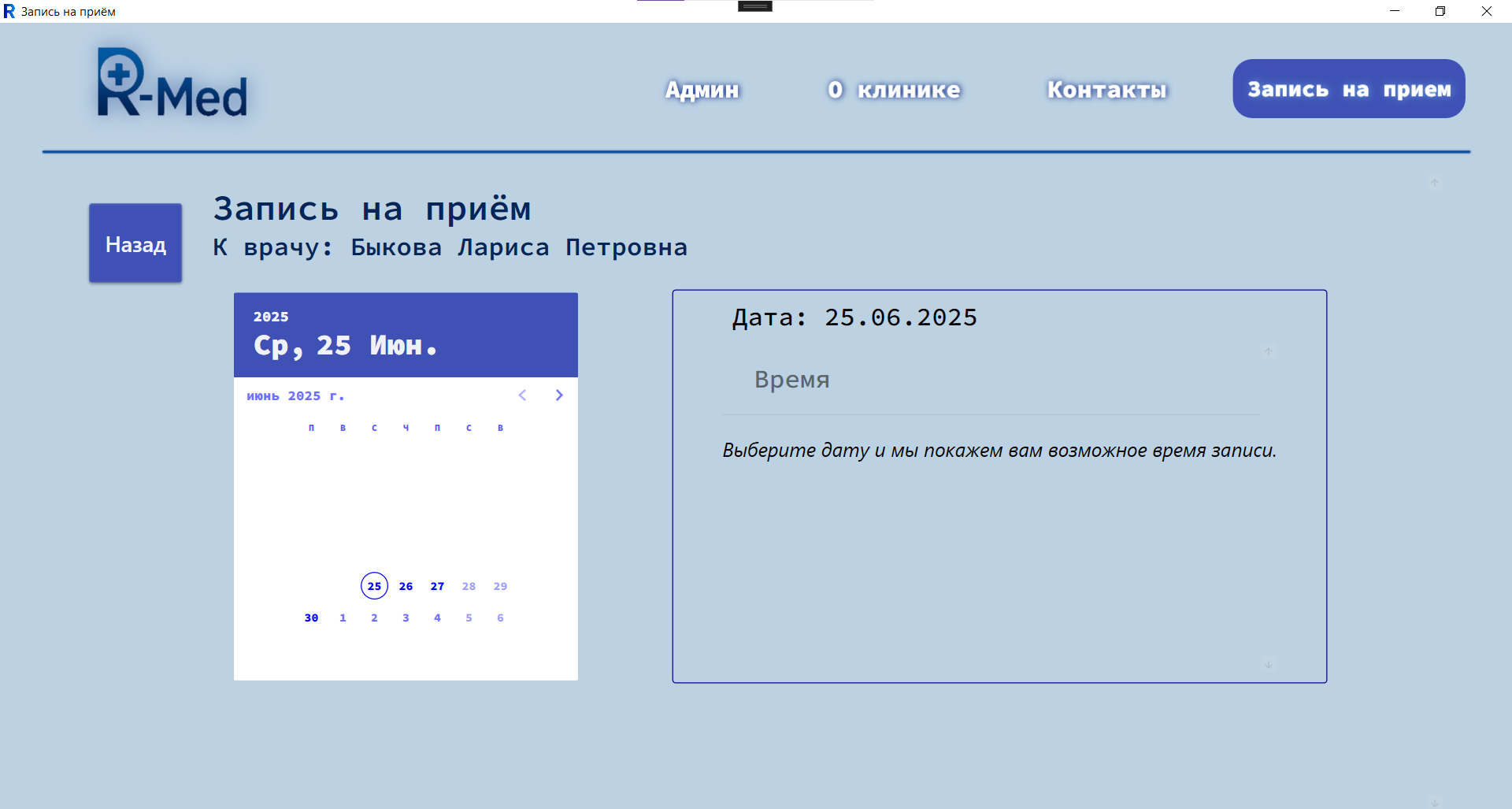


Рисунок 2.3.8 – Окно записи на приём.

Далее необходимо выбрать доступную дату в календаре и время записи в соответствующем поле (рисунок 2.3.9).

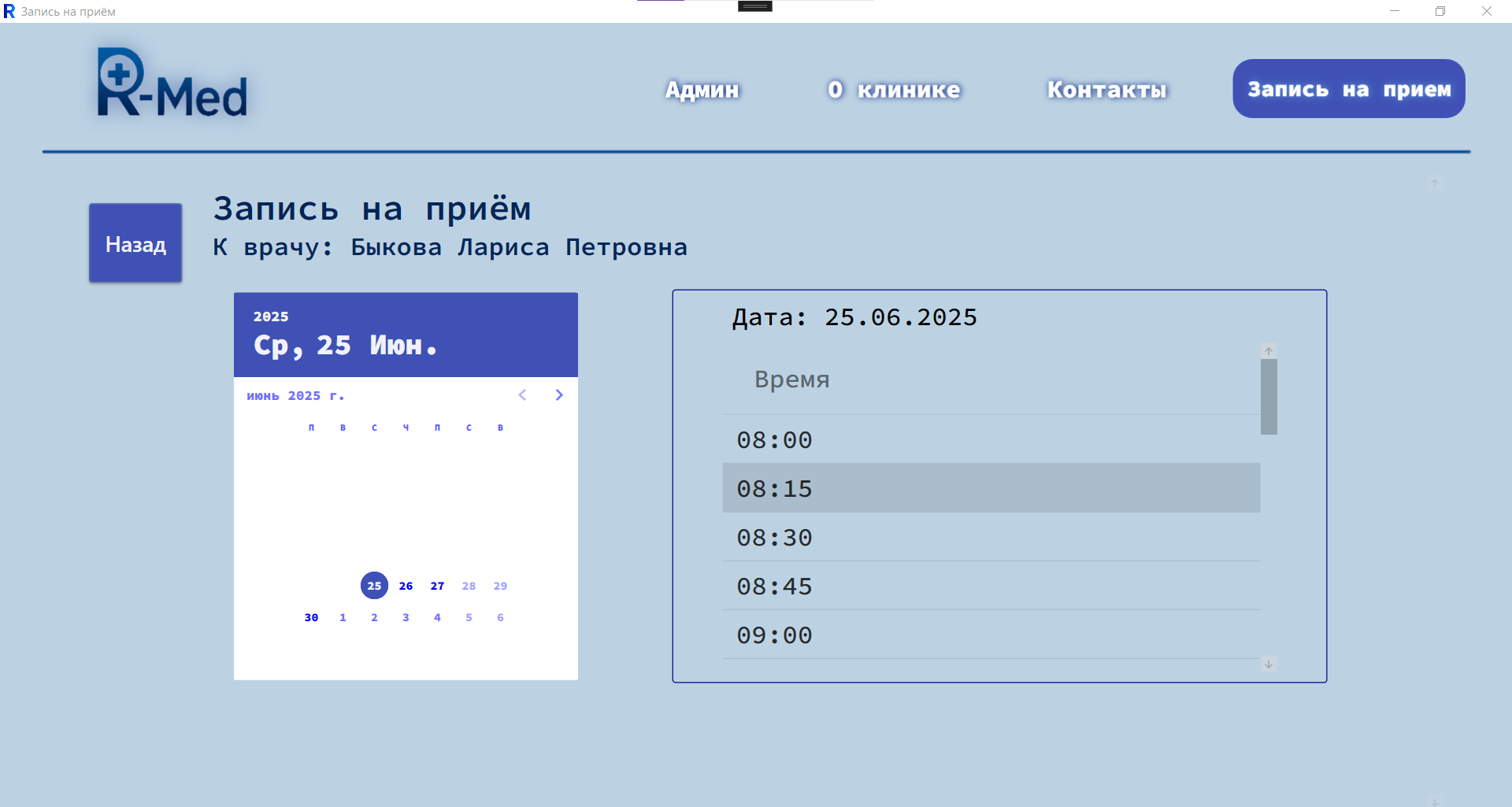


Рисунок 2.3.9 – Окно записи на приём с выбором даты и времени.

Выбрав удобную дату и время для записи на приём к врачу, система перенаправит пользователя чуть ниже для заполнения личных данных и получения кода подтверждения для последующей успешной записи (рисунки 2.3.10 – 2.3.11).

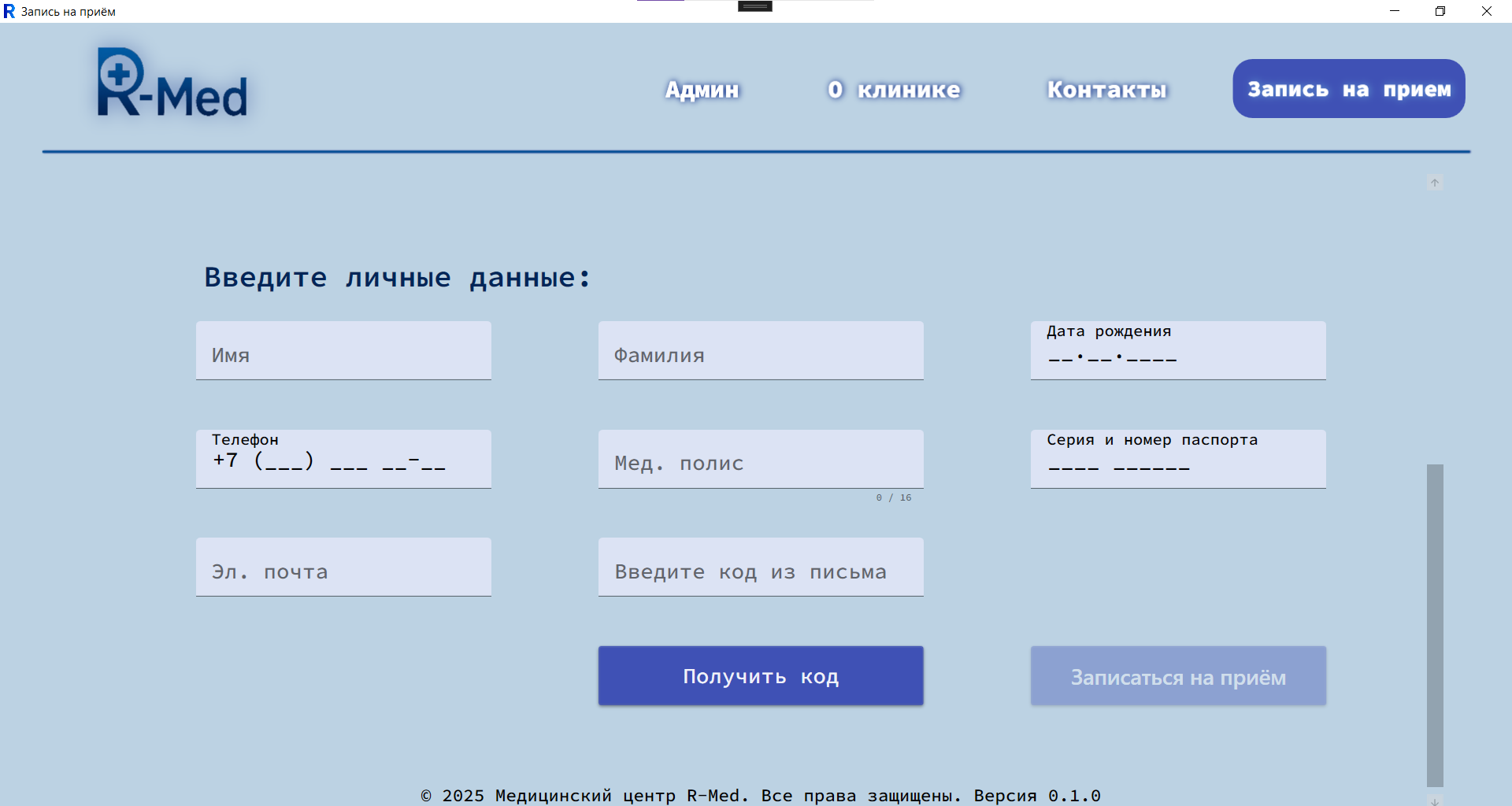


Рисунок 2.3.10 – Окно записи на приём с внесением личных данных.

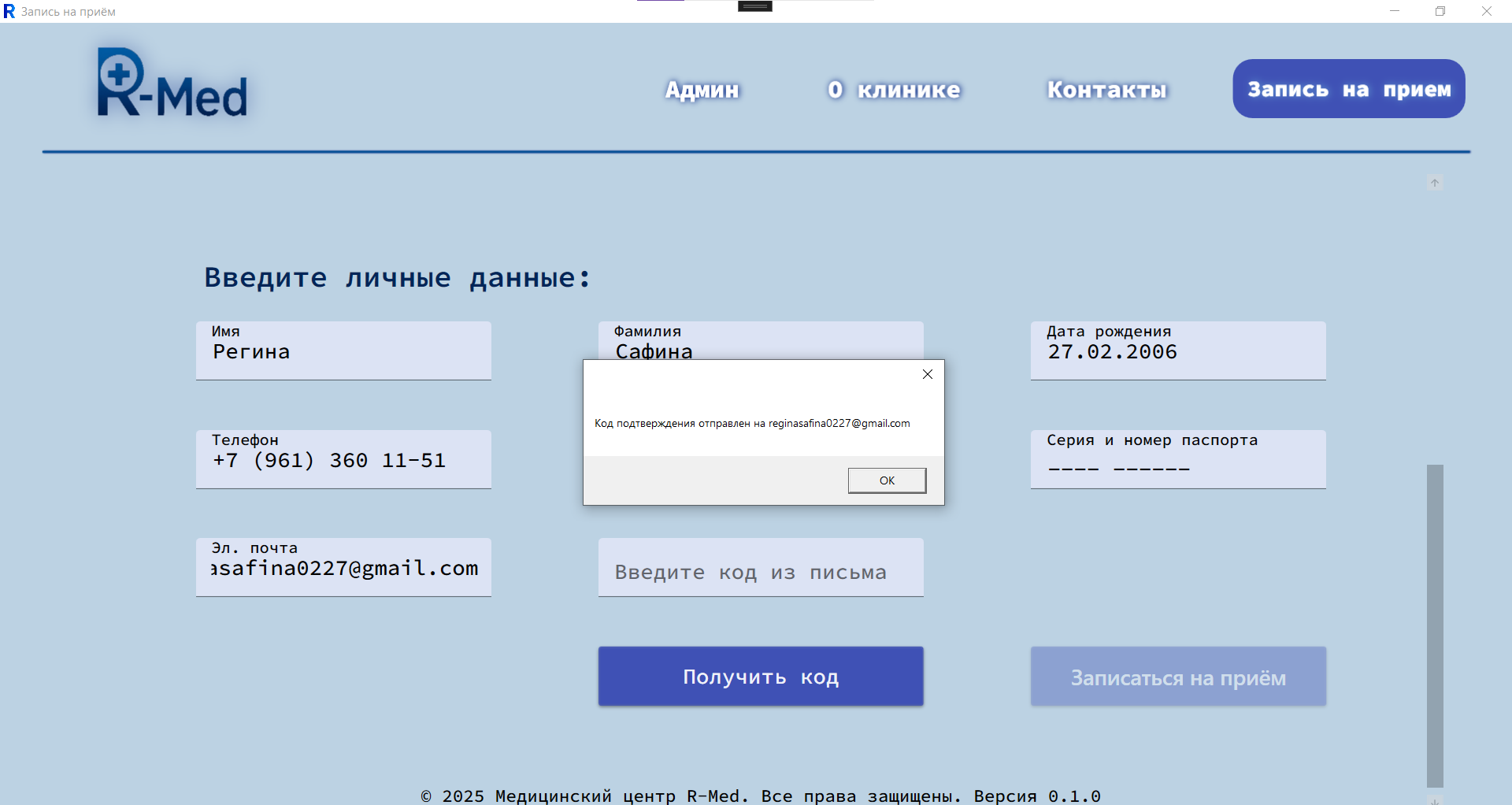


Рисунок 2.3.11 – Окно записи на приём с внесением личных данных.

Уже прикрепленный к клинике пользователь, может не вносить данные медицинского полиса и паспорта, но при этом может их обновить. Незарегистрированному пользователю обязательно нужно внести все свои данные для записи на приём и автоматического прикрепления к клинике (рисунок 2.3.12).

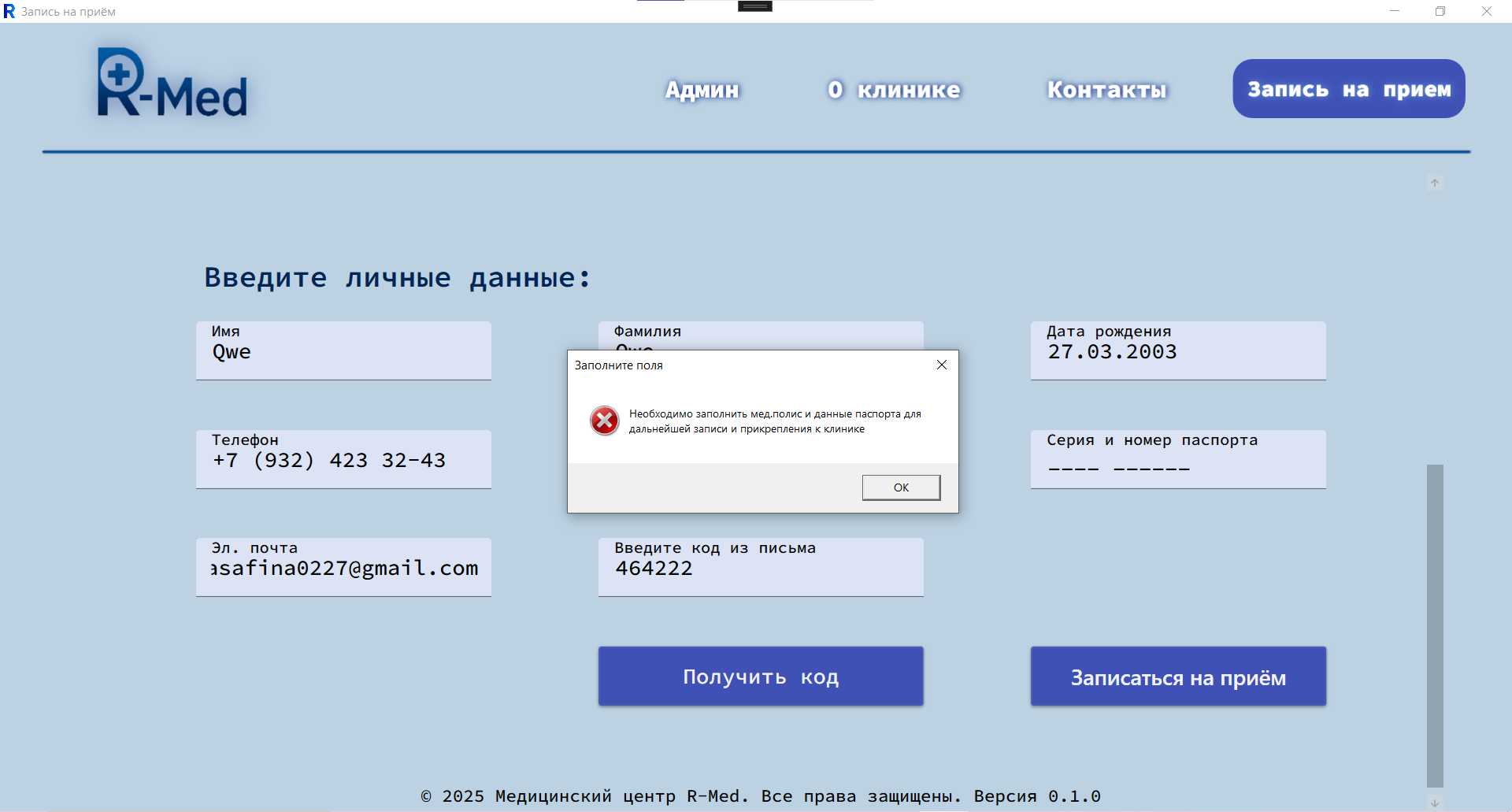


Рисунок 2.3.12 – Сообщение об незарегистрированном пользователе.

Внеся все необходимые данные, пользователя перенаправляет на страницу с талоном об успешной записи, который можно распечатать нажав на кнопку “Распечатать талон”. Также нажав на кнопку “На главную страницу”, пользователь перейдет на главное окно со всеми специальностями. (рисунки 2.3.13 – 2.3.14)

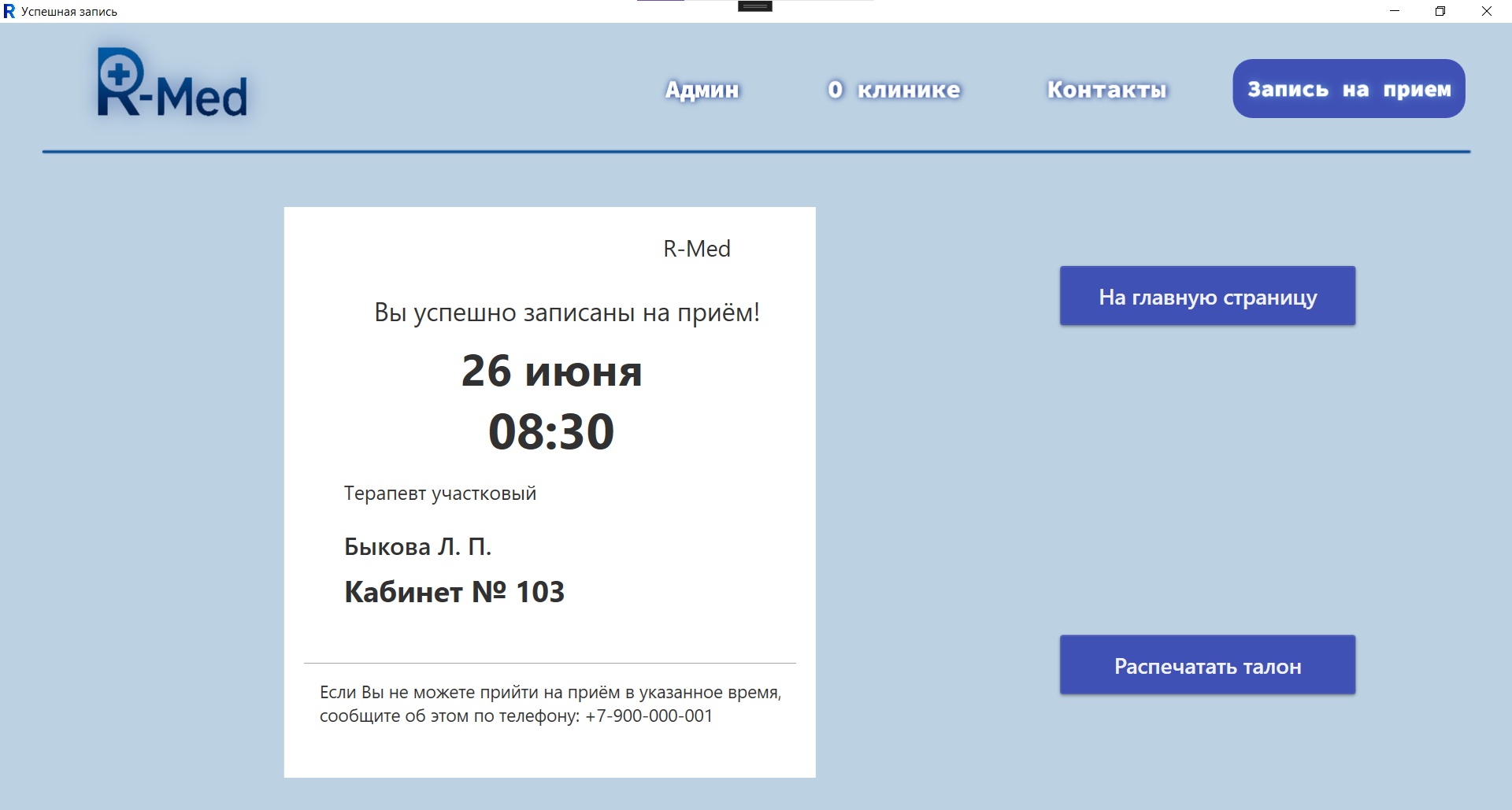
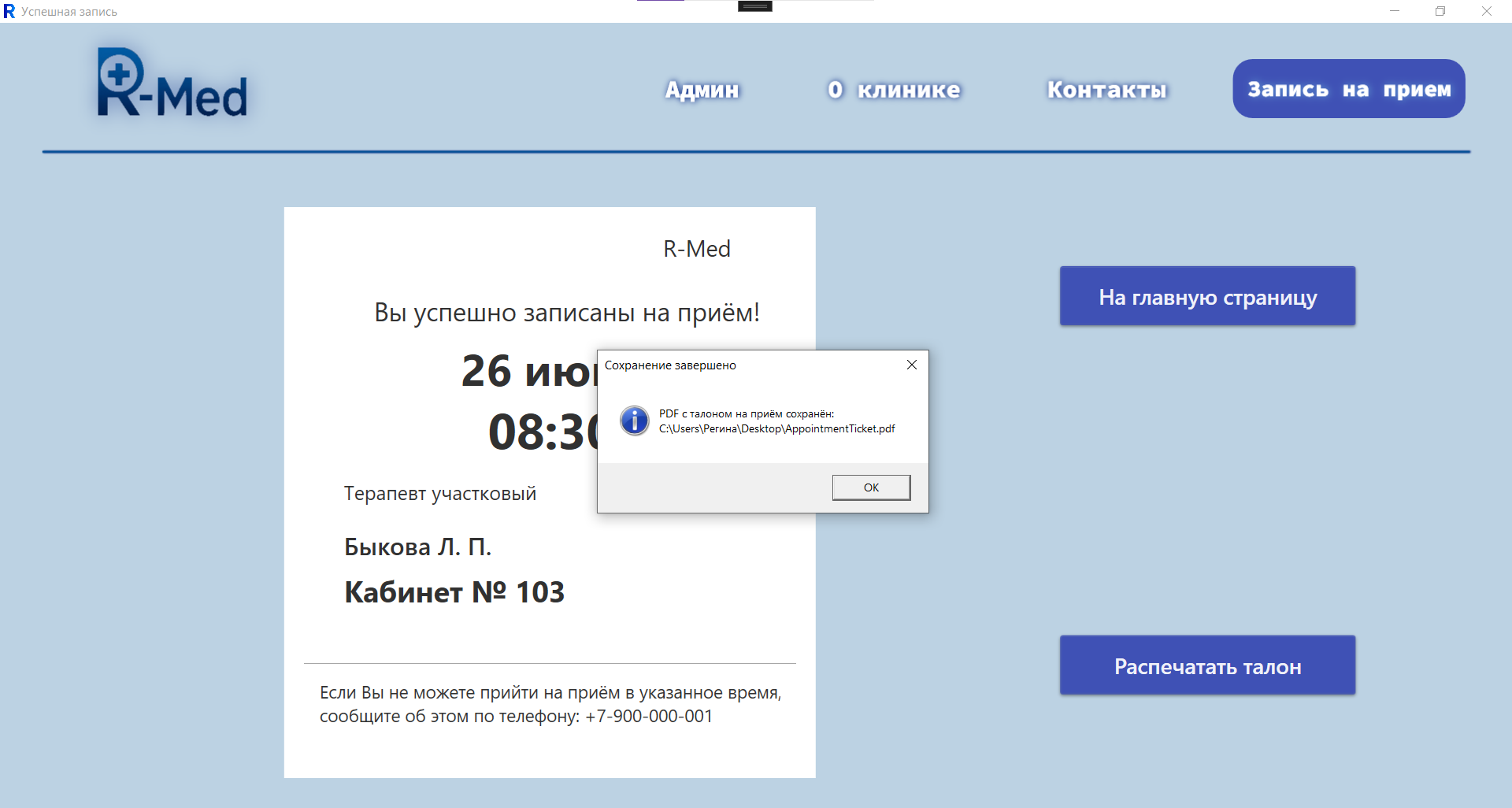


Рисунок 2.3.13 – Талон об успешной записи

Рисунок 2.3.14 – Сообщение об успешном сохранении талона.

Администраторская панель представляет собой несколько страниц с добавлением/редактированием врачей, пользователей, специализаций, графиков на приём (рисунки 2.3.15 – 2.3.16).

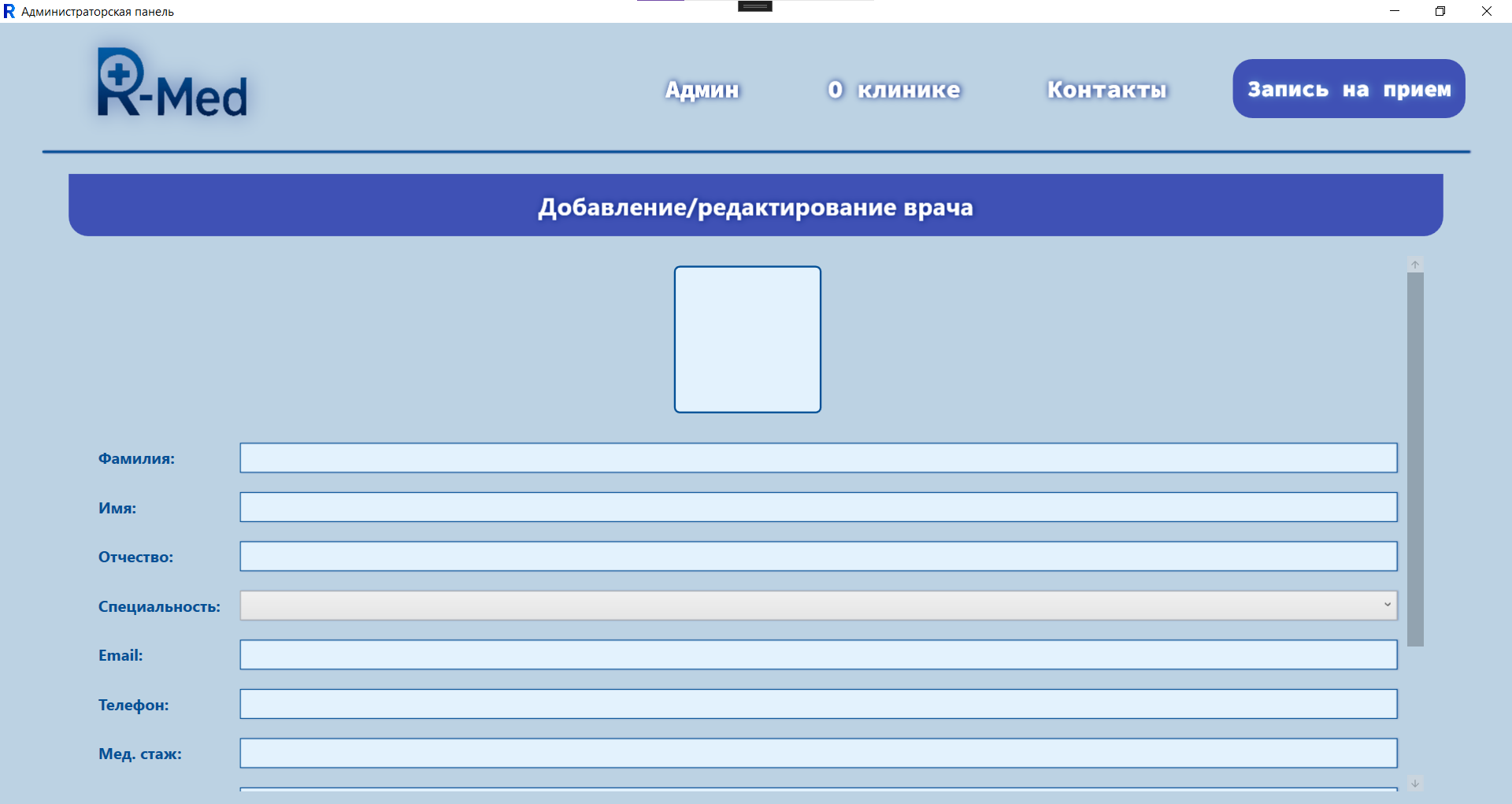
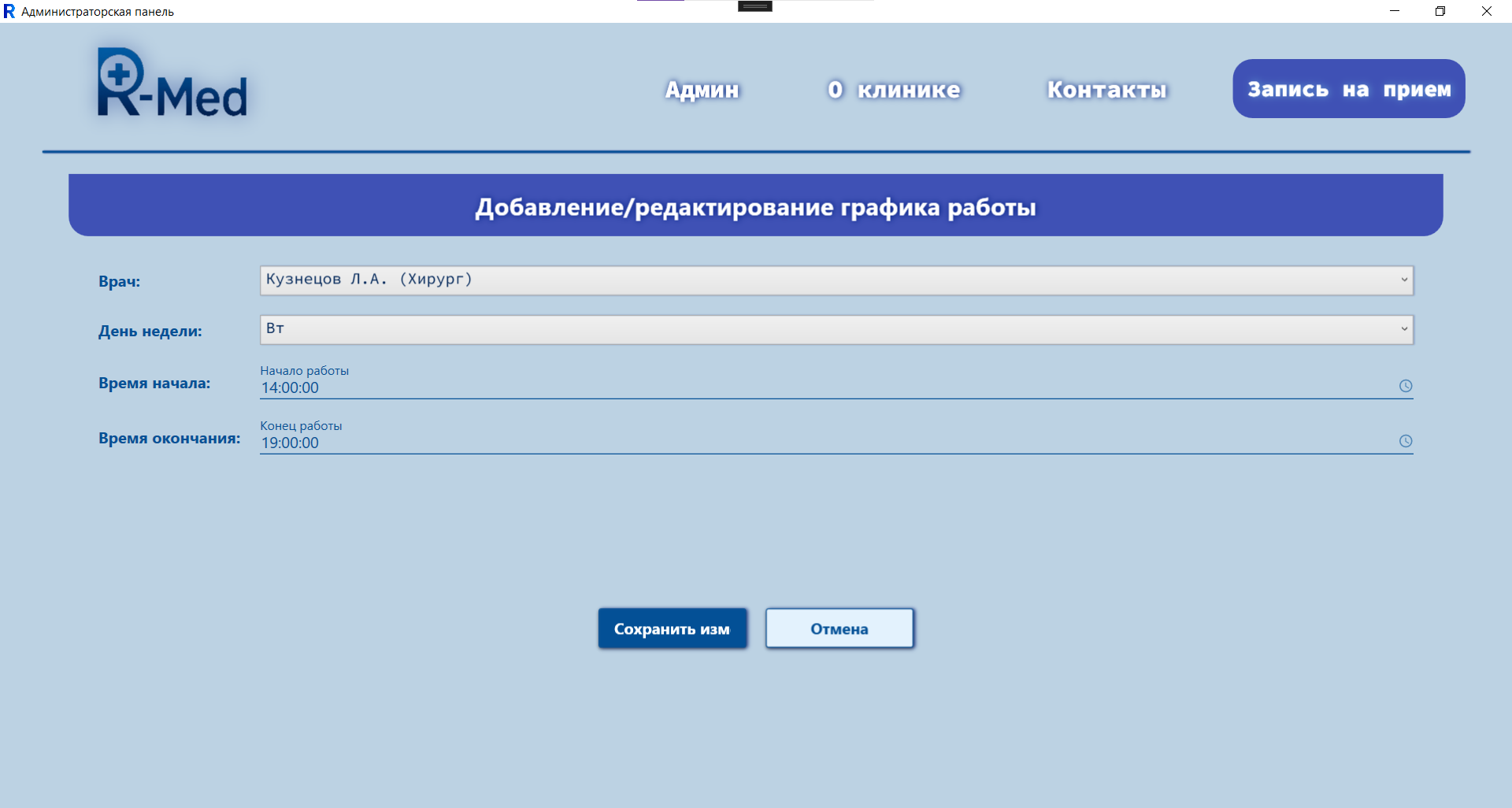


Рисунок 2.3.15 – Список специалистов

Рисунок 2.3.16 – Список специалистов

ЗАКЛЮЧЕНИЕ

В процессе выполнения курсового проекта были разработаны структура и алгоритм работы WPF-приложения «Медицинский центр R-Med».

При этом были изучены особенности реализации компонентов WPF для построения клиентских приложений с визуально привлекательными возможностями взаимодействия с пользователем.

Результатом работы стало создание WPF-приложения, которое сокращает время на обслуживание пациентов, автоматизирует рутинные процессы регистратуры, а также, позволяющее пользователям просматривать доступные записи к врачам, осуществлять запись на приём к конкретному специалисту, искать специалистов по их специализациям и оформлять талоны на приём.

WPF-приложение написано на языке C# в среде разработки Visual Studio 2022 с использованием языка разметки XAML и системы управления базой данных MS SQL Server 2022.

Были проведены опытная эксплуатация и отладочное тестирование WPF приложения. По результатам отладочного тестирования были устранены некоторые недостатки, в частности были обнаружены и исправлены неточности в реализации алгоритма: усовершенствован контроль на входные данные и отформатирован вывод документов. После этого было написано руководство пользователя.

С помощью приложения на основании данных контрольного примера были получены результаты, которые полностью совпадают с выходной информацией контрольного примера.

Шаблоны выходных документов

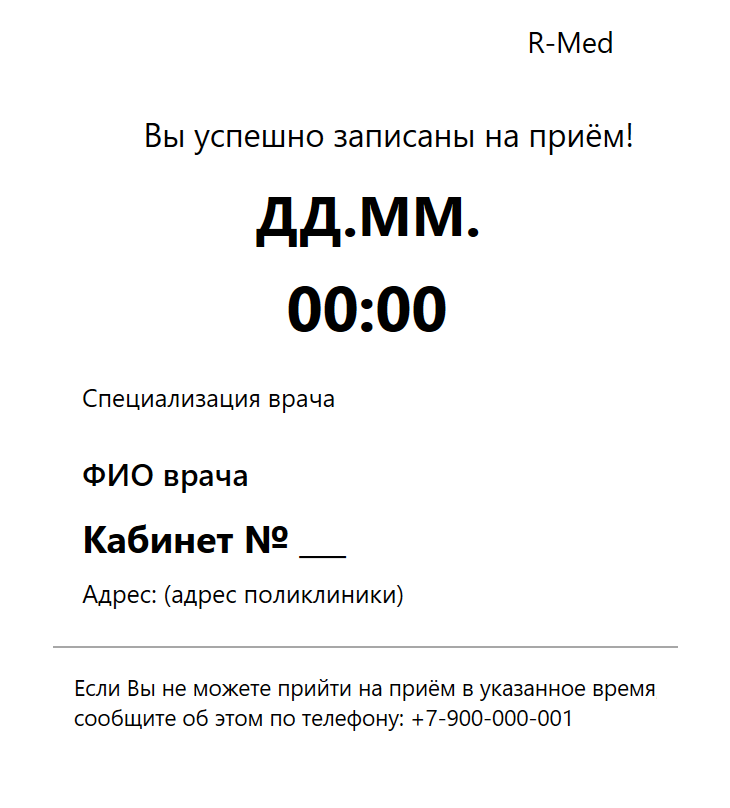


Рисунок А.1 – Шаблон об записи на приём

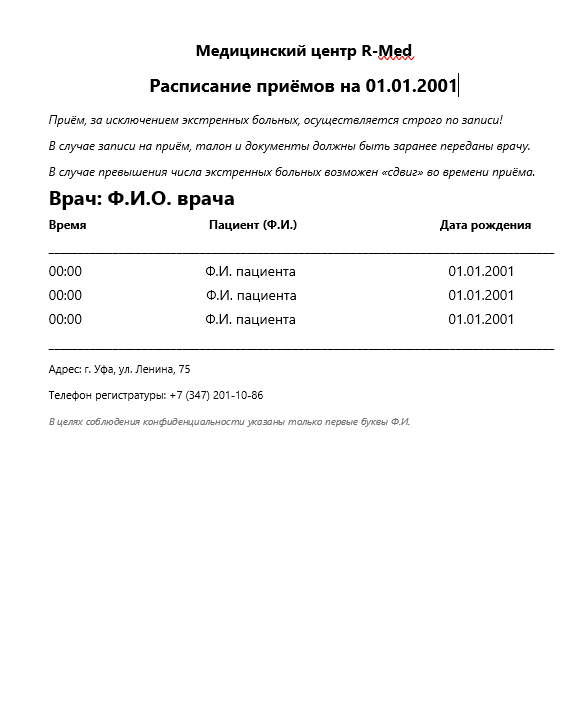


Рисунок А.2 – Шаблон о расписании записей на приём врача

Концептуальная модель базы данных.

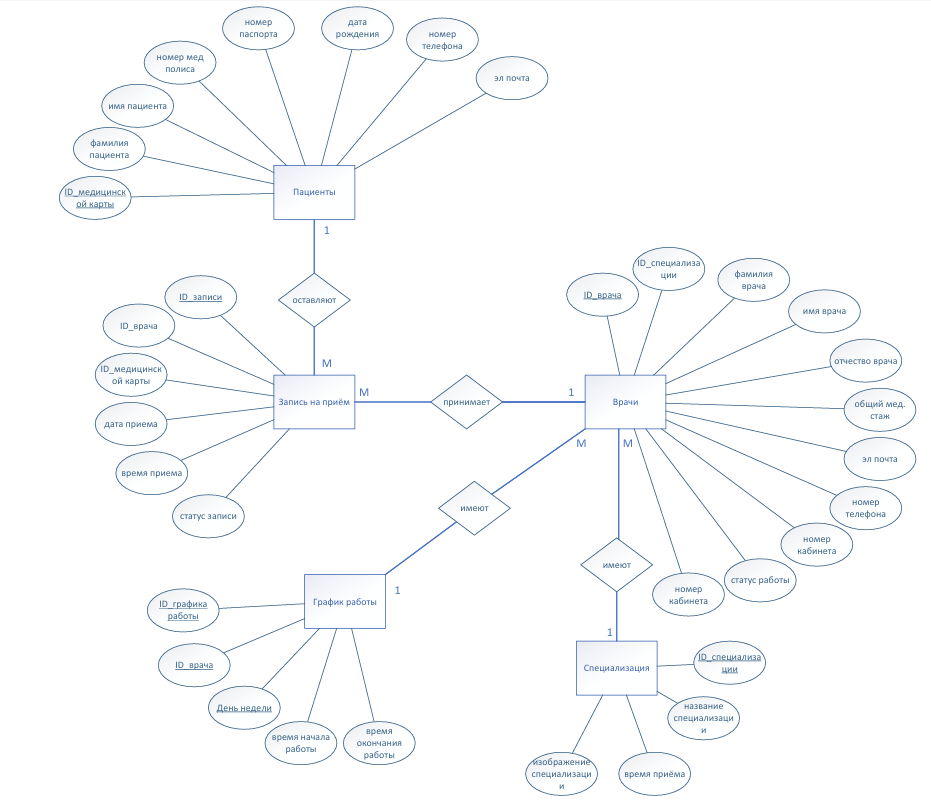


Рисунок Б.1 – Концептуальная модель

Схема данных

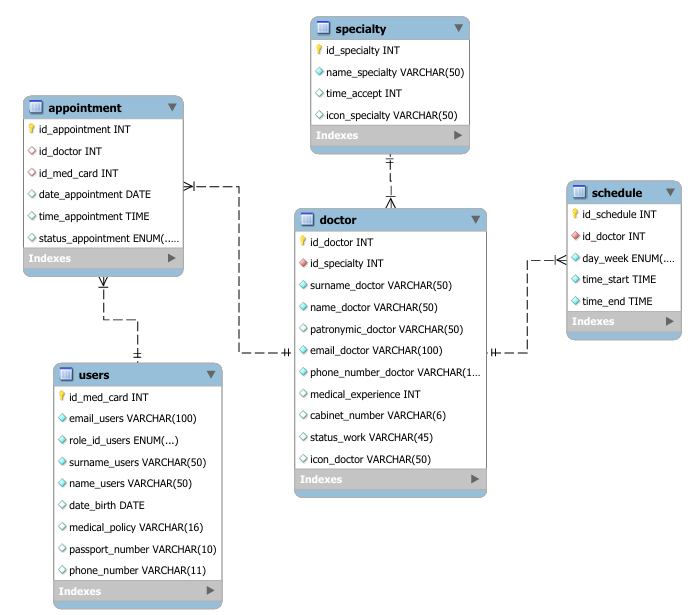


Рисунок В.1 – Схема базы данных

Входные данные контрольного примера

Таблица Г.1 – Справочник врачей

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| ID врача | ID спец-ти | Фамилия врача | Имя врача | Фамилия врача | Эл. почта | Номер телефона | Стаж работы | Номер кабинета | Статус | icon\_doctor |
| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | larisa.jpg |
| 1 | 1 | Быкова | Лариса | Петровна | larisa@mail.ru | 79613601152 | 6 | 103 |  | leonid.jpg |
| 2 | 7 | Кузнецов | Леонид | Андреевич | smirnovv02@yandex.ru | 79063641151 | 15 | 133 |  | moroz.jpg |
| 3 | 6 | Морозова | Татьяна | Владимировна | moroz\_t23@mail.ru | 79870534611 | 10 | 101А |  | komarov.jpg |
| 4 | 10 | Комаров | Юрий | Олегович | komar\_ova22@gmail.com | 79053602849 | 6 | 135 |  | popova.jpg |
| 5 | 5 | Попова | Мария | Кирилловна | popova\_m@clinic.ru | 79261112233 | 8 | 201 | В отпуске | metastazova.jpg |
| 6 | 9 | Смирнова | Василиса | Валерьевна | vasilisa.smirnova@hospital.ru | 79154445566 | 12 | 205 |  | zaitseva.jpg |
| 7 | 2 | Волкова | Екатерина | Дмитриевна | volkova\_e@medcenter.ru | 79677778899 | 7 | 112 |  | krukova.jpg |

Продолжение таблицы Г.1

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 |
| 8 | 2 | Крюкова | Ксения | Романовна | ksenia.krukova@zdorov.ru | 7903888900 | 9 | 211 | Б/л | krukova.jpg |
| 9 | 1 | Филинова | Ольга | Сергеевна | filin\_olya@mail.ru | 79463754483 | 10 | 103 |  | filinova.jpg |

Таблица Г.2 – Справочник графиков работы

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| ID графика | ID врача | День недели | Время начала работы | Время конца работы |
| 1 | 2 | 3 | 4 | 5 |
| 1 | 1 | Пн | 08:00:00 | 14:00:00 |
| 2 | 1 | Вт | 08:00:00 | 14:00:00 |
| 3 | 1 | Ср | 08:00:00 | 14:00:00 |
| 4 | 1 | Чт | 08:00:00 | 14:00:00 |
| 5 | 1 | Пт | 08:00:00 | 14:00:00 |
| 6 | 2 | Пн | 14:00:00 | 19:00:00 |
| 7 | 2 | Вт | 14:00:00 | 19:00:00 |
| 8 | 2 | Ср | 14:00:00 | 19:00:00 |
| 9 | 2 | Чт | 14:00:00 | 19:00:00 |

Таблица Г.3 – Справочник специализаций

|  |  |  |  |
| --- | --- | --- | --- |
| ID спец-ти | Название специальности | Время приёма | icon\_specialty |
| 1 | 2 | 3 | 4 |
| 1 | Терапевт | 15 | therapist.png |
| 2 | Педиатр | 20 | pediatry.png |
| 3 | Офтальмолог | 15 | eyes.png |
| 4 | Отоларинголог | 20 | lor.png |
| 5 | Дерматовенеролог | 20 | dermatology.png |
| 6 | Травматолог | 35 | trauma.png |
| 7 | Хирург | 25 | surgery.png |
| 8 | Аллерголог | 35 | allergy.png |
| 9 | Онколог | 25 | oncology.png |
| 10 | Кардиолог | 25 | cardiology.png |
| 11 | Уролог | 30 | urology.png |
| 12 | Акушер-гинеколог | 30 | ginecolog.png |

Выходные данные контрольного примера

Таблица Д.1 – Справочник записей на приём

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| ID записи | ID врача | ID мед. карты | Дата записи | Время записи | Статус записи |
| 1 | 2 | 3 | 4 | 5 | 6 |
| 12 | 1 | 12 | 2025-06-23 | 09:00:00 | Завершён |
| 13 | 7 | 5 | 2025-04-24 | 10:30:00 | Завершён |
| 14 | 7 | 11 | 2025-04-22 | 11:00:00 | Завершён |
| 26 | 1 | 123 | 2025-06-26 | 08:30:00 | В ожидании |
| 27 | 1 | 6 | 2025-06-25 | 08:30:00 | В ожидании |
| 28 | 1 | 3 | 2025-06-25 | 12:00:00 | В ожидании |

Код программы

DoctorsControl.xaml.cs

public partial class DoctorsControl : UserControl

{

public Doctor currDoct { get; set; }

Specialty specialtyName;

public DoctorsControl(Doctor doctor, Specialty spec)

{

specialtyName = spec;

currDoct = doctor;

InitializeComponent();

DataContext = doctor;

specialtyNameTxt.Text = "Врач - " + spec.NameSpecialty.ToString().ToLower();

}

private void toAppointment\_button\_Click(object sender, RoutedEventArgs e)

{

ClinikMainWindow mainWindow = Application.Current.MainWindow as ClinikMainWindow;

mainWindow.mainframe.NavigationService.Navigate(new AppointmentPage(specialtyName, currDoct));

}

private void informationDoctor\_Click(object sender, RoutedEventArgs e)

{

ClinikMainWindow mainWindow = Application.Current.MainWindow as ClinikMainWindow;

mainWindow.mainframe.NavigationService.Navigate(new InformationDoctorPage(specialtyName, currDoct));

}

}

SpecialtyControl.xaml.cs

public partial class SpecialtyControl : UserControl

{

private Specialty currSpec;

public SpecialtyControl(Specialty specialty)

{

currSpec = specialty;

InitializeComponent();

DataContext = currSpec;

}

private void UserControl\_Loaded(object sender, RoutedEventArgs e)

{

}

private void Click\_toAppointment(object sender, RoutedEventArgs e)

{

using (var db = new DbAppontmentClinikContext())

{

currSpec = db.Specialties.FirstOrDefault(spec => spec.NameSpecialty == selectedSpecialtyTxtbx.Text);

if (currSpec == null)

{

MessageBox.Show("Специальность не найдена!");

return;

}

}

ClinikMainWindow mainWindow = Application.Current.MainWindow as ClinikMainWindow;

if (mainWindow != null && mainWindow.mainframe != null)

{

mainWindow.mainframe.NavigationService.Navigate(new AllSpecialtiesPage(currSpec));

}

}

}

Appointment.cs

public partial class Appointment

{

public int IdAppointment { get; set; }

public int? IdDoctor { get; set; }

public int? IdMedCard { get; set; }

public DateOnly? DateAppointment { get; set; }

public TimeOnly? TimeAppointment { get; set; }

public string? StatusAppointment { get; set; }

public virtual Doctor? IdDoctorNavigation { get; set; }

public virtual User? IdMedCardNavigation { get; set; }

}

Doctor.cs

public partial class Doctor

{

public int IdDoctor { get; set; }

public int IdSpecialty { get; set; }

public string SurnameDoctor { get; set; } = null!;

public string NameDoctor { get; set; } = null!;

public string? PatronymicDoctor { get; set; }

public string EmailDoctor { get; set; } = null!;

public string PhoneNumberDoctor { get; set; } = null!;

public int? MedicalExperience { get; set; }

public string? CabinetNumber { get; set; }

public string? StatusWork { get; set; }

public string? IconDoctor { get; set; }

public virtual ICollection<Appointment> Appointments { get; set; } = new List<Appointment>();

public virtual Specialty IdSpecialtyNavigation { get; set; } = null!;

public virtual ICollection<Schedule> Schedules { get; set; } = new List<Schedule>();

public string DisplayIconDoctor => (System.IO.Path.Combine(Environment.CurrentDirectory, @"..\..\..")) + "\\Resources\\Doctors\\" + ((IconDoctor == null || string.IsNullOrEmpty(IconDoctor)) ? "default\_doctor.png" : IconDoctor);

public string DisplayNameDoctor => $"{SurnameDoctor} {NameDoctor} {PatronymicDoctor}";

public string DisplayMedExperience => $"Мед. стаж: {MedicalExperience}";

public string DisplayStatusWork => string.IsNullOrEmpty(StatusWork) ? $"Статус работы: По графику" : $"Статус работы: {StatusWork}";

}

Schedule.cs

public partial class Schedule

{

public int IdSсhedule { get; set; }

public int IdDoctor { get; set; }

public string DayWeek { get; set; } = null!;

public TimeOnly TimeStart { get; set; }

public TimeOnly TimeEnd { get; set; }

public virtual Doctor IdDoctorNavigation { get; set; } = null!;

}

Specialty.cs

public partial class Specialty

{

public int IdSpecialty { get; set; }

public string NameSpecialty { get; set; } = null!;

public int? TimeAccept { get; set; }

public string? IconSpecialty { get; set; }

public virtual ICollection<Doctor> Doctors { get; set; } = new List<Doctor>();

public string DisplayIconSpecialty => (System.IO.Path.Combine(Environment.CurrentDirectory, @"..\..\..")) + "\\Resources\\" + ((IconSpecialty == null || string.IsNullOrEmpty(IconSpecialty)) ? null : IconSpecialty);

}

User.cs

public partial class User

{

public int IdMedCard { get; set; }

public string EmailUsers { get; set; } = null!;

public string RoleIdUsers { get; set; } = null!;

public string SurnameUsers { get; set; } = null!;

public string NameUsers { get; set; } = null!;

public DateOnly? DateBirth { get; set; }

public string? MedicalPolicy { get; set; }

public string? PassportNumber { get; set; }

public string? PhoneNumber { get; set; }

public virtual ICollection<Appointment> Appointments { get; set; } = new List<Appointment>();

}

Email\_code.cs

internal class Email\_code

{

public const int smtpPort = 587;

public const string smtpServer = "smtp.mail.ru";

public const string smtpUsername = "rr-medd@mail.ru";

public const string smtpPassword = "8z5BeeXoTx8EeHlrRMIF";

//отправка email (кому, тема, текст)

public static void SendMessage(string to, string title, string message)

{

using SmtpClient smtpClient = new SmtpClient(smtpServer, smtpPort);

smtpClient.Credentials = new NetworkCredential(smtpUsername, smtpPassword);

smtpClient.EnableSsl = true;

using MailMessage mailMessage = new MailMessage();

mailMessage.IsBodyHtml = true;

mailMessage.From = new MailAddress(smtpUsername);

mailMessage.To.Add(to);

var logoAttachment = new Attachment("C:\\Users\\Регина\\Desktop\\repositories\\Appointment\_\_Clinik\\Main\_project\\Main\_project\\Resources\\image.png");

logoAttachment.ContentId = "logo"; // Это важно - тот же ID, что и в HTML (cid:logo)

mailMessage.Attachments.Add(logoAttachment);

mailMessage.Subject = title;

mailMessage.Body = message;

try

{

smtpClient.Send(mailMessage);

}

catch (SmtpException ex) when (

ex.Message.Contains("invalid mailbox") ||

ex.Message.Contains("Mailbox unavailable") ||

ex.Message.Contains("recipient verification failed")

)

{

MessageBox.Show($"Ошибка: почта {to} не существует или заблокирована.");

}

catch (Exception ex)

{

MessageBox.Show($"Ошибка отправки сообщения: {ex.Message}");

}

}

//письмо о регистрации

public static List<string> GenerateRegMessage(DateTime? appointment, string name, string id)

{

string formattedDate = appointment.Value.ToString("dd.MM.yyyy");

string formattedTime = appointment.Value.ToString("HH:mm");

string body = $@"

<!DOCTYPE html>

<html>

<head>

<style>

body {{ font-family: 'Segoe UI', Arial, sans-serif; line-height: 1.6; color: #333; max-width: 600px; margin: 0 auto; }}

.header {{

background-color: #4a8bfc;

text-align: center;

border-radius: 10px 10px 0px 0px;

}}

.logo-container {{

display: inline-flex;

flex-direction: column;

align-items: center;

gap: 15px;

margin-bottom: 15px;

}}

.content {{

padding: 25px;

background: #ffffff;

}}

.code {{

font-size: 28px;

font-weight: bold;

color: #4a8bfc;

text-align: center;

margin: 25px 0;

padding: 20px;

background: #f5f9ff;

border-radius: 5px;

border: 2px dashed #4a8bfc;

letter-spacing: 3px;

}}

.footer {{

margin-top: 30px;

padding-top: 20px;

border-top: 1px solid #eee;

font-size: 13px;

color: #777;

text-align: center;

}}

.note {{

font-size: 14px;

color: #666;

margin-top: 20px;

}}

</style>

</head>

<body>

<div class='header'>

<div class='logo-container'>

<img src='cid:logo' style='height: 300px; width: 300px;' alt='R-Med icon'>

</div>

</div>

<div class='content'>

<h1 style='color: #4a8bfc; text-align: center;'>Добро пожаловать в нашу клинику!</h1>

<h1>Уважаемый(ая) {name}!</h1>

<h3>Мы рады сообщить, что вы успешно прикреплены к нашей клинике. Теперь вы можете

записываться на прием к нашим специалистам в любое удобное время.</h3>

<div style='background-color: #e8f4ff; padding: 15px; border-radius: 6px; margin: 20px 0;'>

<p style='margin: 5px 0;'><strong>Ваш номер медицинской карты:</strong> {id}</p>

<p style='margin: 5px 0;'><strong>Дата прикрепления:</strong> {DateTime.Now.ToShortDateString()}</p>

</div>

</div>

<div class='footer'>

<h3>С уважением,<br><strong>❤ Команда R-Med ❤</strong></h3>

<p>Обратная связь: <a href='mailto:rr-medd@mail.ru' style='color: #4a8bfc;'>rr-meddp@mail.ru</a></p>

<p style='margin-top: 10px;'>{formattedDate} в {formattedTime}</p>

</div>

</body>

</html>";

string title = $"Успешное прикрепление к клинике!";

return new List<string> { title, body };

}

//письмо с кодом подтверждения GenerateVerificateMessageAdmin

public static List<string> GenerateVerificateMessageAdmin(DateTime? appointment, string verificationCode)

{

string formattedDate = appointment.Value.ToString("dd.MM.yyyy");

string formattedTime = appointment.Value.ToString("HH:mm");

string body = $"Администратор, добро пожаловать!\r\n\r\n" +

$"Код подтверждения: {verificationCode}\r\n\r\n\r\n";

string title = $"Ваш код подтверждения R-Med";

return new List<string> { title, body };

}

public static List<string> GenerateVerificateMessage(DateTime? appointment, string verificationCode)

{

string formattedDate = appointment.Value.ToString("dd.MM.yyyy");

string formattedTime = appointment.Value.ToString("HH:mm");

string body = $@"

<!DOCTYPE html>

<html>

<head>

<style>

body {{ font-family: 'Segoe UI', Arial, sans-serif; line-height: 1.6; color: #333; max-width: 600px; margin: 0 auto; }}

.header {{

background-color: #4a8bfc;

text-align: center;

border-radius: 10px 10px 0px 0px;

}}

.logo-container {{

display: inline-flex;

flex-direction: column;

align-items: center;

gap: 15px;

margin-bottom: 15px;

}}

.logo-text {{

color: white;

font-size: 24px;

font-weight: bold;

}}

.content {{

padding: 25px;

background: #ffffff;

}}

.code {{

font-size: 28px;

font-weight: bold;

color: #4a8bfc;

text-align: center;

margin: 25px 0;

padding: 20px;

background: #f5f9ff;

border-radius: 5px;

border: 2px dashed #4a8bfc;

letter-spacing: 3px;

}}

.footer {{

margin-top: 30px;

padding-top: 20px;

border-top: 1px solid #eee;

font-size: 13px;

color: #777;

text-align: center;

}}

.note {{

font-size: 14px;

color: #666;

margin-top: 20px;

}}

</style>

</head>

<body>

<div class='header'>

<div class='logo-container'>

<img src='cid:logo' style='height: 300px; width: 300px;' alt='R-Med icon'>

</div>

</div>

<div class='content'>

<h1 style='color: #4a8bfc; text-align: center;'>Код подтверждения</h1>

<h1>Уважаемый пользователь!</h1>

<h3>Для записи на приём введите следующий код подтверждения:</h3>

<div class='code'>{verificationCode}</div>

<p class='note'>Код действителен в течение 5 минут. Никому не сообщайте этот код.</p>

</div>

<div class='footer'>

<h3>С уважением,<br><strong>❤ Команда R-Med ❤</strong></h3>

<p>Обратная связь: <a href='mailto:rr-medd@mail.ru' style='color: #4a8bfc;'>rr-meddp@mail.ru</a></p>

<p style='margin-top: 10px;'>{formattedDate} в {formattedTime}</p>

</div>

</body>

</html>";

string title = $"Ваш код подтверждения R-Med";

return new List<string> { title, body };

}

//генерация кода подтверждения

private static readonly Random \_random = new Random();

public static string GenerateCode()

{

string code = "";

for (int i = 0; i < 6; i++)

{

int randomNumber = \_random.Next(0, 9);

code += randomNumber.ToString();

}

return code;

}

}

PDF.cs

internal class PDF

{

public static void CreateAppointmentTicketPdf(string doctorSpecialization, string doctorName, string cabinetNumber, string clinicAddress, string date, string time)

{

string filePath = @"C:\Users\Регина\Desktop\AppointmentTicket.pdf";

try

{

using (var document = new PdfDocument())

{

// Узкие настройки шрифтов

var headerFont = new XFont("Segoe UI", 12, XFontStyle.Bold);

var titleFont = new XFont("Segoe UI", 14, XFontStyle.Bold);

var dateTimeFont = new XFont("Segoe UI", 18, XFontStyle.Bold);

var dateTimeFont1 = new XFont("Segoe UI", 22, XFontStyle.Bold);

var infoFont = new XFont("Segoe UI", 9);

var smallInfoFont = new XFont("Segoe UI", 7);

// Узкие настройки документа

const double margin = 20; // Минимальные отступы

const double contentWidth = 300; // Фиксированная ширина контента

double yPosition = margin;

PdfPage page = document.AddPage();

XGraphics gfx = XGraphics.FromPdfPage(page);

double centerX = page.Width / 2;

// Заголовок документа (компактный в углу)

gfx.DrawString("R-Med", headerFont, XBrushes.Black,

new XRect(page.Width - margin - 180, yPosition, 50, 20),

XStringFormats.TopLeft);

yPosition += 30;

// Основной текст

gfx.DrawString("Вы успешно записаны на приём!", titleFont, XBrushes.Black,

new XRect(centerX - contentWidth / 2, yPosition, contentWidth, 0),

XStringFormats.TopCenter);

yPosition += 30;

// Дата и время (в одну строку)

string dateTime = $"{date}";

gfx.DrawString(dateTime, dateTimeFont1, XBrushes.Black,

new XRect(centerX - contentWidth / 2, yPosition, contentWidth, 0),

XStringFormats.TopCenter);

yPosition += 40;

// Дата и время (в одну строку)

string Time = $"{time}";

gfx.DrawString(Time, dateTimeFont1, XBrushes.Black,

new XRect(centerX - contentWidth / 2, yPosition, contentWidth, 0),

XStringFormats.TopCenter);

yPosition += 50;

// Информация о враче (плотная группировка)

gfx.DrawString(doctorSpecialization, infoFont, XBrushes.Black,

new XRect(centerX - contentWidth / 2, yPosition, contentWidth, 0),

XStringFormats.TopLeft);

yPosition += 20;

gfx.DrawString($"Врач: {doctorName}", infoFont, XBrushes.Black,

new XRect(centerX - contentWidth / 2, yPosition, contentWidth, 0),

XStringFormats.TopLeft);

yPosition += 20;

gfx.DrawString($"{cabinetNumber}", dateTimeFont, XBrushes.Black,

new XRect(centerX - contentWidth / 2, yPosition, contentWidth, 0),

XStringFormats.TopLeft);

yPosition += 30;

gfx.DrawString($"Адрес: {clinicAddress}", infoFont, XBrushes.Black,

new XRect(centerX - contentWidth / 2, yPosition, contentWidth, 0),

XStringFormats.TopLeft);

yPosition += 25;

// Компактный разделитель

gfx.DrawLine(new XPen(XColors.Black, 0.5),

centerX - contentWidth / 2, yPosition,

centerX + contentWidth / 2, yPosition);

yPosition += 15;

// Компактная контактная информаци

gfx.DrawString("Если ВЫ не можете прийти на приём в указанное время, \nсообщите об этом по телефону: +7 (347) 201-10-86",

smallInfoFont, XBrushes.Gray,

new XRect(centerX - contentWidth / 2, yPosition, contentWidth, 0),

XStringFormats.TopLeft);

// Сохранение документа

document.Save(filePath);

Application.Current.Dispatcher.Invoke(() =>

{

MessageBox.Show(

$"PDF с талоном на приём сохранён:\n{filePath}",

"Сохранение завершено",

MessageBoxButton.OK,

MessageBoxImage.Information);

});

}

}

catch (Exception ex)

{

Application.Current.Dispatcher.Invoke(() =>

{

MessageBox.Show(

$"Ошибка при создании PDF: {ex.Message}",

"Ошибка",

MessageBoxButton.OK,

MessageBoxImage.Error);

});

}

}

public static void CreateTodaysAppointmentsPdf(string doctorFullName)

{

using var db = new DbAppontmentClinikContext();

string clinicAddress = "г. Уфа, ул. Ленина, 75";

// Получаем список сегодняшних записей с датой рождения

var appointments = GetDoctorAppointmentsWithBirthDate(doctorFullName);

string filePath = @"C:\Users\Регина\Desktop\TodaysAppointments.pdf";

try

{

using (var document = new PdfDocument())

{

// Настройки шрифтов

var headerFont = new XFont("Segoe UI", 16, XFontStyle.Bold);

var titleFont = new XFont("Segoe UI", 18, XFontStyle.Bold);

var infoFont = new XFont("Segoe UI", 12, XFontStyle.Italic);

var tableHeaderFont = new XFont("Segoe UI", 12, XFontStyle.Bold);

var tableContentFont = new XFont("Segoe UI", 11);

var footerFont = new XFont("Segoe UI", 9);

var smallInfoFont = new XFont("Segoe UI", 8, XFontStyle.Italic);

PdfPage page = document.AddPage();

XGraphics gfx = XGraphics.FromPdfPage(page);

double pageWidth = page.Width;

double yPosition = 30; // Начальная позиция

const double margin = 40;

// Заголовок документа

gfx.DrawString("Медицинский центр R-Med", headerFont, XBrushes.Black,

new XRect(0, yPosition, pageWidth, 0),

XStringFormats.TopCenter);

yPosition += 30;

// Заголовок

string today = DateTime.Today.ToString("dd.MM.yyyy");

gfx.DrawString($"Расписание приёмов на {today}", titleFont, XBrushes.Black,

new XRect(0, yPosition, pageWidth, 0),

XStringFormats.TopCenter);

yPosition += 35;

// Важная информация

gfx.DrawString("Приём, за исключением экстренных больных, осуществляется строго по записи!", infoFont, XBrushes.Black,

new XRect(margin, yPosition, pageWidth - 2 \* margin, 0),

XStringFormats.TopLeft);

yPosition += 20;

gfx.DrawString($"В случае записи на приём, талон и документы должны быть заранее переданы врачу.", infoFont, XBrushes.Black,

new XRect(margin, yPosition, pageWidth - 2 \* margin, 0),

XStringFormats.TopLeft);

yPosition += 20;

gfx.DrawString("В случае превышения числа экстренных больных возможен «сдвиг» во времени приёма.", infoFont, XBrushes.Black,

new XRect(margin, yPosition, pageWidth - 2 \* margin, 0),

XStringFormats.TopLeft);

yPosition += 30;

// Информация о враче

gfx.DrawString($"Врач: {doctorFullName}", titleFont, XBrushes.Black,

new XRect(margin, yPosition, pageWidth - 2 \* margin, 0),

XStringFormats.TopLeft);

yPosition += 30;

// Создаем таблицу

double tableWidth = pageWidth - 2 \* margin;

double col1Width = tableWidth \* 0.15; // Время

double col2Width = tableWidth \* 0.45; // ФИО

double col3Width = tableWidth \* 0.4; // Дата рождения

// Заголовки таблицы

double tableYStart = yPosition;

gfx.DrawString("Время", tableHeaderFont, XBrushes.Black,

new XRect(margin, yPosition, col1Width, 20),

XStringFormats.TopLeft);

gfx.DrawString("Пациент (Ф.И.)", tableHeaderFont, XBrushes.Black,

new XRect(margin + col1Width, yPosition, col2Width, 20),

XStringFormats.TopLeft);

gfx.DrawString("Дата рождения", tableHeaderFont, XBrushes.Black,

new XRect(margin + col1Width + col2Width, yPosition, col3Width, 20),

XStringFormats.TopLeft);

yPosition += 25;

// Линия под заголовком

gfx.DrawLine(new XPen(XColors.Black, 1),

margin, yPosition,

margin + tableWidth, yPosition);

yPosition += 10;

// Заполняем таблицу данными

if (appointments.Any())

{

foreach (var app in appointments)

{

// Проверяем, не вышли ли за пределы страницы

if (yPosition > page.Height - 70) // Оставляем место для футера

{

// Создаем новую страницу

page = document.AddPage();

gfx = XGraphics.FromPdfPage(page);

yPosition = 30;

// Повторяем заголовок на новой странице

gfx.DrawString("Медицинский центр R-Med", headerFont, XBrushes.Black,

new XRect(0, yPosition, pageWidth, 0),

XStringFormats.TopCenter);

yPosition += 30;

gfx.DrawString($"Расписание приёмов на {today} (продолжение)", titleFont, XBrushes.Black,

new XRect(0, yPosition, pageWidth, 0),

XStringFormats.TopCenter);

yPosition += 35;

// Заголовки таблицы на новой странице

gfx.DrawString("Время", tableHeaderFont, XBrushes.Black,

new XRect(margin, yPosition, col1Width, 20),

XStringFormats.TopLeft);

gfx.DrawString("Пациент (Ф.И.)", tableHeaderFont, XBrushes.Black,

new XRect(margin + col1Width, yPosition, col2Width, 20),

XStringFormats.TopLeft);

gfx.DrawString("Дата рождения", tableHeaderFont, XBrushes.Black,

new XRect(margin + col1Width + col2Width, yPosition, col3Width, 20),

XStringFormats.TopLeft);

yPosition += 25;

gfx.DrawLine(new XPen(XColors.Black, 1),

margin, yPosition,

margin + tableWidth, yPosition);

yPosition += 10;

}

// Содержимое таблицы

gfx.DrawString(app.Time, tableContentFont, XBrushes.Black,

new XRect(margin, yPosition, col1Width, 20),

XStringFormats.TopLeft);

gfx.DrawString(app.PatientName, tableContentFont, XBrushes.Black,

new XRect(margin + col1Width, yPosition, col2Width, 20),

XStringFormats.TopLeft);

gfx.DrawString(app.BirthDate, tableContentFont, XBrushes.Black,

new XRect(margin + col1Width + col2Width, yPosition, col3Width, 20),

XStringFormats.TopLeft);

yPosition += 20;

// Линия между записями

gfx.DrawLine(new XPen(XColors.LightGray, 0.5),

margin, yPosition,

margin + tableWidth, yPosition);

yPosition += 5;

}

}

else

{

gfx.DrawString("На сегодня записей нет", tableContentFont, XBrushes.Black,

new XRect(margin, yPosition, tableWidth, 20),

XStringFormats.TopCenter);

yPosition += 25;

}

// Футер

yPosition += 20;

gfx.DrawLine(new XPen(XColors.Black, 0.5),

margin, yPosition,

margin + tableWidth, yPosition);

yPosition += 10;

gfx.DrawString($"Адрес: {clinicAddress}", footerFont, XBrushes.Black,

new XRect(margin, yPosition, tableWidth, 20),

XStringFormats.TopLeft);

yPosition += 15;

gfx.DrawString("Телефон регистратуры: +7 (347) 201-10-86", footerFont, XBrushes.Black,

new XRect(margin, yPosition, tableWidth, 20),

XStringFormats.TopLeft);

yPosition += 15;

// Конфиденциальность

gfx.DrawString("В целях соблюдения конфиденциальности указаны только первые буквы Ф.И.", smallInfoFont, XBrushes.Gray,

new XRect(margin, yPosition, tableWidth, 20),

XStringFormats.TopLeft);

document.Save(filePath);

Application.Current.Dispatcher.Invoke(() =>

{

MessageBox.Show(

$"PDF с расписанием на сегодня сохранён:\n{filePath}",

"Сохранение завершено",

MessageBoxButton.OK,

MessageBoxImage.Information);

});

}

}

catch (Exception ex)

{

Application.Current.Dispatcher.Invoke(() =>

{

MessageBox.Show(

$"Ошибка при создании PDF: {ex.Message}",

"Ошибка",

MessageBoxButton.OK,

MessageBoxImage.Error);

});

}

}

public static List<(string Time, string PatientName, string BirthDate)> GetDoctorAppointmentsWithBirthDate(string doctorFullName)

{

using var db = new DbAppontmentClinikContext();

var nameParts = doctorFullName.Split(' ');

string surname = nameParts[0];

string name = nameParts[1];

string patronymic = nameParts.Length > 2 ? nameParts[2] : null;

var doctor = db.Doctors

.Where(d => d.SurnameDoctor == surname &&

d.NameDoctor == name &&

(patronymic == null || d.PatronymicDoctor == patronymic))

.FirstOrDefault();

if (doctor == null)

{

MessageBox.Show("Врач не найден");

return new List<(string, string, string)>();

}

var today = DateOnly.FromDateTime(DateTime.Today);

var appointments = db.Appointments

.Where(a => a.IdDoctor == doctor.IdDoctor &&

a.DateAppointment == today)

.Include(a => a.IdMedCardNavigation)

.OrderBy(a => a.TimeAppointment)

.ToList();

var result = new List<(string Time, string PatientName, string BirthDate)>();

foreach (var app in appointments)

{

if (app.IdMedCardNavigation != null && app.TimeAppointment.HasValue)

{

string time = app.TimeAppointment.Value.ToString("HH\\:mm");

string patientName = $"{app.IdMedCardNavigation.SurnameUsers[0]}. {app.IdMedCardNavigation.NameUsers[0]}.";

string birthDate = app.IdMedCardNavigation.DateBirth?.ToString("dd.MM.yyyy") ?? "не указана";

result.Add((time, patientName, birthDate));

}

}

return result;

}

}

AboutClinik.cs

public partial class AboutClinik : Page

{

public AboutClinik()

{

ClinikMainWindow mainWindow = Application.Current.MainWindow as ClinikMainWindow;

mainWindow.Title = "О клинике";

InitializeComponent();

}

}

AddRedactDoctors.cs

public partial class AddRedactDoctors : Page

{

private User \_user;

private Doctor \_doctor;

private bool \_isEditMode;

private List<Specialty> \_specialties;

private List<string> \_statuses = new List<string>

{

"По графику",

"В отпуске",

"На больничном",

};

public AddRedactDoctors(User user)

{

InitializeComponent();

\_user = user;

\_isEditMode = false;

LoadData();

this.DataContext = this;

}

public AddRedactDoctors(Doctor doctor, User user)

{

InitializeComponent();

\_user = user;

\_doctor = doctor;

\_isEditMode = true;

LoadData();

FillDoctorData();

this.DataContext = this;

}

private void LoadData()

{

try

{

using (var db = new DbAppontmentClinikContext())

{

\_specialties = db.Specialties.ToList();

cmbSpecialty.ItemsSource = \_specialties;

cmbSpecialty.DisplayMemberPath = "NameSpecialty";

cmbSpecialty.SelectedValuePath = "IdSpecialty";

}

cmbStatus.ItemsSource = \_statuses;

}

catch (Exception ex)

{

MessageBox.Show($"Ошибка при загрузке данных: {ex.Message}", "Ошибка", MessageBoxButton.OK, MessageBoxImage.Error);

}

}

private void FillDoctorData()

{

if (\_doctor != null)

{

txtSurname.Text = \_doctor.SurnameDoctor;

txtName.Text = \_doctor.NameDoctor;

txtPatronymic.Text = \_doctor.PatronymicDoctor;

cmbSpecialty.SelectedValue = \_doctor.IdSpecialty;

txtEmail.Text = \_doctor.EmailDoctor;

txtPhone.Text = \_doctor.PhoneNumberDoctor;

txtExperience.Text = \_doctor.MedicalExperience?.ToString();

txtCabinet.Text = \_doctor.CabinetNumber;

cmbStatus.SelectedItem = \_doctor.StatusWork;

if (!string.IsNullOrEmpty(\_doctor.IconDoctor))

{

string imagePath = \_doctor.DisplayIconDoctor;

if (File.Exists(imagePath))

{

BitmapImage bitmap = new BitmapImage();

bitmap.BeginInit();

bitmap.UriSource = new Uri(imagePath);

bitmap.EndInit();

doctorImage.Source = bitmap;

}

}

}

}

private void btnCancel\_Click(object sender, RoutedEventArgs e)

{

if (Application.Current.MainWindow is ClinikMainWindow mainWindow)

{

mainWindow.mainframe.Navigate(new Admin\_panel(\_user));

}

}

private void btnSave\_Click(object sender, RoutedEventArgs e)

{

try

{

if (string.IsNullOrWhiteSpace(txtSurname.Text) || string.IsNullOrWhiteSpace(txtName.Text) || cmbSpecialty.SelectedItem == null || string.IsNullOrWhiteSpace(txtEmail.Text) || string.IsNullOrWhiteSpace(txtPhone.Text))

{

MessageBox.Show("Пожалуйста, заполните все обязательные поля!", "Ошибка", MessageBoxButton.OK, MessageBoxImage.Warning);

return;

}

if (!int.TryParse(txtExperience.Text, out int experience) || experience < 0)

{

MessageBox.Show("Пожалуйста, введите корректный стаж работы (целое положительное число)!", "Ошибка", MessageBoxButton.OK, MessageBoxImage.Warning);

return;

}

using (var db = new DbAppontmentClinikContext())

{

if (\_isEditMode)

{

Doctor doctor = db.Doctors.FirstOrDefault(d => d.IdDoctor == \_doctor.IdDoctor);

if (doctor == null)

{

MessageBox.Show("Врач не найден в базе данных!", "Ошибка",

MessageBoxButton.OK, MessageBoxImage.Error);

return;

}

doctor.SurnameDoctor = txtSurname.Text;

doctor.NameDoctor = txtName.Text;

doctor.PatronymicDoctor = txtPatronymic.Text;

doctor.IdSpecialty = (int)cmbSpecialty.SelectedValue;

doctor.EmailDoctor = txtEmail.Text;

doctor.PhoneNumberDoctor = txtPhone.Text;

doctor.MedicalExperience = experience;

doctor.CabinetNumber = txtCabinet.Text;

doctor.StatusWork = cmbStatus.SelectedItem?.ToString();

db.SaveChanges();

MessageBox.Show("Данные врача успешно обновлены!", "Успех",MessageBoxButton.OK, MessageBoxImage.Information);

}

else

{

var newdoctor = new Doctor

{

SurnameDoctor = txtSurname.Text,

NameDoctor = txtName.Text,

PatronymicDoctor = txtPatronymic.Text,

IdSpecialty = (int)cmbSpecialty.SelectedValue,

EmailDoctor = txtEmail.Text,

PhoneNumberDoctor = txtPhone.Text,

MedicalExperience = experience,

CabinetNumber = txtCabinet.Text,

StatusWork = cmbStatus.SelectedItem?.ToString(),

IconDoctor = "default\_doctor.png",

};

db.Doctors.Add(newdoctor);

db.SaveChanges();

MessageBox.Show("Новый врач успешно добавлен!", "Успех",MessageBoxButton.OK, MessageBoxImage.Information);

}

if (Application.Current.MainWindow is ClinikMainWindow mainWindow)

{

mainWindow.mainframe.Navigate(new Admin\_panel(\_user));

}

}

}

catch (Exception ex)

{

MessageBox.Show($"Ошибка при сохранении данных: {ex.Message}", "Ошибка",MessageBoxButton.OK, MessageBoxImage.Error);

}

}

}

AddRedactSched.cs

public partial class AddRedactSched : Page

{

private User \_currentUser;

private Schedule \_editingSchedule;

private bool \_isEditMode;

public AddRedactSched(User currentUser)

{

InitializeComponent();

\_currentUser = currentUser;

\_isEditMode = false;

LoadDoctors();

InitializeDayWeekComboBox();

this.DataContext = this;

}

public AddRedactSched(Schedule scheduleToEdit, User currentUser)

{

InitializeComponent();

\_currentUser = currentUser;

\_editingSchedule = scheduleToEdit;

\_isEditMode = true;

LoadDoctors();

InitializeDayWeekComboBox();

FillScheduleData();

this.DataContext = this;

}

private void InitializeDayWeekComboBox()

{

cmbDayWeek.Items.Clear();

cmbDayWeek.Items.Add(new ComboBoxItem() { Content = "Пн", Tag = "Пн" });

cmbDayWeek.Items.Add(new ComboBoxItem() { Content = "Вт", Tag = "Вт" });

cmbDayWeek.Items.Add(new ComboBoxItem() { Content = "Ср", Tag = "Ср" });

cmbDayWeek.Items.Add(new ComboBoxItem() { Content = "Чт", Tag = "Чт" });

cmbDayWeek.Items.Add(new ComboBoxItem() { Content = "Пт", Tag = "Пт" });

cmbDayWeek.Items.Add(new ComboBoxItem() { Content = "Сб", Tag = "Сб" });

cmbDayWeek.Items.Add(new ComboBoxItem() { Content = "Вс", Tag = "Вс" });

}

private void LoadDoctors()

{

try

{

using (var db = new DbAppontmentClinikContext())

{

var doctors = db.Doctors

.Include(d => d.IdSpecialtyNavigation)

.Select(d => new

{

d.IdDoctor,

DisplayText = $"{d.SurnameDoctor} {d.NameDoctor[0]}.{(!string.IsNullOrEmpty(d.PatronymicDoctor) ? d.PatronymicDoctor[0] + "." : "")} ({d.IdSpecialtyNavigation.NameSpecialty})"

})

.ToList();

cmbDoctor.DisplayMemberPath = "DisplayText";

cmbDoctor.SelectedValuePath = "IdDoctor";

cmbDoctor.ItemsSource = doctors;

}

}

catch (Exception ex)

{

MessageBox.Show($"Ошибка при загрузке списка врачей: {ex.Message}", "Ошибка",

MessageBoxButton.OK, MessageBoxImage.Error);

}

}

private void FillScheduleData()

{

if (\_editingSchedule != null)

{

cmbDoctor.SelectedValue = \_editingSchedule.IdDoctor;

foreach (ComboBoxItem item in cmbDayWeek.Items)

{

if (item.Tag.ToString() == \_editingSchedule.DayWeek)

{

cmbDayWeek.SelectedItem = item;

break;

}

}

timeStart.Text = \_editingSchedule.TimeStart.ToString("HH:mm:ss");

timeEnd.Text = \_editingSchedule.TimeEnd.ToString("HH:mm:ss");

btnSave.Content = "Сохранить изменения";

Title = "Редактирование графика работы";

}

}

private bool CheckExistingSchedule(int doctorId, string dayOfWeek, int? excludeScheduleId = null)

{

using (var db = new DbAppontmentClinikContext())

{

var query = db.Schedules

.Where(s => s.IdDoctor == doctorId && s.DayWeek == dayOfWeek);

if (excludeScheduleId.HasValue)

{

query = query.Where(s => s.IdSсhedule != excludeScheduleId.Value);

}

return query.Any();

}

}

private void btnCancel\_Click(object sender, RoutedEventArgs e)

{

if (Application.Current.MainWindow is ClinikMainWindow mainWindow)

{

mainWindow.mainframe.Navigate(new Admin\_panel(\_currentUser));

}

}

private void btnSave\_Click(object sender, RoutedEventArgs e)

{

try

{

if (cmbDoctor.SelectedValue == null)

{

MessageBox.Show("Пожалуйста, выберите врача!", "Ошибка",

MessageBoxButton.OK, MessageBoxImage.Warning);

return;

}

if (cmbDayWeek.SelectedItem == null)

{

MessageBox.Show("Пожалуйста, выберите день недели!", "Ошибка",

MessageBoxButton.OK, MessageBoxImage.Warning);

return;

}

if (string.IsNullOrWhiteSpace(timeStart.Text) || string.IsNullOrWhiteSpace(timeEnd.Text))

{

MessageBox.Show("Пожалуйста, укажите время начала и окончания работы!", "Ошибка",

MessageBoxButton.OK, MessageBoxImage.Warning);

return;

}

if (!TimeSpan.TryParse(timeStart.Text, out TimeSpan startTime) ||

!TimeSpan.TryParse(timeEnd.Text, out TimeSpan endTime))

{

MessageBox.Show("Пожалуйста, укажите время в корректном формате (HH:mm:ss)!", "Ошибка",

MessageBoxButton.OK, MessageBoxImage.Warning);

return;

}

if (startTime >= endTime)

{

MessageBox.Show("Время окончания должно быть позже времени начала!", "Ошибка",

MessageBoxButton.OK, MessageBoxImage.Warning);

return;

}

int doctorId = (int)cmbDoctor.SelectedValue;

string dayOfWeek = ((ComboBoxItem)cmbDayWeek.SelectedItem).Tag.ToString();

if (!\_isEditMode && CheckExistingSchedule(doctorId, dayOfWeek))

{

MessageBox.Show("У выбранного врача уже есть график в этот день недели!\n" +

"Пожалуйста, выберите другой день или отредактируйте существующий график.",

"Ошибка", MessageBoxButton.OK, MessageBoxImage.Warning);

return;

}

using (var db = new DbAppontmentClinikContext())

{

if (\_isEditMode)

{

if (CheckExistingSchedule(doctorId, dayOfWeek, \_editingSchedule.IdSсhedule))

{

MessageBox.Show("У выбранного врача уже есть график в этот день недели!\n" +

"Пожалуйста, выберите другой день.",

"Ошибка", MessageBoxButton.OK, MessageBoxImage.Warning);

return;

}

var schedule = db.Schedules.Find(\_editingSchedule.IdSсhedule);

if (schedule == null)

{

MessageBox.Show("График работы не найден в базе данных!", "Ошибка",

MessageBoxButton.OK, MessageBoxImage.Error);

return;

}

schedule.IdDoctor = doctorId;

schedule.DayWeek = dayOfWeek;

schedule.TimeStart = TimeOnly.FromTimeSpan(startTime);

schedule.TimeEnd = TimeOnly.FromTimeSpan(endTime);

db.SaveChanges();

MessageBox.Show("График работы успешно обновлен!", "Успех",

MessageBoxButton.OK, MessageBoxImage.Information);

}

else

{

var newSchedule = new Schedule

{

IdDoctor = doctorId,

DayWeek = dayOfWeek,

TimeStart = TimeOnly.FromTimeSpan(startTime),

TimeEnd = TimeOnly.FromTimeSpan(endTime)

};

db.Schedules.Add(newSchedule);

db.SaveChanges();

MessageBox.Show("Новый график работы успешно добавлен!", "Успех",

MessageBoxButton.OK, MessageBoxImage.Information);

}

if (Application.Current.MainWindow is ClinikMainWindow mainWindow)

{

mainWindow.mainframe.Navigate(new Admin\_panel(\_currentUser));

}

}

}

catch (Exception ex)

{

MessageBox.Show($"Ошибка при сохранении данных: {ex.Message}", "Ошибка",

MessageBoxButton.OK, MessageBoxImage.Error);

}

}

}

AddRedactSpecialities.cs

public partial class AddRedactSpecialities : Page

{

private User \_currentUser;

private Specialty \_editingSpecialty;

private bool \_isEditMode;

public AddRedactSpecialities(User currentUser)

{

InitializeComponent();

\_currentUser = currentUser;

\_isEditMode = false;

this.DataContext = this;

}

public AddRedactSpecialities(Specialty specialtyToEdit, User currentUser)

{

InitializeComponent();

\_currentUser = currentUser;

\_editingSpecialty = specialtyToEdit;

\_isEditMode = true;

FillSpecialtyData();

this.DataContext = this;

}

private void FillSpecialtyData()

{

if (\_editingSpecialty != null)

{

txtName.Text = \_editingSpecialty.NameSpecialty;

txtTimeAccept.Text = \_editingSpecialty.TimeAccept?.ToString();

btnSave.Content = "Сохранить изменения";

Title = "Редактирование специальности";

}

}

private void btnCancel\_Click(object sender, RoutedEventArgs e)

{

if (Application.Current.MainWindow is ClinikMainWindow mainWindow)

{

mainWindow.mainframe.Navigate(new Admin\_panel(\_currentUser));

}

}

private void btnSave\_Click(object sender, RoutedEventArgs e)

{

try

{

if (string.IsNullOrWhiteSpace(txtName.Text))

{

MessageBox.Show("Пожалуйста, укажите название специальности!", "Ошибка",

MessageBoxButton.OK, MessageBoxImage.Warning);

return;

}

if (!int.TryParse(txtTimeAccept.Text, out int timeAccept) || timeAccept <= 0)

{

MessageBox.Show("Пожалуйста, укажите корректное время приема (положительное число)!", "Ошибка",

MessageBoxButton.OK, MessageBoxImage.Warning);

return;

}

using (var db = new DbAppontmentClinikContext())

{

if (\_isEditMode)

{

var specialty = db.Specialties.Find(\_editingSpecialty.IdSpecialty);

if (specialty == null)

{

MessageBox.Show("Специальность не найдена в базе данных!", "Ошибка",

MessageBoxButton.OK, MessageBoxImage.Error);

return;

}

specialty.NameSpecialty = txtName.Text;

specialty.TimeAccept = timeAccept;

specialty.IconSpecialty = null;

db.SaveChanges();

MessageBox.Show("Данные специальности успешно обновлены!", "Успех",

MessageBoxButton.OK, MessageBoxImage.Information);

}

else

{

var newSpecialty = new Specialty

{

NameSpecialty = txtName.Text,

TimeAccept = timeAccept,

IconSpecialty = null

};

db.Specialties.Add(newSpecialty);

db.SaveChanges();

MessageBox.Show("Новая специальность успешно добавлена!", "Успех",

MessageBoxButton.OK, MessageBoxImage.Information);

}

if (Application.Current.MainWindow is ClinikMainWindow mainWindow)

{

mainWindow.mainframe.Navigate(new Admin\_panel(\_currentUser));

}

}

}

catch (Exception ex)

{

MessageBox.Show($"Ошибка при сохранении данных: {ex.Message}", "Ошибка",

MessageBoxButton.OK, MessageBoxImage.Error);

}

}

}

AddRedactUsers.cs

public partial class AddRedactUsers : Page

{

private User \_currentUser;

private User \_editingUser;

private bool \_isEditMode;

private List<string> \_roles = new List<string> { "Администратор", "Пользователь" };

public AddRedactUsers(User currentUser)

{

InitializeComponent();

\_currentUser = currentUser;

\_isEditMode = false;

LoadData();

this.DataContext = this;

}

public AddRedactUsers(User userToEdit, User currentUser)

{

InitializeComponent();

\_currentUser = currentUser;

\_editingUser = userToEdit;

\_isEditMode = true;

LoadData();

FillUserData();

this.DataContext = this;

}

private void LoadData()

{

try

{

cmbRole.ItemsSource = \_roles;

if (!\_isEditMode)

{

cmbRole.SelectedIndex = 1;

dpBirthDate.Text = DateTime.Now.ToString("dd/MM/yyyy");

}

}

catch (Exception ex)

{

MessageBox.Show($"Ошибка при загрузке данных: {ex.Message}", "Ошибка",

MessageBoxButton.OK, MessageBoxImage.Error);

}

}

private void FillUserData()

{

if (\_editingUser != null)

{

txtSurname.Text = \_editingUser.SurnameUsers;

txtName.Text = \_editingUser.NameUsers;

txtEmail.Text = \_editingUser.EmailUsers;

if (\_editingUser.DateBirth.HasValue)

{

dpBirthDate.Text = \_editingUser.DateBirth.Value.ToString("dd/MM/yyyy");

}

cmbRole.SelectedItem = \_editingUser.RoleIdUsers;

txtPhone.Text = \_editingUser.PhoneNumber;

txtMedicalPolicy.Text = \_editingUser.MedicalPolicy;

txtPassport.Text = \_editingUser.PassportNumber;

btnSave.Content = "Сохранить изменения";

Title = "Редактирование пользователя";

}

}

private void btnCancel\_Click(object sender, RoutedEventArgs e)

{

if (Application.Current.MainWindow is ClinikMainWindow mainWindow)

{

mainWindow.mainframe.Navigate(new Admin\_panel(\_currentUser));

}

}

private void btnSave\_Click(object sender, RoutedEventArgs e)

{

try

{

if (string.IsNullOrWhiteSpace(txtSurname.Text) ||

string.IsNullOrWhiteSpace(txtName.Text) ||

string.IsNullOrWhiteSpace(txtEmail.Text) ||

cmbRole.SelectedItem == null ||

string.IsNullOrWhiteSpace(dpBirthDate.Text) ||

!IsValidDate(dpBirthDate.Text))

{

MessageBox.Show("Пожалуйста, заполните все обязательные поля корректно!", "Ошибка",

MessageBoxButton.OK, MessageBoxImage.Warning);

return;

}

if (!DateTime.TryParseExact(dpBirthDate.Text, "dd/MM/yyyy", null, System.Globalization.DateTimeStyles.None, out DateTime birthDate))

{

MessageBox.Show("Пожалуйста, введите корректную дату рождения в формате ДД/ММ/ГГГГ!", "Ошибка",

MessageBoxButton.OK, MessageBoxImage.Warning);

return;

}

using (var db = new DbAppontmentClinikContext())

{

if (\_isEditMode)

{

var user = db.Users.FirstOrDefault(u => u.IdMedCard == \_editingUser.IdMedCard);

if (user == null)

{

MessageBox.Show("Пользователь не найден в базе данных!", "Ошибка",

MessageBoxButton.OK, MessageBoxImage.Error);

return;

}

user.SurnameUsers = txtSurname.Text;

user.NameUsers = txtName.Text;

user.EmailUsers = txtEmail.Text;

user.RoleIdUsers = cmbRole.SelectedItem.ToString();

user.PhoneNumber = txtPhone.Text;

user.MedicalPolicy = txtMedicalPolicy.Text;

user.PassportNumber = txtPassport.Text;

user.DateBirth = DateOnly.FromDateTime(birthDate);

db.SaveChanges();

MessageBox.Show("Данные пользователя успешно обновлены!", "Успех",

MessageBoxButton.OK, MessageBoxImage.Information);

}

else

{

var newUser = new User

{

SurnameUsers = txtSurname.Text,

NameUsers = txtName.Text,

EmailUsers = txtEmail.Text,

RoleIdUsers = cmbRole.SelectedItem.ToString(),

PhoneNumber = txtPhone.Text,

MedicalPolicy = txtMedicalPolicy.Text,

PassportNumber = txtPassport.Text,

DateBirth = DateOnly.FromDateTime(birthDate)

};

db.Users.Add(newUser);

db.SaveChanges();

MessageBox.Show("Новый пользователь успешно добавлен!", "Успех",

MessageBoxButton.OK, MessageBoxImage.Information);

}

if (Application.Current.MainWindow is ClinikMainWindow mainWindow)

{

mainWindow.mainframe.Navigate(new Admin\_panel(\_currentUser));

}

}

}

catch (Exception ex)

{

MessageBox.Show($"Ошибка при сохранении данных: {ex.Message}", "Ошибка",

MessageBoxButton.OK, MessageBoxImage.Error);

}

}

private bool IsValidDate(string dateString)

{

return DateTime.TryParseExact(dateString, "dd/MM/yyyy", null, System.Globalization.DateTimeStyles.None, out \_);

}

}

Admin\_panel.cs

public partial class Admin\_panel : Page

{

private User \_user;

private Specialty \_specialty;

public int total\_items;

public int total\_pages;

public int current\_page = 1;

public ObservableCollection<Doctor> doctors;

public ObservableCollection<Appointment> appointment;

public ObservableCollection<Specialty> specialties;

public ObservableCollection<Schedule> schedules;

public List<User> users;

public Admin\_panel(User user)

{

InitializeComponent();

ClinikMainWindow mainWindow = Application.Current.MainWindow as ClinikMainWindow;

mainWindow.Title = "Администраторская панель";

\_user = user;

DataContext = user;

getData();

LoadData();

datagrid.SelectionChanged += Datagrid\_SelectionChanged;

}

private void Datagrid\_SelectionChanged(object sender, SelectionChangedEventArgs e)

{

// Показываем кнопку печати только для врачей и когда есть выделенный элемент

btnPrint.Visibility = (cboxMode.SelectedIndex == 0 && datagrid.SelectedItem != null)

? Visibility.Visible

: Visibility.Collapsed;

}

private void getData()

{

using var db = new DbAppontmentClinikContext();

doctors = new ObservableCollection<Doctor>(db.Doctors.ToList());

users = db.Users.ToList();

appointment = new ObservableCollection<Appointment>(db.Appointments.ToList());

specialties = new ObservableCollection<Specialty>(db.Specialties.ToList());

schedules = new ObservableCollection<Schedule>(db.Schedules.ToList());

}

public void LoadDoctors()

{

using var db = new DbAppontmentClinikContext();

if (doctors is not null)

{

total\_items = doctors.Count();

total\_pages = (int)Math.Ceiling((double)total\_items / 10);

if (current\_page > total\_pages && total\_pages > 0)

{

current\_page = total\_pages;

}

else if (total\_pages == 0){ current\_page = 1; }

var doctorsfiltered = doctors.Select(d => new

{

d.IdDoctor,

d.IdSpecialty,

SpecialtyName = db.Specialties.FirstOrDefault(s => s.IdSpecialty == d.IdSpecialty)?.NameSpecialty,

d.SurnameDoctor,

d.NameDoctor,

d.PatronymicDoctor,

d.EmailDoctor,

d.PhoneNumberDoctor,

d.MedicalExperience,

d.CabinetNumber,

d.StatusWork,

d.IconDoctor

}).ToList();

var pagedGames = doctorsfiltered.Skip((current\_page - 1) \* 10).Take(10).ToList();

currPageTblock.Text = $"{current\_page}";

datagrid.ItemsSource = pagedGames;

btnPrint.Visibility = (datagrid.SelectedItem != null)

? Visibility.Visible

: Visibility.Collapsed;

}

}

public void LoadUsers()

{

using var db = new DbAppontmentClinikContext();

if (users is not null)

{

total\_items = users.Count();

total\_pages = (int)Math.Ceiling((double)total\_items / 10);

if (current\_page > total\_pages && total\_pages > 0)

{

current\_page = total\_pages;

}

else if (total\_pages == 0)

{

current\_page = 1;

}

var usersfiltered = users.Select(g => new

{

g.IdMedCard,

g.EmailUsers,

g.RoleIdUsers,

g.SurnameUsers,

g.NameUsers,

g.DateBirth,

g.MedicalPolicy,

g.PassportNumber,

g.PhoneNumber,

}).ToList();

var pagedGames = usersfiltered.Skip((current\_page - 1) \* 10).Take(10).ToList();

currPageTblock.Text = $"{current\_page}";

datagrid.ItemsSource = pagedGames;

}

}

private void LoadAppointment()

{

using var db = new DbAppontmentClinikContext();

if (appointment is not null)

{

total\_items = appointment.Count();

total\_pages = (int)Math.Ceiling((double)total\_items / 10);

if (current\_page > total\_pages && total\_pages > 0)

{

current\_page = total\_pages;

}

else if (total\_pages == 0)

{

current\_page = 1;

}

var appfiltered = appointment.Select(g => new

{

g.IdDoctor,

DoctorName = db.Doctors.FirstOrDefault(d => d.IdDoctor == g.IdDoctor)?.SurnameDoctor + " " +

db.Doctors.FirstOrDefault(d => d.IdDoctor == g.IdDoctor)?.NameDoctor.Substring(0, 1) + "." +

(string.IsNullOrEmpty(db.Doctors.FirstOrDefault(d => d.IdDoctor == g.IdDoctor)?.PatronymicDoctor) ? "" :

db.Doctors.FirstOrDefault(d => d.IdDoctor == g.IdDoctor)?.PatronymicDoctor.Substring(0, 1) + "."),

PatientName = db.Users.FirstOrDefault(u => u.IdMedCard == g.IdMedCard)?.SurnameUsers + " " + db.Users.FirstOrDefault(u => u.IdMedCard == g.IdMedCard)?.NameUsers,

g.DateAppointment,

g.TimeAppointment,

g.StatusAppointment

}).ToList();

var pagedGames = appfiltered.Skip((current\_page - 1) \* 10).Take(10).ToList();

currPageTblock.Text = $"{current\_page}";

datagrid.ItemsSource = pagedGames;

}

}

private void LoadSpecialties()

{

using var db = new DbAppontmentClinikContext();

if (specialties is not null)

{

total\_items = specialties.Count();

total\_pages = (int)Math.Ceiling((double)total\_items / 10);

if (current\_page > total\_pages && total\_pages > 0)

{

current\_page = total\_pages;

}

else if (total\_pages == 0)

{

current\_page = 1;

}

var specialtiesFiltered = specialties.Select(s => new

{

s.IdSpecialty,

s.NameSpecialty,

s.TimeAccept,

s.IconSpecialty

}).ToList();

var pagedItems = specialtiesFiltered.Skip((current\_page - 1) \* 10).Take(10).ToList();

currPageTblock.Text = $"{current\_page}";

datagrid.ItemsSource = pagedItems;

}

}

private void LoadSchedules()

{

using var db = new DbAppontmentClinikContext();

if (schedules is not null)

{

total\_items = schedules.Count();

total\_pages = (int)Math.Ceiling((double)total\_items / 10);

if (current\_page > total\_pages && total\_pages > 0)

{

current\_page = total\_pages;

}

else if (total\_pages == 0)

{

current\_page = 1;

}

var schedulesFiltered = schedules.Select(s => new

{

s.IdSсhedule,

s.IdDoctor,

DoctorName = db.Doctors.FirstOrDefault(d => d.IdDoctor == s.IdDoctor)?.SurnameDoctor + " " +

db.Doctors.FirstOrDefault(d => d.IdDoctor == s.IdDoctor)?.NameDoctor.Substring(0, 1) + "." +

(string.IsNullOrEmpty(db.Doctors.FirstOrDefault(d => d.IdDoctor == s.IdDoctor)?.PatronymicDoctor) ? "" :

db.Doctors.FirstOrDefault(d => d.IdDoctor == s.IdDoctor)?.PatronymicDoctor.Substring(0, 1) + "."),

s.DayWeek,

s.TimeStart,

s.TimeEnd

}).ToList();

var pagedItems = schedulesFiltered.Skip((current\_page - 1) \* 10).Take(10).ToList();

currPageTblock.Text = $"{current\_page}";

datagrid.ItemsSource = pagedItems;

}

}

public void LoadData()

{

var showButtons = new[] { 0, 1, 3, 4 }.Contains(cboxMode.SelectedIndex);

if (btnAdd != null)

{

btnAdd.Visibility = showButtons ? Visibility.Visible : Visibility.Collapsed;

btnEdit.Visibility = showButtons ? Visibility.Visible : Visibility.Collapsed;

btnDelete.Visibility = showButtons ? Visibility.Visible : Visibility.Collapsed;

}

switch (cboxMode.SelectedIndex)

{

case 0: LoadDoctors(); break;

case 1: LoadUsers(); break;

case 2: LoadAppointment(); break;

case 3: LoadSchedules(); break;

case 4: LoadSpecialties(); break;

}

}

private void arrowLeft\_MouseLeftButtonDown(object sender, MouseButtonEventArgs e)

{

if (current\_page > 1){ current\_page--; LoadData(); }

}

private void arrowRight\_MouseLeftButtonDown(object sender, MouseButtonEventArgs e)

{

int totalpages = 0;

switch (cboxMode.SelectedIndex)

{

case 0: // Врачи

totalpages = (int)Math.Ceiling((double)doctors.Count / 10);

break;

case 1: // Пользователи

totalpages = (int)Math.Ceiling((double)users.Count / 10);

break;

case 2: // Записи на приём

totalpages = (int)Math.Ceiling((double)appointment.Count / 10);

break;

case 3: // График работы

totalpages = (int)Math.Ceiling((double)schedules.Count / 10);

break;

case 4: // Специальности

totalpages = (int)Math.Ceiling((double)specialties.Count / 10);

break;

}

if (current\_page < totalpages){current\_page++; LoadData(); }

}

private void cboxMode\_SelectionChanged(object sender, SelectionChangedEventArgs e)

{

LoadData();

}

private void btnAdd\_Click(object sender, RoutedEventArgs e)

{

if (Application.Current.MainWindow is ClinikMainWindow mainWindow)

{

switch (cboxMode.SelectedIndex)

{

case 0: // Врачи

mainWindow.mainframe.NavigationService.Navigate(new AddRedactDoctors(\_user));

break;

case 1: // Пользователи

mainWindow.mainframe.NavigationService.Navigate(new AddRedactUsers(\_user));

break;

case 4: // Специальности

mainWindow.mainframe.NavigationService.Navigate(new AddRedactSpecialities(\_user));

break;

case 3: // График работы

mainWindow.mainframe.NavigationService.Navigate(new AddRedactSched(\_user));

break;

}

}

}

private void btnEdit\_Click(object sender, RoutedEventArgs e)

{

if (Application.Current.MainWindow is ClinikMainWindow mainWindow && datagrid.SelectedItem != null)

{

try

{

dynamic selectedItem = datagrid.SelectedItem;

switch (cboxMode.SelectedIndex)

{

case 0: // Врачи

var doctorIdProperty = selectedItem.GetType().GetProperty("IdDoctor");

if (doctorIdProperty != null)

{

object idValue = doctorIdProperty.GetValue(selectedItem);

if (idValue != null && int.TryParse(idValue.ToString(), out int doctorId))

{

var doctor = doctors.FirstOrDefault(d => d.IdDoctor == doctorId);

if (doctor != null)

{

mainWindow.mainframe.Navigate(new AddRedactDoctors(doctor, \_user));

}

else

{

MessageBox.Show("Врач не найден в базе данных", "Ошибка",

MessageBoxButton.OK, MessageBoxImage.Warning);

}

}

}

break;

case 1: // Пользователи

var userIdProperty = selectedItem.GetType().GetProperty("IdMedCard");

if (userIdProperty != null)

{

object idValue = userIdProperty.GetValue(selectedItem);

if (idValue != null && int.TryParse(idValue.ToString(), out int userId))

{

using (var db = new DbAppontmentClinikContext())

{

var user = db.Users.FirstOrDefault(u => u.IdMedCard == userId);

if (user != null)

{

mainWindow.mainframe.Navigate(new AddRedactUsers(user, \_user));

}

else

{

MessageBox.Show("Пользователь не найден в базе данных", "Ошибка",

MessageBoxButton.OK, MessageBoxImage.Warning);

}

}

}

}

break;

case 4: // Специальности

var specialityIdProperty = selectedItem.GetType().GetProperty("IdSpecialty");

if (specialityIdProperty != null)

{

object idValue = specialityIdProperty.GetValue(selectedItem);

if (idValue != null && int.TryParse(idValue.ToString(), out int specialityId))

{

using (var db = new DbAppontmentClinikContext())

{

var speciality = db.Specialties.FirstOrDefault(s => s.IdSpecialty == specialityId);

if (speciality != null)

{

mainWindow.mainframe.Navigate(new AddRedactSpecialities(speciality, \_user));

}

else

{

MessageBox.Show("Специальность не найдена в базе данных", "Ошибка",

MessageBoxButton.OK, MessageBoxImage.Warning);

}

}

}

}

break;

case 3: // График работы

var scheduleIdProperty = selectedItem.GetType().GetProperty("IdSсhedule");

if (scheduleIdProperty != null)

{

object idValue = scheduleIdProperty.GetValue(selectedItem);

if (idValue != null && int.TryParse(idValue.ToString(), out int scheduleId))

{

using (var db = new DbAppontmentClinikContext())

{

var schedule = db.Schedules

.Include(s => s.IdDoctorNavigation)

.FirstOrDefault(s => s.IdSсhedule == scheduleId);

if (schedule != null)

{

mainWindow.mainframe.Navigate(new AddRedactSched(schedule, \_user));

}

else

{

MessageBox.Show("График работы не найден в базе данных", "Ошибка",

MessageBoxButton.OK, MessageBoxImage.Warning);

}

}

}

}

break;

default:

MessageBox.Show("Режим редактирования не поддерживается для этого типа данных", "Информация",

MessageBoxButton.OK, MessageBoxImage.Information);

break;

}

}

catch (Exception ex)

{

MessageBox.Show($"Ошибка при редактировании: {ex.Message}", "Ошибка",

MessageBoxButton.OK, MessageBoxImage.Error);

}

}

}

private void btnPrint\_Click(object sender, RoutedEventArgs e)

{

if (datagrid.SelectedItem != null && cboxMode.SelectedIndex == 0)

{

try

{

dynamic selectedItem = datagrid.SelectedItem;

var surname = selectedItem.SurnameDoctor as string;

var name = selectedItem.NameDoctor as string;

var patronymic = selectedItem.PatronymicDoctor as string;

string fullName = $"{surname} {name} {patronymic}".Trim();

PDF.CreateTodaysAppointmentsPdf(fullName);

}

catch (Exception ex)

{

MessageBox.Show($"Ошибка при получении данных врача: {ex.Message}",

"Ошибка", MessageBoxButton.OK, MessageBoxImage.Error);

}

}

}

}

AdminAuth.cs

public partial class AdminAuth : Page

{

private string \_lastVerificationCode;

User currUser;

public AdminAuth()

{

InitializeComponent();

ClinikMainWindow mainWindow = Application.Current.MainWindow as ClinikMainWindow;

mainWindow.Title = "Вход в админ-панель";

}

private void code\_button\_Click(object sender, RoutedEventArgs e)

{

if (string.IsNullOrWhiteSpace(email\_txtbx.Text) || !email\_txtbx.Text.Contains("@"))

{

MessageBox.Show("Введите корректный email адрес");

return;

}

using var db = new DbAppontmentClinikContext();

var admin = db.Users.FirstOrDefault(u => u.EmailUsers == email\_txtbx.Text && u.RoleIdUsers == "Администратор");

if (admin == null)

{

MessageBox.Show("У вас недостаточно прав для доступа к администраторской панели");

ClinikMainWindow mainWindow = Application.Current.MainWindow as ClinikMainWindow;

mainWindow.mainframe.NavigationService.Navigate(new SpecialtiesPage());

return;

}

try

{

string verificationCode = Email\_code.GenerateCode();

List<string> emailContent = Email\_code.GenerateVerificateMessageAdmin(DateTime.Now, verificationCode);

this.\_lastVerificationCode = verificationCode;

Email\_code.SendMessage(email\_txtbx.Text, emailContent[0], emailContent[1]);

MessageBox.Show($"Код подтверждения отправлен на {email\_txtbx.Text}\n");

enter\_button.IsEnabled = true;

}

catch (Exception ex)

{

MessageBox.Show($"Ошибка при отправке кода: {ex.Message}");

}

}

private void enter\_button\_Click(object sender, RoutedEventArgs e)

{

//if (!Verify(code\_txtbx.Text))

//{

// MessageBox.Show("Неверный или просроченный код", "Ошибка", MessageBoxButton.OK, MessageBoxImage.Error);

// return;

//}

//else

//{

ClinikMainWindow mainWindow = Application.Current.MainWindow as ClinikMainWindow;

mainWindow.mainframe.NavigationService.Navigate(new Admin\_panel(currUser));

//}

}

private bool Verify(string user\_code)

{

return !string.IsNullOrEmpty(\_lastVerificationCode) && \_lastVerificationCode == user\_code;

}

}

AppointmentPage.cs

public partial class AppointmentPage : Page

{

Specialty selectedSpecialty;

Doctor currDoctor;

public string selected\_date;

public string selected\_time;

public AppointmentPage(Specialty spec, Doctor doct)

{

if (spec == null || doct == null) return;

currDoctor = doct;

selectedSpecialty = spec;

InitializeComponent();

DataContext = currDoctor;

StartInitialize();

}

private void StartInitialize()

{

CalendarLimit(); // ограничения календаря

bookAppointmentButton.IsEnabled = false;

}

private void CalendarLimit()

{

try

{

DateTime startDate = DateTime.Today;

DateTime endDate = DateTime.Today.AddDays(14); //запись на ближ 2 недель

calendarSelectedDateTxt.Text = "Дата: " + DateTime.Today.ToShortDateString();

appointmentCalendar.DisplayDateStart = startDate;

appointmentCalendar.DisplayDateEnd = endDate;

if (currDoctor == null) appointmentCalendar.IsEnabled = false; //если врач не выбран

NoAppointmentsText.Visibility = Visibility.Visible;

using var db = new DbAppontmentClinikContext();

var scheduleDays = db.Schedules.Where(s => s.IdDoctor == currDoctor.IdDoctor).Select(sc => sc.DayWeek).ToList();

for (DateTime date = startDate; date <= endDate; date = date.AddDays(1))

{

string dayOfWeekString = date.ToString("ddd", new CultureInfo("ru-RU"));

dayOfWeekString = dayOfWeekString[..1].ToUpper() + dayOfWeekString[1..].ToLower();

if (!scheduleDays.Contains(dayOfWeekString)) //блок если дня нет в расписании

{

appointmentCalendar.BlackoutDates.Add(new CalendarDateRange(date));

}

if (date.DayOfWeek == DayOfWeek.Sunday) //блок воскресенья

{

appointmentCalendar.BlackoutDates.Add(new CalendarDateRange(date));

}

}

}

catch (Exception ex)

{

MessageBox.Show($"Ошибка в ограничении календаря: {ex.Message}");

appointmentCalendar.IsEnabled = false;

}

}

private void Back\_button\_Click(object sender, RoutedEventArgs e)

{

if (Application.Current.MainWindow is not ClinikMainWindow mainWindow)

{

MessageBox.Show("Ошибка навигации");

return;

}

mainWindow.mainframe.NavigationService?.Navigate(new AllSpecialtiesPage(selectedSpecialty));

}

private void AppointmentCalendar\_SelectedDatesChanged(object sender, SelectionChangedEventArgs e)

{

LoadAvailableAppointments(); //загрузка записей

}

private void LoadAvailableAppointments()

{

if (!appointmentCalendar.SelectedDate.HasValue) return;

DateTime select = appointmentCalendar.SelectedDate.Value;

selected\_date = select.ToString("d MMMM", new CultureInfo("ru-RU"));

if (currDoctor == null) return;

try

{

var selectedDate = appointmentCalendar.SelectedDate.Value.Date;

var dayOfWeek = CultureInfo.GetCultureInfo("ru-RU").DateTimeFormat.GetAbbreviatedDayName(selectedDate.DayOfWeek);

dayOfWeek = char.ToUpper(dayOfWeek[0]) + dayOfWeek[1..].ToLower();

calendarSelectedDateTxt.Text = $"Дата: {selectedDate:d}";

using var db = new DbAppontmentClinikContext();

if (db == null) { MessageBox.Show("Ошибка в подключении к базе данных"); return; }

var schedule = db.Schedules.FirstOrDefault(s => s.IdDoctor == currDoctor.IdDoctor && s.DayWeek == dayOfWeek);

//если нет расписания для этого дня

if (schedule == null)

{

availableAppointmentsList.ItemsSource = null; //пустой список

NoAppointmentsText.Visibility = Visibility.Visible; //записи отсутсвуют

return;

}

//длит-ть приёма, по умолчанию 15

var time\_accept = db.Specialties.Where(d => d.IdSpecialty == currDoctor.IdSpecialty).Select(t => t.TimeAccept).FirstOrDefault() ?? 15;

//записи в опред день

var existingApp = db.Appointments

.Where(a => a.IdDoctor == currDoctor.IdDoctor &&

a.DateAppointment == DateOnly.FromDateTime(selectedDate) && a.TimeAppointment.HasValue)

.Select(a => a.TimeAppointment!.Value).ToList();

var availableAppList = GenerateTimeApp(selectedDate.Add(schedule.TimeStart.ToTimeSpan()),selectedDate.Add(schedule.TimeEnd.ToTimeSpan()),TimeSpan.FromMinutes((double)time\_accept),existingApp);

availableAppointmentsList.ItemsSource = availableAppList;

NoAppointmentsText.Visibility = availableAppList.Count == 0 ? Visibility.Visible : Visibility.Hidden;

}

catch (Exception ex)

{

MessageBox.Show($"Ошибка загрузки записей: {ex.Message}");

availableAppointmentsList.ItemsSource = null;

NoAppointmentsText.Visibility = Visibility.Visible;

}

}

private static List<Schedule> GenerateTimeApp(DateTime startTime, DateTime endTime, TimeSpan interval, List<TimeOnly> bookedTimes)

{

var appList = new List<Schedule>();

if (startTime >= endTime) return appList;

if (interval <= TimeSpan.Zero) return appList;

//генерация слотов с интервалом и проверкой на занятость

for (var i = startTime; i <= endTime; i += interval)

{

if (!bookedTimes.Contains(TimeOnly.FromDateTime(i)))

{

appList.Add(new Schedule

{

TimeStart = TimeOnly.FromDateTime(i)

});

}

}

return appList;

}

private void AppointmentCalendar\_DisplayDateChanged(object sender, CalendarDateChangedEventArgs e)

{

if (appointmentCalendar.DisplayDate.Month != DateTime.Now.Month || appointmentCalendar.DisplayDate.Year != DateTime.Now.Year)

{

appointmentCalendar.DisplayDate = DateTime.Now;

}

}

private async void AvailableAppointmentsList\_SelectionChanged(object sender, SelectionChangedEventArgs e)

{

if (availableAppointmentsList.SelectedItem != null && mainScrollViewer != null)

{

if (availableAppointmentsList.SelectedItem is Schedule selectedSchedule)

{

selected\_time = selectedSchedule.TimeStart.ToString("HH:mm"); // Берём время

}

information\_user\_grid.Visibility = Visibility.Visible;

await Task.Delay(50);

Dispatcher.Invoke(() =>

{

mainScrollViewer.BeginAnimation(ScrollViewerBehavior.VerticalOffsetProperty, null);

double targetOffset = mainScrollViewer.VerticalOffset + 550;

targetOffset = Math.Min(targetOffset, mainScrollViewer.ScrollableHeight);

DoubleAnimation animation = new()

{

To = targetOffset,

Duration = TimeSpan.FromMilliseconds(800),

EasingFunction = new CubicEase { EasingMode = EasingMode.EaseOut },

FillBehavior = FillBehavior.Stop

};

animation.Completed += (s, \_) =>

{

mainScrollViewer.BeginAnimation(ScrollViewerBehavior.VerticalOffsetProperty, null);

mainScrollViewer.ScrollToVerticalOffset(targetOffset);

};

mainScrollViewer.BeginAnimation(ScrollViewerBehavior.VerticalOffsetProperty, animation);

}, DispatcherPriority.Render);

}

}

private string \_lastVerificationCode;

private void code\_button\_Click(object sender, RoutedEventArgs e)

{

if (!ValidateTxtBox())

{

return;

}

string name = CapitalizeFirstLetter(name\_user\_txtbx.Text.Trim());

string surname = CapitalizeFirstLetter(surname\_user\_txtbx.Text.Trim());

DateOnly date = FormatDate(date\_user\_txtbx.Text.Trim());

string phone = new string(phone\_user\_txtbx.Text.Trim().Where(char.IsDigit).ToArray());

string email = email\_user\_txtbx.Text.Trim().ToLower();

try

{

string verificationCode = Email\_code.GenerateCode();

List<string> emailContent = Email\_code.GenerateVerificateMessage(DateTime.Now, verificationCode);

this.\_lastVerificationCode = verificationCode;

Email\_code.SendMessage(email, emailContent[0], emailContent[1]);

MessageBox.Show($"Код подтверждения отправлен на {email\_user\_txtbx.Text}\n");

bookAppointmentButton.IsEnabled = true;

}

catch (Exception ex)

{

MessageBox.Show($"Ошибка при отправке кода: {ex.Message}");

}

}

private bool ValidateTxtBox()

{

if (string.IsNullOrWhiteSpace(name\_user\_txtbx.Text) || string.IsNullOrWhiteSpace(surname\_user\_txtbx.Text) || string.IsNullOrWhiteSpace(date\_user\_txtbx.Text) || string.IsNullOrWhiteSpace(phone\_user\_txtbx.Text) || string.IsNullOrWhiteSpace(email\_user\_txtbx.Text))

{

return false;

}

if (date\_user\_txtbx.Text.Length < 10) { MessageBox.Show("Запишите корректную дату рождения"); return false; }

if (new string(phone\_user\_txtbx.Text.Trim().Where(char.IsDigit).ToArray()).Length < 11) { MessageBox.Show("Запишите корректный номер телефона"); return false; }

if (name\_user\_txtbx.Text.Length < 3) { MessageBox.Show("Запишите корректное имя"); return false; }

if (surname\_user\_txtbx.Text.Length < 2) { MessageBox.Show("Запишите корректную фамилию"); return false; }

if (email\_user\_txtbx.Text.Length < 8) { MessageBox.Show("Запишите корректную эл. почту"); return false; }

if (!IsValidDateFormat(date\_user\_txtbx.Text))

{

MessageBox.Show("Некорректный формат даты!");

return false;

}

return true;

}

public static bool IsValidDateFormat(string input)

{

return Regex.IsMatch(input, @"^\d{1,2}\.\d{1,2}\.\d{4}$");

}

private void BookAppointmentButton\_Click(object sender, RoutedEventArgs e)

{

if (!Verify(code\_txtbx.Text))

{

MessageBox.Show("Неверный или просроченный код", "Ошибка", MessageBoxButton.OK, MessageBoxImage.Error);

return;

}

using var db = new DbAppontmentClinikContext();

var check\_user = db.Users.Where(us => us.EmailUsers == email\_user\_txtbx.Text).FirstOrDefault();

string oms = oms\_user\_txtbx.Text;

string passport = passport\_user\_txtbx.Text.Replace("\_", "").Replace(" ", "");

DateOnly appointmentDate = DateOnly.FromDateTime(appointmentCalendar.SelectedDate.Value);

TimeOnly appointmentTime = TimeOnly.Parse(selected\_time);

if (check\_user == null) // если его нет

{

if (string.IsNullOrWhiteSpace(oms) || string.IsNullOrWhiteSpace(passport))

{

MessageBox.Show("Необходимо заполнить мед.полис и данные паспорта для дальнейшей записи и прикрепления к клинике", "Заполните поля", MessageBoxButton.OK, MessageBoxImage.Error);

return;

}

else

{

var newUser = new User

{

EmailUsers = email\_user\_txtbx.Text,

RoleIdUsers = "Пользователь",

SurnameUsers = surname\_user\_txtbx.Text,

NameUsers = name\_user\_txtbx.Text,

DateBirth = FormatDate(date\_user\_txtbx.Text.Trim()),

MedicalPolicy = oms,

PassportNumber = passport,

PhoneNumber = new string(phone\_user\_txtbx.Text.Trim().Where(char.IsDigit).ToArray())

};

try

{

db.Users.Add(newUser);

db.SaveChanges();

var c = db.Users.Where(s => s.EmailUsers == newUser.EmailUsers).FirstOrDefault();

List<string> emailContent = Email\_code.GenerateRegMessage(DateTime.Now, name\_user\_txtbx.Text, c.IdMedCard.ToString());

Email\_code.SendMessage(email\_user\_txtbx.Text, emailContent[0], emailContent[1]);

var newAppointment = new Appointment

{

IdDoctor = currDoctor.IdDoctor,

IdMedCard = newUser.IdMedCard,

DateAppointment = appointmentDate,

TimeAppointment = appointmentTime,

StatusAppointment = "В ожидании"

};

try

{

db.Appointments.Add(newAppointment);

db.SaveChanges();

}

catch(Exception ex)

{

MessageBox.Show($"Ошибка при записи: {ex.Message}", "Ошибка", MessageBoxButton.OK, MessageBoxImage.Error);

}

NavigateToEnd();

}

catch (Exception ex)

{

MessageBox.Show($"Ошибка при прикреплении: {ex.Message}", "Ошибка", MessageBoxButton.OK, MessageBoxImage.Error);

}

}

}

else // если он есть

{

if(string.IsNullOrWhiteSpace(oms) && string.IsNullOrWhiteSpace(passport)) //если пустые поля, то продолжаем

{

var newAppointment = new Appointment

{

IdDoctor = currDoctor.IdDoctor,

IdMedCard = check\_user.IdMedCard,

DateAppointment = appointmentDate,

TimeAppointment = appointmentTime,

StatusAppointment = "В ожидании"

};

try

{

db.Appointments.Add(newAppointment);

db.SaveChanges();

}

catch (Exception ex)

{

MessageBox.Show($"Ошибка при записи: {ex.Message}", "Ошибка", MessageBoxButton.OK, MessageBoxImage.Error);

}

MessageBox.Show($"{check\_user.NameUsers}, вы успешно записались на приём!\n\nВаш номер мед.карты: {check\_user.IdMedCard}");

NavigateToEnd();

}

else if(string.IsNullOrWhiteSpace(oms))//если поле с мед полисом пустое, то обновляем паспорт

{

if(check\_user.PassportNumber != passport)

{

check\_user.PassportNumber = passport;

var newAppointment = new Appointment

{

IdDoctor = currDoctor.IdDoctor,

IdMedCard = check\_user.IdMedCard,

DateAppointment = appointmentDate,

TimeAppointment = appointmentTime,

StatusAppointment = "В ожидании"

};

try

{

db.Appointments.Add(newAppointment);

db.SaveChanges();

}

catch (Exception ex)

{

MessageBox.Show($"Ошибка при записи: {ex.Message}", "Ошибка", MessageBoxButton.OK, MessageBoxImage.Error);

}

MessageBox.Show($"Паспорт был успешно обновлён.\n\n{check\_user.NameUsers}, вы успешно записались на приём!\n\nВаш номер мед.карты: {check\_user.IdMedCard}");

NavigateToEnd();

}

else {

var newAppointment = new Appointment

{

IdDoctor = currDoctor.IdDoctor,

IdMedCard = check\_user.IdMedCard,

DateAppointment = appointmentDate,

TimeAppointment = appointmentTime,

StatusAppointment = "В ожидании"

};

try

{

db.Appointments.Add(newAppointment);

db.SaveChanges();

}

catch (Exception ex)

{

MessageBox.Show($"Ошибка при записи: {ex.Message}", "Ошибка", MessageBoxButton.OK, MessageBoxImage.Error);

}

MessageBox.Show($"{check\_user.NameUsers}, вы успешно записались на приём!\n\nВаш номер мед.карты: {check\_user.IdMedCard}");

NavigateToEnd();

}

}

else if(string.IsNullOrWhiteSpace(passport)) // обновляем мед полис

{

if(check\_user.MedicalPolicy != oms)

{

check\_user.MedicalPolicy = oms;

var newAppointment = new Appointment

{

IdDoctor = currDoctor.IdDoctor,

IdMedCard = check\_user.IdMedCard,

DateAppointment = appointmentDate,

TimeAppointment = appointmentTime,

StatusAppointment = "В ожидании"

};

try

{

db.Appointments.Add(newAppointment);

db.SaveChanges();

}

catch (Exception ex)

{

MessageBox.Show($"Ошибка при записи: {ex.Message}", "Ошибка", MessageBoxButton.OK, MessageBoxImage.Error);

}

MessageBox.Show($"Медицинский полис был успешно обновлён.\n\n{check\_user.NameUsers}, вы успешно записались на приём!\n\nВаш номер мед.карты: {check\_user.IdMedCard}");

NavigateToEnd();

}

else { MessageBox.Show($"{check\_user.NameUsers}, вы успешно записались на приём!\n\nВаш номер мед.карты: {check\_user.IdMedCard}");

NavigateToEnd(); }

}

else

{

check\_user.PassportNumber = passport;

check\_user.MedicalPolicy = oms;

var newAppointment = new Appointment

{

IdDoctor = currDoctor.IdDoctor,

IdMedCard = check\_user.IdMedCard,

DateAppointment = appointmentDate,

TimeAppointment = appointmentTime,

StatusAppointment = "В ожидании"

};

try

{

db.Appointments.Add(newAppointment);

db.SaveChanges();

}

catch (Exception ex)

{

MessageBox.Show($"Ошибка при записи: {ex.Message}", "Ошибка", MessageBoxButton.OK, MessageBoxImage.Error);

}

MessageBox.Show($"Медицинский полис и паспорт были успешно обновлены.\n\n{check\_user.NameUsers}, вы успешно записались на приём!\n\nВаш номер мед.карты: {check\_user.IdMedCard}");

NavigateToEnd(); }

}

}

private bool Verify(string user\_code)

{

return !string.IsNullOrEmpty(\_lastVerificationCode) && \_lastVerificationCode == user\_code;

}

private void name\_surname\_TextChanged(object sender, TextChangedEventArgs e)

{

string name = name\_user\_txtbx.Text;

string surname = surname\_user\_txtbx.Text;

if (!string.IsNullOrEmpty(name) || !string.IsNullOrEmpty(surname))

{

name\_user\_txtbx.Text = CapitalizeFirstLetter(name);

surname\_user\_txtbx.Text = CapitalizeFirstLetter(surname);

name\_user\_txtbx.CaretIndex = name.Length;

surname\_user\_txtbx.CaretIndex = surname.Length;

}

}

private string CapitalizeFirstLetter(string input)

{

if (string.IsNullOrWhiteSpace(input))

return input;

return char.ToUpper(input[0]) + input.Substring(1).ToLower();

}

private DateOnly FormatDate(string dateInput)

{

return DateOnly.Parse(dateInput);

}

private void NoLetter\_PreviewTextInput(object sender, TextCompositionEventArgs e)

{

e.Handled = !e.Text.All(char.IsDigit);

}

private void email\_user\_txtbx\_PreviewTextInput(object sender, TextCompositionEventArgs e)

{

Regex regex = new Regex("[а-яА-ЯёЁ]");

e.Handled = regex.IsMatch(e.Text);

}

private void NavigateToEnd()

{

if (Application.Current.MainWindow is ClinikMainWindow mainWindow)

{

mainWindow.mainframe.NavigationService.Navigate(new EndAppointment(selected\_date, selected\_time, selectedSpecialty.NameSpecialty, selectedSpecialtyTxtbx.Text.Replace("К врачу: ", ""), currDoctor.CabinetNumber));

}

}

}

public static class ScrollViewerBehavior

{

public static readonly DependencyProperty VerticalOffsetProperty =

DependencyProperty.RegisterAttached("VerticalOffset", typeof(double), typeof(ScrollViewerBehavior),

new UIPropertyMetadata(0.0, OnVerticalOffsetChanged));

private static void OnVerticalOffsetChanged(DependencyObject target, DependencyPropertyChangedEventArgs e)

{

if (target is ScrollViewer scrollViewer)

scrollViewer.ScrollToVerticalOffset((double)e.NewValue);

}

}

ClinikMainWindow.cs

public partial class ClinikMainWindow : Window

{

public ClinikMainWindow()

{

InitializeComponent();

UpdateAppointmentStatuses();

mainframe.NavigationService.Navigate(new SpecialtiesPage());

}

private void UpdateAppointmentStatuses()

{

try

{

using (var db = new DbAppontmentClinikContext())

{

var currentDate = DateOnly.FromDateTime(DateTime.Now);

var currentTime = TimeOnly.FromDateTime(DateTime.Now);

var appointmentsToUpdate = db.Appointments.Where(a => (a.DateAppointment < currentDate ||(a.DateAppointment == currentDate && a.TimeAppointment <= currentTime)) &&(a.StatusAppointment == null || a.StatusAppointment != "Завершён")).ToList();

foreach (var appointment in appointmentsToUpdate)

{

appointment.StatusAppointment = "Завершён";

}

if (appointmentsToUpdate.Any())

{

int changes = db.SaveChanges();

}

}

}

catch (Exception ex)

{

MessageBox.Show($"Ошибка при обновлении статусов записей: {ex.Message}\n\n{ex.InnerException?.Message}",

"Ошибка", MessageBoxButton.OK, MessageBoxImage.Error);

}

}

private void Click\_aboutClinik\_MouseLeftButtonDown(object sender, MouseButtonEventArgs e)

{

mainframe.NavigationService.Navigate(new AboutClinik());

}

private void Click\_contactsClinik\_MouseLeftButtonDown(object sender, MouseButtonEventArgs e)

{

mainframe.NavigationService.Navigate(new ContactsClinik());

}

private void Click\_appointmentClinik\_MouseLeftButtonDown(object sender, MouseButtonEventArgs e)

{

mainframe.NavigationService.Navigate(new SpecialtiesPage());

}

private void mainframe\_Navigated(object sender, System.Windows.Navigation.NavigationEventArgs e)

{

}

private void Admin\_panel\_MouseLeftButtonDown(object sender, MouseButtonEventArgs e)

{

mainframe.NavigationService.Navigate(new AdminAuth());

}

}

EndAppointment.cs

public partial class EndAppointment : Page

{

public EndAppointment(string date, string time, string specialty, string fio, string? cabinet)

{

InitializeComponent();

date\_lbl.Content = date;

time\_lbl.Content = time;

if(specialty == "Терапевт") { specialty = "Терапевт участковый"; }

specialty\_lbl.Content = specialty;

fio\_lbl.Content = GetShortName(fio);

cabinet\_lbl.Content = $"Кабинет № {cabinet}";

ClinikMainWindow mainWindow = Application.Current.MainWindow as ClinikMainWindow;

mainWindow.Title = "Успешная запись";

}

private static string GetShortName(string fullName)

{

if (string.IsNullOrWhiteSpace(fullName)) return fullName;

string[] parts = fullName.Split(new[] { ' ' }, StringSplitOptions.RemoveEmptyEntries);

if (parts.Length == 3){return $"{parts[0]} {parts[1][0]}. {parts[2][0]}.";}

else if (parts.Length == 2){return $"{parts[0]} {parts[1][0]}.";}

else {return fullName;}

}

private void on\_main\_Click(object sender, RoutedEventArgs e)

{

ClinikMainWindow mainWindow = Application.Current.MainWindow as ClinikMainWindow;

mainWindow.mainframe.NavigationService.Navigate(new SpecialtiesPage());

}

private void print\_talon\_Click(object sender, RoutedEventArgs e)

{

string doctorSpecialization = specialty\_lbl.Content.ToString();

string doctorName = fio\_lbl.Content.ToString();

string cabinetNumber = cabinet\_lbl.Content.ToString();

string clinicAddress = "г. Уфа, ул. Ленина, д. 75";

string date = date\_lbl.Content.ToString();

string time = time\_lbl.Content.ToString();

PDF.CreateAppointmentTicketPdf(doctorSpecialization, doctorName, cabinetNumber, clinicAddress, date, time);

}

}

InformationDoctorPage.cs

public partial class InformationDoctorPage : Page

{

Specialty specialtyName;

Doctor currDoctor1 { get; set; }

public InformationDoctorPage(Specialty specialty, Doctor currDoctor)

{

specialtyName = specialty;

currDoctor1 = currDoctor;

InitializeComponent();

DataContext = currDoctor1;

specialtyNameTxt.Text = $"Врач - {specialtyName.NameSpecialty.ToLower()}";

IconDoctorView.Source = new BitmapImage(new Uri(currDoctor.DisplayIconDoctor));

}

private void Back\_button\_Click(object sender, RoutedEventArgs e)

{

ClinikMainWindow mainWindow = Application.Current.MainWindow as ClinikMainWindow;

mainWindow.mainframe.NavigationService.Navigate(new AllSpecialtiesPage(specialtyName));

}

}

SpecialtiesPage.cs

public partial class SpecialtiesPage : Page

{

public List<Specialty> allSpecialties { get; set; }

public List<Specialty> selectedSpecialty { get; set; }

public SpecialtiesPage()

{

InitializeComponent();

ClinikMainWindow mainWindow = Application.Current.MainWindow as ClinikMainWindow;

mainWindow.Title = "Запись на приём";

LoadSpecialty();

UpdateSpecialtyListView();

}

private void LoadSpecialty()

{

using (var db = new DbAppontmentClinikContext())

{

allSpecialties = db.Specialties.ToList();

}

selectedSpecialty = allSpecialties.ToList();

UpdateSpecialtyListView();

}

private void UpdateSpecialtyListView()

{

specialtyListView.Items.Clear();

foreach (Specialty specialty in selectedSpecialty)

{

SpecialtyControl specItem = new SpecialtyControl(specialty);

specialtyListView.Items.Add(specItem);

}

}

private void searchtxtbox\_TextChanged(object sender, TextChangedEventArgs e)

{

var search = searchtbox.Text?.ToLower();

if (!string.IsNullOrEmpty(search))

{

selectedSpecialty = allSpecialties.Where(s =>

!string.IsNullOrEmpty(s.NameSpecialty) && s.NameSpecialty.ToLower().Contains(search)).ToList();

}

else

{

selectedSpecialty = allSpecialties.ToList();

}

UpdateSpecialtyListView();

}

}

AllSpecialtiesPage.cs

public partial class AllSpecialtiesPage : Page

{

Specialty specialtyName;

public AllSpecialtiesPage(Specialty selectedSpecialty)

{

specialtyName = selectedSpecialty;

InitializeComponent();

selectedSpecialtyTxtbx.Text = "Специализация: " + specialtyName.NameSpecialty;

EmptySpecTxt.Visibility = Visibility.Hidden;

LoadDoctors();

}

private void LoadDoctors()

{

using (var db = new DbAppontmentClinikContext())

{

var doctors = db.Doctors.Include(spec => spec.IdSpecialtyNavigation).Where(d => d.IdSpecialtyNavigation.NameSpecialty == specialtyName.NameSpecialty);

if (!doctors.Any())

{

EmptySpecTxt.Visibility = Visibility.Visible;

EmptySpecTxt.Text = "К сожалению специалистов данного направления нет...";

}

else

{

foreach (var d in doctors)

{

doctorsListView.Items.Add(new DoctorsControl(d, specialtyName));

}

}

}

}

private void Back\_button\_Click(object sender, RoutedEventArgs e)

{

ClinikMainWindow mainWindow = Application.Current.MainWindow as ClinikMainWindow;

mainWindow.mainframe.NavigationService.Navigate(new SpecialtiesPage());

}

}

# СПИСОК ИСПОЛЬЗУЕМЫХ ИСТОЧНИКОВ

1. ГОСТ Р 7.0.100-2018. Библиографическая запись. Библиографическое описание. Общие требования и правила составления: национальный стандарт Российской Федерации : дата введения 2019-07-01 / Федеральное агентство по техническому регулированию. – Изд. официальное. – Москва : Стандартинформ, 2018. – 124 с.
2. ГОСТ Р 51583-2014. Защита информации. Порядок создания автоматизированных систем в защищенном исполнении. Общие положения : дата введения 2014-09-01 / Федеральное агентство по техническому регулированию и метрологии. – Изд. официальное. – Москва: Стандартинформ, 2018. – 14 с.
3. ГОСТ 19.701-90 ЕСПД. Схемы алгоритмов, программ, данных и систем. Обозначения условные и правила выполнения : дата введения 1992-01-01. – Изд. официальное. – Москва: Стандартинформ, 2010. – 158 с.
4. ГОСТ 34.601-90 Информационная технология. Комплекс стандартов на автоматизированные системы. Автоматизированные системы. Стадии создания : дата введения 2009-09-01. – Изд. официальное. – Москва: Стандартинформ, 2009.
5. Ботрос, С. MySQL по максимуму. / С. Ботрос, Д. Тинли – 4-е изд. – Санкт-Петербург : Питер, 2023. – 432 с. – (Бестселлеры O’Reilly). – ISBN 978-5-4461-2261-5. – Текст: непосредственный.
6. Лок, Э. ASP.NET Core в действии. / Э. Лок. – 3-е изд. – Москва : ДМК Пресс, 2024. – 1046 с. – ISBN 978-5-93700-183-2. – Текст: непосредственный.
7. Мартин, Р. Чистый код: создание, анализ и рефакторинг. / Р. Мартин. – Санкт-Петербург : Питер, 2020. – 464 с. – (Библиотека программиста) – ISBN 978-5-4461-0960-9. – Текст: непосредственный.
8. Прайс, М. C# 10 и .NET 6. Современная кросс-платформенная разработка. / М. Прайс. – Санкт-Петербург : Питер, 2023. – 848с. – (Для профессионалов). – ISBN 978-5-4461-2249-3. – Текст: непосредственный.
9. Документация C#. – 2025. – URL: <https://learn.microsoft.com/ru-ru/dotnet/csharp/> (дата обращения 07.03.2025). – Текст: электронный.
10. Документация MySQL. – 2025. – URL: <https://dev.mysql.com/doc/> (дата образщения 16.02.2025). – Текст: электронный.
11. Потопахин В. Искусство алгоритмизации. [Текст] / В. Потопахин. – Москва: ДМК Пресс, 2018. – 320 с.
12. Семакин И. Основы алгоритмизации и программирования. Учебник [Текст] / И. Семакин, А. Шестаков. – 2-е изд., Москва: Академия, 2018. – 304 с.
13. Скиена С. Алгоритмы. Руководство по разработке [Текст] / С. Скиена 2-е изд. – СПб. БХВ-Петербург, 2016. – 720 с. Шарп, Дж. Microsoft Visual C#. Подробное руководство. [Текст]: учеб. / Дж. Шарп – 8-е изд., СПб.: Питер СПб, 2017 – 848 с.
14. Программирование на C, C# и Java [Электронный ресурс] / MySQL и C# работаем с базой данных. – Режим доступа https://vscode.ru/proglessons/mysql-c-sharp.html, свободный
15. Metanit [Электронный ресурс] / Руководство по языку C#. – Режим доступа https://metanit.com/sharp/tutorial/, свободный
16. Metanit [Электронный ресурс] / Руководство по WPF. – Режим доступа https://metanit.com/sharp/wpf/, свободный
17. Microsoft [Электронный ресурс] / Руководство по языку C#. – Режим доступа https://docs.microsoft.com/ru-ru/dotnet/csharp/, свободный
18. Professor Web [Электронный ресурс] / Основы WPF. – Режим доступа https://professorweb.ru/my/WPF/base\_WPF/level1/info\_WPF.php, свободный
19. Ruseller [Электронный ресурс] / Введение в хранимые процедуры MySQL 5. – Режим доступа https://ruseller.com/lessons.php?id=1189, свободный
20. Sql-tutorial [Электронный ресурс] / SQL Задачи и решения. – Режим доступа http://www.sql-tutorial.ru/ru/content.html, свободный