СОДЕРЖАНИЕ

[ВВЕДЕНИЕ 5](#_Toc120381854)

[1 Постановка задачи 6](#_Toc120381855)

[1.1 Анализ предметной области 6](#_Toc120381856)

[1.2 Техническое задание на разработку ГОСТ 19.201-78 7](#_Toc120381857)

[1.2.1 Введение 7](#_Toc120381858)

[1.2.2 Основания для разработки 7](#_Toc120381859)

[1.2.3 Назначение разработки 7](#_Toc120381860)

[1.2.4 Требования к программе или программному изделию 7](#_Toc120381861)

[1.2.5 Технико-экономические показатели 10](#_Toc120381862)

[1.2.6 Стадии и этапы разработки 11](#_Toc120381863)

[1.2.6.3 Содержание работ по этапам 12](#_Toc120381864)

[1.2.7 Порядок контроля и приемки 12](#_Toc120381865)

[2. Проектирование 13](#_Toc120381866)

[2.1 Функциональная структура программы 13](#_Toc120381867)

[2.2 Схема модулей программы 15](#_Toc120381868)

[2.3 Паспорта основных модулей 16](#_Toc120381869)

[2.3.1 Модуль «Запись на прием ко врачу» 16](#_Toc120381870)

[2.3.2 Модуль «Поиск врача по ФИО» 17](#_Toc120381871)

[2.3.3 Модуль «Фильтр врачей по специализации» 18](#_Toc120381872)

[2.3.4 Модуль «Удаление врача из базы данных» 19](#_Toc120381873)

[3 Руководство оператора ГОСТ 19.505-79 20](#_Toc120381874)

[3.1 Назначение программы 20](#_Toc120381875)

[3.1.1 Функциональное назначение 20](#_Toc120381876)

[3.1.2 Состав функций 20](#_Toc120381877)

[3.2 Условия выполнения программы 20](#_Toc120381878)

[3.2.1 Климатические условия эксплуатации 20](#_Toc120381879)

[3.2.2 Требования к составу и параметрам технических средств 21](#_Toc120381880)

[3.2.3 Требования к программным средствам, используемым программой 21](#_Toc120381881)

[3.3 Требования к персоналу (пользователю) 21](#_Toc120381882)

[3.4 Выполнение программы 21](#_Toc120381883)

[3.4.1 Загрузка и запуск программы 22](#_Toc120381884)

[3.4.2 Выполнение программы 22](#_Toc120381885)

[3.5 Сообщения оператору 30](#_Toc120381886)

[4 Тестирование программы 31](#_Toc120381887)

[4.1 Программа, методика и результаты испытаний ГОСТ 19.301-79 31](#_Toc120381888)

[4.1.1 Объект испытаний 31](#_Toc120381889)

[4.1.2 Цель испытаний 31](#_Toc120381890)

[4.1.3 Требования к программе 33](#_Toc120381891)

[4.1.4 Требования к программной документации 33](#_Toc120381892)

[4.1.5 Средства и порядок испытаний 33](#_Toc120381893)

[4.1.6. Методика испытаний 34](#_Toc120381894)

[4.2 Результаты испытаний 35](#_Toc120381895)

[СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ 39](#_Toc120381896)

[ПРИЛОЖЕНИЕ А 40](#_Toc120381897)

# ВВЕДЕНИЕ

В современных условиях, человек вынужден работать с гигантскими объемами информации. Системы автоматизированного учета обязаны представлять собой мощные средства, способные обрабатывать гигантские потоки данных высокой структурной сложности за минимум затраченного времени, обеспечивая дружественный диалог с пользователем

Цель данного курсового проекта состоит в проектировании, реализации и тестирования программного продукта, реализующего работу с информационной системой.

Данные будут хранится в виде таблицах. Для взаимодействия с данными будет применяться СУБДДомоуправлениеи среда разработки Visual Studio на языке программирования С#.

Для того чтобы достичь цели курсового проекта необходимо поставить перед собой следующие задачи:

* Более углубленно изучить язык программирования С#;
* Изучить СУБД Домоуправление;
* Изучить набор данных в Microsoft Visual Studio 2022;
* Изучить нормализацию БД;
* Изучить и проанализировать соответствующую литературу;

1. **Постановка задачи**

## 1.1 Анализ предметной области

Курсовой проект будет реализовывать программный продукт с для автоматизации работы с данными поликлиники. Будет использоваться удаленный сервер для хранения данных и клиентское приложение. Интерфейс должен быть удобен и интуитивно понятен для пользователя с минимальным опытом знакомства с подобными системами. Основные данные в БД будут представлять данные врачей, пациентов и приемов. Так же должна быть возможность организовать возможность выписывать больничные. В современном мире базы данных стали довольно доступным инструментом, упрощающим работу миллионов людей по всему миру, и медицинские учреждения не исключение. Грамотное хранение данных о врачах, пациентах, больничных позволяет ускорить поток обслуживания граждан, сделать посещение поликлиники более удобным.

## 1.2 Техническое задание на разработку ГОСТ 19.201-78

### 1.2.1 Введение

#### 1.2.1.1 Наименование программы

Наименование программы – «Polyclinic».

#### 1.2.1.2 Краткая характеристика области применения

Автоматизации работы с данными поликлиники.

### 1.2.2 Основания для разработки

#### 1.2.2.1 Основание для проведения разработки

Основанием для проведения разработки является задание на курсовой проект.

#### 1.2.2.2 Наименование и условное обозначение темы разработки

Наименование темы разработки – «Разработка информационной системы «Поликлиника». Условное обозначение темы разработки (шифр темы) – «18/09.02.07».

### 1.2.3 Назначение разработки

#### 1.2.3.1 Функциональное назначение

Функциональным назначением программы является работа с данными.

#### 1.2.3.2 Эксплуатационное назначение

Не имеет.

### 1.2.4 Требования к программе или программному изделию

#### 1.2.4.1 Требования к функциональным характеристикам

Программа должна обеспечивать возможность выполнения следующих функций:

* Просмотр данных врачей;
* Добавление в базу нового врача;
* Изменение данных врача;
* Удаление врача из базы данных;
* Просмотр данных медсестер;
* Добавление в базу нового медсестры;
* Изменение данных медсестры;
* Удаление медсестры из базы данных;
* Просмотр данных пациентов;
* Добавление в базу нового пациента;
* Изменение данных пациента;
* Удаление пациента из базы данных;
* Возможность выписать больничный;
* Возможность записать пациента на прием ко врачу.

#### 1.2.4.2 Требования к надежности

Требования к обеспечению надежного функционирования программы не предъявляются.

#### 1.2.4.3 Время восстановления после отказа

Время восстановления после не фатального сбоя ОС не должно превышать 10-ти минут при условии соблюдения условий эксплуатации технических и программных средств.

Время восстановления после фатального сбоя не должно превышать времени, требуемого для устранения неисправностей технических средств и переустановки ПО.

#### 1.2.4.4 Отказы из-за некорректных действий оператора

Отказы программы возможны вследствие некорректных действий оператора (пользователя) при взаимодействии с операционной системой. Во избежание возникновения отказов программы по указанной выше причине следует обеспечить работу пользователя без предоставления ему административных привилегий.

#### 1.2.4.5 Условия эксплуатации

Климатические условия эксплуатации, при которых должны осуществляться заданные характеристики, должны удовлетворять требованиям, предъявляемым к техническим средствам (ПК) в части условий эксплуатации.

#### 1.2.4.6 Требования к видам обслуживания

Программа не требует проведения каких-либо видов обслуживания.

#### 1.2.4.7 Требования к численности и квалификации персонала

Минимальное количество персонала должно составлять 2 штатных единиц - системный администратор и пользователь ПК – оператор.

Системный администратор должен иметь высшее профильное образование. В перечень задач системного администратора входят:

* поддержание работоспособности технических средств;
* установка и поддержка работоспособности системных программных средств – ОС;
* установка программ.

Оператор должен уметь работать с графическим пользовательским интерфейсом ОС.

#### 1.2.4.8 Требования к составу и параметрам технических средств

В состав технических средств должен входить IBM-совместимый персональный компьютер, включающий в себя:

* Процессор: Pentium 4 2.5 GHz (одноядерный) или выше;
* Свободное место на жестком диске: от 1 Гб;
* ОЗУ: 4 Гб и выше;
* ОС: Windows 10;

#### 1.2.4.9 Требования к информационной и программной совместимости

**Требования к информационным структурам и методам решения**

Требования к информационным структурам (записям) на входе, а также к методам решения не предъявляются. Информационная структура должна включать отображение входных данных и результата на экране.

**Требования к исходным кодам и языкам программирования**

Исходные коды программы должны быть реализованы на языке C#. В качестве интеграционной среды разработки программы должна быть использована среда Microsoft Visual Studio 2022 и СУБД MS SQL.

**Требования к защите информации и программ**

Требования к защите информации и программ не предъявляются.

**Специальные требования**

Программа должна обеспечивать взаимодействие с пользователем посредством графического интерфейса.

#### 1.2.4.10. Требования к маркировке и упаковке

**Требование к маркировке**

Программное изделие [оптический диск] должно иметь маркировку с обозначением фамилии студента, номера группы, наименования программы. Маркировка должна быть нанесена на программное изделие в виде надписи, маркером на диск или ручкой на упаковку диска.

**Требования к упаковке**

Упаковка программного изделия должна осуществляться в конверте для дисков.

#### 1.2.4.11 Требования к программной документации

В состав сопровождающей документации должны входить:

* Техническое задание на разработку ГОСТ 19.201-78
* Руководство оператора ГОСТ 19.505-79
* Программа, методика и результаты испытаний ГОСТ 19.301-79

### 1.2.5 Технико-экономические показатели

**Экономические преимущества разработки**

Коммерческое использование не предусмотрено.

### 1.2.6 Стадии и этапы разработки

#### 1.2.6.1 Стадии разработки

Разработка должна быть проведена в две стадии:

* Разработка технического задания;
* Разработка технического проекта.

#### 1.2.6.2 Этапы разработки

Этапы разработки:

* Разработка технического задания
* разработка программы;
* разработка программной документации;
* испытания программы.

#### **1.2.6.3 Содержание работ по этапам**

На этапе разработки технического задания должны быть выполнены следующие работы:

* Постановка задачи
* Определение и уточнение требований к техническим средствам
* Определение требований к программе
* Определение стадий, этапов и сроков разработки программы и документации на неё
* Выбор языка программирования и среды разработки
* Согласование и утверждение технического задания

На этапе разработки программной документации должна быть выполнена разработка программных документов в соответствии с требованиями ГОСТ.

На этапе испытаний программы долины быть выполнены следующие виды работ:

* Разработка, согласование и утверждение программы и методики испытаний
* Проведение приемосдаточных испытаний
* Корректировка программы и программной документации по результатам испытаний

На этапе подготовки и передачи программы должна быть выполнена работа по сдаче программы и программной документации преподавателю.

### 1.2.7 Порядок контроля и приемки

Приемосдаточные испытания должны проводиться в колледже преподавателем до декабря 2022 года.

Приемосдаточные испытания программы должны проводиться согласно разработанной студентом и согласованной преподавателем «Программы и методики испытаний».

# 2. Проектирование

## 2.1 Функциональная структура программы

Ниже приведена таблица, в которую включаются все возможные функции, которые планируются использовать в программе в соответствии с таблицей 2.1.

Таблица 2.1 – Таблица спецификации функций

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Название | Действие | | Объект | Вид |
| Основные действия с БД | | | | |
| Добавить врача в базу данных | Добавляет в базу данных новую запись о враче и отображает ее на странице врачей | | Таблица | Основная |
| Изменить данные врача | Возможность внести правки в запись выбранного врача | | Таблица | Основная |
| Удалить врача | Удаляет врача из базы данных | | Таблица | Основная |
| Добавить медсестру в базу данных | Добавляет в базу данных новую запись о медсестре и отображает ее на странице медсестер | | Таблица | Основная |
| Изменить данные медсестры | Возможность внести правки в запись выбранной медсестры | | Таблица | Основная |
| Удалить медсестру | Удаляет медсестру из базы данных | | Таблица | Основная |
| Добавить пациента в базу данных | Добавляет в базу данных новую запись о пациенте и отображает ее на странице пациентов | | Таблица | Основная |
| Изменить данные пациента | Возможность внести правки в запись выбранного пациента | | Таблица | Основная |
| Удалить пациента | Удаляет пациента из базы данных | | Таблица | Основная |
| Выписать больничный | Добавляет в базу данных новую запись о больничном и отображает ее на странице пациентов | | Таблица | Основная |
| Изменить данные больничного | Возможность внести правки в запись выбранного больничного | | Таблица | Основная |
| Удалить больничный | Удаляет больничный из базы данных | | Таблица | Основная |
| Записать пациента на прием | Создает запись в базе данных о новом приеме | | Таблица | Основная |
| Вспомогательные функции | | | | |
| Фильтр врачей по специализации | | Отбирает врачей из базы данных по выбранной специализации | Окно | Вспомогательная |
| Поиск по ФИО | | Позволяет найти врача / медсестру/пациента по ФИО | Окно | Вспомогательная |
| Сервисные функции | | | | |
| Выход | Функция осуществляет выход из приложения | | Приложения | Сервисная |

## 2.2 Схема модулей программы



Рисунок 1 – ERD диаграмма в нотации IDEF1x

## Паспорта основных модулей

### 2.3.1 Модуль «Запись на прием ко врачу»

Таблица 2.2 -Структура данных функции «Запись на прием ко врачу»

|  |  |  |
| --- | --- | --- |
| Сигнатура | Описание | Параметры |
| void SaveButton\_Click(object sender, RoutedEventArgs e) | Проверяет корректность введенных данных и создает запись на прием ко врачу. В случае существования такой записи – редактирует в ней данные | Входные параметры являются служебными и в модуле практического применения не имеют |



Рисунок 2 – Общий алгоритм функции «Запись на прием ко врачу»

### 2.3.2 Модуль «Поиск врача по ФИО»

Таблица 2.3 -Структура данных функции «Поиск врача по ФИО»

|  |  |  |
| --- | --- | --- |
| Сигнатура | Описание | Параметры |
| void TxtFind\_TextChanged(object sender, TextChangedEventArgs e) | Производит поиск врача по ФИО | Входные параметры являются служебными и в модуле практического применения не имеют |



Рисунок 3 – Общий алгоритм функции «Поиск врача по ФИО»

Поиск по ФИО также используется для выборки из таблиц медсестер и пациентов. Реализация идентична вышеописанному модулю, за исключение отсутствия установки фильтра по специализации.

### 2.3.3 Модуль «Фильтр врачей по специализации»

Таблица 2.4 -Структура данных функции «Фильтр врачей по специализации»

|  |  |  |
| --- | --- | --- |
| Сигнатура | Описание | Параметры |
| void CmbSpecFilter\_SelectionChanged (object sender, SelectionChangedEventArgs e) | Производит выборку врачей по специализации | Входные параметры являются служебными и в модуле практического применения не имеют |



Рисунок 4 – Общий алгоритм функции «Фильтр врачей по специализации»

### 2.3.4 Модуль «Удаление врача из базы данных»

Таблица 2.5 -Структура данных функции «Удаление врача из базы данных»

|  |  |  |
| --- | --- | --- |
| Сигнатура | Описание | Параметры |
| void DeleteButton\_Click(object sender,RoutedEventArgs e) | Удаляет врача из базы данных | Входные параметры являются служебными и в модуле практического применения не имеют |



Рисунок 5 – Общий алгоритм функции «Удаление врача из базы данных»

# 3 Руководство оператора ГОСТ 19.505-79

## 3.1 Назначение программы

## 3.1.1 Функциональное назначение

Функциональным назначением программы является работа с данными.

**3.1.2 Состав функций**

Программа должна обеспечивать возможность выполнения следующих функций:

* Просмотр данных врачей;
* Добавление в базу нового врача;
* Изменение данных врача;
* Удаление врача из базы данных;
* Просмотр данных медсестер;
* Добавление в базу нового медсестры;
* Изменение данных медсестры;
* Удаление медсестры из базы данных;
* Просмотр данных пациентов;
* Добавление в базу нового пациента;
* Изменение данных пациента;
* Удаление пациента из базы данных;
* Возможность выписать больничный;
* Возможность записать пациента на прием ко врачу.

**3.2 Условия выполнения программы**

**3.2.1 Климатические условия эксплуатации**

Климатические условия эксплуатации, при которых должны обеспечиваться заданные характеристики, должны удовлетворять требованиям, предъявляемым к техническим средствам в части условий их эксплуатации.

**3.2.2 Требования к составу и параметрам технических средств**

В состав технических средств должен входить IBM-совместимый персональный компьютер (ПЭВМ), включающий в себя:

* Процессор: Pentium 4 2.5 GHz (одноядерный) или выше;
* Свободное место на жестком диске: от 1 Гб;
* ОЗУ: 4 Гб и выше;
* ОС: Windows 10;

**3.2.3 Требования к программным средствам, используемым программой**

Исходные коды программы должны быть реализованы на языке C#. В качестве интеграционной среды разработки программы должна быть использована среда Microsoft Visual Studio 2022 и СУБД MS SQL.

**3.3 Требования к персоналу (пользователю)**

Минимальное количество персонала, требуемого для работы программы, должно составлять не менее 2 штатных единиц – системный администратор и пользователь программы – оператор.

Системный администратор должен иметь высшее профильное образование и сертификаты компании-производителя операционной системы. В перечень задач, выполняемых системным администратором, должны входить:

* задача поддержания работоспособности технических средств;
* задачи установки (инсталляции) и поддержания работоспособности системных программных средств – операционной системы;
* задача установки (инсталляции) программы.

Пользователь программы (оператор) должен обладать практическими навыками работы с графическим пользовательским интерфейсом операционной системы.

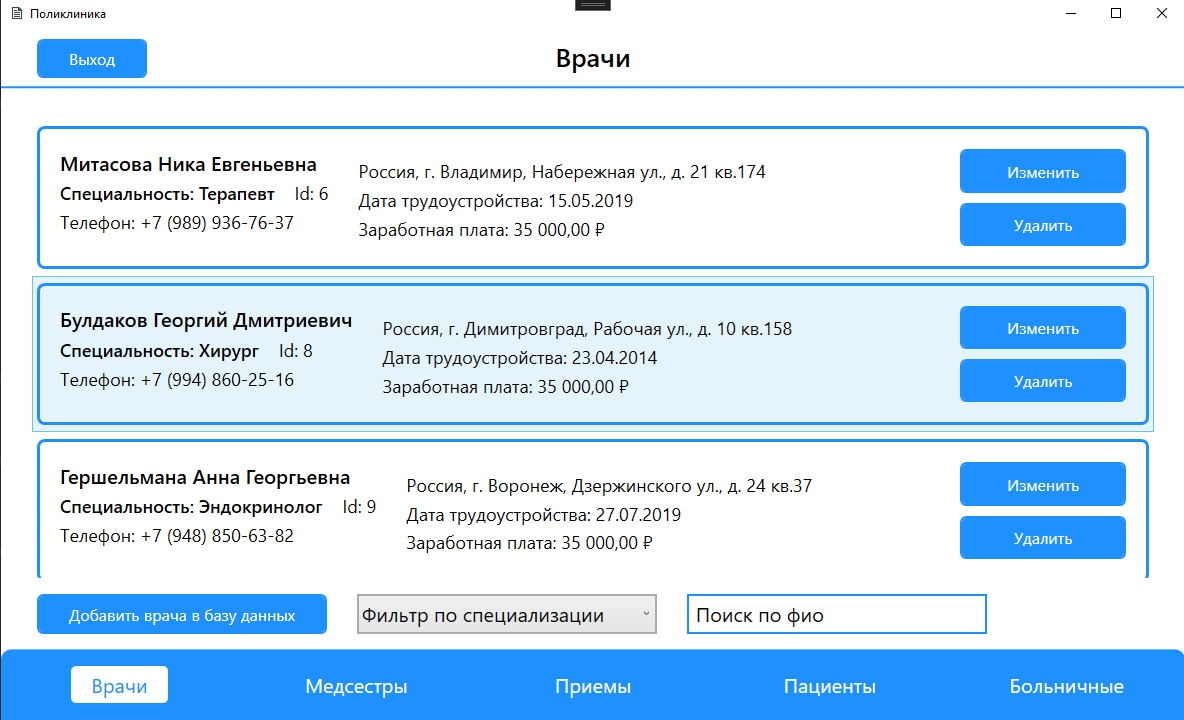
**3.4 Выполнение программы**

* Загрузка и запуск программы;
* Выполнение программы;
* Завершение работы программы.

**3.4.1 Загрузка и запуск программы**

Загрузка и запуск программы осуществляется способами, детальные сведения о которых изложены в руководстве пользователя операционной системы.

В случае успешного запуска программы на рабочем столе будет отображена стартовая страница приложения в соответствии с рисунком 6.

Рисунок 6 – Главное окно приложения.

**3.4.2 Выполнение программы**

**3.4.2.1 Страница врачей**

Стартовая страница – страница врачей. На ней отображены врачи поликлиники. Вы можете ускорить поиск нужного, используя фильтр по специализации или поиск по ФИО. Для того, чтобы добавить в базу нового врача следует нажать на кнопку «Добавить врача в базу данных», находящуюся в левом нижнем углу страницы. Откроется страница для ввода данных, соответствующая рисунку 7.

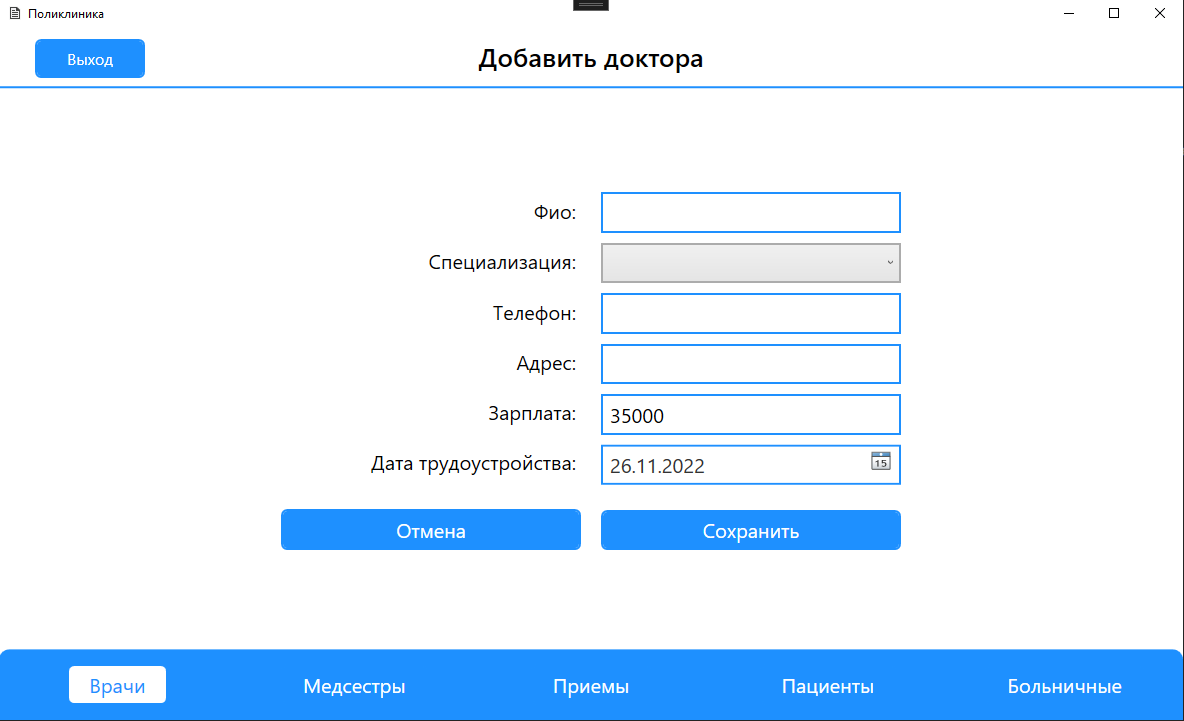
****

Рисунок 7 – Страница для ввода данных доктора.

После ввода корректных данных нажмите кнопу «Сохранить». Врач будет добавлен в базу данных, а приложение покажет страницу врачей. Нажатие кнопки изменить перенаправит вас на страницу изменения данных аналогичной странице, изображенной на рисунке 7.

**3.4.2.2 Страница медсестер**

Немаловажные сотрудники больницы – медсестры. Они собраны на странице медсестер, изображенной на рисунке 8. Вы можете ускорить поиск нужного сотрудника, используя поиск по ФИО или поиск по номеру кабинета, за которым закреплена медсестра.

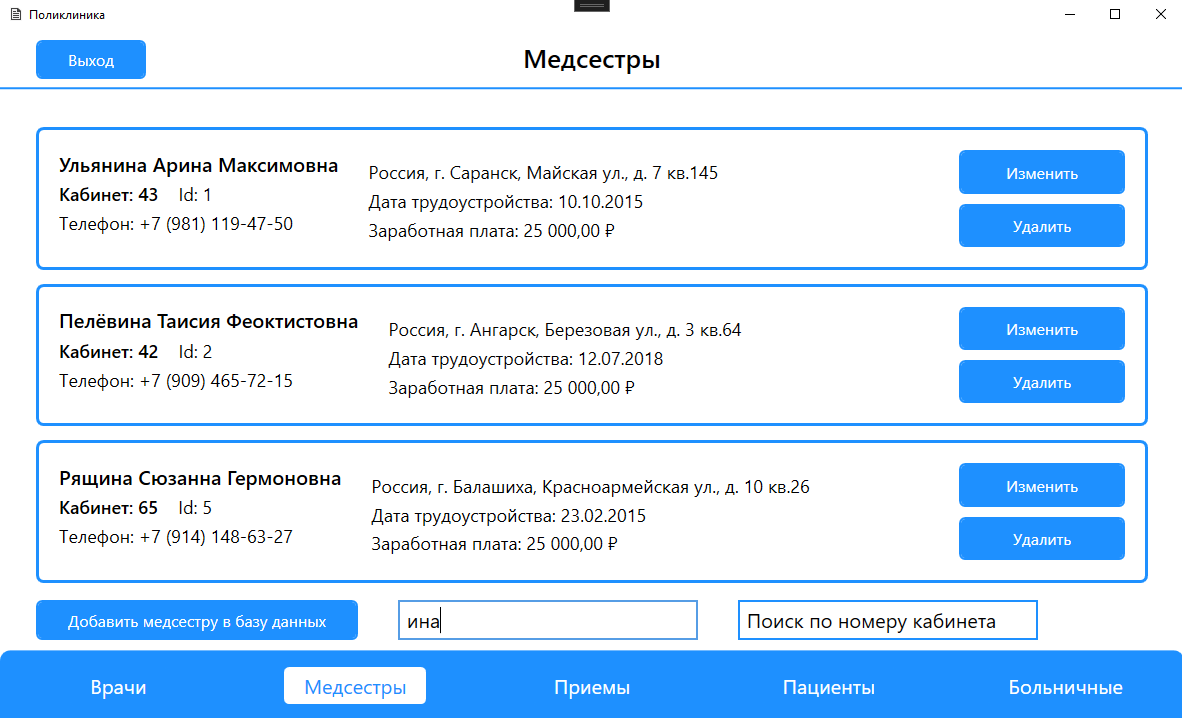


Рисунок 8 – Страница медсестер.

Для того, чтобы добавить в базу новую медсестру следует нажать на кнопку «Добавить медсестру в базу данных», находящуюся в левом нижнем углу страницы. Откроется страница для ввода данных, соответствующая рисунку 9.

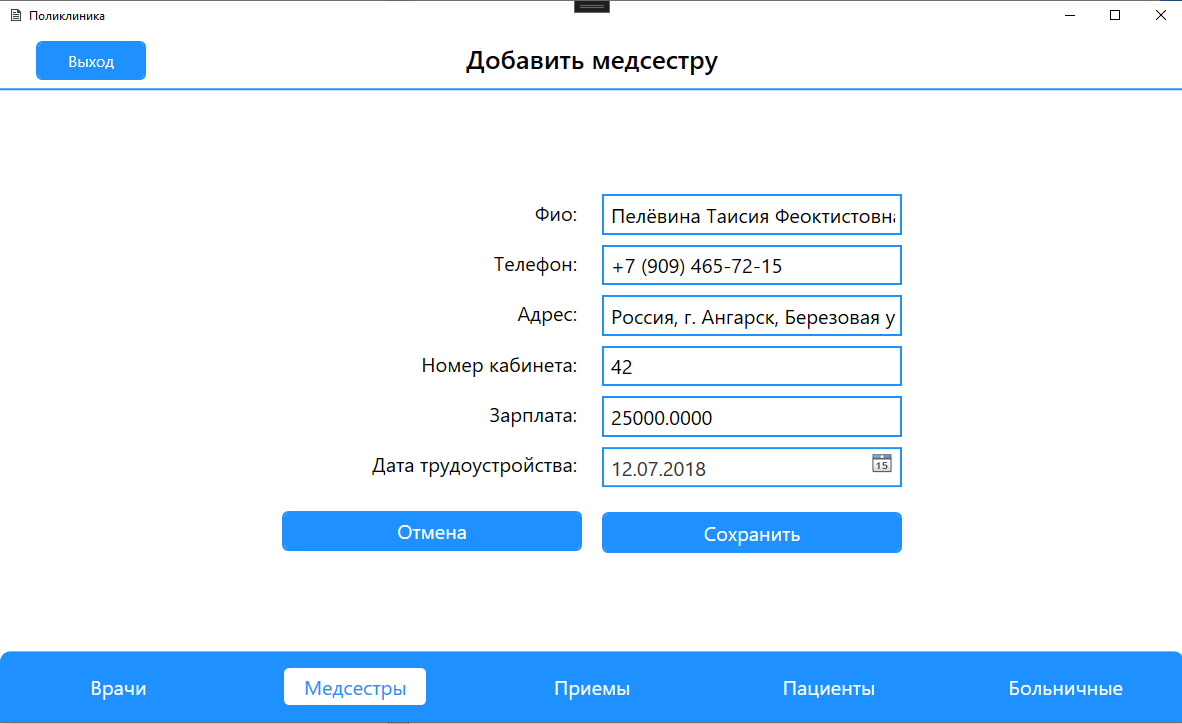
****

Рисунок 9 – Страница для ввода данных медсестры.

После ввода корректных данных нажмите кнопу «Сохранить». Медсестра будет добавлена в базу данных, а приложение покажет страницу медсестер. Нажатие кнопки изменить перенаправит вас на страницу изменения данных аналогичной странице, изображенной на рисунке 9.

**3.4.2.3 Страница пациентов**

Пациентов поликлиники можно найти на странице пациентов, изображенной на рисунке 10. Вы можете ускорить поиск нужного пациента, используя поиск по ФИО.

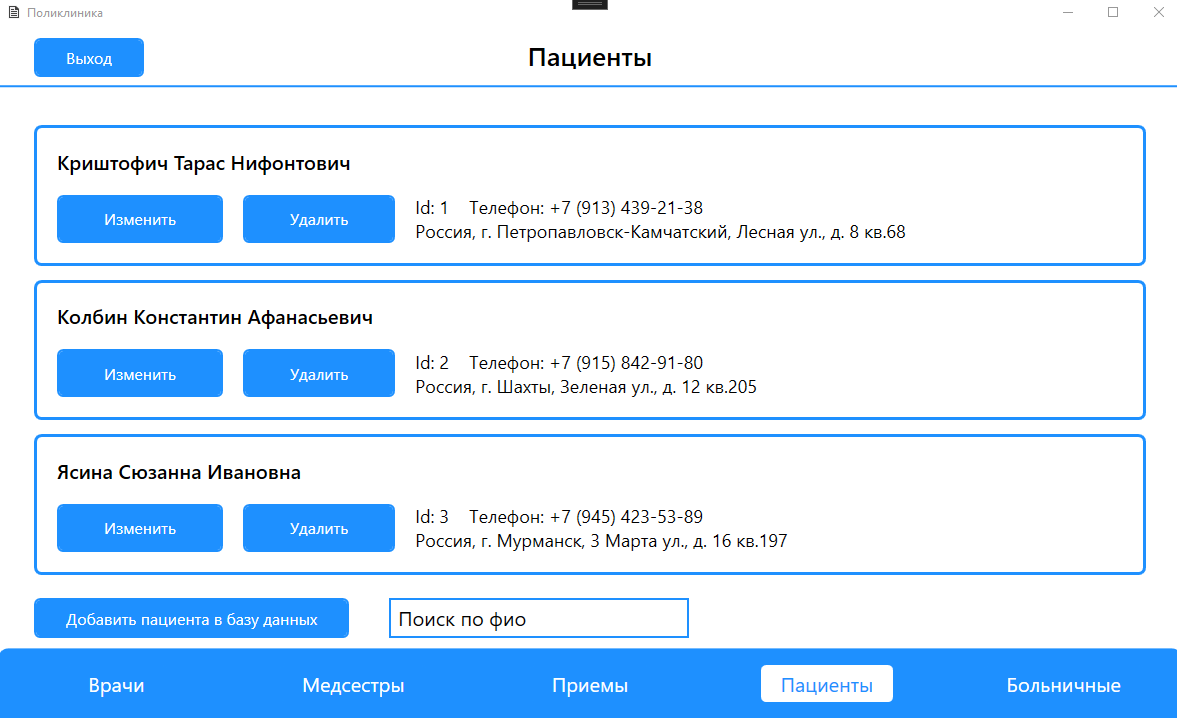


Рисунок 10 – Страница пациентов.

Для того, чтобы добавить в базу нового пациента следует нажать на кнопку «Добавить пациента в базу данных», находящуюся в левом нижнем углу страницы. Откроется страница для ввода данных, соответствующая рисунку 11.

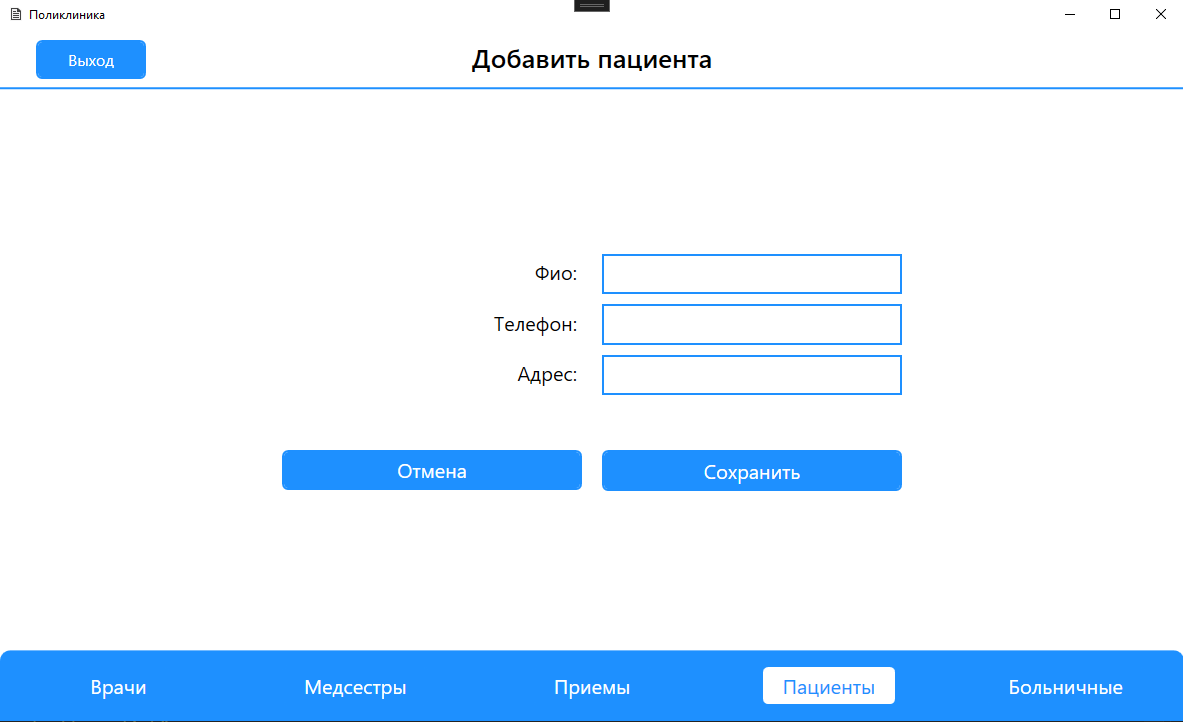
****

Рисунок 11 – Страница для ввода данных пациента.

После ввода корректных данных нажмите кнопу «Сохранить». Пациент будет добавлен в базу данных, а приложение покажет страницу пациентов. Нажатие кнопки изменить перенаправит вас на страницу изменения данных аналогичной странице, изображенной на рисунке 11.

**3.4.2.4 Страница приемов**

На странице приемов, изображенной на рисунке 10, собраны все записи на прием в поликлинику. Вы можете ускорить поиск нужной записи, используя поиск по дате и по специализации врача.

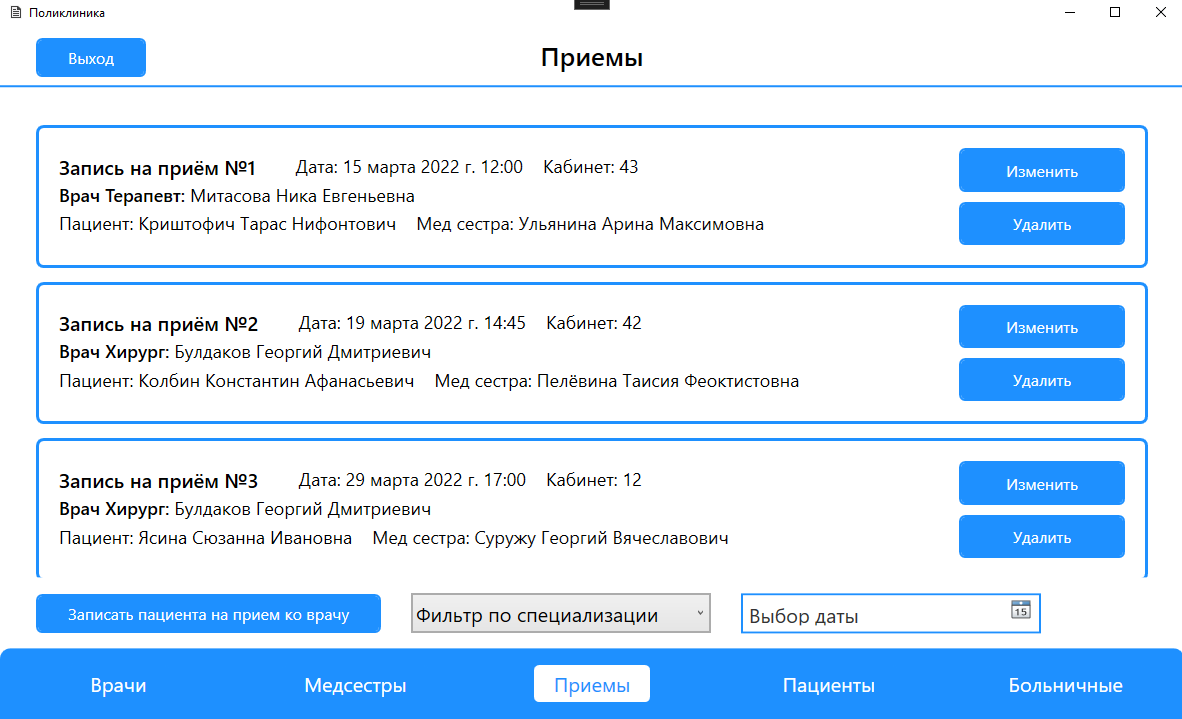


Рисунок 12 – Страница приемов.

Для того, чтобы добавить в базу новую запись следует нажать на кнопку «Записать пациента на прием ко врачу», находящуюся в левом нижнем углу страницы. Откроется страница для ввода данных, соответствующая рисунку 13.

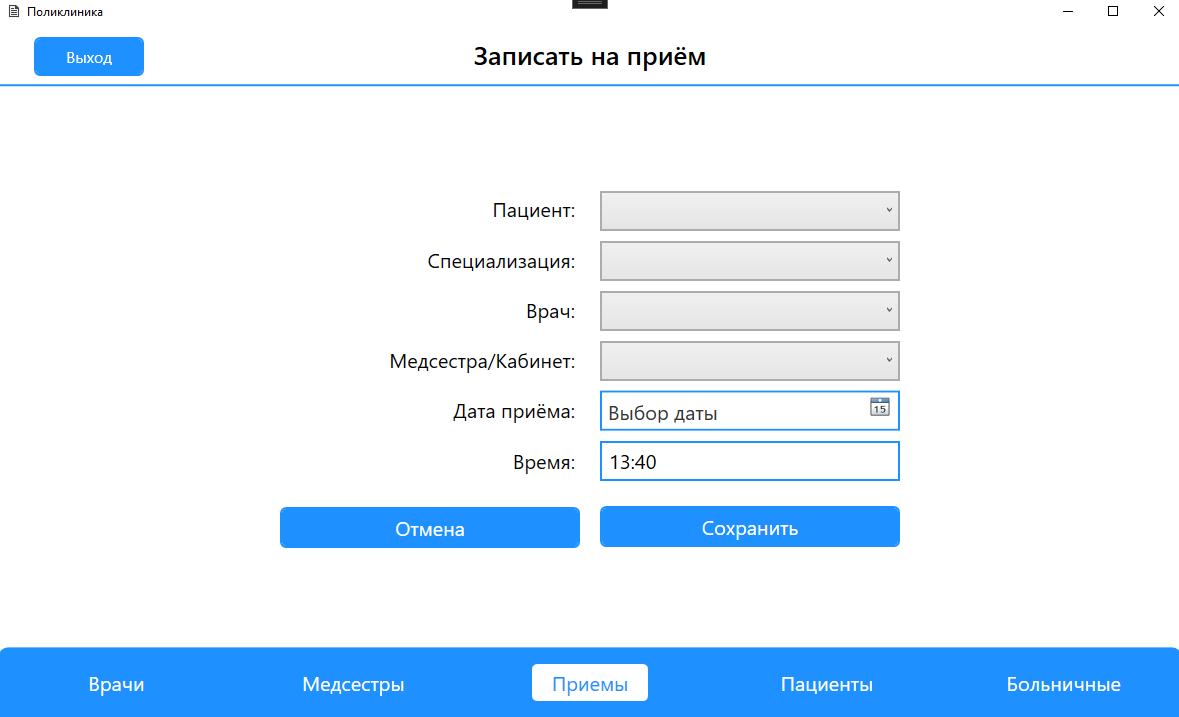
****

Рисунок 13 – Страница записи на прием.

После ввода корректных данных нажмите кнопу «Сохранить». Запись будет добавлена в базу данных, а приложение покажет страницу приемов. Нажатие кнопки изменить перенаправит вас на страницу изменения данных аналогичной странице, изображенной на рисунке 13.

**3.4.2.5 Страница больничных**

На странице больничных, изображенной на рисунке 14, собраны все записи о больничных. Вы можете ускорить поиск нужной записи, используя поиск номеру больничного.

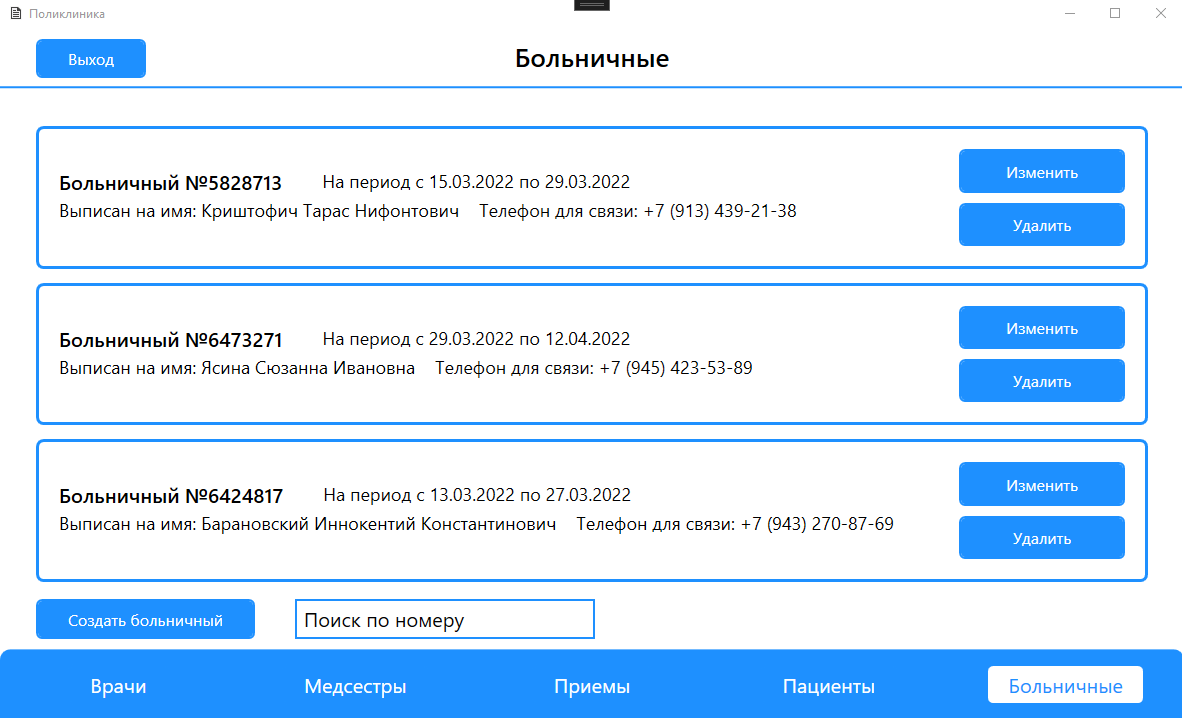


Рисунок 14 – Страница больничных.

Для того, чтобы добавить в базу новую запись следует нажать на кнопку «Создать больничный», находящуюся в левом нижнем углу страницы. Откроется страница для ввода данных, соответствующая рисунку 15.

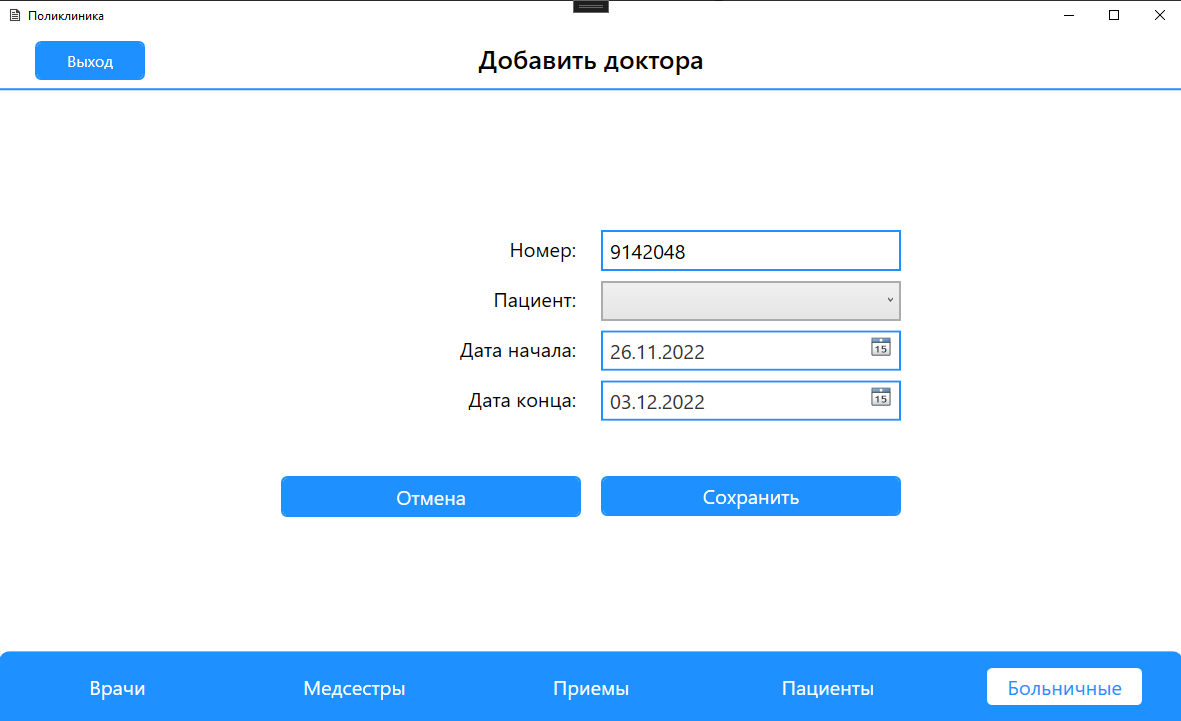
****

Рисунок 15 – Страница создания больничного.

После ввода корректных данных нажмите кнопу «Сохранить». Запись будет добавлена в базу данных, а приложение покажет страницу больничных. Нажатие кнопки изменить перенаправит вас на страницу изменения данных аналогичной странице, изображенной на рисунке 15.

**3.5 Сообщения оператору**

**3.5.1 Ошибка в веденных данных**

Такое сообщение появляется при попытке сохранить запись с некорректными данными. Ошибка подсказывает пользователю в каком поле он ошибся. Пример сообщения показан на рисунке 16.

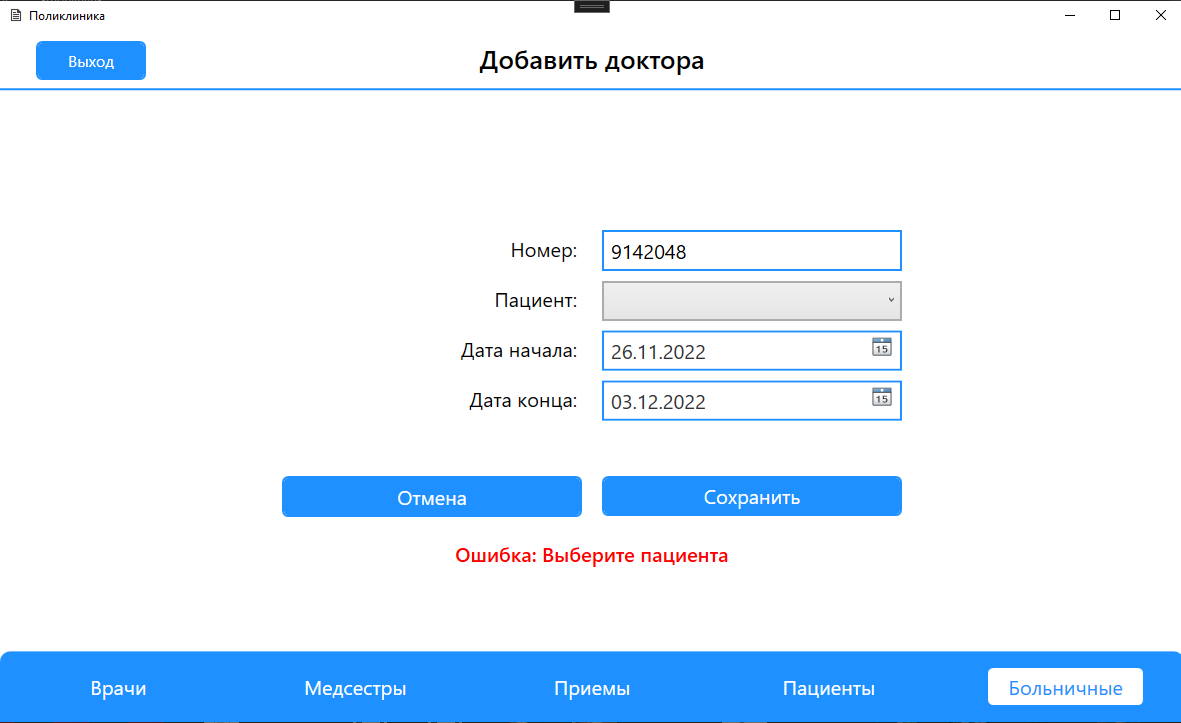


Рисунок 16 – Сообщение оператору с ошибкой.

**3.5.1 Подтверждение удаления записи**

Такое сообщение появляется при попытке удалить запись в любой таблице. Чтобы удалить запись, нажмите кнопку «Да» в диалоговом окне. Пример сообщения показан на рисунке 17.

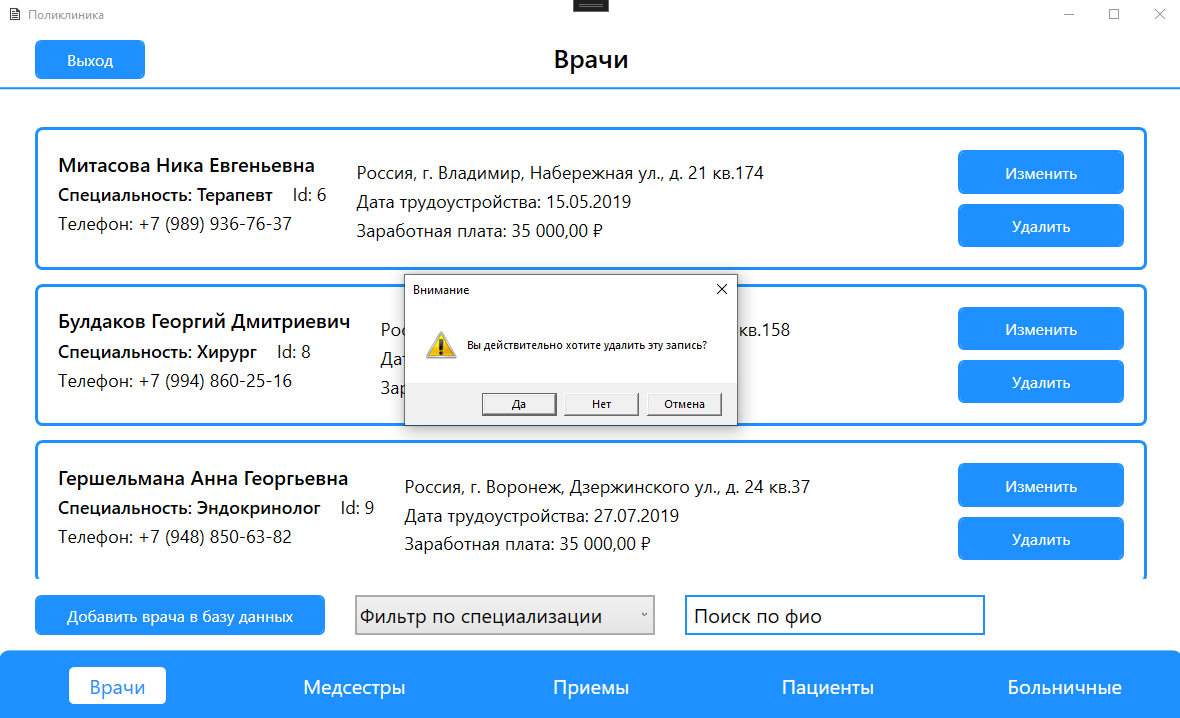


Рисунок 17 – Сообщению подтверждение удаление записи.

**4 Тестирование программы**

## 4.1 Программа, методика и результаты испытаний ГОСТ 19.301-79

### 4.1.1 Объект испытаний

#### 4.1.1.1 Наименование программы

Наименование программы – «Polyclinic».

#### 4.1.1.2 Назначение разработки

Функциональным назначением программы является работа с данными.

### 4.1.2 Цель испытаний

Цель проведения испытаний - проверка соответствия характеристик разработанной программы (программного изделия) функциональным и отдельным иным видам требований, изложенным в документе Техническое задание.

#### 4.1.2.1 Основания для проведения испытаний

Основанием для проведения испытаний является задание на курсовой проект.

#### 4.1.2.2 Место и продолжительность испытаний

Дом, Колледж. Срок 3 месяца.

#### 4.1.2.3 Перечень документов, предъявляемых на испытания

Состав программной документации должен включать в себя:

* Техническое задание на разработку ГОСТ 19.201-78
* Руководство оператора ГОСТ 19.505-79
* Программа, методика и результаты испытаний ГОСТ 19.301-79

#### 4.1.2.4 Объем испытаний

**Перечень этапов испытаний**

Испытания проводятся в два этапа:

* ознакомительный;
* испытания.

##### 4.1.2.4.1 Перечень проверок, проводимых на 1 этапе испытаний

Перечень проверок, проводимых на 1-м этапе испытаний, должен включать в себя:

* проверку комплектности программной документации;
* проверку комплектности состава технических и программных средств.

##### 4.1.2.4.2 Перечень проверок, проводимых на 2 этапе испытаний

Перечень проверок, проводимых на 2-м этапе испытаний, должен включать в себя:

* проверку соответствия технических характеристик программы;
* проверку степени выполнения требований функционального назначения программы.

##### 4.1.2.4.3 Количественные и качественные характеристики, подлежащие оценке

В ходе проведения приемосдаточных испытаний оценке подлежат качественные (функциональные) характеристики программы. Проверке подлежит возможность выполнения программой перечисленных ниже функций:

* Просмотр данных врачей;
* Добавление в базу нового врача;
* Изменение данных врача;
* Удаление врача из базы данных;
* Просмотр данных медсестер;
* Добавление в базу нового медсестры;
* Изменение данных медсестры;
* Удаление медсестры из базы данных;
* Просмотр данных пациентов;
* Добавление в базу нового пациента;
* Изменение данных пациента;
* Удаление пациента из базы данных;
* Возможность выписать больничный;
* Возможность записать пациента на прием ко врачу.

### 4.1.3 Требования к программе

При проведении испытаний функциональные характеристики (возможности) программы подлежат проверке на соответствие требованиям, изложенным в п. «Требования к функциональным выполняемых функций» технического задания.

### 4.1.4 Требования к программной документации

Состав программной документации должен включать в себя:

* Техническое задание на разработку ГОСТ 19.201-78
* Руководство оператора ГОСТ 19.505-79
* Программа, методика и результаты испытаний ГОСТ 19.301-79

### 4.1.5 Средства и порядок испытаний

#### 4.1.5.1 Технические средства, используемые во время испытаний

В состав технических средств должен входить IBM-совместимый персональный компьютер, включающий в себя:

* Процессор: Pentium 4 2.5 GHz (одноядерный) или выше;
* Свободное место на жестком диске: от 1 Гб;
* ОЗУ: 4 Гб и выше;
* ОС: Windows 10;

##### 4.1.5.1.1 Порядок проведения испытаний

Испытания должны проводиться поэтапно согласно п. «Перечень этапов испытаний» настоящего документа.

##### 4.1.5.2 Условия и порядок проведения испытаний

##### 4.1.5.2.1 Условия проведения испытаний

Испытания должны проводиться в нормальных климатических условиях по ГОСТ 22261-94. Условия проведения испытаний приведены ниже:

* температура окружающего воздуха, °С - 20 ± 5;
* относительная влажность, % - от 30 до 80;
* атмосферное давление, кПа - от 84 до 106;
* частота питающей электросети, Гц - 50 ± 0,5;
* напряжение питающей сети переменного тока, В - 220 ± 4,4.

##### 4.1.5.2.2 Условия начала и завершения отдельных этапов испытаний

Необходимым и достаточным условием завершения 1 этапа испытаний и начала 2 этапа испытаний является успешное завершение проверок, проводимых на 1 этапе (см. п. «Перечень проверок, проводимых на 1 этапе испытаний»).

Условием завершения 2 этапа испытаний является успешное завершение проверок, проводимых на 2 этапе испытаний.

##### 4.1.5.2.3 Требования к техническому обслуживанию

Требования к техническому обслуживанию не предъявляются.

**4.1.6. Методика испытаний**

**Методы проведения проверки комплектности программной документации**

Проверка комплектности программной документации на программное изделие производится визуально преподавателем. В ходе проверки сопоставляется состав и комплектность программной документации, представленной исполнителем, с перечнем программной документации.

Проверка считается завершенной в случае соответствия состава и комплектности программной документации, представленной исполнителем, перечню программной документации, приведенному в указанном выше пункте.

**Методы проведения проверки степени** **выполнения требований функционального назначения программы**

Проверка выполнения требований функционального назначения производится методом тестирования программного продукта. Тестирование будет проводится методом белого ящика по критерию C1 или условия критерия тестирования ветвей. Набор тестов в совокупности должен обеспечить прохождение каждого критерия не менее одного раза. Но применяется при тестировании ветвей. Это достаточно экономичный критерий, поскольку множество ветвей в тестируемом приложении конечно и не так велико.

## 4.2 Результаты испытаний

Результаты предварительного тестирования в соответствии с   
таблицей 4.1.

Таблица 4.1 **- Функциональное тестирование «белого ящика»**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **Тестовая спецификация** | | | | |  | | |
| **Разработано:**  Потапкин П.А. | | **Начальные установки:** нет | | | | | |
| **Источник тестовых данных:** | | **Цели:**  Проверки степени выполнения требований функционального назначения программы | | | | | |
| **Test Case #** | **Описание** | **Шаги**  **теста** | **Ожидаемые**  **результаты** | **Реальные результаты** | | **Прошел**  **/Провалился** | **Тестер**  **/Дата** |
| **Тестирование функций страницы врачей** | | | | | | | |
| 1 | Тестирование функциональности добавления врача | Нажать кнопку «Добавить врача в базу данных», ввести данные, нажать кнопку «Сохранить» | Появление новой записи в списке врачей | Появление новой записи в списке врачей | | Прошел | Челяев Н.Ю.  20.11.2022 |
| 2 | Тестирование функциональности изменения данных врача | Нажать кнопку «Изменить», изменить данные, нажать кнопку «Сохранить» | Изменение данных записи в списке врачей | Изменение данных записи в списке врачей | | Прошел | Челяев Н.Ю.  20.11.2022 |
| 3 | Тестирование функциональности удаления врача | Нажать кнопку «Удалить», подтвердить удаление | Исчезновение записи из списка врачей | Исчезновение записи из списка врачей | | Прошел | Челяев Н.Ю.  20.11.2022 |
| 4 | Тестирование функциональности кнопки «Выход» | Нажатие на кнопку «Выход» | Закрытие программы | Закрытие программы | | Прошел | Челяев Н.Ю.  20.11.2022 |
| **Тестирование функций поиска и фильтрации** | | | | | | | |
| 5 | Тестирование функциональности поиска по ФИО | Открыть таблицу врачей, ввести ФИО существующего в базе врача | Отображение врача с совпавшим ФИО | Отображение врача с совпавшим ФИО | | Прошел | Челяев Н.Ю.  20.11.2022 |
| 6 | Тестирование функциональности фильтра по специализации | Открыть таблицу приемов, выбрать специализацию в выпадающем списке | Отображение приемов, имеющих выбранную специализацию | Отображение приемов, имеющих выбранную специализацию | | Прошел | Челяев Н.Ю.  20.11.2022 |
| 7 | Тестирование функциональности поиска по номеру больничного | Открыть таблицу больничных, ввести номер больничного существующего в базе врача | Отображение больничного с совпавшим номером | Отображение больничного с совпавшим номером | | Прошел | Челяев Н.Ю.  20.11.2022 |
| **Тестирование функции «Запись на прием ко врачу»** | | | | | | | |
| 8 | Тестирования функциональности записи на прием ко врачу | Нажатие на кнопку «Записать пациента на прием ко врачу», ввести данные, нажать на кнопку «Сохранить» | Появление новой записи | Появление новой записи | | Прошел | Челяев Н.Ю.  20.11.2022 |
| 9 | Тестирования функциональности записи на прием ко врачу, который занят | Нажатие на кнопку «Записать пациента на прием ко врачу», ввести данные, указать занятое время, нажать на кнопку «Сохранить» | Появление ошибки «На данное время данный врач занят» | Появление ошибки «На данное время данный врач занят» | | Прошел | Челяев Н.Ю.  20.11.2022 |

**ЗАКЛЮЧЕНИЕ**

При выполнении курсового проекта удалось достигнуть всех поставленных целей. Я получила новые знания и навыки в разработке приложений, взаимодействующих с базами данных, а также систематизировал старые. У меня получилось создать удобный и красивый проект, подходящий для эксплуатации. Приложение позволяет структурно хранить данные поликлиники. Я изучил инфологическое устройство поликлиники и расширил свои знания в проектировании устойчивых баз данных. Грамотная структура базы обеспечивают легкую масштабируемость для будущего расширения. Были реализованы все поставленные задачи обработки информации о пациентах, врачах, медсёстрах, приёмах и т.д.

# СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. ГОСТ 19.201-78. Единая система программной документации. Техническое задание, требование к содержанию и оформлению [Текст]. – Введ. 2002-07-01. – М.: ФГУП «СТАНДАРТИНФОРМ», 2008. – 27с.
2. ГОСТ 19.505-79. Единая система программной документации. Руководство оператора. Требования к содержанию и оформлению [Текст]. – Введ. 2002-07-01. – М.: ФГУП «СТАНДАРТИНФОРМ», 2008. – 27с.
3. ГОСТ 19.301-79. Единая система программной документации. Программа и методика испытаний. Требования к содержанию и оформлению [Текст]. – Введ. 2002-07-01. – М.: ФГУП «СТАНДАРТИНФОРМ», 2008. – 27с.
4. ГОСТ 7.32-2001 СИБИД. Отчет о научно-исследовательской работе. Структура и правила оформления [Текст]. – Введ. 2002-07-01. – М.: ФГУП «СТАНДАРТИНФОРМ», 2008. – 27с.
5. ГОСТ 7.1-2003 СИБИД. Библиографическая запись. Библиографическое описание. Общие требования и правила составления [Текст]. – Введ. 2003-07-02. – М.: ФГУП «СТАНДАРТИНФОРМ», 2010. – 49с.
6. Разработка Android-приложений на C# с использованием Xamarin c нуля [Текст] Е.Д. Умрихин – СПб,: БВХ-Петербург, 2021. – 336 с.
7. Шарп, Джон Microsoft Visual C#. Подробное руководство [Текст] : Практическое руководство / Джон Шарп. – 8-е изд., испр. И доп. – СПб.: Издательский дом «Питер», 2017. – 848с.
8. Изучаем SQL. – Пер. с англ. [Текст] – СПб: Символ-Плюс, 2007. – 312 с., ил.
9. Грофф, Джеймс Р., Вайнберг, Пол Н., Оппель, Эндрю Дж.   
   SQL: Полное руководство, 3-е изд [Текст] : Пер. с англ. – СПб. : ООО «Диалектика», 2019. – 960 с. : ил.

# ПРИЛОЖЕНИЕ А

**Листинг** **программы по модулям:**

**Листинг 1 - DoctorsPage**

public partial class DoctorsPage : Page

{

private readonly DbEntities \_db = DbEntities.GetContext();

public DoctorsPage()

{

InitializeComponent();

}

/// <summary>

/// Вызывается при переходе на страницу

/// </summary>

private void Page\_Loaded(object sender, System.Windows.RoutedEventArgs e)

{

//Загружаем таблицу врачей и специализаций и выводим их на форму

\_db.Doctors.Load();

\_db.Specializations.Load();

//Заполняем combobox специализациями

var listSpecs = \_db.Specializations.ToList();

//Вставляем первым элементом заглушку для сброса фильтра

listSpecs.Insert(0, new Specialization() { Title = "Фильтр по специализации" });

cmbSpecFilter.ItemsSource = listSpecs;

//Выбираем заглушку, при которой фильтр отключен

cmbSpecFilter.SelectedIndex = 0;

}

private void AddButton\_Click(object sender, System.Windows.RoutedEventArgs e)

{

//Создаем нового врача, заполняем значениями по умолчанию

Doctor doctor = new Doctor()

{

Worker = new Worker()

{

Salary = 35000,

DateEnployment = DateTime.Now.Date

}

};

//Переходим на страницу заполнения

this.NavigationService.Navigate(new ExtensionPages.AddDoctorPage(doctor));

}

private void EditButton\_Click(object sender, System.Windows.RoutedEventArgs e)

{

//Получаем врача из кнопки

var doctor = (sender as Button).DataContext as Doctor;

//Переходим на страницу заполнения

this.NavigationService.Navigate(new ExtensionPages.AddDoctorPage(doctor));

}

private void DeleteButton\_Click(object sender, System.Windows.RoutedEventArgs e)

{

var asnwer = MessageBox.Show("Вы действительно хотите удалить эту запись?", "Внимание", MessageBoxButton.YesNoCancel, MessageBoxImage.Warning);

if (asnwer != MessageBoxResult.Yes) return;

//Получаем врача из кнопки

var doctor = (sender as Button).DataContext as Doctor;

//Удаляем и сохраняем

\_db.Workers.Remove(doctor.Worker);

\_db.Doctors.Remove(doctor);

\_db.SaveChanges();

}

/// <summary>

/// Фильтр по специализациям

/// </summary>

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

{

if (cmbSpecFilter.SelectedItem == null) return;

var spec = cmbSpecFilter.SelectedItem as Specialization; //Получаем выбранную специализацию

if (spec.Title == "Фильтр по специализации") //Если это заглушка, выводим все записи

{

collectionView.ItemsSource = \_db.Doctors.Local.ToBindingList();

return;

}

//Если это НЕ заглушка, проводим выборку по специализациям

collectionView.ItemsSource = \_db.Doctors.Local.Where(p => p.Specialization == spec).ToList();

txtFind.Text = "Поиск по фио"; //Сбрасываем фильтр по фио

}

/// <summary>

/// Поиск по фио

/// </summary>

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

{

if (txtFind.Text == null) return;

if (collectionView.ItemsSource == null) return;

if (txtFind.Text == "Поиск по фио") return; //Если это заглушка, ничего не делаем

//Получаем записи из списка на форме и преобразуем их в List

var entities = collectionView.ItemsSource;

List<Doctor> list = entities.Cast<Doctor>().ToList();

//Если поиск пуст, выводим все записи выбранной специальности, принудительно вызвав фильтр

if (string.IsNullOrEmpty(txtFind.Text))

{

CmbSpecFilter\_SelectionChanged(null, null);

return;

}

//Если это НЕ заглушка, проводим выборку по фио

collectionView.ItemsSource = list.Where(p => p.Worker.FIO.Contains(txtFind.Text));

}

}

**Листинг 2 - NursesPage**

public partial class HouseInfoPage : Page

public partial class NursesPage : Page

{

private readonly DbEntities \_db = DbEntities.GetContext();

public NursesPage()

{

InitializeComponent();

}

/// <summary>

/// Вызывается при переходе на страницу

/// </summary>

private void Page\_Loaded(object sender, System.Windows.RoutedEventArgs e)

{

//Загружаем таблицу медсестер и выводим их на форму

\_db.Nurses.Load();

collectionView.ItemsSource = \_db.Nurses.Local.ToBindingList();

//Сбрасываем фильтры

txtFindFio.Text = "Поиск по фио";

txtFindOffice.Text = "Поиск по номеру кабинета";

}

private void AddButton\_Click(object sender, System.Windows.RoutedEventArgs e)

{

//Создаем новую медсестру, заполняем значениями по умолчанию

Nurse nurse = new Nurse()

{

Worker = new Worker()

{

Salary = 15000,

DateEnployment = DateTime.Now.Date

}

};

//Переходим на страницу заполнения

this.NavigationService.Navigate(new ExtensionPages.AddNursePage(nurse));

}

private void EditButton\_Click(object sender, System.Windows.RoutedEventArgs e)

{

//Получаем медсестру из кнопки

var nurse = (sender as Button).DataContext as Nurse;

//Переходим на страницу заполнения

this.NavigationService.Navigate(new ExtensionPages.AddNursePage(nurse));

}

private void DeleteButton\_Click(object sender, System.Windows.RoutedEventArgs e)

{

var asnwer = MessageBox.Show("Вы действительно хотите удалить эту запись?", "Внимание", MessageBoxButton.YesNoCancel, MessageBoxImage.Warning);

if (asnwer != MessageBoxResult.Yes) return;

//Получаем медсестру из кнопки

var nurse = (sender as Button).DataContext as Nurse;

//Удаляем и сохраняем

\_db.Workers.Remove(nurse.Worker);

\_db.Nurses.Remove(nurse);

\_db.SaveChanges();

}

/// <summary>

/// Поиск по фио

/// </summary>

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

{

if (txtFindFio == null) return;

if (txtFindOffice == null) return;

if (txtFindFio.Text == "Поиск по фио") return; //Если это заглушка, ничего не делаем

txtFindOffice.Text = "Поиск по номеру кабинета"; //Сбрасываем другой поиск

if (string.IsNullOrEmpty(txtFindFio.Text)) //Если текстбокс пустой, выводим все записи

{

collectionView.ItemsSource = \_db.Nurses.Local.ToBindingList();

return;

}

List<Nurse> list = collectionView.ItemsSource?.Cast<Nurse>().ToList();

if (list == null || list.Count == 0) list = \_db.Nurses.Local.ToList();

collectionView.ItemsSource = list.Where(p => p.Worker.FIO.Contains(txtFindFio.Text));

}

/// <summary>

/// Поиск по номеру кабинета

/// </summary>

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

{

if (txtFindOffice == null) return;

if (txtFindFio == null) return;

if (txtFindOffice.Text == "Поиск по номеру кабинета") return;

txtFindFio.Text = "Поиск по фио";

if (string.IsNullOrEmpty(txtFindOffice.Text))

{

collectionView.ItemsSource = \_db.Nurses.Local.ToBindingList();

return;

}

//Если список пустой, для выборки наполним его записями из таблицы

List<Nurse> list = collectionView.ItemsSource?.Cast<Nurse>().ToList();

if (list == null || list.Count == 0) list = \_db.Nurses.Local.ToList();

//Проводим выборку по фио

collectionView.ItemsSource = list.Where(p => p.Office.Value.ToString() == txtFindOffice.Text);

}

}

**Листинг 3 - AppoitmentsPage**

public partial class AppoitmentsPage : Page

{

private readonly DbEntities \_db = DbEntities.GetContext();

public AppoitmentsPage()

{

InitializeComponent();

}

/// <summary>

/// Вызывается при переходе на страницу

/// </summary>

private void Page\_Loaded(object sender, System.Windows.RoutedEventArgs e)

{

//Загружаем таблицу приемов и выводим ее на форму

\_db.Appointments.Load();

collectionView.ItemsSource = \_db.Appointments.Local.ToBindingList();

//Заполняем combobox специализациями

var listSpecs = \_db.Specializations.ToList();

//Вставляем первым элементом заглушку для сброса фильтра

listSpecs.Insert(0, new Specialization() { Title = "Фильтр по специализации" });

cmbSpecFilter.ItemsSource = listSpecs;

//Выбираем заглушку, при которой фильтр отключен

cmbSpecFilter.SelectedIndex = 0;

}

private void AddButton\_Click(object sender, System.Windows.RoutedEventArgs e)

{

//Создаем новый прием

Appointment appointment = new Appointment();

//Переходим на страницу заполнения

this.NavigationService.Navigate(new ExtensionPages.AddAppointmentPage(appointment));

}

private void EditButton\_Click(object sender, System.Windows.RoutedEventArgs e)

{

//Получаем прием из кнопки

var appointment = (sender as Button).DataContext as Appointment;

//Переходим на страницу заполнения

this.NavigationService.Navigate(new ExtensionPages.AddAppointmentPage(appointment));

}

private void DeleteButton\_Click(object sender, System.Windows.RoutedEventArgs e)

{

var asnwer = MessageBox.Show("Вы действительно хотите удалить эту запись?", "Внимание", MessageBoxButton.YesNoCancel, MessageBoxImage.Warning);

if (asnwer != MessageBoxResult.Yes) return;

//Получаем прием из кнопки

var appointment = (sender as Button).DataContext as Appointment;

//Удаляем и сохраняем

\_db.Appointments.Remove(appointment);

\_db.SaveChanges();

}

/// <summary>

/// Фильтр по специализациям

/// </summary>

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

{

if (cmbSpecFilter.SelectedItem == null) return;

var spec = cmbSpecFilter.SelectedItem as Specialization; //Получаем выбранную специализацию

if (spec.Title == "Фильтр по специализации") //Если это заглушка, выводим все записи

{

collectionView.ItemsSource = \_db.Appointments.Local.ToBindingList();

return;

}

//Если это НЕ заглушка, проводим выборку по специализациям

collectionView.ItemsSource = \_db.Appointments.Local.Where(p => p.Doctor.Specialization == spec).ToList();

if (txtFind.SelectedDate != null) //Сбрасываем фильтр по дате

{

txtFind.SelectedDate = null;

}

}

/// <summary>

/// Фильтр по дате приема

/// </summary>

private void TxtFind\_SelectedDateChanged(object sender, SelectionChangedEventArgs e)

{

if (txtFind == null) return;

//Если ничего не выбрано, или список на форме пустой, обновим его принудительным вызовом фильтра

if (txtFind.Text == string.Empty || collectionView.ItemsSource.Cast<Appointment>().Count() == 0)

{

CmbSpecFilter\_SelectionChanged(null, null);

return;

}

//Получаем записи из списка на форме и преобразуем их в List

var entities = collectionView.ItemsSource;

List<Appointment> list = entities.Cast<Appointment>().ToList();

//Проводим выборку по дате

collectionView.ItemsSource = list.Where(p => p.Date.Value.Date == txtFind.SelectedDate.Value.Date);

}

}

**Листинг 4 - PatientsPage**

public partial class PatientsPage : Page

{

private readonly DbEntities \_db = DbEntities.GetContext();

public PatientsPage()

{

InitializeComponent();

}

/// <summary>

/// Вызывается при переходе на страницу

/// </summary>

private void Page\_Loaded(object sender, System.Windows.RoutedEventArgs e)

{

//Загружаем таблицу паицентов и выводим их на форму

\_db.MedicalCards.Load();

collectionView.ItemsSource = \_db.MedicalCards.Local.ToBindingList();

txtFindFio.Text = "Поиск по фио"; //Сбрасываем фильтр

}

private void AddButton\_Click(object sender, System.Windows.RoutedEventArgs e)

{

//Создаем нового врача

MedicalCard patient = new MedicalCard();

//Переходим на страницу заполнения

this.NavigationService.Navigate(new ExtensionPages.AddPatientPage(patient));

}

private void EditButton\_Click(object sender, System.Windows.RoutedEventArgs e)

{

//Получаем пациента из кнопки

var patient = (sender as Button).DataContext as MedicalCard;

//Переходим на страницу заполнения

this.NavigationService.Navigate(new ExtensionPages.AddPatientPage(patient));

}

private void DeleteButton\_Click(object sender, System.Windows.RoutedEventArgs e)

{

var asnwer = MessageBox.Show("Вы действительно хотите удалить эту запись?", "Внимание", MessageBoxButton.YesNoCancel, MessageBoxImage.Warning);

if (asnwer != MessageBoxResult.Yes) return;

//Получаем пациента из кнопки

var patient = (sender as Button).DataContext as MedicalCard;

//Удаляем и сохраняем

\_db.MedicalCards.Remove(patient);

\_db.SaveChanges();

}

/// <summary>

/// Поиск по фио

/// </summary>

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

{

if (txtFindFio == null) return;

if (txtFindFio.Text == "Поиск по фио") return; //Если это заглушка, ничего не делаем

//Если пустой текстбокс, выводим все записи

if (!string.IsNullOrEmpty(txtFindFio.Text)) collectionView.ItemsSource = \_db.MedicalCards.Local.ToBindingList();

//Если список пустой, для выборки наполним его записями из таблицы

List<MedicalCard> list = collectionView.ItemsSource?.Cast<MedicalCard>().ToList();

if (list == null || list.Count == 0) list = \_db.MedicalCards.Local.ToList();

//Проводим выборку по фио

collectionView.ItemsSource = list.Where(p => p.FIO.Contains(txtFindFio.Text));

}

}

**Листинг 5 - SicksPage**

public partial class SicksPage : Page

{

private readonly DbEntities \_db = DbEntities.GetContext();

public SicksPage()

{

InitializeComponent();

}

/// <summary>

/// Вызывается при переходе на страницу

/// </summary>

private void Page\_Loaded(object sender, System.Windows.RoutedEventArgs e)

{

\_db.Sicks.Load();

collectionView.ItemsSource = \_db.Sicks.Local.ToBindingList();

txtFind.Text = "Поиск по номеру";

}

private void AddButton\_Click(object sender, System.Windows.RoutedEventArgs e)

{

//Генерируем случайный номер для боличного, проверяем на уникальность

Random random = new Random();

int sickNumber = 0;

do

{

sickNumber = random.Next(1231981, 9982794);

} while (\_db.Sicks.Count(p => p.Number == sickNumber) != 0);

//Создаем больничный, заполняем значениями по умолчанию

Sick sick = new Sick()

{

Number = sickNumber,

DateStart = DateTime.Now,

DateEnd = DateTime.Now.AddDays(7),

};

//Переходим на страницу заполнения

this.NavigationService.Navigate(new ExtensionPages.AddSickPage(sick));

}

private void EditButton\_Click(object sender, System.Windows.RoutedEventArgs e)

{

//Получаем больничный из кнопки

var sick = (sender as Button).DataContext as Sick;

//Переходим на страницу заполнения

this.NavigationService.Navigate(new ExtensionPages.AddSickPage(sick));

}

private void DeleteButton\_Click(object sender, System.Windows.RoutedEventArgs e)

{

var asnwer = MessageBox.Show("Вы действительно хотите удалить эту запись?", "Внимание", MessageBoxButton.YesNoCancel, MessageBoxImage.Warning);

if (asnwer != MessageBoxResult.Yes) return;

//Получаем больничный из кнопки

var sick = (sender as Button).DataContext as Sick;

//Удаляем и сохраняем

\_db.Sicks.Remove(sick);

\_db.SaveChanges();

}

/// <summary>

/// Поиск по номеру больничного

/// </summary>

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

{

if (txtFind == null) return;

if (txtFind.Text == "Поиск по номеру") return; //Если это заглушка, ничего не делаем

//Если пустой текстбокс, выводим все записи

if (!string.IsNullOrEmpty(txtFind.Text)) collectionView.ItemsSource = \_db.Sicks.Local.ToBindingList();

//Если список пустой, для выборки наполним его записями из таблицы

List<Sick> list = collectionView.ItemsSource?.Cast<Sick>().ToList();

if (list == null || list.Count == 0) list = \_db.Sicks.Local.ToList();

//Проводим выборку по номеру

collectionView.ItemsSource = list.Where(p => p.Number.ToString().Contains(txtFind.Text));

}

}

**Листинг 6 - AddAppointmentPage**

public partial class AddAppointmentPage : Page

{

private readonly DbEntities \_db = DbEntities.GetContext();

public AddAppointmentPage(Appointment appointment)

{

InitializeComponent();

//Получаем объект для редактирования и привязываем его

this.DataContext = appointment;

//Устанавливаем дату если есть

if (appointment.Date != null)

{

txtTime.Text = appointment.Date.Value.ToString("HH") + ":" + appointment.Date.Value.ToString("mm");

}

//Загружаем данные, заполняем combobox'ы

\_db.Specializations.Load();

\_db.Doctors.Load();

\_db.Nurses.Load();

\_db.MedicalCards.Load();

cmbSpec.ItemsSource = \_db.Specializations.ToList();

cmbMedCard.ItemsSource = \_db.MedicalCards.ToList();

cmbNurse.ItemsSource = \_db.Nurses.ToList();

}

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

{

this.NavigationService.GoBack();

}

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

{

//Получаем объект из привязки

Appointment appointment = this.DataContext as Appointment;

txtErrorMessage.Text = string.Empty; //Очищаем строку с ошибкой

//Валидация данных

if (cmbSpec.SelectedItem == null)

{

txtErrorMessage.Text = "Ошибка: Выберите специализацию";

return;

}

if (cmbDoctor.SelectedItem == null)

{

txtErrorMessage.Text = "Ошибка: Выберите врача";

return;

}

if (cmbMedCard.SelectedItem == null)

{

txtErrorMessage.Text = "Ошибка: Выберите пациента";

return;

}

if (cmbNurse.SelectedItem == null)

{

txtErrorMessage.Text = "Ошибка: Выберите медсестру";

return;

}

if (appointment.Date == null)

{

txtErrorMessage.Text = "Ошибка: Выберите дату приема";

return;

}

if (string.IsNullOrEmpty(txtTime.Text) || txtTime.Text.Length != 5)

{

txtErrorMessage.Text = "Ошибка: Укажите время с 07:00 по 20:59";

return;

}

if (!int.TryParse(txtTime.Text.Substring(0, 2), out int hours))

{

txtErrorMessage.Text = "Ошибка: Укажите время с 07:00 по 20:59";

return;

}

if (!int.TryParse(txtTime.Text.Substring(3, 2), out int minutes))

{

txtErrorMessage.Text = "Ошибка: Укажите время с 07:00 по 20:59";

return;

}

if (hours < 7 || hours > 20)

{

txtErrorMessage.Text = "Ошибка: Укажите время с 07:00 по 20:59";

return;

}

if (minutes < 0 || minutes > 59)

{

txtErrorMessage.Text = "Ошибка: Укажите время с 07:00 по 20:59";

return;

}

//Добавляем время из текстбокса в дату из датапикера

var old = appointment.Date.Value;

appointment.Date = new DateTime(old.Year, old.Month, old.Day, hours, minutes, 0);

if (appointment.Date < DateTime.Now)

{

txtErrorMessage.Text = "Ошибка: Выбрана дата в прошлом";

return;

}

//Указываем id для связей в бд

appointment.MedicalCardId = appointment.MedicalCard.Id;

appointment.NurseId = appointment.Nurse.Id;

appointment.DoctorId = appointment.Doctor.Id;

//Если такой записи еще нет, значит добавим ее

if (\_db.Appointments.Find(appointment.Id) == null)

{

//Получение расписания приемов у выбранного доктора

var list = \_db.Appointments.Where(p => p.DoctorId == appointment.DoctorId).ToList();

//Если в 15 минутный промежуток в этот день у врача уже есть запись, сообщаем об этом

if (list.Count(p => p.Date > appointment.Date.Value.AddMinutes(-15) && p.Date < appointment.Date.Value.AddMinutes(15)) > 0)

{

txtErrorMessage.Text = "Ошибка: На данное время к данному врачу уже существует запись";

return;

}

//Добавляем запись в базу данных

\_db.Appointments.Add(appointment);

}

try

{

\_db.SaveChanges();

}

catch (Exception ex)

{

txtErrorMessage.Text = ex.Message + ex.InnerException?.Message;

return;

}

this.NavigationService.GoBack();

}

private void SpecChanged(object sender, SelectionChangedEventArgs e)

{

if (cmbSpec.SelectedItem == null) return;

//Получаем выбранную специализацию из комбобокса

var spec = cmbSpec.SelectedItem as Specialization;

//Заполняем комбобокс врачей врачами выбранной специализации

cmbDoctor.ItemsSource = \_db.Doctors.Where(p => p.SpecializationId == spec.Id).ToList();

}

}

**Листинг 7 - AddDoctorPage**

public partial class AddDoctorPage : Page

{

private readonly DbEntities \_db = DbEntities.GetContext();

public AddDoctorPage(Doctor doctor)

{

InitializeComponent();

//Получаем объект для редактирования и привязываем его

this.DataContext = doctor;

//Загружаем данные, заполняем comboboх

\_db.Specializations.Load();

cmb.ItemsSource = \_db.Specializations.ToList();

}

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

{

this.NavigationService.GoBack();

}

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

{

//Получаем объект из привязки

var doctor = this.DataContext as Doctor;

txtErrorMessage.Text = string.Empty; //Очищаем строку с ошибкой

//Валидация данных

if (cmb.SelectedItem == null)

{

txtErrorMessage.Text = "Ошибка: Выберите специализацию";

return;

}

if (string.IsNullOrEmpty(doctor.Worker.FIO))

{

txtErrorMessage.Text = "Ошибка: Укажите фио";

return;

}

if (string.IsNullOrEmpty(doctor.Worker.Adress))

{

txtErrorMessage.Text = "Ошибка: Укажите адрес";

return;

}

if (doctor.Worker.DateEnployment == null)

{

txtErrorMessage.Text = "Ошибка: Укажите дату трудоустройства";

return;

}

//Указываем id для связей в бд

doctor.SpecializationId = doctor.Specialization.Id;

//Если такой записи еще нет, значит добавим ее

if (\_db.Workers.Find(doctor.Id) == null)

{

//Добавим сначала работника, к которому привяжем нового врача

\_db.Workers.Add(doctor.Worker);

doctor.Id = doctor.Worker.Id; //Указываем что новый рабочий будет новым врачом

//Добавляем врача

\_db.Doctors.Add(doctor);

}

try

{

\_db.SaveChanges();

}

catch (Exception ex)

{

txtErrorMessage.Text = ex.Message + ex.InnerException?.Message;

return;

}

this.NavigationService.GoBack();

}

}

**Листинг 8 - AddNursePage**

public partial class AddNursePage : Page

{

private readonly DbEntities \_db = DbEntities.GetContext();

public AddNursePage(Nurse nurse)

{

InitializeComponent();

//Получаем объект для редактирования и привязываем его

this.DataContext = nurse;

}

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

{

this.NavigationService.GoBack();

}

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

{

//Получаем объект из привязки

var nurse = this.DataContext as Nurse;

txtErrorMessage.Text = string.Empty; //Очищаем строку с ошибкой

//Валидация данных

if (string.IsNullOrEmpty(nurse.Worker.FIO))

{

txtErrorMessage.Text = "Ошибка: Укажите фио";

return;

}

if (string.IsNullOrEmpty(nurse.Worker.Adress))

{

txtErrorMessage.Text = "Ошибка: Укажите адрес";

return;

}

if (nurse.Office == null | nurse.Office == 0)

{

txtErrorMessage.Text = "Ошибка: Укажите номер кабинета";

return;

}

if (nurse.Worker.DateEnployment == null)

{

txtErrorMessage.Text = "Ошибка: Укажите дату трудоустройства";

return;

}

//Если такой записи еще нет, значит добавим ее

if (\_db.Workers.Find(nurse.Id) == null)

{

//Добавим сначала работника, к которому привяжем новую медсестру

\_db.Workers.Add(nurse.Worker);

nurse.Id = nurse.Worker.Id; //Указываем что новый рабочий будет новой медсестрой

//Добавляем медсестру

\_db.Nurses.Add(nurse);

}

try

{

\_db.SaveChanges();

}

catch (Exception ex)

{

txtErrorMessage.Text = ex.Message + ex.InnerException?.Message;

return;

}

this.NavigationService.GoBack();

}

}

**Листинг 9 - AddPatientPage**

public partial class AddPatientPage : Page

{

private readonly DbEntities \_db = DbEntities.GetContext();

public AddPatientPage(MedicalCard patient)

{

InitializeComponent();

//Получаем объект для редактирования и привязываем его

this.DataContext = patient;

}

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

{

this.NavigationService.GoBack();

}

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

{

//Получаем объект из привязки

var medicalCard = this.DataContext as MedicalCard;

txtErrorMessage.Text = string.Empty; //Очищаем строку с ошибкой

//Валидация данных

if (string.IsNullOrEmpty(medicalCard.FIO))

{

txtErrorMessage.Text = "Ошибка: Укажите фио";

return;

}

if (string.IsNullOrEmpty(medicalCard.Adress))

{

txtErrorMessage.Text = "Ошибка: Укажите адрес";

return;

}

if (string.IsNullOrEmpty(medicalCard.Phone))

{

txtErrorMessage.Text = "Ошибка: Укажите телефон";

return;

}

//Если такой записи еще нет, значит добавим ее

if (\_db.MedicalCards.Find(medicalCard.Id) == null)

{

\_db.MedicalCards.Add(medicalCard);

}

try

{

\_db.SaveChanges();

}

catch (Exception ex)

{

txtErrorMessage.Text = ex.Message + ex.InnerException?.Message;

return;

}

this.NavigationService.GoBack();

}

}

**Листинг 10 - AddSickPage**

public partial class AddSickPage : Page

{

private readonly DbEntities \_db = DbEntities.GetContext();

public AddSickPage(Sick sick)

{

InitializeComponent();

//Получаем объект для редактирования и привязываем его

this.DataContext = sick;

//Загружаем данные, заполняем combobox

\_db.MedicalCards.Load();

cmb.ItemsSource = \_db.MedicalCards.ToList();

}

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

{

this.NavigationService.GoBack();

}

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

{

//Получаем объект из привязки

var sick = this.DataContext as Sick;

txtErrorMessage.Text = string.Empty; //Очищаем строку с ошибкой

//Валидация данных

if (cmb.SelectedItem == null)

{

txtErrorMessage.Text = "Ошибка: Выберите пациента";

return;

}

if (sick.DateStart == null)

{

txtErrorMessage.Text = "Ошибка: Укажите дату начала больничного";

return;

}

if (sick.DateEnd == null)

{

txtErrorMessage.Text = "Ошибка: Укажите дату выхода с больничного";

return;

}

if (sick.DateEnd <= sick.DateStart)

{

txtErrorMessage.Text = "Ошибка: Дата конца раньше даты начала";

return;

}

if (\_db.Sicks.Count(p => p.MedicalCardId == sick.MedicalCard.Id && p.DateEnd >= DateTime.Now) > 0)

{

txtErrorMessage.Text = "Ошибка: Пациент уже на больничном";

return;

}

//Указываем id для связей в бд

sick.MedicalCardId = sick.MedicalCard.Id;

//Если такой записи еще нет, значит добавим ее

if (\_db.Sicks.Find(sick.Id) == null)

{

\_db.Sicks.Add(sick);

}

try

{

\_db.SaveChanges();

}

catch (Exception ex)

{

txtErrorMessage.Text = ex.Message + ex.InnerException?.Message;

return;

}

this.NavigationService.GoBack();

}

}