МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ

ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ

УЧРЕЖДЕНИЕ

ВЫСШЕГО ОБРАЗОВАНИЯ

“НОВОСИБИРСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ”

Кафедра АВТОМАТИЗИРОВАННЫХ СИСТЕМ УПРАВЛЕНИЯ

Пояснительная записка

к курсовой работе

по дисциплине ПРОГРАММИРОВАНИЕ

на тему: Разработка АРМ врача поликлиники

АП-227

2

Студента \_\_\_курса \_\_\_\_\_\_\_\_ группы

Бузмакова Антона Игоревича

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

(Фамилия Имя Отчество студента)

Эстрайх Игорь Викторович,

Руководитель\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

старший преподаватель кафедры АСУ

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

(Ф.И.О., должность, ученое звание, ученая степень)

3

Новосибирск 202\_

Федеральное государственное бюджетное образовательное учреждение

высшего образования

“Новосибирский государственный технический университет”

Кафедра автоматизированных систем управления

ЗАДАНИЕ НА КУРСОВОЙ ПРОЕКТ (РАБОТУ)

Студент

Бузмаков Антон Игоревич, АП-227

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

(Ф.И.О., группа)

Тема курсовой работы

Разработка АРМ врача поликлиники

Утверждена на заседании кафедры от

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_протокол № \_\_\_\_\_

Срок защиты работы

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

Краткая аннотация задания: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Научный руководитель \_\_\_\_\_

Задание принял к исполнению \_\_\_\_\_

Бузмаков А.И.

Дата \_\_\_\_\_\_\_\_

**Реферат курсового проекта**

Курсовая работа по дисциплине “Программирование” на тему “Информацион­ные и управля­ющие системы. АРМ врача поликлиники” студента второго курса Бузмакова А.И.

Количество страниц – 37, рисунков – 14, таблиц – 1, количество используемых источников – 7, количество приложений – 2.

Ключевые слова, идентифицирующие предметную область –информацион­ные и управля­ющие системы, программный продукт, автоматизированное рабочее место.

**Объект исследования** – автоматизированное рабочее место врача поликлиники.

**Предмет исследования** – методы и технологии автоматизации, используемые на рабочем месте врача поликлиники, включая программное обеспечение, оборудование и процессы обработки данных.

**Цель работы** – анализ эффективности автоматизированных рабочих мест врачей и разработка предложений по их оптимизации для повышения производительности и удобства работы.

**Использованные методы** – ООП, C#, Visual Studio, Windows Forms, тестирования ПО, UML и литературный анализ.

Результатом стало функциональное приложение с удобным интерфейсом, обработкой персональной информации, модульностью, надежностью и документированной архитектурой, демонстрирующее успешное применение современных методов программирования.

**Оглавление**

[Постановка задачи 5](#_Toc154435109)

[Описание классов 6](#_Toc154435110)

[Взаимосвязь классов 7](#_Toc154435111)

[Описание алгоритмов 9](#_Toc154435112)

[Описание разработанного приложения 11](#_Toc154435113)

[Тестирование 17](#_Toc154435114)

[Заключение 21](#_Toc154435115)

[Список использованной литературы 24](#_Toc154435116)

[Приложения 25](#_Toc154435117)

[Приложение 1 25](#_Toc154435118)

[Приложение 2 26](#_Toc154435119)

# **Постановка задачи**

**Цель работы** – анализ эффективности автоматизированных рабочих мест медицинского персонала и разработка предложений по их оптимизации для повышения производительности и удобства работы.

**Задачи работы:**

1. Изучить существующие решения и технологии, используемые на автоматизированных рабочих местах медицинского персонала.
2. Проанализировать преимущества и недостатки автоматизации рабочего места врача.
3. Определить ключевые факторы, влияющие на эффективность работы врача.
4. Разработать рекомендации по улучшению автоматизированных систем на рабочем месте врачей, учитывая текущие тенденции и инновации.
5. Оценить потенциальное влияние предложенных улучшений на производительность труда и клиентский опыт.

# **Описание классов**

**Класс Database**

Класс, содержащий список всех пациентов **List<Patient> Patients**.

**Класс Patient**

Класс, представляющий данные о пациенте в системе. Содержит персональную информацию **PersonalInfo PersonalInfo**, список записей к врачу **List<Appointment> Appointments** и список анализов **List<Test> Tests**.

**Класс PersonalInfo**

Класс представляет главную персональную информацию пациента, а именно: имя **string Name**, фамилию **string Surname**, отчество **string Fathername**, дату рождения **DateTime Birthday**, пол **string Gender**.

**Класс MedicalHistory**

Абстрактный класс для представления медицинской записи события, содержит описание события **string Info**, дату, когда событие произошло **DateTime Date**, тип события **string Type** – меняется в зависимости от события.

**Класс Appointment**

Класс Посещение. Наследует класс MedicalHistory, имеет свойство **bool Status**, описывающее статус посещения (**true – состоялось, false – не состоялось**).

**Класс Test**

Класс Анализ. Наследует класс MedicalHistory, имеет свойство **string Results**, описывающее результаты анализа.

**Класс MainForm**

Класс для главного пользовательского интерфейса программы, содержит **Database db** для хранения списка пациентов во время работы, **DataTable dt** для визуального отображения базы данных пациентов в интерфейсе программы.

**Класс PersonalCard**

Класс для медицинской карточки пациента. Включает в себя данные о пациенте **Patient \_patient,** **List<MedicalHistory> \_medicalHistory** – медицинская история пациента, **DataTable dt** для визуального отображения медицинской истории в интерфейсе программы.

**Класс PersonalCard**

Класс для создания медицинского документа (справки) для пациента.

**Класс TestForm**

Класс для создания печатного варианта результатов анализа для пациента.

# **Взаимосвязь классов**

Основная взаимосвязь между классами MainForm и Database иллюстрирует поток данных и контроль в приложении. MainForm взаимодействует с пользователем, обрабатывает его ввод и управляет другими объектами для выполнения операций. Это включает в себя отображение информации о пациентах, управление действиями пользователя (например, созданием медицинского документа, добавлением и удалением пациентов и др.), а также чтение и запись информации из/в файл базы данных.

Класс Patient служит центральным хранилищем для объектов PersonalInfo и MedicalHistory. Это ключевой компонент, который отвечает за данные о каждом пациенте.

Класс PersonalInfo содержит в себе основную информацию о пациенте, что добавляет в приложение функциональность создания медицинского документа в классе DocumentForm без необходимости получения полной информации о пациенте.

Класс MedicalHistory используется для перечисления медицинской истории пациента. Так же этот класс наследуется классами Appointment и Test (Рисунок 1), что в свою очередь позволяет получить информацию о посещениях и, при необходимости, создать отчет об анализе пациента, используя TestForm.

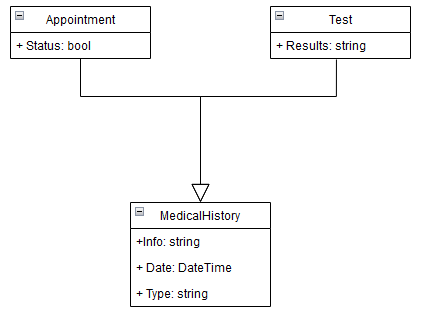


Рисунок 1 - Иерархия наследования

**Использование диаграммы**

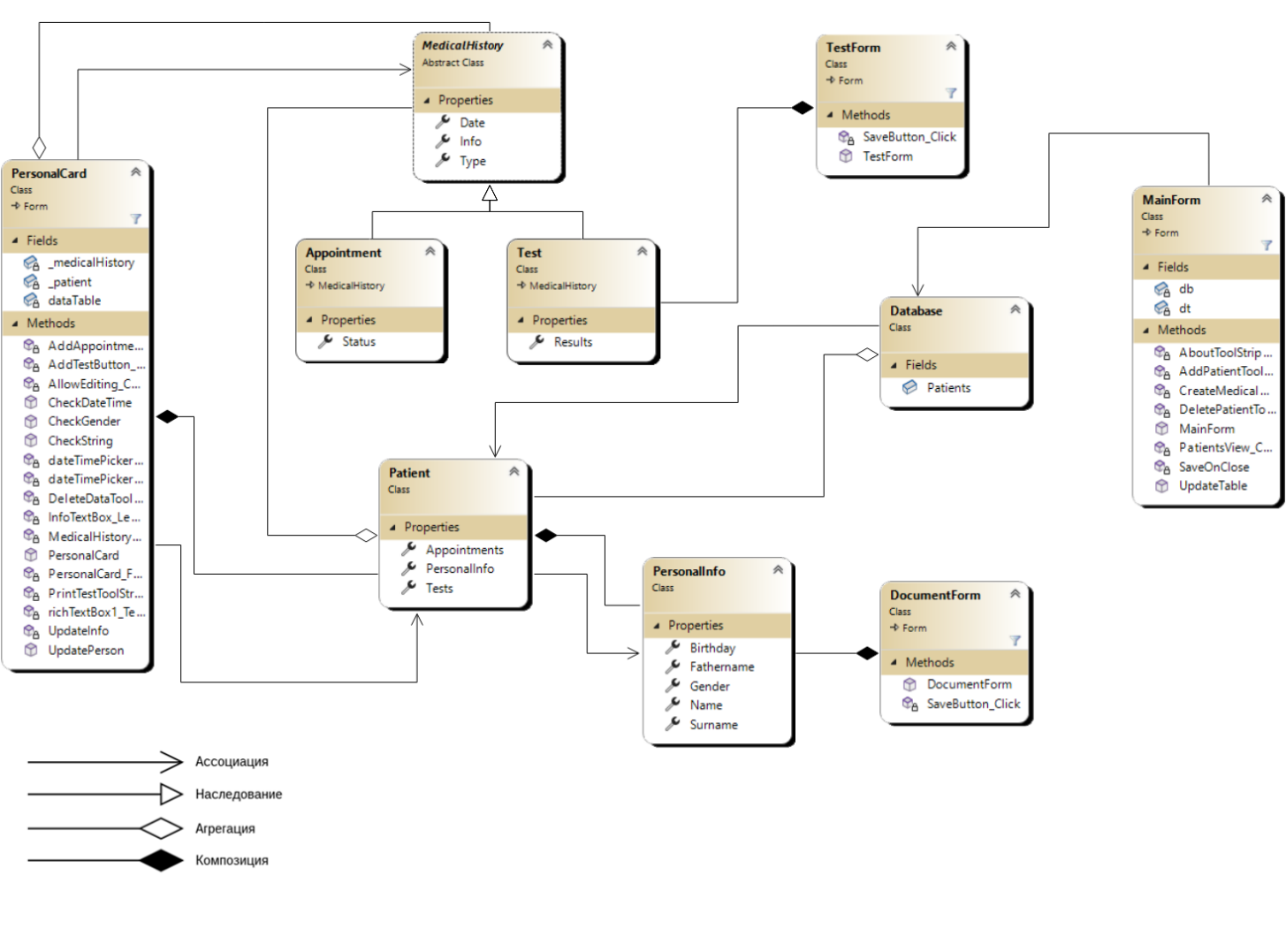


Рисунок 2 - Диаграмма UML

Диаграмма UML (Рисунок 2) предоставляет визуальное представление архитектуры программы, что является ценным инструментом при разработке, тестировании и отладке приложения. Она помогает разработчикам и аналитикам понять структуру программы и взаимодействие между ее компонентами, а также облегчает внесение изменений и добавление новых функций.

# **Описание алгоритмов**

**Алгоритм Добавления нового пациента**

Описание: Этот алгоритм обрабатывает добавление нового пациента в базу данных. При выборе опции “Добавить пациента” в контекстном меню, программа должна открыть новое окно для ввода информации о пациенте. Эта форма, в свою очередь, проверяет введенные пользователем данные на наличие ошибок: в случае их отсутствия программа сохраняет информацию о пациенте, в обратном случае выдает предупреждение с подробным описанием проблемы. При закрытии формы создания пациента программа автоматически добавляет его в базу данных, после чего обновляется таблица и пользователь может увидеть созданного пользователя.

Шаги:

1. Получение данных из полей ввода и опций выбора (ФИО, Дата рождения, Пол).
2. Проверка данных на валидность (не является ли одно из значений пустым, содержит ли оно недопустимые символы).
3. Добавление корректных значений в объект Patient.
4. При корректном вводе всех значений и закрытии окна – добавление нового пациента в базу данных.
5. Обновление отображаемой информации о пациентах.

**Алгоритм Удаления пациента**

Описание: Этот алгоритм используется для удаления выбранного пациента из базы данных.

Шаги:

1. Определение выбранного пациента в интерфейсе пользователя.
2. Если пациент выбран – удаление его из базы данных.
3. Обновление отображаемой информации о пациентах.

**Алгоритм Добавления записи в медицинской истории**

Описание: Алгоритм создает запись в медицинской истории

Шаги:

1. Создание объекта записи в медицинской истории.
2. Передача значений, введенных пользователем в интерфейсе, в объект.
3. Сохранение объекта в списке медицинской истории пациента.
4. Обновление отображаемой информации о пациенте.

**Алгоритм Удаления записи в медицинской истории**

Описание: Этот алгоритм используется для удаления выбранной записи из медицинской истории пациента.

Шаги:

1. Определение выбранной записи в интерфейсе пользователя.
2. Если запись выбрана – удаление его из медицинской истории.
3. Обновление отображаемой информации о пациенте.

**Алгоритм Создания справки пациенту**

Описание: Этот алгоритм используется для создания справки, выдаваемой пациенту.

Шаги:

1. Заполнение текста в элементах интерфейса программы данными о пациенте.
   1. При необходимости пользователь может ввести дополнительную информацию в основной части медицинского документа
2. Сохранение справки в формате .jpeg

**Алгоритм Создания отчета об анализе**

Описание: Этот алгоритм используется для создания отчета об анализе, выдаваемого пациенту.

Шаги:

1. Заполнение текста в элементах интерфейса программы данными о пациенте и анализе.
2. Сохранение отчета в формате .jpeg

Каждый из этих алгоритмов важен для обеспечения функциональности автоматизированного рабочего места врача. Они позволяют эффективно обрабатывать действия пользователя и обеспечивают корректную логику работы программы.

# **Описание разработанного приложения**

**Интерфейс Пользователя**

Приложение "Автоматизированное рабочее место врача поликлиники" имеет интуитивно понятный и удобный пользовательский интерфейс, предназначенный для медицинской деятельности. Интерфейс главного окна содержит следующие элементы управления (Рисунок 3):

* Список всех пациентов в базе данных.
* Кнопки для получения информации о программе, контекстное меню для работы с базой данных, контекстное меню для работы с пациентами.

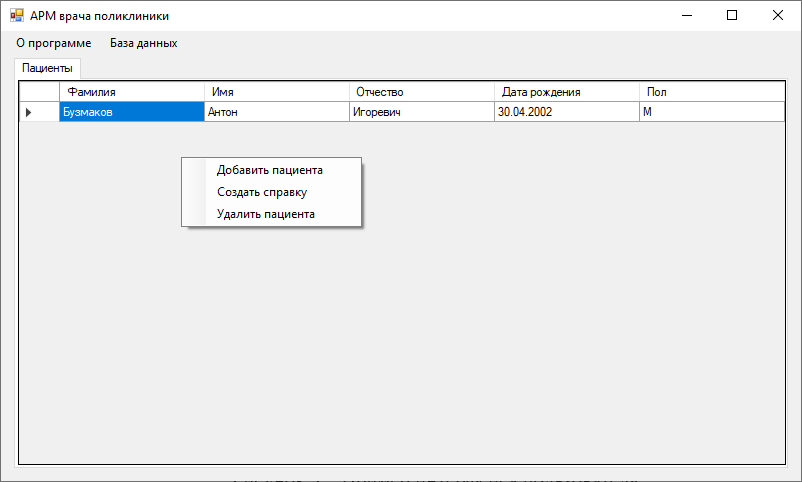


Рисунок 3 – Интерфейс главного окна программы

Для работы с персональными данными пациентов используется окно медицинской карточки. Оно содержит следующие элементы управления (Рисунок 4):

* Поля для ввода и выбора информации о пациенте.
* Список с медицинской историей пациента.
* Контекстное меню для работы с медицинской историей пациента.
* Контейнер с текстом для ознакомления c/изменения информации о записи в медицинской истории.

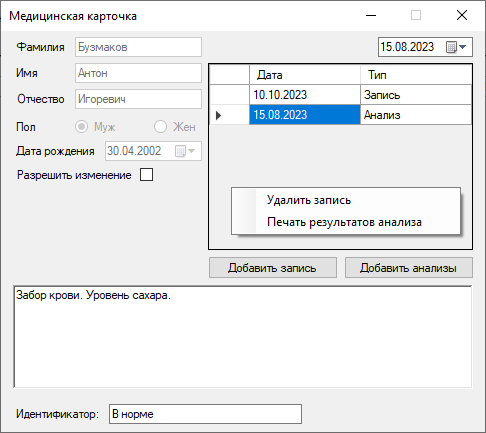


Рисунок 4 - Интерфейс медицинской карты пациента

Для создания отчета об анализе используется соответствующее окно, содержащее в себе следующие элементы управления (Рисунок 5):

* Контейнеры с информацией об анализе для ознакомления.
* Кнопку для сохранения отчета.

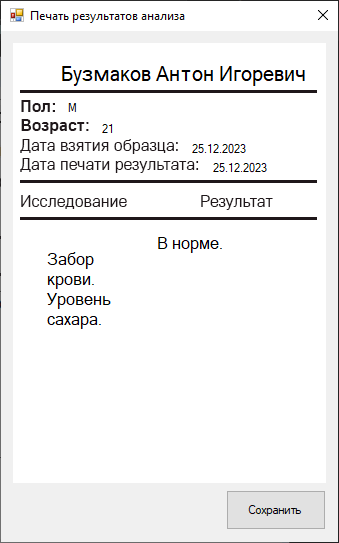


Рисунок 5 - Интерфейс создания отчета об анализе

Для создания справки используется соответствующее окно, содержащее в себе следующие элементы управления (Рисунок 6):

* Контейнеры с информацией о пациенте для ознакомления.
* Кнопку для сохранения документа.

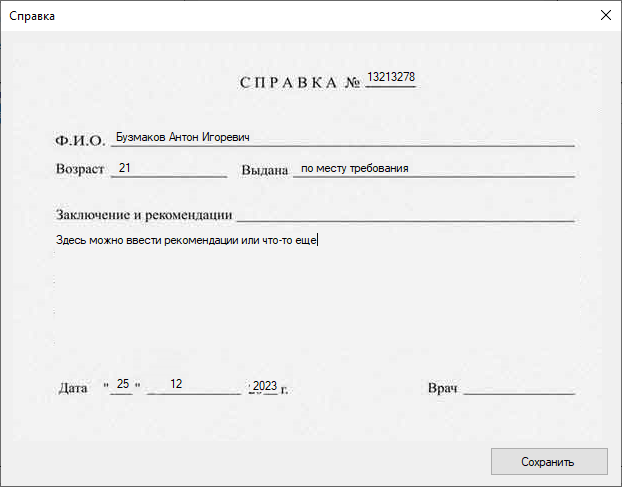


Рисунок 6 - Интерфейс окна для создания справки

**Инструкция Пользователя**

Добавление пациента в базу данных:

1. Нажмите правой кнопкой мыши по пустому месту в интерфейсе программы (Рисунок 3).
2. В появившемся окне медицинской карты пациента (Рисунок 4) нажмите на кнопку “Разрешить изменение” и введите необходимую информацию. Также при необходимости можно добавить записи в медицинскую историю.
3. После заполнения всех нужных полей повторно нажмите на кнопку “Разрешить изменение” и закройте окно.
4. Повторите процедуру для добавления новых пациентов.

Удаление пациента из базы данных:

1. Нажмите правой кнопкой мыши по пустому месту в интерфейсе программы (Рисунок 3).
2. Нажмите кнопку "Удалить пациента", чтобы удалить выбранного пациента.
3. В появившемся окне подтвердите удаление (Рисунок 7).
4. Для удаления других пациентов повторите процедуру.

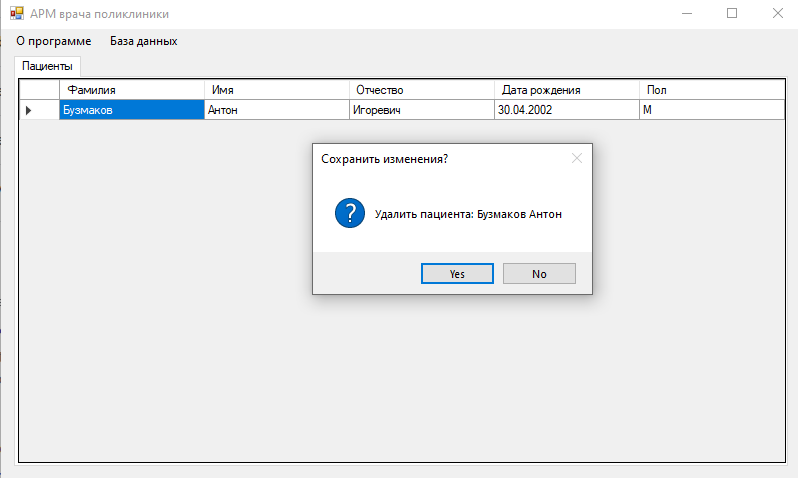


Рисунок 7 - Удаление пациента

Изменение информации о пациенте:

1. Двойным нажатием левой кнопки мыши в основном интерфейсе программы (Рисунок 3) выберите пациента, информацию о котором необходимо изменить.
2. В появившемся окне медицинской карты пациента (Рисунок 4) нажмите на кнопку “Разрешить изменение” и измените необходимую информацию. Также при необходимости можно изменить записи в медицинской истории.
3. После изменения всех нужных полей повторно нажмите на кнопку “Разрешить изменение” и закройте окно.
4. Повторите процедуру для повторного изменения информации о пациенте.

Создание отчета об анализе (Рисунок 5):

1. Двойным нажатием левой кнопки мыши в основном интерфейсе программы (Рисунок 3) выберите пациента, отчет для которого необходимо составить.
2. В списке медицинской истории выберите необходимую запись с типом “Анализ” и нажмите на нее правой кнопкой мыши.
3. В появившемся контекстном меню выберите пункт “Печать результатов анализа” (Рисунок 8).
4. В появившемся окне (Рисунок 5) нажмите кнопку “Сохранить” для сохранения отчета.
5. При необходимости печати других отчетов повторите процедуру.

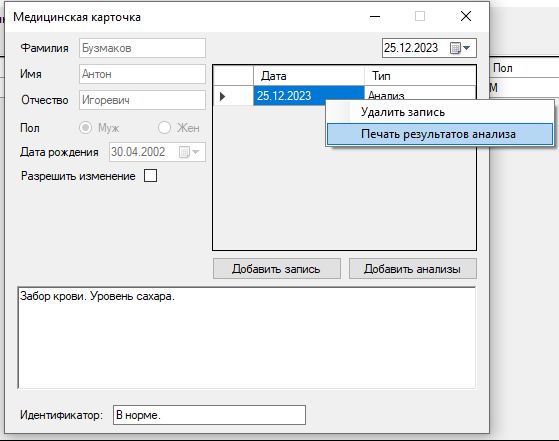


Рисунок 8 - Контекстное меню печати анализа

Создание справки (Рисунок 6):

1. Одиночным нажатием левой кнопки мыши в основном интерфейсе программы (Рисунок 3) выберите пациента, справку для которого необходимо составить.
2. Нажмите правую кнопку мыши и в появившемся контекстном меню выберите пункт “Создать справку” (Рисунок 9).
3. В появившемся окне введите необходимую информацию в контейнер для ввода текста (Рисунок 6) нажмите кнопку “Сохранить” для сохранения отчета.
4. При необходимости печати других справок повторите процедуру.

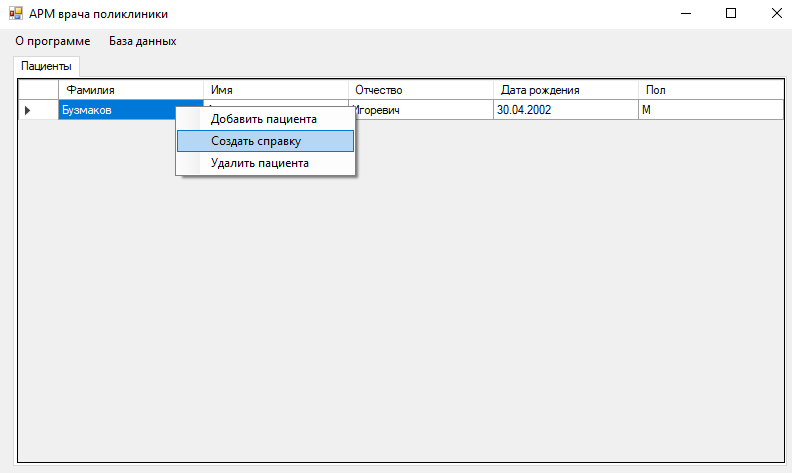


Рисунок 9 - Контекстное меню для печати справки

# **Тестирование**

Тестирование приложения проводилось с целью подтвердить его работоспособность и корректность выполнения основных функций. Рассмотрим контрольные примеры для каждой из ключевых функций.

**Тестирование Добавления нового пациента**

Цель теста: Проверка функции добавления пациента в базу данных.

Шаги:

1. Запуск приложения.
2. Открытие окна для создания нового пациента.
3. Ввод информации о пациенте (Рисунок 10): Фамилия – “Бузмаков”, Имя – “Игорь”, Отчество – “Витальевич”, Пол – “Мужской”, Дата рождения – “24.11.1965”.
4. Закрытие окна.
5. Ожидаемый результат: в списке пациентов отображается информация о пациенте: Фамилия – “Бузмаков”, Имя – “Игорь”, Отчество – “Витальевич”, Пол – “М”, Дата рождения – “24.11.1965”.

Скриншот результата тестирования - Рисунок 11.

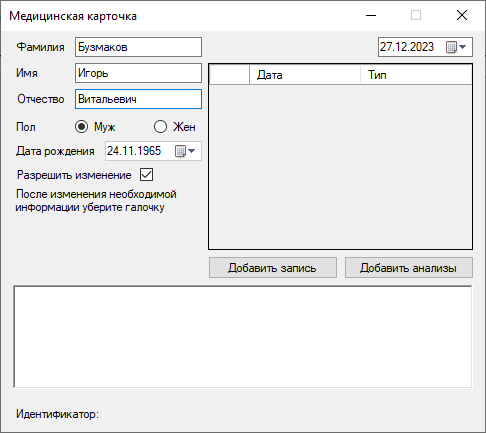


Рисунок 10 - Медицинская карточка нового пациента

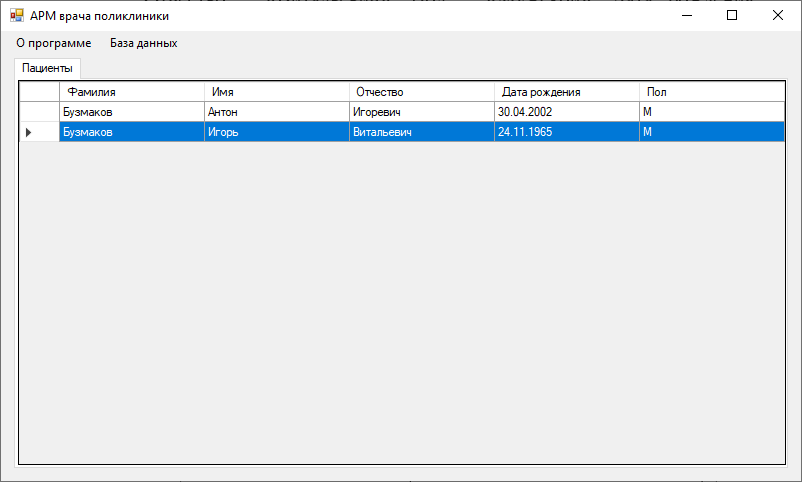


Рисунок 11 - Новый пациент в списке

**Тестирование Удаления пациента из базы данных**

Цель теста: Проверка функции удаления пациента из базы данных.

Шаги:

1. Выбор пациента для удаления: Пациент – Бузмаков Антон.
2. Нажатие кнопки “Удалить пациента” в контекстном меню и подтверждение удаления (Рисунок 7).
3. Ожидаемый результат: Пациент “Бузмаков Антон” исчезает из списка.

Скриншот результата тестирования – Рисунок 12.

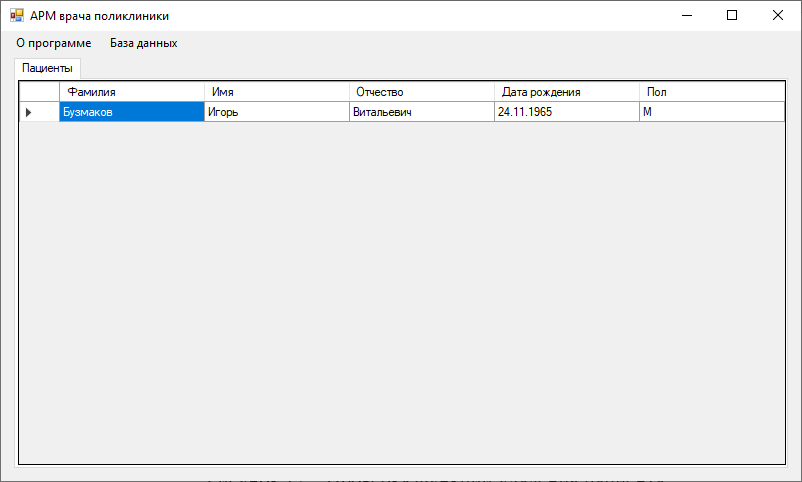


Рисунок 12 – Проверка функции удаления пациента

**Тестирование Печать отчета об анализе**

Цель теста: Проверка функционала печати отчета об анализе.

Шаги:

1. Выбор пациента для печати анализа.
2. Выбор анализа для печати.
3. Нажатие кнопки “Печать результатов анализа” в контекстном меню.
4. Нажатие кнопки “Сохранить” в появившемся окне.
5. Ожидаемый результат: Файл в папке с программой – отчет об анализе.

Скриншот результата тестирования –Рисунок 13.

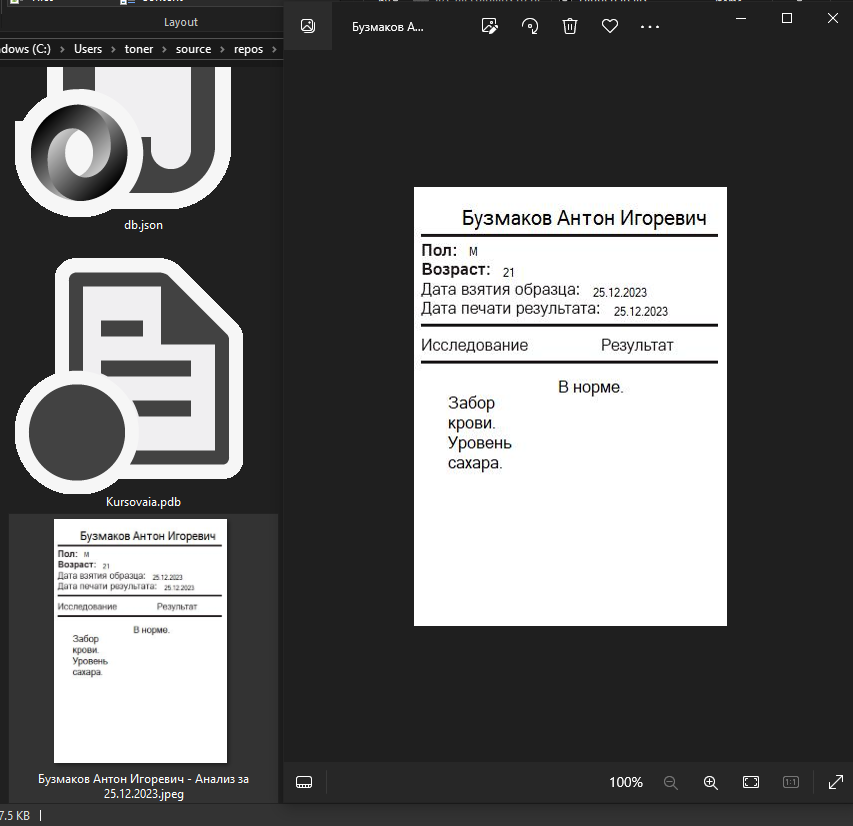


Рисунок 13 - Отчет об анализе в папке с программой

**Тестирование Создания справки**

Цель теста: Проверка создания справки.

Шаги:

1. Выбор пациента для печати справки.
2. Выбор опции “Создать справку” в контекстном меню.
3. Ввод информации в появившемся окне в поле для текста: “Здоров. К занятиям физической культурой допущен.”
4. Нажатие кнопки “Сохранить”

Скриншот результата тестирования – Рисунок 14.

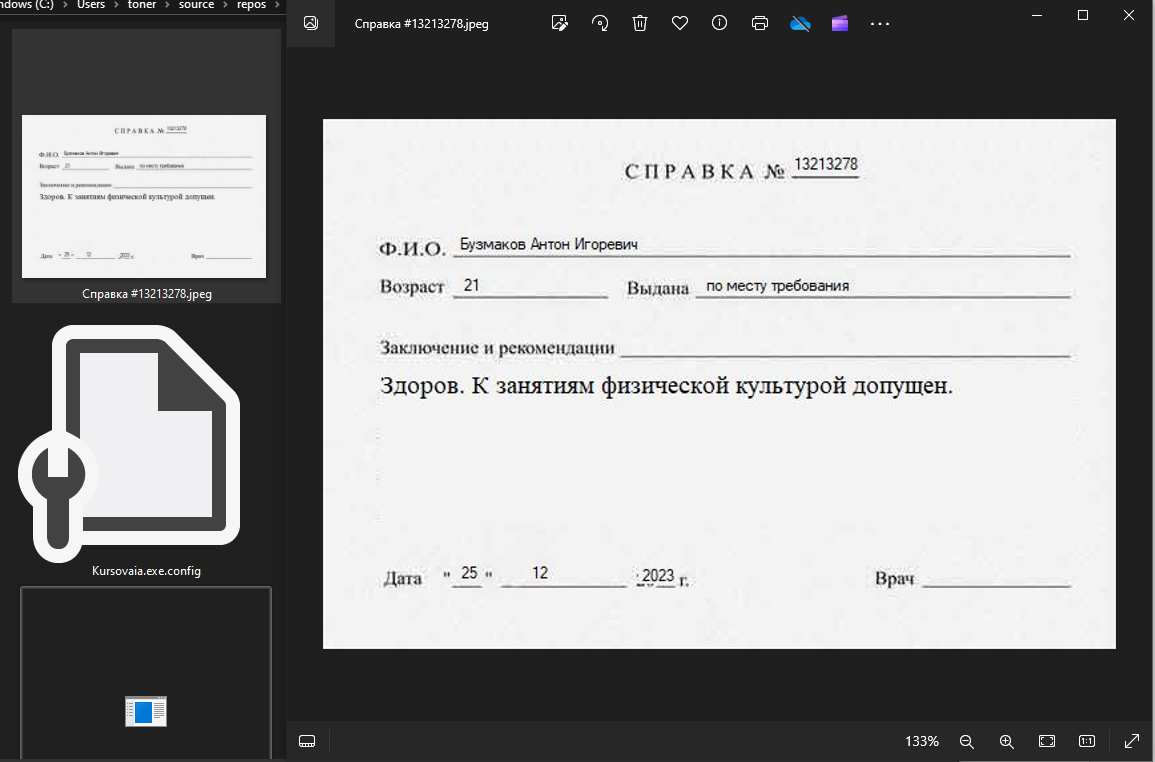


Рисунок 14 - Справка в папке с программой

Эти тесты демонстрируют основные функции приложения и подтверждают его работоспособность.

# **Заключение**

В ходе выполнения курсовой работы была разработана и успешно протестирована программа "Автоматизированное рабочее место врача поликлиники", предназначенная для упрощения и оптимизации процесса работы медицинского персонала. Основная цель – автоматизация действий врача и улучшение взаимодействия с пациентами – была достигнута благодаря реализации ряда ключевых функций:

1. Добавление и удаление пациентов, изменение информации о них в базе данных.
2. Ведение цифровой медицинской истории пациентов.
3. Автоматизированная печать важных медицинских документов.

В проекте были использованы принципы объектно-ориентированного программирования (ООП), что обеспечило его гибкость и масштабируемость. Среди ключевых ООП-приемов, примененных в данной работе:

* Инкапсуляция: Скрытие деталей реализации внутри классов PersonalCard, MainForm, предоставляя четкий и удобный интерфейс для взаимодействия с этими объектами.
* Наследование и полиморфизм: Использование абстрактного класса для упрощения масштабирования программы.
* Абстракция: Определение общих структур, таких как классы посещений врача, анализов, персональной информации и медицинской истории, что упрощает понимание логики программы и ее дальнейшее расширение.

Были реализованы все ключевые требования, изложенные в постановке задачи, включая удобный пользовательский интерфейс, автоматизацию и упрощение работы с медицинскими картами и документами. В приложении 1 представлен анализ достигнутых результатов.

**Пожелания по дальнейшему развитию проекта:**

1. Расширение функционала по работе с пациентами: Возможность добавлять родственников, место жительства и другие важные данные.
2. Добавление модуля управления пользователями: Регистрация и аутентификация персонала для обеспечения безопасности и персонализации рабочего процесса.
3. Разработка мобильной версии приложения: Создание мобильного приложения для обеспечения большей гибкости и удобства использования.
4. Интеграция с облачным хранилищем данных и внедрение шифрования данных: Возможность загружать нужные базы данных о различных районах, а также шифрование для обеспечения безопасности и сохранности персональных данных.

Завершая, стоит отметить, что реализованный проект успешно демонстрирует базовые принципы создания программных продуктов и может служить отличной основой для дальнейшего развития и модернизации в соответствии с растущими и изменяющимися требованиями к медицинскому программному обеспечению.

# **Список использованной литературы**

1. Троелсен, Э. Язык программирования C# 7 и платформы .NET и .NET Core : учебное пособие / Э. Троелсен, Ф. Джепикс. – Санкт-Петербург: Питер, 2019. – 832 с. – ISBN 978-5-4461-0999-7.
2. Рихтер, Джеффри. CLR via C#. Программирование на платформе Microsoft .NET Framework 4.5 на языке C#: учебное пособие / Джеффри Рихтер. – 4-е изд. – Москва: Вильямс, 2018. – 896 с. – ISBN 978-5-8459-2099-3.
3. Коннолли, Томас. Базы данных: проектирование, реализация и обслуживание: теория и практика: учебник / Томас Коннолли, Кэролин Бегг. – 6-е изд. – Москва: Вильямс, 2017. – 1440 с. – ISBN 978-5-8459-2028-3.
4. Макконнелл, Стив. Совершенный код. Мастер-класс : учебное пособие / Стив Макконнелл. – 3-е изд. – Москва: Вильямс, 2018. – 896 с. – ISBN 978-5-8459-2117-4.
5. Фаулер, Мартин. Рефакторинг. Улучшение существующего кода: учебное пособие / Мартин Фаулер. – Москва: Вильямс, 2018. – 448 с. – ISBN 978-5-8459-2125-9.
6. Мартин, Роберт. Чистый код. Создание, анализ и рефакторинг : учебное пособие / Роберт Мартин. – Москва: Вильямс, 2020. – 464 с. – ISBN 978-5-8459-2141-9.
7. Фридель, Джоэл. Windows Forms 2.0. Программирование на платформе .NET 2.0 с использованием C#: учебное пособие / Джоэл Фридель. – Москва : Вильямс, 2016. – 656 с. – ISBN 978-5-8459-1987-4.

# **Приложения**

## **Приложение 1**

Таблица 1 - Анализ достигнутых результатов

|  |  |  |
| --- | --- | --- |
| № | Цели и задачи | Достигнутый результат |
| 1 | Создание интегрированной системы для упрощения и ускорения обслуживания пациентов | Интегрированная система успешно разработана и тестируется |
| 2 | Повышение точности и эффективности работы медицинского персонала | Достигнута высокая точность обработки данных |
| 3 | Автоматизация процесса создания медицинских документов | Реализовано быстрое и точное создание основных медицинских документов |
| 4 | Уровень доступности медицинских данных | Реализовано доступное и своевременное получение данных для врачей |
| 5 | Экономическая эффективность | Достигнуто уменьшение затрат на бумажную документацию, сокращение ошибок в учёте медицинских процедур |
| 6 | Интеграция с другими системами | Разработаны возможности для интеграции с другими системами |
| 7 | Удобство интерфейса | Интерфейс является интуитивно понятным и легким в использовании |
| 8 | Масштабируемость приложения | Приложение способно адаптироваться к изменяющимся объемам информации |
| 9 | Учет особенностей рынка | Приложение адаптируемо под различные форматы медицинской деятельности |

## **Приложение 2**

**Листинг разработанной программы**

|  |
| --- |
| Database.cs |
| using System.Collections.Generic;  using Newtonsoft.Json;  namespace Kursovaia  {  public class Database  {  [JsonProperty("patients")]  public List<Patient> Patients;  }  } |

|  |
| --- |
| Patient.cs |
| using Newtonsoft.Json;  using System.Collections.Generic;  namespace Kursovaia  {  public class Patient  {  [JsonProperty("personalInfo")]  public PersonalInfo PersonalInfo { get; set; }  [JsonProperty("appointments")]  public List<Appointment> Appointments { get; set; }  [JsonProperty("tests")]  public List<Test> Tests { get; set; }  }  } |

|  |
| --- |
| PersonalInfo.cs |
| using Newtonsoft.Json;  using System;  namespace Kursovaia  {  public class PersonalInfo  {  [JsonProperty("name")]  public string Name { get; set; }  [JsonProperty("surname")]  public string Surname { get; set; }  [JsonProperty("fathername")]  public string Fathername { get; set; }  [JsonProperty("birthday")]  public DateTime Birthday { get; set; }  [JsonProperty("gender")]  public string Gender { get; set; }  }  } |

|  |
| --- |
| MedicalHistory.cs |
| using Newtonsoft.Json;  using System;  namespace Kursovaia  {  public abstract class MedicalHistory  {  [JsonProperty("info")]  public string Info { get; set; }  [JsonProperty("date")]  public DateTime Date { get; set; }  [JsonProperty("type")]  public string Type { get; set; }  }  } |

|  |
| --- |
| Appointment.cs |
| using Newtonsoft.Json;  namespace Kursovaia  {  public class Appointment : MedicalHistory  {  [JsonProperty("status")]  public bool Status { get; set; }  }  } |

|  |
| --- |
| Test.cs |
| using Newtonsoft.Json;  namespace Kursovaia  {  public class Test : MedicalHistory  {  [JsonProperty("results")]  public string Results { get; set; }  }  } |

|  |
| --- |
| MainForm.cs |
| using Newtonsoft.Json;  using System;  using System.Collections.Generic;  using System.Data;  using System.IO;  using System.Windows.Forms;  namespace Kursovaia  {  public partial class MainForm : Form  {  private Database db = new Database();  private DataTable dt = new DataTable();  public MainForm()  {  InitializeComponent();  dt.Columns.Add("Фамилия");  dt.Columns.Add("Имя");  dt.Columns.Add("Отчество");  dt.Columns.Add("Дата рождения");  dt.Columns.Add("Пол");  try  {  db = JsonConvert.DeserializeObject<Database>(File.ReadAllText("db.json"));  if (db == null)  {  db = new Database();  db.Patients = new List<Patient>();  DataBaseToolStripMenuItem.Enabled = false;  }  UpdateTable();  }  catch  {  if (MessageBox.Show(  $"Программа не нашла файл базы данных пациентов. Создать новый файл?",  "Выберите действие",  MessageBoxButtons.YesNo,  MessageBoxIcon.Question) == DialogResult.Yes)  {  File.Create("db.json");  }  else { DatabaseTab.Visible = false; }  }  }  // О программе  private void AboutToolStripMenuItem\_Click(object sender, EventArgs e)  {  MessageBox.Show("Курсовая работа по дисциплине 'Программирование', автор: Бузмаков Антон АП-227");  }  // Открыть медицинскую карту выбранного пациента  private void PatientsView\_CellMouseDoubleClick(object sender, DataGridViewCellMouseEventArgs e)  {  if (e.RowIndex != -1 && e.ColumnIndex != -1)  {  Form card = new PersonalCard(db.Patients[e.RowIndex]);  card.FormClosed += delegate  {  UpdateTable();  };  card.ShowDialog();  }  }  public void UpdateTable()  {  if (db != null)  {  dt.Clear();  foreach (Patient patient in db.Patients)  {  PersonalInfo personalInfo = patient.PersonalInfo;  dt.Rows.Add(  personalInfo.Surname,  personalInfo.Name,  personalInfo.Fathername,  personalInfo.Birthday.ToString("dd.MM.yyyy"),  personalInfo.Gender);  }  PatientsView.DataSource = dt;  }  }  // Сохранение на выходе  private void SaveOnClose(object sender, FormClosingEventArgs e)  {  if (MessageBox.Show("Подтвердите изменение", "Сохранить изменения?", MessageBoxButtons.YesNo, MessageBoxIcon.Question) == DialogResult.Yes)  {  string json = JsonConvert.SerializeObject(db, Formatting.Indented);  File.WriteAllText("db.json", json);  }  }  // Удалить пациента  private void DeletePatientToolStripMenuItem\_Click(object sender, EventArgs e)  {  if(db.Patients.Count != 0)  if (PatientsView.CurrentCell.RowIndex != -1 && PatientsView.CurrentCell.ColumnIndex != -1)  {  if (MessageBox.Show(  $"Удалить пациента: {db.Patients[PatientsView.CurrentCell.RowIndex].PersonalInfo.Surname} {db.Patients[PatientsView.CurrentCell.RowIndex].PersonalInfo.Name}",  "Сохранить изменения?",  MessageBoxButtons.YesNo,  MessageBoxIcon.Question) == DialogResult.Yes)  {  db.Patients.RemoveAt(PatientsView.CurrentCell.RowIndex);  UpdateTable();  }  }  }  // Создать справку  private void CreateMedicalDocumentToolStripMenuItem\_Click(object sender, EventArgs e)  {  if (db.Patients.Count != 0)  if (PatientsView.CurrentCell.RowIndex != -1 && PatientsView.CurrentCell.ColumnIndex != -1)  {  Form DocumentForm = new DocumentForm(db.Patients[PatientsView.CurrentCell.RowIndex]);  DocumentForm.ShowDialog();  }  else  {  MessageBox.Show("Не выбран пациент");  }  }  // Добавить пациента  private void AddPatientToolStripMenuItem\_Click(object sender, EventArgs e)  {  Patient newPatient = new Patient();    Form newCard = new PersonalCard(newPatient);  newCard.FormClosed += delegate  {  if (newPatient.PersonalInfo != null)  {  db.Patients.Add(newPatient);  UpdateTable();  }  };  newCard.ShowDialog();  }  // Создать новую базу данных  private void CreateNewDBToolStripMenuItem\_Click(object sender, EventArgs e)  {  if (MessageBox.Show(  $"Вы уверены что хотите создать новую базу данных пациентов? Все старые данные будут удалены без возможности восстановления.",  "Выберите действие",  MessageBoxButtons.YesNo,  MessageBoxIcon.Question) == DialogResult.Yes)  {  if (File.Exists("db.json"))  {  File.WriteAllText("db.json", "");  UpdateTable();  }  else  {  File.Create("db.json");  UpdateTable();  }  }  }  }  } |

|  |
| --- |
| PersonalCard.cs |
| using Kursovaia.Forms;  using System;  using System.Collections.Generic;  using System.Data;  using System.Linq;  using System.Text.RegularExpressions;  using System.Windows.Forms;  namespace Kursovaia  {  public partial class PersonalCard : Form  {  private Patient \_patient = null;  private DataTable dataTable = new DataTable();  private List<MedicalHistory> \_medicalHistory = new List<MedicalHistory>();  public PersonalCard(Patient patient)  {  this.\_patient = patient;  InitializeComponent();  StatusCompleted.Visible = false;  StatusUncompleted.Visible = false;  ResultTextBox.Visible = false;  // Заполняем поля данными  dataTable.Columns.Add("Дата");  dataTable.Columns.Add("Тип");  if (patient.Appointments != null)  \_medicalHistory.AddRange(patient.Appointments);  else patient.Appointments = new List<Appointment>();  if (patient.Tests != null)  \_medicalHistory.AddRange(patient.Tests);  else patient.Tests = new List<Test>();  if (\_patient.PersonalInfo != null)  UpdateInfo();  else  {  \_patient.PersonalInfo = new PersonalInfo();  }  BirthdayDateTimePicker.Value = DateTime.Now;  BirthdayDateTimePicker.MaxDate = DateTime.Now;  }  private void MedicalHistoryView\_CellClick(object sender, DataGridViewCellEventArgs e)  {  // Показать информацию  if (e.RowIndex != -1 && e.ColumnIndex != -1)  {  MedicalHistoryDateTimePicker.Value = \_medicalHistory[e.RowIndex].Date;  InfoTextBox.Text = \_medicalHistory[e.RowIndex].Info;  if (\_medicalHistory[e.RowIndex].Type.Equals("Анализ"))  {  StatusCompleted.Visible = false;  StatusUncompleted.Visible = false;  ResultTextBox.Visible = true;  Test temp = (Test)\_medicalHistory[e.RowIndex];  ResultTextBox.Text = temp.Results;  }  if (\_medicalHistory[e.RowIndex].Type.Equals("Запись"))  {  ResultTextBox.Visible = false;  StatusCompleted.Visible = true;  StatusUncompleted.Visible = true;  Appointment temp = (Appointment)\_medicalHistory[e.RowIndex];  if(temp.Status == true)  StatusCompleted.Checked = true;  else StatusUncompleted.Checked = true;  }  }  }  private void AllowEditing\_CheckStateChanged(object sender, EventArgs e)  {  // Включить/выключить изменение данных  SurnameTextBox.Enabled = AllowEditing.Checked;  NameTextBox.Enabled = AllowEditing.Checked;  FathernameTextBox.Enabled = AllowEditing.Checked;  GenderFemale.Enabled = AllowEditing.Checked;  GenderMale.Enabled = AllowEditing.Checked;  BirthdayDateTimePicker.Enabled = AllowEditing.Checked;  GenderFemale.Enabled = AllowEditing.Checked;  GenderMale.Enabled = AllowEditing.Checked;  WarningLabel.Visible = AllowEditing.Checked;  if (!AllowEditing.Checked) // Обновление данных  {  UpdatePerson(\_patient, SurnameTextBox.Text, 1);  UpdatePerson(\_patient, NameTextBox.Text, 2);  UpdatePerson(\_patient, FathernameTextBox.Text, 3);  UpdatePerson(\_patient, BirthdayDateTimePicker.Text, 4);  if (GenderFemale.Checked || GenderMale.Checked)  \_patient.PersonalInfo.Gender = (GenderMale.Checked) ? "М" : "Ж";  else MessageBox.Show("Ошибка: Выберите пол пациента.");  UpdateInfo();  }  }  // Добавить посещение  private void AddAppointmentButton\_Click(object sender, EventArgs e)  {  dataTable.Rows.Add(MedicalHistoryDateTimePicker.Value.ToString("dd.MM.yyyy"), "");  MedicalHistory mh = new Appointment();  mh.Date = MedicalHistoryDateTimePicker.Value;  mh.Type = "Запись";  \_medicalHistory.Add(mh);  UpdateInfo();  }  // Добавить анализ  private void AddTestButton\_Click(object sender, EventArgs e)  {  dataTable.Rows.Add(MedicalHistoryDateTimePicker.Value.ToString("dd.MM.yyyy"), "");  MedicalHistory mh = new Test();  mh.Date = MedicalHistoryDateTimePicker.Value;  mh.Type = "Анализ";  \_medicalHistory.Add(mh);  UpdateInfo();  }  private void InfoTextBox\_TextChanged(object sender, EventArgs e)  {  if (MedicalHistoryView.SelectedCells.Count > 0 && MedicalHistoryView.CurrentCell.RowIndex != -1 && MedicalHistoryView.CurrentCell.ColumnIndex != -1)  {  string[] infoSeparators = new string[] { "\n\n" };  \_medicalHistory[MedicalHistoryView.CurrentCell.RowIndex].Info = InfoTextBox.Text.Split(infoSeparators, StringSplitOptions.None).First();  }  }  private void dateTimePicker2\_ValueChanged(object sender, EventArgs e)  {  if(MedicalHistoryView.SelectedCells.Count > 0 && MedicalHistoryView.CurrentCell.RowIndex != -1 && MedicalHistoryView.CurrentCell.ColumnIndex != -1)  {  \_medicalHistory[MedicalHistoryView.CurrentCell.RowIndex].Date = MedicalHistoryDateTimePicker.Value;  }  }  // Обновить интерфейс  private void UpdateInfo()  {  // Персональная информация  SurnameTextBox.Text = \_patient.PersonalInfo.Surname;  NameTextBox.Text = \_patient.PersonalInfo.Name;  FathernameTextBox.Text = \_patient.PersonalInfo.Fathername;  if (\_patient.PersonalInfo.Gender != null && \_patient.PersonalInfo.Gender.Equals("М"))  GenderMale.Checked = true;  else GenderFemale.Checked = true;  if (\_patient.PersonalInfo.Birthday >= BirthdayDateTimePicker.MinDate && \_patient.PersonalInfo.Birthday <= BirthdayDateTimePicker.MaxDate)  BirthdayDateTimePicker.Value = \_patient.PersonalInfo.Birthday;  // Записи  dataTable.Clear();  foreach (MedicalHistory history in \_medicalHistory)  {  dataTable.Rows.Add(  history.Date.ToString("dd.MM.yyy"),  history.Type  );  }  MedicalHistoryView.DataSource = dataTable;  }  private void dateTimePicker2\_Leave(object sender, EventArgs e)  {  UpdateInfo();  }  private void InfoTextBox\_Leave(object sender, EventArgs e)  {  UpdateInfo();  }  public static void CheckString(string value)  {  if (string.IsNullOrEmpty(value))  throw new Exception("Поле не может быть пустым");  if (!Regex.IsMatch(value, @"^[А-Яа-я]\*$"))  throw new Exception("Поле содержит недопустимые символы");  }  public static void CheckDateTime(DateTime value)  {  if (value > DateTime.Now)  throw new Exception("Дата не может быть больше текущей");  }  public static void UpdatePerson(Patient p, string info, int index)  {  switch (index)  {  case 1:  {  try { CheckString(info); }  catch (Exception e) { MessageBox.Show("Ошибка в поле 'Фамилия': " + e.Message); break; }  p.PersonalInfo.Surname = info;  break;  }  case 2:  {  try { CheckString(info); }  catch (Exception e) { MessageBox.Show("Ошибка в поле 'Имя': " + e.Message); break; }  p.PersonalInfo.Name = info;  break;  }  case 3:  {  try { CheckString(info); }  catch (Exception e) { MessageBox.Show("Ошибка в поле 'Отчество': " + e.Message); break; }  p.PersonalInfo.Fathername = info;  break;  }  case 4:  {  try { CheckDateTime(DateTime.ParseExact(info, "dd.MM.yyyy", System.Globalization.CultureInfo.InvariantCulture)); }  catch (Exception e) { MessageBox.Show("Ошибка в поле 'Дата рождения': " + e.Message); break; }  p.PersonalInfo.Birthday = DateTime.ParseExact(info, "dd.MM.yyyy", System.Globalization.CultureInfo.InvariantCulture);  break;  }  default: break;  }  }  private void PersonalCard\_FormClosing(object sender, FormClosingEventArgs e)  {  \_patient.Appointments.Clear();  \_patient.Tests.Clear();  \_patient.Appointments = \_medicalHistory  .Where(history => history is Appointment)  .Cast<Appointment>()  .ToList();  \_patient.Tests = \_medicalHistory  .Where(history => history is Test)  .Cast<Test>()  .ToList();  if (\_patient.PersonalInfo.Name == null || \_patient.PersonalInfo.Surname == null || \_patient.PersonalInfo.Fathername == null || \_patient.PersonalInfo.Gender == null)  {  \_patient.PersonalInfo = null;  MessageBox.Show("Заполнены не все необходимые поля, изменения не сохранены.");  }  }  // Удалить запись  private void DeleteDataToolStripMenuItem\_Click(object sender, EventArgs e)  {  if(\_medicalHistory.Count != 0)  if (MedicalHistoryView.CurrentCell.RowIndex != -1 && MedicalHistoryView.CurrentCell.ColumnIndex != -1)  {  if (MessageBox.Show(  $"Удалить информацию: {\_medicalHistory[MedicalHistoryView.CurrentCell.RowIndex].Date.ToString("dd.MM.yyyy")}: {\_medicalHistory[MedicalHistoryView.CurrentCell.RowIndex].Type}",  "Сохранить изменения?",  MessageBoxButtons.YesNo,  MessageBoxIcon.Question) == DialogResult.Yes)  {  \_medicalHistory.RemoveAt(MedicalHistoryView.CurrentCell.RowIndex);  UpdateInfo();  }  }  }  // Напечатать отчет об анализе  private void PrintTestToolStripMenuItem\_Click(object sender, EventArgs e)  {  if(\_medicalHistory.Count != 0)  if(MedicalHistoryView.CurrentCell.RowIndex != -1 && MedicalHistoryView.CurrentCell.ColumnIndex != -1)  {  if (\_medicalHistory.ElementAt(MedicalHistoryView.CurrentCell.RowIndex).Type.Equals("Анализ"))  {  TestForm PrintTest = new TestForm((Test)\_medicalHistory.ElementAt(MedicalHistoryView.CurrentCell.RowIndex), \_patient);  PrintTest.ShowDialog();  }  else  {  MessageBox.Show("Выбран неверный элемент: выберите строку с типом 'Анализ'");  }  }  }  private void GenderMale\_CheckedChanged(object sender, EventArgs e)  {  if (GenderMale.Checked)  {  GenderFemale.Checked = false;  \_patient.PersonalInfo.Gender = "М";  }  }  private void GenderFemale\_CheckedChanged(object sender, EventArgs e)  {  if (GenderFemale.Checked)  {  GenderMale.Checked = false;  \_patient.PersonalInfo.Gender = "Ж";  }  }  private void StatusCompleted\_CheckedChanged(object sender, EventArgs e)  {  if (StatusCompleted.Checked )  {  Appointment temp = (Appointment)\_medicalHistory[MedicalHistoryView.CurrentCell.RowIndex];  temp.Status = true;  \_medicalHistory[MedicalHistoryView.CurrentCell.RowIndex] = temp;  }  }  private void StatusUncompleted\_CheckedChanged(object sender, EventArgs e)  {  if (StatusUncompleted.Checked)  {  Appointment temp = (Appointment)\_medicalHistory[MedicalHistoryView.CurrentCell.RowIndex];  temp.Status = false;  \_medicalHistory[MedicalHistoryView.CurrentCell.RowIndex] = temp;  }  }  private void ResultTextBox\_TextChanged(object sender, EventArgs e)  {  Test temp = (Test)\_medicalHistory[MedicalHistoryView.CurrentCell.RowIndex];  temp.Results = ResultTextBox.Text;  \_medicalHistory[MedicalHistoryView.CurrentCell.RowIndex] = temp;  }  }  } |

|  |
| --- |
| TestForm.cs |
| using System;  using System.Drawing;  using System.Windows.Forms;  namespace Kursovaia.Forms  {  public partial class TestForm : Form  {  public TestForm(Test test, Patient patient)  {  InitializeComponent();  // Присваеваем элементам управления информацию о пациенте и анализе  FIOLabel.Text = patient.PersonalInfo.Surname + " " + patient.PersonalInfo.Name + " " + patient.PersonalInfo.Fathername;  var age = DateTime.Today.Year - patient.PersonalInfo.Birthday.Year;  if (patient.PersonalInfo.Birthday.Date > DateTime.Today.AddYears(-age)) age--;  GenderLabel.Text = patient.PersonalInfo.Gender;  AgeLabel.Text = age.ToString();  TestDateLabel.Text = test.Date.ToString("dd.MM.yyyy");  PrintDateLabel.Text = DateTime.Today.ToString("dd.MM.yyyy");  TestNameLabel.Text = test.Info;  TestInfoTextBox.Text = test.Results;  TestPictureBox.Controls.Add(FIOLabel);  TestPictureBox.Controls.Add(GenderLabel);  TestPictureBox.Controls.Add(AgeLabel);  TestPictureBox.Controls.Add(TestDateLabel);  TestPictureBox.Controls.Add(PrintDateLabel);  TestPictureBox.Controls.Add(TestNameLabel);  TestPictureBox.Controls.Add(TestInfoTextBox);  }  // Сохранить в файл  private void SaveButton\_Click(object sender, EventArgs e)  {  var bm = new Bitmap(TestPictureBox.Width, TestPictureBox.Height);  TestPictureBox.DrawToBitmap(bm, new Rectangle(0, 0, TestPictureBox.Width, TestPictureBox.Height));  bm.Save($"{FIOLabel.Text.Trim()} - Анализ за {TestDateLabel.Text}.jpeg", System.Drawing.Imaging.ImageFormat.Jpeg);  bm.Dispose();  }  }  } |

|  |
| --- |
| DocumentForm.cs |
| using System;  using System.Drawing;  using System.Windows.Forms;  namespace Kursovaia  {  public partial class DocumentForm : Form  {  public DocumentForm(Patient p)  {  InitializeComponent();  //Добавляем информацию о пациенте  FIOLabel.Text = p.PersonalInfo.Surname + " " + p.PersonalInfo.Name + " " + p.PersonalInfo.Fathername;  var age = DateTime.Today.Year - p.PersonalInfo.Birthday.Year;  if (p.PersonalInfo.Birthday.Date > DateTime.Today.AddYears(-age)) age--;  AgeLabel.Text = age.ToString();  TodayDayLabel.Text = DateTime.Today.Day.ToString("");  TodayMonthLabel.Text = DateTime.Today.Month.ToString();  TodayYearLabel.Text = DateTime.Today.Year.ToString("");  DocumentNumberLabel.Text = this.GetHashCode().ToString();  DocumentPictureBox.Controls.Add(FIOLabel);  DocumentPictureBox.Controls.Add(AgeLabel);  DocumentPictureBox.Controls.Add(PlaceLabel);  DocumentPictureBox.Controls.Add(TodayDayLabel);  DocumentPictureBox.Controls.Add(TodayMonthLabel);  DocumentPictureBox.Controls.Add(TodayYearLabel);  DocumentPictureBox.Controls.Add(DocumentNumberLabel);  DocumentPictureBox.Controls.Add(InfoTextBox);  }  // Сохранить в файл  private void SaveButton\_Click(object sender, EventArgs e) // Доделать  {  var bm = new Bitmap(DocumentPictureBox.Width, DocumentPictureBox.Height);  DocumentPictureBox.DrawToBitmap(bm, new Rectangle(0, 0, DocumentPictureBox.Width, DocumentPictureBox.Height));  bm.Save("Cправка #" + DocumentNumberLabel.Text + ".jpeg", System.Drawing.Imaging.ImageFormat.Jpeg);  bm.Dispose();  }  }  } |

|  |
| --- |
| Program.cs |
| using System;  using System.Collections.Generic;  using System.Linq;  using System.Threading.Tasks;  using System.Windows.Forms;  namespace Kursovaia  {  internal static class Program  {  [STAThread]  static void Main()  {  Application.EnableVisualStyles();  Application.SetCompatibleTextRenderingDefault(false);  Application.Run(new MainForm());  }  }  } |