**МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ**

**УЧРЕЖДЕНИЕ ОБРАЗОВАНИЯ**

**«ГОМЕЛЬСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ ИМЕНИ П. О. СУХОГО»**

**Кафедра «Информатика»**

**ПОЯСНИТЕЛЬНАЯ ЗАПИСКА**

**к курсовому проекту**

**по дисциплине « \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ »**

**на тему: «\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ »**

Исполнитель: студент гр. ИП-21

В.А. Дранев

Руководитель: старший преподаватель

В.Н. Шибеко

Дата проверки: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Дата допуска к защите: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Дата защиты: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Оценка работы:\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Подписи членов комиссии

по защите курсового проекта: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Гомель 2020

Содержание

[ВВЕДЕНИЕ 5](#_Toc40306774)

[1 ПОСТАНОВКА ЗАДАЧИ 7](#_Toc40306775)

[1.1 Постановка задачи 7](#_Toc40306776)

[1.2 Пример решения аналогичной задачи 7](#_Toc40306777)

[2 АРХИТЕКТУРА ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ 9](#_Toc40306778)

[2.1 Анализ задания на проектирование 9](#_Toc40306779)

[2.1.1 Регистрация вызова 9](#_Toc40306780)

[2.1.2 Обработка вызова 10](#_Toc40306781)

[2.1.3 Хронические больные 11](#_Toc40306782)

[2.2 Доменные классы 12](#_Toc40306783)

[3 СТРУКТУРА И АЛГОРИТМЫ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ 20](#_Toc40306784)

[3.1 Язык и среда программирования 20](#_Toc40306785)

[3.2 Основы графических интерфейсов 21](#_Toc40306786)

[3.3 Структурная схема приложения и описание 24](#_Toc40306787)

[3.4 Вспомогательные классы 25](#_Toc40306788)

[3.5 Интерфейс пользователя 27](#_Toc40306789)

[4 ВЕРИФИКАЦИЯ 37](#_Toc40306790)

[ЗАКЛЮЧЕНИЕ 38](#_Toc40306791)

[СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ 39](#_Toc40306792)

[ПРИЛОЖЕНИЕ А 40](#_Toc40306793)

Изм.

Лист

№ докум.

Подпись

Дата

Лист

5

Курсовая Работа

# **ВВЕДЕНИЕ**

Уровень внедрения комплексных медицинских информационных систем в российском здравоохранении, по сдержанным оценкам, не превышает в настоящее время 10% от общей численности лечебно-профилактических организаций.

Наиболее актуальной и сложной до настоящего времени является проблема полной автоматизации процесса управления службой скорой медицинской помощи. Развитие данной службы невозможно без полной автоматизации процесса управления в условиях постоянного роста количества выполняемых вызовов.

Основой информационного обеспечения является система учета и отчетности, определяющая документальное обеспечение деятельности службы и взаимодействие ее структурных подразделений. Сегодня выбор медицинских информационных систем, с учетом их высокой стоимости, является одним из самых рискованных управленческих решений для руководителя любого уровня.

Существующая и по сей день ручная технология сбора и обработки оперативной информации не позволяет быстро и эффективно управлять выездными бригадами скорой медицинской помощи, оперативно проводить всесторонний анализ качественных и количественных показателей деятельности службы.

Так как работа скорой помощи очень важна, появляется необходимость оптимизировать ее работу для увеличения количества вызовов, которые она может выполнять за одну рабочую смену. Данная программа должна улучшить качество работы, скорость реагирования на вызовы, учет статистических и экономических данных.

Качество работы: в процессе обработки вызовов за счет заранее подготовленных данных, таких как улица, заболевание, наименование медикаментов, сложнее ошибиться.

Скорость реагирования: чем быстрее будет передаваться вызов бригаде, тем больше вызовов можно сделать.

Учет статистических и экономических данных: благодаря базе данных, в которых хранятся вызова, проще рассчитывать расход медикаментов и количество топлива для машин скорой помощи.

База данных представляет собой определенный набор данных, которые, как правило, связаны объединяющим признаком либо свойством (или несколькими). Эти данные упорядочены, например, по алфавиту. Обилие различных данных, которые могут быть помещены в единую базу, ведет к множеству вариаций того, что может быть записано: личные данные пользователей, записи, даты, заказы и так далее.

В первую очередь это удобно тем, что информацию можно быстро заносить в базу данных и так же быстро ее извлекать при необходимости. Немаловажной является и взаимосвязь информации в базе данных: изменение одной строчки может привести к значительным изменениям других строк. Работать с данными таким образом гораздо проще и быстрее, чем если бы изменения касались только одного места в базе данных.

Изм.

Лист

№ докум.

Подпись

Дата

Лист

6

Курсовая Работа

**Цель работы**:

Создать приложение позволяющее облегчить и автоматизировать работу станций скорой медицинской помощи. Программа должна осуществлять передачу информации о вызовах между подстанциями и вести учет больных и их хронических заболеваний. Отслеживать работу бригад скорой помощи и вести учет вызовов, на которые реагировала бригада.

**Задача работы**:

Приложение должно осуществлять прием вызовов, выполнять обработку их, хранить данные в базе и осуществлять поиск информации о вызове. Выводить информацию о больных и о вызовах сделанных каждой бригадой.

1. **ПОСТАНОВКА ЗАДАЧИ**

Изм.

Лист

№ докум.

Подпись

Дата

Лист

7

Курсовая Работа

## Постановка задачи

Целью курсовой работы является спроектировать приложение позволяющее автоматизировать работу диспетчера скорой медицинской помощи.

Программа должна моделировать клиент – серверное приложение. На стороне сервера должна содержаться вся нужна информация для принятия вызова, и его обработки, а именно :

1. Название улиц
2. Название диагнозов
3. Название подстанций скорой помощи
4. Список типов людей, которые вызывают
5. Список типов бригад
6. Список мест обслуживания
7. Наименования лекарств и действий скорой помощи

Приложение должно осуществлять регистрацию вызова, передачу его бригадам, ввод бригад, учет статистических данных исходя из обработанных вызовов, учет хронических больных.

После регистрации вызова, он передается на сервер, где хранятся все не обслуженных вызова. Просмотр этих вызовов доступен всем подстанциям. Когда освобождается бригада, ей передается вызов для обслуживания. Бригада, выполнив вызов и записав точную информацию о пациенте, передает все данные диспетчеру для редактирования и финальной обработки вызова.

Обработанный вызов передается на сервер, где сохраняется в базу данных и может потом использоваться для учета статистических данных.

Ведение справочника хронических больных осуществляется с помощью введение в всей нужной информации, по обслуживанию пациента и регистрации вызова, в базу, что позволит быстро принять заявку и передать его бригаде.

Просмотр всех вызовов, включая попутные, и подробной информации каждого вызова.

Просмотр вызовов выполненные каждой бригадой за смену.

## Пример решения аналогичной задачи

В настоящее время, станции скорой медицинской помощи используют программу, позволяющая частично автоматизировать работу диспетчера скорой помощи.

На телефон диспетчера поступает вызов, звонящий называет фамилию пациента, его возраст, улицу, на которой находится больной, номер дома и номер квартиры, также называет контактные телефоны. Далее вызов передается диспетчеру конкретной подстанции, расположенной ближе других к этому адресу.

Изм.

Лист

№ докум.

Подпись

Дата

Лист

8

Курсовая Работа

Вызов от диспетчера подстанции передается конкретной свободной бригаде. На месте вызова бригадой уточняется информация такая как: место прописки пациента, его ФИО, возраст и заболевания. По возвращении всея уточненная информация передается диспетчеру для обработки и занесения в базу.

Также есть возможность просмотра статистики по вызовам, по подстанциям, по бригадам.

Достоинствами такой системы являются:

1. Скорость передачи вызова между подстанциями.
2. Хранение данных о больных и сотрудниках в базе.
3. Скорость реагирования на вызов.

Недостатки:

1. Ручной ввод вызовов
2. Ручная обработка выполненных вызовов
3. Неполность продуманный интерфейс программы.
4. **АРХИТЕКТУРА ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ**

Изм.

Лист

№ докум.

Подпись

Дата

Лист

9

Курсовая Работа

## Анализ задания на проектирование

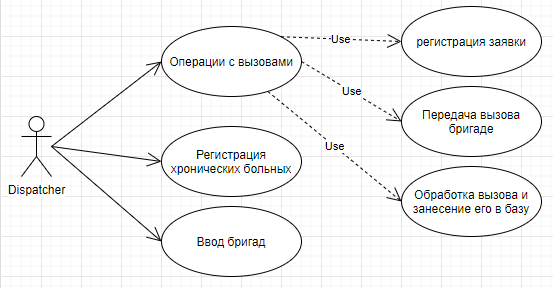
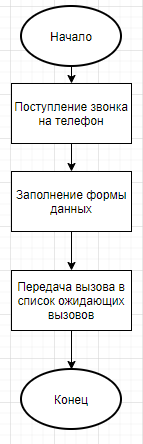


Рисунок 1 – Действия диспетчера

В обязанности диспетчера скорой помощи входит регистрация вызовов, их передача, финальная обработка ввод бригад скорой помощи и регистрация хронических больных.

### **Регистрация вызова**

Вызов поступает на телефон станции, диспетчер записывает фамилию, имя, отчество, возраст пациента, его состояние, местоположение и кем приходится пациенту вызывающий. Все это заносится в поля формы и передается в список не выполненных вызовов.



Изм.

Лист

№ докум.

Подпись

Дата

Лист

10

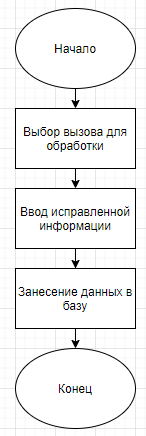
Курсовая Работа

Рисунок 2 – Алгоритм регистрации вызова

В случае ошибочного заполнения, доступна функция очистки формы.

### **Обработка вызова**

После выполнения вызова, он поступает в список вызовов для обработки. Диспетчер должен заменить неточную информацию о вызове на информацию, уточненную у пациента.



Изм.

Лист

№ докум.

Подпись

Дата

Лист

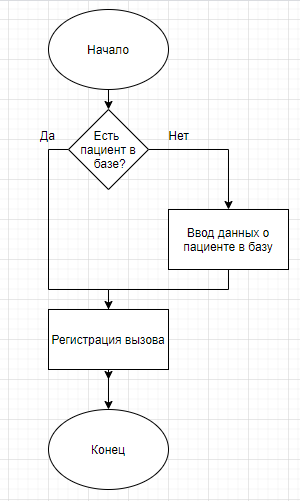
11

Курсовая Работа

Рисунок 3 – Алгоритм обработки вызова

### **Хронические больные**

Диспетчер осуществляет прием вызова хронически больного и занесение всей нужной информации в базу данных, что позволяет быстрее принимать вызов от этого же пациента.



Изм.

Лист

№ докум.

Подпись

Дата

Лист

12

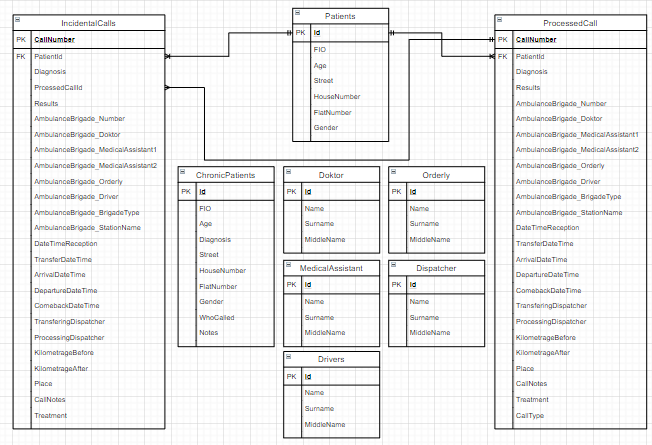
Курсовая Работа

Рисунок 4 – Алгоритм принятия вызова хронического больного

## Доменные классы

Проанализировав предметную область, можно выделить следующие девять доменных классов :

* (Пациент) Patients
* (Хронически больной) ChronicPatients
* (Диспетчер) Dispatcher
* (Доктор) Doktor
* (Фельдшер) MedicalAssistant
* (Санитар) Orderly
* (Водитель) Driver
* (Обработанный вызов) ProcessedCall
* (Попутный вызов) IncidentalCall



Изм.

Лист

№ докум.

Подпись

Дата

Лист

13

Курсовая Работа

Рисунок 5 – Диаграмма доменных классов

Класс Patient.

Описывает сущность «Пациент», которого обслуживает бригада скорой помощи.

Таблица 1 – Класс Patient.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | Модификатор | Имя | Наименование | Тип | Комментарий |
| С  в  о  й  с  т  в  а | public | Id | ID | int |  |
| public | FIO | Фамилия Имя Отчество | string | Фамилия Имя Отчество пациента |
| public | Age | Возраст | double | Возраст пациента |
| public | Diagnosis | Диагноз | string | Диагноз поставленный бригадой |
| public | Street | Улица адреса прописки | string | Улица на которой прописан пациент |
| public | HouseNumber | Номер дома адреса прописки | string | Номер дома в котором прописан |
| public | FlatNumber | Номер квартиры адреса прописки | string | Номер квартиры в которой прописан |
| public | Gender | Пол пациента | string | Пол пациента |

Изм.

Лист

№ докум.

Подпись

Дата

Лист

14

Курсовая Работа

Класс ChronicPatients.

Описывает сущность «Пациент с хроническими заболеваниями» и содержит в себе вид его заболевания.

Таблица 2 – Класс ChronicPatients

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | Модификатор | Имя | Наименование | Тип | Комментарий |
| С  в  о  й  с  т  в  а | public | Id | ID | int | Идентификационный номер |
| public | FIO | Фамилия Имя Отчество | string | Фамилия Имя Отчество пациента |
| public | Age | Возраст | double | Возраст пациента |
| public | Diagnosis | Диагноз | string | Диагноз поставленный бригадой |
| public | Street | Улица адреса прописки | string | Улица на которой прописан пациент |
| public | HouseNumber | Номер дома адреса прописки | string | Номер дома в котором прописан |
| public | FlatNumber | Номер квартиры адреса прописки | string | Номер квартиры в которой прописан |
| public | Gender | Пол пациента | string | Пол пациента |
| public | WhoCalled | Кто вызывает скорую | string | Кто вызывает скорую для пациента |
| public | Notes | Заметки | string | Заметки к вызову |

Класс Dispatcher.

Описывает сущность «Диспетчер скорой помощи», который работает с программой.

Таблица 3 – Класс Dispatcher

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | Модификатор | Имя | Наименование | Тип | Комментарий |
| С  в  о  й  с  т  в | public | Id | ID | int | Идентификационный номер |
| public | Name | Имя | string | Имя сотрудника |
| public | Surname | Фамилия | string | Фамилия сотрудника |
| public | MiddleName | Отчество | string | Отчество сотрудника |
| public | FIO | Ф.И.О. | string | Позволяет получить ФИО в полном виде |
| public | GetPosition | Должность сотрудника | string | Позволяет получить должность сотрудника |

Изм.

Лист

№ докум.

Подпись

Дата

Лист

15

Курсовая Работа

Класс Doktor

Описывает сущность «Доктор выездной бригады» скорой помощи.

Таблица 4 – Класс Doktor

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | Модификатор | Имя | Наименование | Тип | Комментарий |
| С  в  о  й  с  т  в | public | Id | ID | int | Идентификационный номер |
| public | Name | Имя | string | Имя сотрудника |
| public | Surname | Фамилия | string | Фамилия сотрудника |
| public | MiddleName | Отчество | string | Отчество сотрудника |
| public | FIO | Ф.И.О. | string | Позволяет получить ФИО в полном виде |
| public | GetPosition | Должность сотрудника | string | Позволяет получить должность сотрудника |

Класс MedicalAssistant.

Описывает сущность «Фельдшер выездной бригады» скорой помощи.

Таблица 5 – Класс MedicalAssistant

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | Модификатор | Имя | Наименование | Тип | Комментарий |
| С  в  о  й  с  т  в | public | Id | ID | int | Идентификационный номер |
| public | Name | Имя | string | Имя сотрудника |
| public | Surname | Фамилия | string | Фамилия сотрудника |
| public | MiddleName | Отчество | string | Отчество сотрудника |
| public | FIO | Ф.И.О. | string | Позволяет получить ФИО в полном виде |
| public | GetPosition | Должность сотрудника | string | Позволяет получить должность сотрудника |

Изм.

Лист

№ докум.

Подпись

Дата

Лист

16

Курсовая Работа

Класс Orderly.

Описывает сущность «Санитар выездной бригады» скорой помощи.

Таблица 6 – Класс Orderly

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | Модификатор | Имя | Наименование | Тип | Комментарий |
| С  в  о  й  с  т  в | public | Id | ID | int | Идентификационный номер |
| public | Name | Имя | string | Имя сотрудника |
| public | Surname | Фамилия | string | Фамилия сотрудника |
| public | MiddleName | Отчество | string | Отчество сотрудника |
| public | FIO | Ф.И.О. | string | Позволяет получить ФИО в полном виде |
| public | GetPosition | Должность сотрудника | string | Позволяет получить должность сотрудника |

Класс Driver.

Описывает сущность «Водитель машины выездной бригады» скорой помощи.

Таблица 7 – Класс Driver

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | Модификатор | Имя | Наименование | Тип | Комментарий |
| С  в  о  й  с  т  в  а | public | Id | ID | int | Идентификационный номер |
| public | Name | Имя | string | Имя сотрудника |
| public | Surname | Фамилия | string | Фамилия сотрудника |
| public | MiddleName | Отчество | string | Отчество сотрудника |
| public | FIO | Ф.И.О. | string | Позволяет получить ФИО в полном виде |
| public | GetPosition | Должность сотрудника | string | Позволяет получить должность сотрудника |

Изм.

Лист

№ докум.

Подпись

Дата

Лист

17

Курсовая Работа

Класс ProccessedCall.

Описывает сущность «Обработанный вызов», содержит в себе информацию о пациенте, о бригаде, которая выполняла вызов, о лечении и результате лечения.

Таблциа 8 – Класс ProcessedCall

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | Модифика  тор | Имя | Наименование | Тип | Комментарий |
| С  в  о  й  с  т  в  а | public | CallNumber | Номер вызова | uint | Уникальный номер вызова |
| public | PatientId | Номер пациента | int | Номер пациента |
| public | Patient | Пациент | Patient | Информация о пациенте |
| public | Results | Результат | string | Результата вызова |
| public | AmbulanceBrigade | Бригада | AmbulanceBrigade | Информация о бригаде, обслужившая вызов |
| public | DateTimeReception | Дата и время получения | DateTime | Дата и время получения вызова |
| public | TransferDateTime | Дата и время передачи | DateTime | Дата и время передачи вызова |
| public | ArrivalDateTime | Дата и время прибытия | DateTime | Дата и время прибытия вызова |
| public | DepartureDateTime | Дата и время отправления | DateTime | Дата и время выезда с вызова |
| public | ComeBackDateTime | Дата и время возвращения | DateTime | Дата и время возвращения на подстанцию |
| public | TransferringDispatcher | Передающий диспетчер | string | Диспетчер передавший вызов |
| public | ProcessingDispatcher | Обрабатывающий диспетчер | string | Диспетчер обработавший вызов |
| public | KilometrageBefor | Километраж до | int | Километраж машины до вызова |
| public | KilometrageAfter | Километраж после | int | Километраж машины после вызова |
| public | Place | Место | string | Место обслуживания пациента |
| public | CallNotes | Заметки | string | Заметки к вызову |
| public | Treatment | Лечение | string | Лечение |
| public | CallType | Тип вызова | string | Тип вызова |

Изм.

Лист

№ докум.

Подпись

Дата

Лист

18

Курсовая Работа

Класс IncidentalCalls.

Описывает сущность «Попутный вызов», содержит в себе, туже информацию что и класс ProcessedCall, но, так же, содержит в себе номер основного вызова.

Таблица 9 – Класс IncidentalCall

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | Моди  фика  тор | Имя | Наименование | Тип | Комментарий |
| С  в  о  й  с  т  в  а | public | CallNumber | Номер вызова | uint | Уникальный номер вызова |
| public | ProcessedCallId | Номер основного вызова | uint | Номер основного вызова |
| public | PatientId | Номер пациента | Int | Номер пациента |
| public | Patient | Пациент | Patient | Информация о пациенте |
| public | Results | Результат | string | Результата вызова |
| public | AmbulanceBrigade | Бригада | AmbulanceBrigade | Информация о бригаде, обслужившая вызов |
| public | DateTimeReception | Дата и время получения | DateTime | Дата и время получения вызова |
| public | TransferDateTime | Дата и время передачи | DateTime | Дата и время передачи вызова |
| public | ArrivalDateTime | Дата и время прибытия | DateTime | Дата и время прибытия вызова |
| public | DepartureDateTime | Дата и время отправления | DateTime | Дата и время выезда с вызова |
| public | ComeBackDateTime | Дата и время возвращения | DateTime | Дата и время возвращения на подстанцию |
| public | TransferringDispatcher | Передающий диспетчер | string | Диспетчер передавший вызов |
| public | ProcessingDispatcher | Обрабатывающий диспетчер | string | Диспетчер обработавший вызов |
| public | KilometrageBefor | Километраж до | int | Километраж машины до вызова |
| public | KilometrageAfter | Километраж после | int | Километраж машины после вызова |
| public | Place | Место | string | Место обслуживания пациента |
| public | CallNotes | Заметки | string | Заметки к вызову |
| public | Treatment | Лечение | string | Лечение |
| public | CallType | Тип вызова | string | Тип вызова |

Изм.

Лист

№ докум.

Подпись

Дата

Лист

19

Курсовая Работа

1. **СТРУКТУРА И АЛГОРИТМЫ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ**

## Язык и среда программирования

C# является объектно-ориентированным языком. Разработка современных приложений все больше тяготеет к созданию программных компонентов в форме автономных и самоописательных пакетов, реализующих отдельные функциональные возможности.

Главная особенность таких компонентов в том, что они представляют собой модель программирования со свойствами, методами и событиями. У них есть атрибуты, предоставляющие декларативные сведения о компоненте. Они включают в себя собственную документацию. C# предоставляет языковые конструкции, непосредственно поддерживающие такую концепцию работы. Благодаря этому C# подходит для создания и применения программных компонентов.

Когда говорят C#, нередко имеют в виду технологии платформы .NET (Windows Forms, WPF, ASP.NET, Xamarin). И, наоборот, когда говорят .NET, нередко имеют в виду C#. Язык C# был создан специально для работы с фреймворком .NET, однако само понятие .NET несколько шире. Фреймворк .NET представляет мощную платформу для создания приложений. Можно выделить следующие ее основные черты:

1. Поддержка нескольких языков. Основой платформы является общеязыковая среда исполнения Common Language Runtime (CLR), благодаря чему .NET поддерживает несколько языков: наряду с C# это также VB.NET, C++, F#, а также различные диалекты других языков, привязанные к .NET, например, Delphi.NET. При компиляции код на любом из этих языков компилируется в сборку на общем языке CIL (Common Intermediate Language) - своего рода ассемблер платформы .NET. Поэтому мы можем сделать отдельные модули одного приложения на отдельных языках.
2. Кроссплатформенность. .NET является переносимой платформой (с некоторыми ограничениями). Последняя версия платформы на данный момент .NET Core поддерживается на большинстве современных ОС Windows, MacOS, Linux. Используя различные технологии на платформе .NET, можно разрабатывать приложения на языке C# для самых разных платформ - Windows, MacOS, Linux, Android, iOS, Tizen.
3. Мощная библиотека классов. .NET представляет единую для всех поддерживаемых языков библиотеку классов.

Изм.

Лист

№ докум.

Подпись

Дата

Лист

20

Курсовая Работа

## Основы графических интерфейсов

Изм.

Лист

№ докум.

Подпись

Дата

Лист

21

Курсовая Работа

**Windows Forms** — [интерфейс программирования приложений](https://ru.wikipedia.org/wiki/%D0%98%D0%BD%D1%82%D0%B5%D1%80%D1%84%D0%B5%D0%B9%D1%81_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D1%8F_%D0%BF%D1%80%D0%B8%D0%BB%D0%BE%D0%B6%D0%B5%D0%BD%D0%B8%D0%B9) (API), отвечающий за [графический интерфейс пользователя](https://ru.wikipedia.org/wiki/%D0%93%D1%80%D0%B0%D1%84%D0%B8%D1%87%D0%B5%D1%81%D0%BA%D0%B8%D0%B9_%D0%B8%D0%BD%D1%82%D0%B5%D1%80%D1%84%D0%B5%D0%B9%D1%81_%D0%BF%D0%BE%D0%BB%D1%8C%D0%B7%D0%BE%D0%B2%D0%B0%D1%82%D0%B5%D0%BB%D1%8F) и являющийся частью [Microsoft](https://ru.wikipedia.org/wiki/Microsoft) [.NET Framework](https://ru.wikipedia.org/wiki/.NET_Framework). Данный интерфейс упрощает доступ к элементам интерфейса [Microsoft Windows](https://ru.wikipedia.org/wiki/Microsoft_Windows) за счет создания обёртки для существующего [Win32 API](https://ru.wikipedia.org/wiki/Windows_API) в [управляемом коде](https://ru.wikipedia.org/wiki/%D0%A3%D0%BF%D1%80%D0%B0%D0%B2%D0%BB%D1%8F%D0%B5%D0%BC%D1%8B%D0%B9_%D0%BA%D0%BE%D0%B4). Причём управляемый код — классы, реализующие [API](https://ru.wikipedia.org/wiki/API) для Windows Forms, не зависят от языка разработки. То есть программист одинаково может использовать Windows Forms как при написании ПО на C#, С++, так и на VB.Net, J# и др.

С одной стороны, Windows Forms рассматривается как замена более старой и сложной библиотеке [MFC](https://ru.wikipedia.org/wiki/Microsoft_Foundation_Classes), изначально написанной на языке [C++](https://ru.wikipedia.org/wiki/C%2B%2B). С другой стороны, WF не предлагает парадигму, сравнимую с [MVC](https://ru.wikipedia.org/wiki/Model-View-Controller). Для исправления этой ситуации и реализации данной функциональности в WF существуют сторонние библиотеки.

Windows Forms включает широкий набор элементов управления, которые можно добавлять на формы: текстовые поля, кнопки, раскрывающиеся списки, переключатели и даже веб-страницы. Список всех элементов управления, которые можно использовать в форме, представлены в разделе [Элементы управления для использования в формах Windows Forms](https://docs.microsoft.com/ru-ru/dotnet/framework/winforms/controls/controls-to-use-on-windows-forms). Если существующий элемент управления не удовлетворяет потребностям, в Windows Forms можно создать пользовательские элементы управления с помощью класса [UserControl](https://docs.microsoft.com/ru-ru/dotnet/api/system.windows.forms.usercontrol).

В состав Windows Forms входят многофункциональные элементы пользовательского интерфейса, позволяющие воссоздавать возможности таких сложных приложений, как Microsoft Office. Используя элементы управления [ToolStrip](https://docs.microsoft.com/ru-ru/dotnet/api/system.windows.forms.toolstrip) и [MenuStrip](https://docs.microsoft.com/ru-ru/dotnet/api/system.windows.forms.menustrip), можно создавать панели инструментов и меню, содержащие текст и рисунки, подменю и другие элементы управления, такие как текстовые поля и поля со списками.

Во многих приложениях нужно отображать данные из базы данных, XML-файла, веб-службы XML или другого источника данных. Windows Forms предоставляет гибкий элемент управления с именем [DataGridView](https://docs.microsoft.com/ru-ru/dotnet/api/system.windows.forms.datagridview) для отображения таких табличных данных в традиционном формате строк и столбцов так, что каждый фрагмент данных занимает свою собственную ячейку. С помощью [DataGridView](https://docs.microsoft.com/ru-ru/dotnet/api/system.windows.forms.datagridview) можно, помимо прочего, настроить внешний вид отдельных ячеек, зафиксировать строки и столбцы на своем месте, а также обеспечить отображение сложных элементов управления внутри ячеек.

При использовании интеллектуальных клиентов Windows Forms можно легко подключаться к источникам данных по сети. Компонент [BindingSource](https://docs.microsoft.com/ru-ru/dotnet/api/system.windows.forms.bindingsource) представляет соединение с источником данных и предоставляет методы для привязки данных к элементам управления, перехода к предыдущим и следующим записям, изменения записей и сохранения изменений в исходном источнике. Элемент управления [BindingNavigator](https://docs.microsoft.com/ru-ru/dotnet/api/system.windows.forms.bindingnavigator) предоставляет простой интерфейс на основе компонента [BindingSource](https://docs.microsoft.com/ru-ru/dotnet/api/system.windows.forms.bindingsource) для перехода между записями.

Изм.

Лист

№ докум.

Подпись

Дата

Лист

22

Курсовая Работа

Вы можете легко создавать элементы управления с привязкой к данным с помощью окна "Источники данных". В нем приводятся имеющиеся в проекте источники данных, такие как базы данных, веб-службы и объекты. Создавать элементы управления с привязкой к данным можно путем перетаскивания объектов из этого окна в формы проекта. Также можно связывать существующие элементы управления с данными, перетаскивая объекты из окна "Источники данных" в существующие элементы управления.

**Технология WPF** (Windows Presentation Foundation) является часть экосистемы платформы .NET и представляет собой подсистему для построения графических интерфейсов.

Если при создании традиционных приложений на основе WinForms за отрисовку элементов управления и графики отвечали такие части ОС Windows, как User32 и GDI+, то приложения WPF основаны на DirectX. В этом состоит ключевая особенность рендеринга графики в WPF: используя WPF, значительная часть работы по отрисовке графики, как простейших кнопочек, так и сложных 3D-моделей, ложиться на графический процессор на видеокарте, что также позволяет воспользоваться аппаратным ускорением графики.

Одной из важных особенностей является использование языка декларативной разметки интерфейса XAML, основанного на XML. XAML (Extensible Application Markup Language — расширяемый язык разметки приложений) представляет собой язык разметки, используемый для создания экземпляров объектов .NET. Хотя язык XAML — это технология, которая может быть применима ко многим различным предметным областям, его главное назначение — конструирование пользовательских интерфейсов WPF.

WPF позволяет разрабатывать приложения, используя как разметку, так и код программной части, что привычно для разработчиков на ASP.NET. Разметка XAML обычно используется для определения внешнего вида приложения, а управляемые языки программирования (код программной части) — для реализации его поведения. Такое разделение внешнего вида и поведения имеет ряд преимуществ.

1. Затраты на разработку и обслуживание снижаются, так как разметка, определяющая внешний вид, не связана тесно с кодом, обуславливающим поведение.
2. Повышается эффективность разработки, так как дизайнеры, занимающиеся внешним видом приложения, могут работать параллельно с разработчиками, реализующими поведение приложения.

Платформа WPF предоставляет широкий, гибкий и масштабируемый набор графических функций, который обладает перечисленными ниже преимуществами.

Изм.

Лист

№ докум.

Подпись

Дата

Лист

23

Курсовая Работа

1. Независимость графики от разрешения и устройства. Основной единицей измерения в графической системе WPF является аппаратно-независимый пиксель, размер которого составляет 1/96 дюйма вне зависимости от разрешения экрана. Это создает основу для, независимой от разрешения и аппаратной платформы, отрисовки. Каждый аппаратно-независимый пиксель автоматически масштабируется в соответствии с заданным в системе количеством точек на дюйм (DPI).
2. Повышение точности. Система координат WPF основана на числах двойной точности с плавающей запятой, а не числах одинарной точности. Значения преобразования и прозрачности также выражаются числами двойной точности. Платформа WPF также поддерживает широкую цветовую палитру (scRGB) и имеет встроенную поддержку управления входными данными из разных цветовых схем.
3. Аппаратное ускорение. Система графики WPF использует возможности графического оборудования, чтобы снизить нагрузку на центральный процессор.

В результате, язык C# и принципы объектно - ориентированного программирования подходят для реализации программы, а технология WPF не только проста в использовании, но и обеспечит кроссплатформенность, что позволит запускать программу на практически любом компьютере.

## Структурная схема приложения и описание

Изм.

Лист

№ докум.

Подпись

Дата

Лист

24

Курсовая Работа

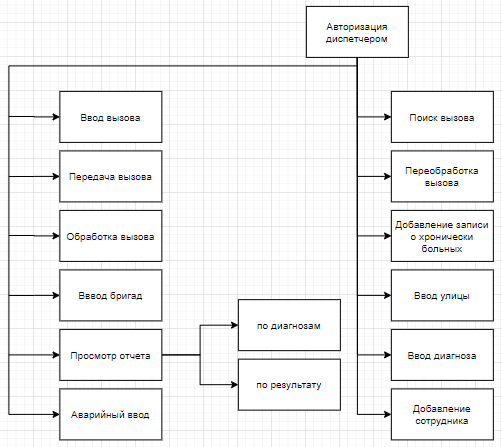


Рисунок 6 – Структурная схема программы

Формы и их назначения:

* MainWindow (Основное окно для остальных)
* ChronicPatientWindow (Позволяет быстро принять вызов хронически больного)
* AddRemoveWindow (Добавление или удаление диагноза или улицы)
* CallInformation (Подробная информация о вызове бригады)
* CallMonitoring (Обзор не обслуженных вызовов)
* CarsInfrmationWindow (Просмотр вызовов сделанные бригадой)
* DataWindow (Окно с добавлением бригад и хронических пациентов)
* DispatcherRegistrationWindow (Ввод диспетчера)
* EditingCalls (Редактирование обработанных вызовов)
* FullCallInfrmationWindow (Окно просмотра подробных данных вызова)
* InputCall (Аварийный ввод или ввод попутного вызова)
* ReportWindow (Окно с отчетной информацией)
* ReprocessingCallsWindow (Окно обработки вызовов)
* SearchForCallInfrmation (Поиск информации о вызове)
* TretmentWindow (Окно для ввода лечения)

Изм.

Лист

№ докум.

Подпись

Дата

Лист

25

Курсовая Работа

* WorkerWindow (Окно для добавления сотрудников)

## Вспомогательные классы

Вспомогательные классы:

1. AmbulanceBrigade
2. ServerService
3. AmbulanceDBContext

Класс AmbulanceBrigade представляет бригаду скорой помощи. Содержит в себе сотрудников конкретной бригады, ее номер и тип.

Таблица 10 – Класс AmbulanceBrigade

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | Моди  фика  тор | Имя | Наименование | Тип | Комментарий |
| С  в  о  й  с  т  в  а | public | Number | Номер бригады | uint | Уникальный номер бригады |
| public | Doktor | Доктор | string | ФИО врача |
| public | MedicalAssistants1 | Фельдшер | string | ФИО фельдшера |
| public | MedicalAssistants2 | Фельдшер | string | ФИО фельдшера |
| public | Orderly | Санитар | string | ФИО санитара |
| public | Driver | Водитель | string | ФИО водителя |
| public | BrigadeType | Тип бригады | string | Тип бригады |
| public | StationName | Название станции | string | Название станции |
| public | StreetOfLastCall | Улица | string | Улица последнего вызова |
| public | HouseNumberOfLastCall | Номер дом | string | Номер дома последнего вызова |
| public | NumberOfLastCall | Номер квартиры | string | Номер квартиры последнего вызова |
| public | ActionTime | Время действия | string | Время действия |
| public | ActionDate | Дата действия | string | Дата действия |
| public | IsOnCall | На вызове ли | Bool | Находится на вызове или нет |
| public | AmbulanceCalls | Список вызовов | List  <AmbulanceBrigade> | Список выполненных вызовов |
| М  е  т  о  д  ы | public | AtTheStation | На станции | void |  |
| public | SendCall | Передать вызов | void |  |
| private | OnACall | На вызове | void |  |

Изм.

Лист

№ докум.

Подпись

Дата

Лист

26

Курсовая Работа

Класс AmbulanceCall представляет временную версию вызова, возможно содержащая не точную информацию. С использованием данных этого вызова проводится финальная обработка.

Таблица 11 – Класс AmbulanceCall

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | Моди  фикатор | Имя | Наименование | Тип | Комментарий |
| С  в  о  й  с  т  в  а | public | CallNumber | Номер вызова | uint | Уникальный номер бригады |
| public | Patient | Пациент | Patient | Информация о пациенте |
| public | DateTimeReception | Дата и время получения | DateTime | Дата и время получения |
| public | TransferDateTime | Дата и время передачи | DateTime | Дата и время передачи |
| public | CallNotes | Заметки | string | Заметки к вызову |
| public | WhoCalled | Кто вызвал | string | Кто вызвал |
| public | Type | Тип | string | Тип вызова |
| public | ReceivingDispatcher | Диспетчер принявший | string | Диспетчер принявший |
| public | TransferringDispatcher | Диспетчер передавший | string | Диспетчер передавший |
| public | Department | Станция | string | На которую поступил вызов |

Класс ServerService представляет собой класс, имитирующий доступ к серверу, на котором содержится вся нужна информация для регистрации и обработки вызова.

Таблица 12 – Методы ServerService

|  |  |  |  |
| --- | --- | --- | --- |
| Имя | Тип | Модификатор | Комментарии |
| GetDepartments | List<string> | public static | Возвращает список подстанций |
| GetCallers | List<string> | public static | Возвращает список возможных вызывающих |
| GetBrigadeTypes | List<string> | public static | Возвращает список типов бригад |
| AddDiagnosis | string | public static | Добавляет диагноз в список |
| RemoveDiagnosis | string | public static | Удаляет диагноз в список |
| GetDiagnoses | List<string> | public static | Возвращает список диагнозов |
| AddStreet | string | public static | Добавляет улицу в список |
| RemoveStreet | string | public static | Удаляет улицу из списка |
| GetStreets | List<string> | public static | Возвращает список улиц |
| GetPlaces | List<string> | public static | Возвращает список мест обслуживания |
| GetTreatment | List<string> | public static | Возвращает список лекарств |
| GetResults | List<string> | public static | Возвращает список результатов вызова |

Изм.

Лист

№ докум.

Подпись

Дата

Лист

27

Курсовая Работа

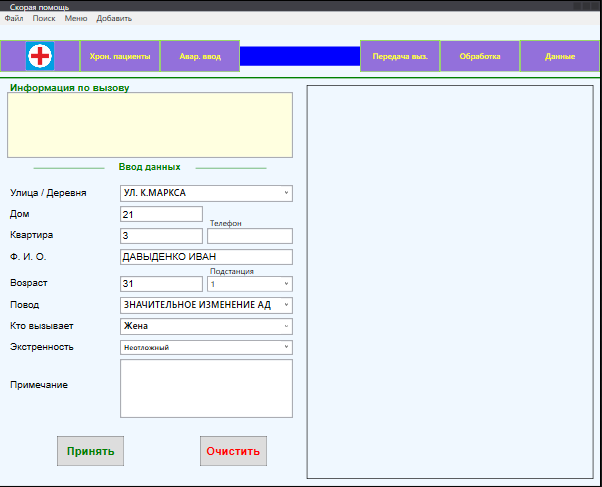
## Интерфейс пользователя

Пользовательский интерфейс - это совокупность программных и аппаратных средств, обеспечивающих взаимодействие пользователя с компьютером. Основу такого взаимодействия составляют диалоги. Под диалогом в данном случае понимают регламентированный обмен информацией между человеком и компьютером, осуществляемый в реальном масштабе времени и направленный на совместное решение конкретной задачи. Каждый диалог состоит из отдельных процессов ввода / вывода, которые физически обеспечивают связь пользователя и компьютера.

Все окна содержат обработку исключительных ситуаций, таких как: ввод текста в поле для числа, ввод слишком большого числа, не заполненное поле, и контроль вводимых данных.

Окно MainWindow.

Основное окно программы, с помощью которого можно получить доступ к другим окнам. Задача окна: содержание области для регистрации вызова и доступ к другим окнам.



Изм.

Лист

№ докум.

Подпись

Дата

Лист

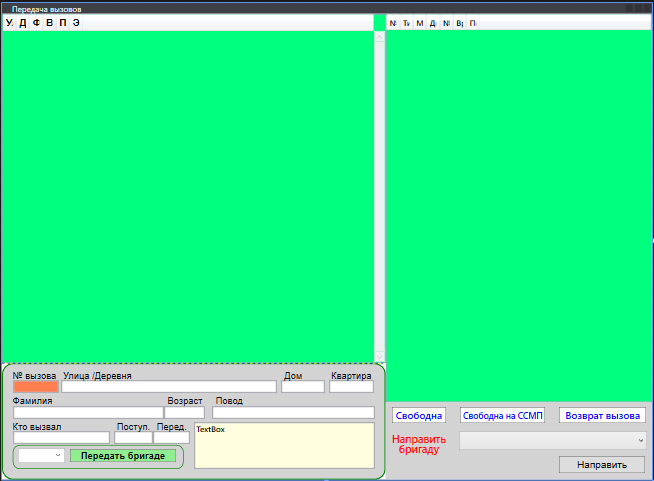
28

Курсовая Работа

Рисунок 7 – Основное окно

Окно CallMonitoring.

Цель окна – доступ к списку невыполненных вызовов, передача их бригадам, просмотр данных о вызове. Также, возможно управление статусом бригады: ремонт, обед, на АЗС.



Изм.

Лист

№ докум.

Подпись

Дата

Лист

29

Курсовая Работа

Рисунок 8 – Передача вызова

Окно EditingCalls.

Цель окна – обработка вызова. Вводится уточненная информация и после обработки вызов поступает в базу данных. Если во время вызова был выполнен попутный, имеется возможность ввести его, он будет связан с основным.

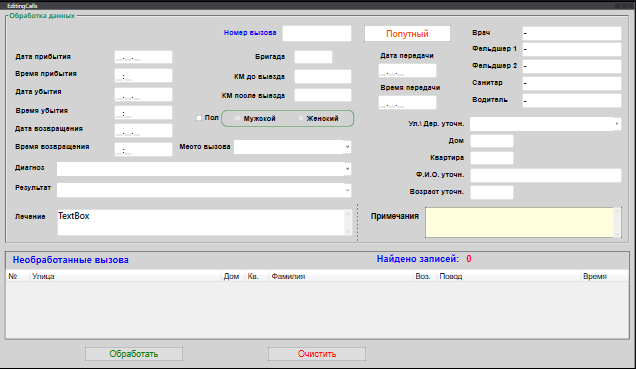


Рисунок 9 – Обработка вызова

Окно CarsInfrmationWindow.

Изм.

Лист

№ докум.

Подпись

Дата

Лист

30

Курсовая Работа

Выводит информацию о бригаде, а именно: ФИО врача, фельдшеров, санитара и водителя. Также возможен просмотр, выполненных бригадой, вызовов.

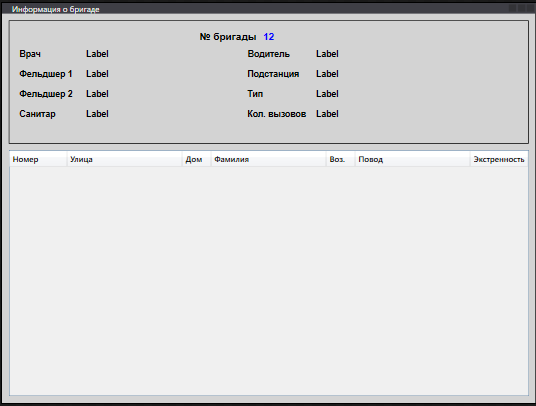


Рисунок 10 – Просмотр информации о бригаде

Окно AddRemoveWindow.

Используется для ввода или удаления диагнозов и улиц.

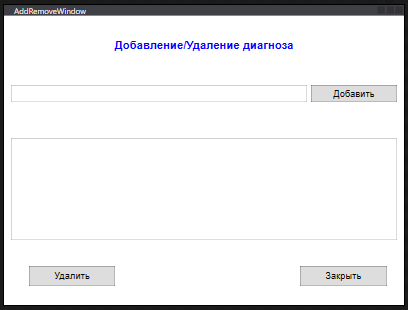


Рисунок 11 – Окно добавления или удаления

Изм.

Лист

№ докум.

Подпись

Дата

Лист

31

Курсовая Работа

Окно CallInformation.

Позволяет просмотреть подробную информацию о вызове, на который направляется бригада.

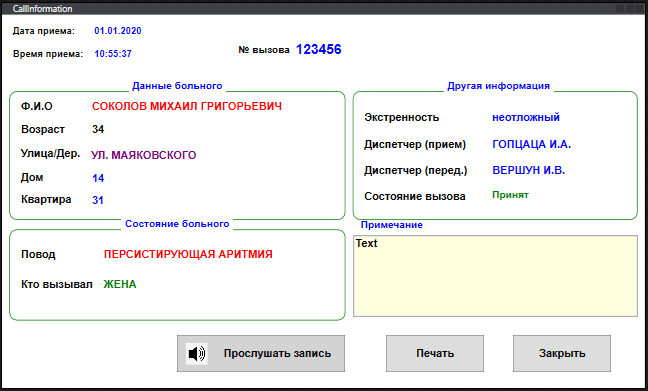
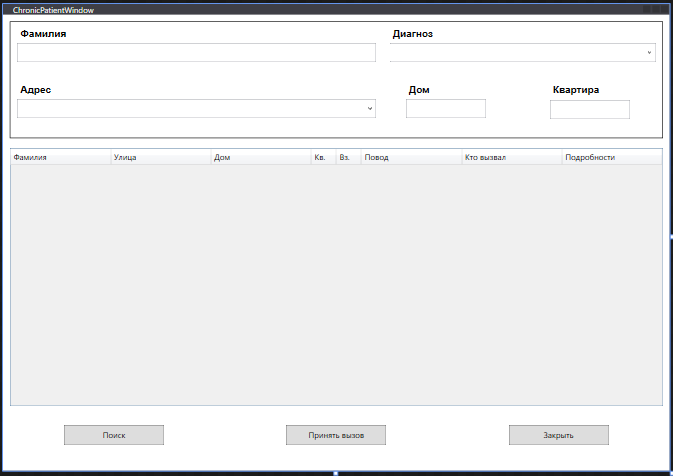


Рисунок 12 – Просмотр информации вызова

Окно ChronicPatientWindow.

Окно позволяет быстро найти и принять вызов от хронически больного пациента, это сделано для удобства ввода вызова, который появляется с завидной периодичностью.



Изм.

Лист

№ докум.

Подпись

Дата

Лист

32

Курсовая Работа

Рисунок 13 – Принятие вызова хронического больного

Окно DataWindow.

Позволяет вводить или удалять бригады. И вводить данные хронических больных или удалять записи о них.

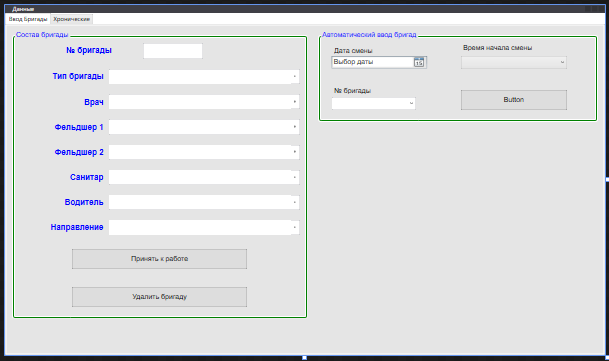
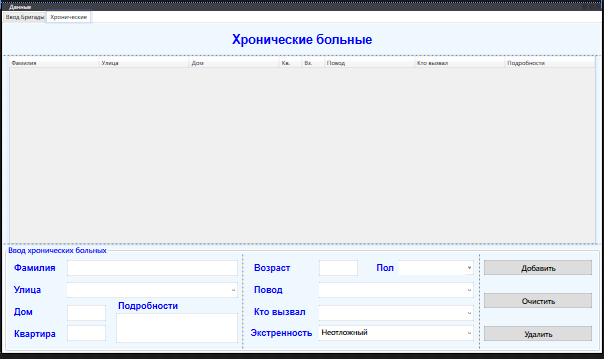


Рисунок 14 – Ввод бригад



Изм.

Лист

№ докум.

Подпись

Дата

Лист

33

Курсовая Работа

Рисунок 15 – Ввод хронических больных

Окно DispatcherRegistrationWindow.

Используется для ввода диспетчера в программу.

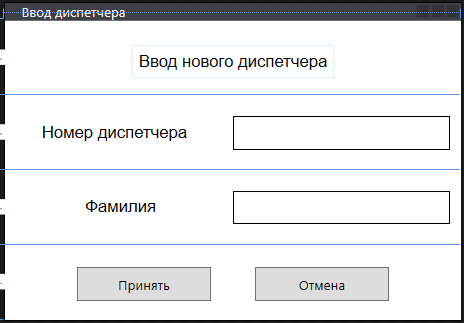
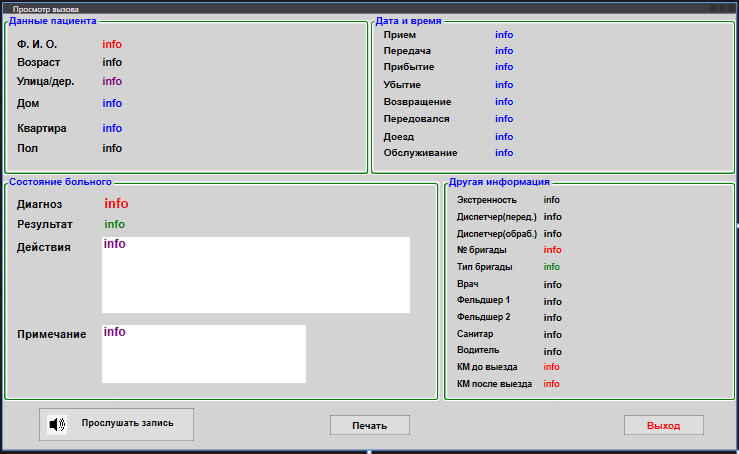


Рисунок 16 – Регистрация диспетчера в программе

Окно FullCallInformationWindow.

Позволяет просмотреть всю информацию обработанного вызова из базы данных.



Изм.

Лист

№ докум.

Подпись

Дата

Лист

34

Курсовая Работа

Рисунок 17 – Просмотр обработанного вызова

Окно InputCall.

Используется для аварийного ввода вызовов или для ввода попутных вызовов.

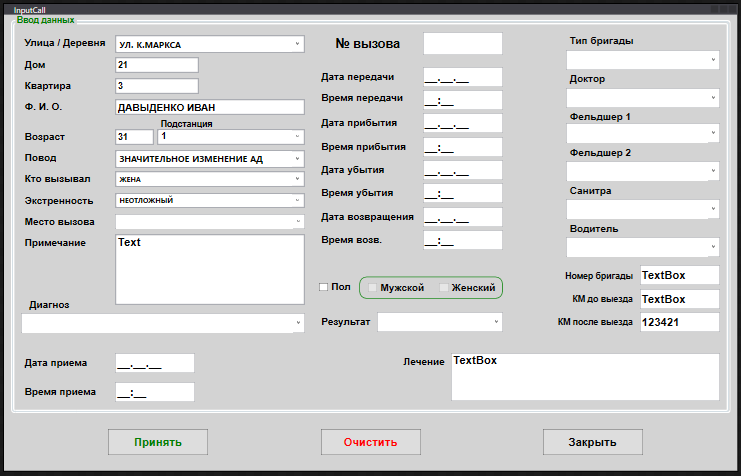


Рисунок 18 – Аварийный ввод

Окно ReportWindow.

Изм.

Лист

№ докум.

Подпись

Дата

Лист

35

Курсовая Работа

Позволяет просмотреть отчет по работе скорой помощи. Возможен просмотр количества вызовов по диагнозу или по результату.

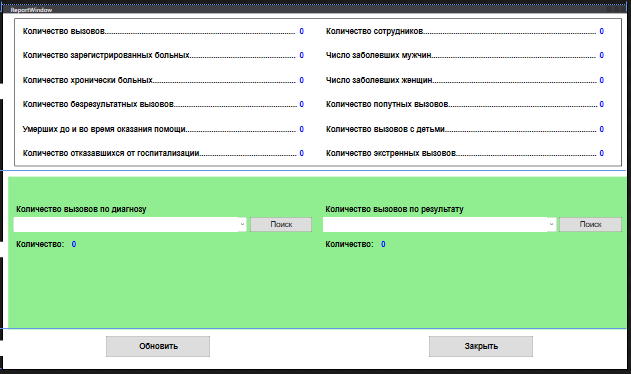


Рисунок 19 – Окно отчета

Окно ReprocessingCallsWindow.

Позволяет исправить данные вызова из базы данных.

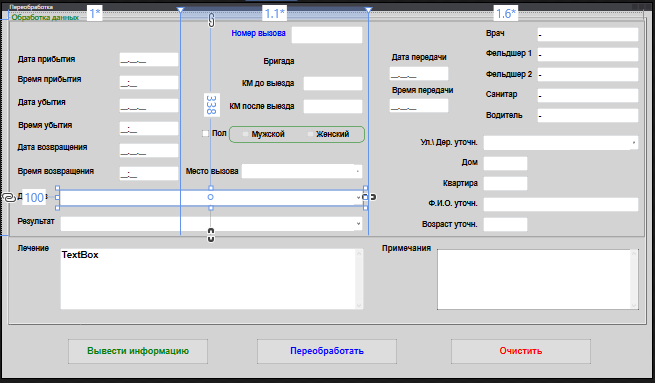


Рисунок 20 – Окно переобработки вызова

Окно SearchForCallInformation.

Изм.

Лист

№ докум.

Подпись

Дата

Лист

36

Курсовая Работа

Позволяет найти вызов по критериям и просмотреть полную информацию о нем.

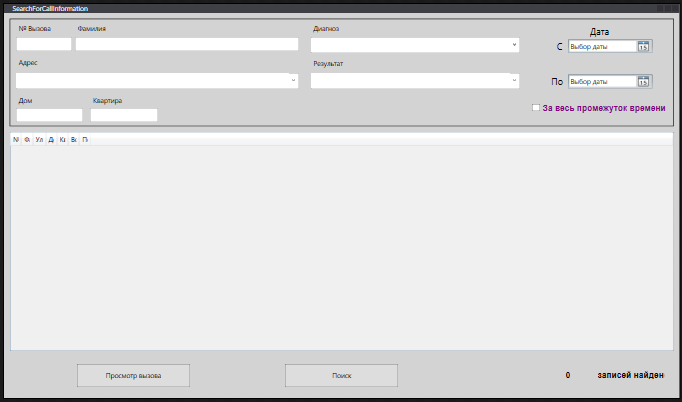


Рисунок 21 – Просмотр всех обработанных вызовов

Окно TreatmentWindow.

Используется для ввода лечения при обработке вызова диспетчером.

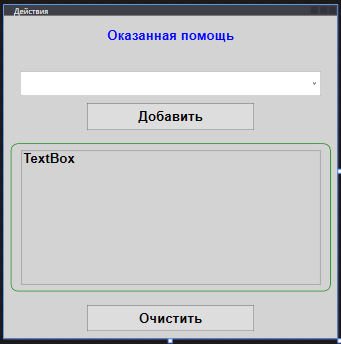


Рисунок 22 – Ввод действий бригады

Окно WorkerWindow.

Используется для ввода сотрудника или поиска информации о нем.

Изм.

Лист

№ докум.

Подпись

Дата

Лист

37

Курсовая Работа

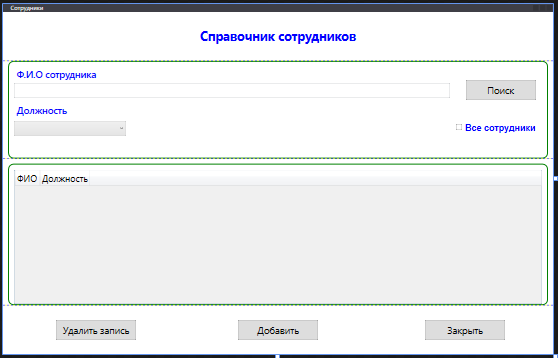


Рисунок 23 – Окно добавления или удаления сотрудников

1. **ВЕРИФИКАЦИЯ**

Таблица 13 – Соответствие Задание - Окно

|  |  |
| --- | --- |
| **Функция** | **Окно** |
| Ведение справочников | ChronicPatientWindow, WorkerWindow |
| Регистрация вызова | MainWindow |
| Результат выезда бригады | SearchForCallInformation |
| Ведомости работы бригад | CarsInfrmationWindow,  CallInformation |
| Анализ деятельности отделения скорой помощи | ReportWindow |

Таблица 14 – Соответствие Сценарий - Окно

|  |  |
| --- | --- |
| **Сценарий** | **Форма** |
| Регистрация вызова | MainWindow |
| Обработка вызова | EditingCalls |
| Ввод бригад | DataWindow |
| Поиск вызова в базе | SearchForCallInformation |
| Редактирование вызова | ReprocessingCallsWindow |
| Просмотр статистки | ReportWindow |
| Ввод хронических больных | DataWindow |
| Аварийный ввод | InputCall |
| Ввод попутного вызова | InputCall |

**ЗАКЛЮЧЕНИЕ**

Изм.

Лист

№ докум.

Подпись

Дата

Лист

38

Курсовая Работа

В ходе данного курсового проекта была смоделирована программа, автоматизирующая работу диспетчера скорой помощи. Она позволяет регистрировать вызова, передавать их бригаде, обрабатывать и хранить их в базе данных. Также возможен ввод бригады и список ее сотрудников. Присутствует функция просмотра статистики, которая отражает результативность работы скорой помощи, количество вызовов с определенным диагнозом или результатом. В процессе создания программы были созданы логическая модель, отношения между сущностями в базе данных и разработан графический интерфейс, который акцентирует внимание диспетчера на важные пункты вызова.

Были реализованы поставленные задачи и проведены тесты, для исключения непредвиденных ситуаций в ходе использования программы пользователями.

**СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ**

Изм.

Лист

№ докум.

Подпись

Дата

Лист

39

Курсовая Работа

1. Мак-Дональд, Мэтью WPF: Windows Presentation Foundation в .NET 4.5 с примерами на C# 5.0 для профессионалов / Мэтью Мак-Дональд. - Москва: Вильямс. 2019. – 1024 с
2. Натан, Адам WPF 4 Подробное руководство / Адам Натан. - Санкт-Петербург: Символ-Плюс. 2019. – 878 с
3. Рихтер, Джефри CLR via C#: Программирование на платформе Microsoft .NET Framework 4.5 на языке C# / Джефри Рихтер. - Санкт-Петербург: Питер, 2013. – 893 с
4. Петцольд, Чарльз программирование для Microsoft Windows 8 / Чарльз Петцольд. - Санкт-Петербург: Питер, 2014. – 1008 с

**ПРИЛОЖЕНИЕ А**

**MainWindow.xaml**

using System;

using System.Collections.ObjectModel;

using System.Windows;

using System.Linq;

using System.Windows.Threading;

using Ambulance.DBAccess;

using Ambulance.Classes;

namespace Ambulance

{

/// <summary>

/// Interaction logic for MainWindow.xaml

/// </summary>

public partial class MainWindow : Window

{

public static ObservableCollection<AmbulanceBrigade> Brigade = new ObservableCollection<AmbulanceBrigade>();

public static ObservableCollection<ProcessedCall> AmbulanceCallsForProcessing = new ObservableCollection<ProcessedCall>();

private static readonly ObservableCollection<AmbulanceCall> ambulanceCalls = new ObservableCollection<AmbulanceCall>();

private string \_dispatcher { get; set; }

public MainWindow()

{

InitializeComponent();

DispatcherTimer LiveTime = new DispatcherTimer

{

Interval = TimeSpan.FromSeconds(1)

};

LiveTime.Tick += Timer\_Tick;

LiveTime.Start();

using (var cont = new AmbulanceDBContext())

{

AmbulanceCall.\_nextcallNumber = ((uint)cont.ProcessedCalls.Count() +1);

}

ComboStreets.ItemsSource = ServerService.GetStreets();

ComboDepartment.ItemsSource = ServerService.GetDepartments();

ComboReason.ItemsSource = ServerService.GetDiagnoses();

ComboCaller.ItemsSource = ServerService.GetCallers();

}

void Timer\_Tick(object sender, EventArgs e)

{

DatePanel1.Text = DateTime.Now.GetDateTimeFormats('F')[1];

if (DateTime.Now.ToString("dd.MM") == "01.05")

{

AmbulanceCall.UpdateCallNumber();

}

}

public void DispetcherClick(object sender, RoutedEventArgs e)

{

DispatcherRegistrationWindow dispatcherRegistrationWindow = new DispatcherRegistrationWindow

{

Owner = this

};

dispatcherRegistrationWindow.ShowDialog();

\_dispatcher = dispatcherRegistrationWindow.AmbDispatcher;

}

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

{

ComboStreets.Text = "";

textHouseNumber.Clear();

textFlatNumber.Clear();

textTelephoneNumber.Clear();

textFIO.Clear();

textAge.Clear();

ComboDepartment.Text = "";

ComboReason.Text = "";

ComboCaller.Text = "";

ComboUrgency.Text = "";

textBoxNote.Clear();

}

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

{

if (\_dispatcher != null)

{

var callMonitoring = new CallMonitoring(ambulanceCalls, \_dispatcher);

callMonitoring.ShowDialog();

}

else

{

MessageBox.Show("Вы не зарегистрировались как диспетчер");

}

}

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

{

if (\_dispatcher != null)

{

var editCallsWindow = new EditingCalls(\_dispatcher);

editCallsWindow.Owner = this;

editCallsWindow.Show();

}

else

{

MessageBox.Show("Вы не зарегистрировались как диспетчер");

}

}

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

{

if (\_dispatcher != null)

{

var dataWindow = new DataWindow(Brigade);

dataWindow.Show();

}

else

{

MessageBox.Show("Вы не зарегистрировались как диспетчер");

}

}

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

{

if (\_dispatcher != null)

{

var searchCallInformation = new SearchInformationAboutCall();

searchCallInformation.Show();

}

else

{

MessageBox.Show("Вы не зарегистрировались как диспетчер");

}

}

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

{

if (\_dispatcher != null)

{

try

{

var street = ComboStreets.Text;

var houseNumber = textHouseNumber.Text;

var flatNumber = textFlatNumber.Text;

var FIO = textFIO.Text;

var age = double.Parse(textAge.Text);

var reason = ComboReason.Text;

var whoCalled = ComboCaller.Text;

var callType = ComboUrgency.Text;

var phoneNumber = textTelephoneNumber.Text;

var callNotes = textBoxNote.Text + "\n" + phoneNumber;

var patient = new Patient { FIO = FIO, Age = age, Street = street, HouseNumber = houseNumber, FlatNumber = flatNumber };

var ambulanceCall = new AmbulanceCall(patient, DateTime.Now, reason, callNotes, whoCalled, callType)

{

ReceivingDispatcher = \_dispatcher,

DateTimeReception = DateTime.Now

};

ambulanceCalls.Add(ambulanceCall);

MessageBox.Show("OK");

}

catch (Exception ex)

{

MessageBox.Show(ex.Message);

}

}

else

{

MessageBox.Show("Вы не зарегистрировались как диспетчер");

}

}

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

{

var workerWindow = new WorkerWindow

{

Owner = this

};

workerWindow.ShowDialog();

}

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

{

var addDiagnosisWindow = new AddDiagnosis(true)

{

Owner = this

};

addDiagnosisWindow.ShowDialog();

}

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

{

var addDiagnosisWindow = new AddDiagnosis(false)

{

Owner = this

};

addDiagnosisWindow.ShowDialog();

}

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

{

if (\_dispatcher != null)

{

var inputCallWindow = new InputCall(WindowTypes.EnterCall, \_dispatcher)

{

Owner = this

};

inputCallWindow.Show();

}

else

{

MessageBox.Show("Вы не зарегистрировались как диспетчер");

}

}

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

{

if (\_dispatcher != null)

{

var reprocessingCallsWindow = new ReprocessingCallsWindow

{

Owner = this

};

reprocessingCallsWindow.ShowDialog();

}

else

{

MessageBox.Show("Вы не зарегистрировались как диспетчер");

}

}

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

{

var reportWindow = new ReportWindow()

{

Owner = this

};

reportWindow.Show();

}

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

{

if (\_dispatcher != null)

{

var chronicPatientWindow = new ChronicPatientWindow();

chronicPatientWindow.Owner = this;

chronicPatientWindow.ShowDialog();

}

else

{

MessageBox.Show("Вы не зарегистрировались как диспетчер");

}

}

public static void AddAmbulanceCall(AmbulanceCall ambulanceCall)

{

ambulanceCalls.Add(ambulanceCall);

}

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

{

if (\_dispatcher != null)

{

if (MessageBox.Show("Действительно выйти?", "Диспетчер", MessageBoxButton.YesNo) == MessageBoxResult.Yes)

{

\_dispatcher = null;

this.Title = "Скорая помощь";

}

}

else

{

MessageBox.Show("Диспетчер не введен!");

}

}

}

}

**AddRemoveWindow.xaml**

using System.Windows;

using Ambulance.Classes;

namespace Ambulance

{

/// <summary>

/// Interaction logic for AddDiagnosis.xaml

/// </summary>

public partial class AddDiagnosis : Window

{

private bool \_isDiagnosis;

public AddDiagnosis(bool isDiagnsis)

{

InitializeComponent();

\_isDiagnosis = isDiagnsis;

if (isDiagnsis)

{

Lable.Content = "Добавление/Удаление диагноза";

ListDiagnoses.ItemsSource = ServerService.GetDiagnoses();

}

else

{

Lable.Content = "Добавление/Удаление улицы";

ListDiagnoses.ItemsSource = ServerService.GetStreets();

}

}

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

{

var text = text1.Text;

if (!string.IsNullOrEmpty(text))

{

if (\_isDiagnosis)

{

ServerService.AddDiagnosis(text);

ListDiagnoses.ItemsSource = null;

ListDiagnoses.ItemsSource = ServerService.GetDiagnoses();

}

else

{

ServerService.AddStreet(text);

ListDiagnoses.ItemsSource = null;

ListDiagnoses.ItemsSource = ServerService.GetStreets();

}

text1.Text = "";

}

}

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

{

try

{

if (\_isDiagnosis)

{

var text = ServerService.GetDiagnoses()[ListDiagnoses.SelectedIndex];

if (!string.IsNullOrEmpty(text))

{

ServerService.RemoveDiagnosis(text);

ListDiagnoses.ItemsSource = null;

ListDiagnoses.ItemsSource = ServerService.GetDiagnoses();

}

}

else

{

var text = ServerService.GetStreets()[ListDiagnoses.SelectedIndex];

if (!string.IsNullOrEmpty(text))

{

ServerService.RemoveStreet(text);

ListDiagnoses.ItemsSource = null;

ListDiagnoses.ItemsSource = ServerService.GetStreets();

}

}

}

catch

{

}

}

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

{

this.Close();

}

}

}

**CallInformation.xaml**

using System.Windows;

using Ambulance.Classes;

namespace Ambulance

{

/// <summary>

/// Interaction logic for CallInformation.xaml

/// </summary>

public partial class CallInformation : Window

{

public CallInformation(AmbulanceCall call)

{

InitializeComponent();

lableCallNumber.Content = call.CallNumber;

lableCallReceivingDate.Content = call.DateTimeReception.ToString("dd.MM.yyyy");

lableCallReceivingTime.Content = call.DateTimeReception.ToString("hh:mm");

lableFIOInformation.Content = call.Patient.FIO;

lableAgeInformation.Content = call.Patient.Age;

lableStreetInformation.Content = call.Patient.Street;

lableHouseNumberInformation.Content = call.Patient.HouseNumber;

lableFlatNumberInformation.Content = call.Patient.FlatNumber;

lableDiagnosis.Content = call.Diagnosis;

lableWhoCalled.Content = call.WhoCalled;

lableUrgency.Content = call.Type;

lableDispatcherAccepted.Content = call.ReceivingDispatcher;

lableDispatcherPassed.Content = call.TransferringDispatcher;

textboxCallInformation.Text = call.CallNotes;

}

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

{

this.Close();

}

}

}

**CallMonitoring.xaml**

using System;

using System.Windows;

using System.Windows.Input;

using System.Linq;

using System.Collections.ObjectModel;

using Ambulance.Classes;

namespace Ambulance

{

/// <summary>

/// Interaction logic for CallMonitoring.xaml

/// </summary>

public partial class CallMonitoring : Window

{

private ObservableCollection<AmbulanceCall> \_calls { get; }

private ObservableCollection<uint> \_listOfBrigadeNumbers { get; set; } = new ObservableCollection<uint>();

private string \_dispatcher;

public CallMonitoring(ObservableCollection<AmbulanceCall> ambulanceCalls, string dispatcher)

{

InitializeComponent();

\_calls = ambulanceCalls;

\_dispatcher = dispatcher;

callDataGrid.ItemsSource = \_calls;

carsDataGrid.ItemsSource = MainWindow.Brigade;

FillListOfBrigadeNumbers();

comboBrigadeNumbers.ItemsSource = \_listOfBrigadeNumbers;

comboDirection.ItemsSource = ServerService.GetDirections();

}

private void FillListOfBrigadeNumbers()

{

foreach (var item in MainWindow.Brigade)

{

\_listOfBrigadeNumbers.Add(item.Number);

}

}

private void callData\_MouseLeftButtonUp(object sender, MouseButtonEventArgs e)

{

if (this.callDataGrid.SelectedItem != null)

{

textCallMonitoringNumber.Text = \_calls[this.callDataGrid.SelectedIndex].CallNumber.ToString();

textCallMonitoringFIO.Text = \_calls[this.callDataGrid.SelectedIndex].Patient.FIO;

textCallMonitoringStreet.Text = \_calls[this.callDataGrid.SelectedIndex].Patient.Street;

textCallMonitoringHouse.Text = \_calls[this.callDataGrid.SelectedIndex].Patient.HouseNumber.ToString();

textCallMonitoringFlat.Text = \_calls[this.callDataGrid.SelectedIndex].Patient.FlatNumber.ToString();

textCallMonitoringAge.Text = \_calls[this.callDataGrid.SelectedIndex].Patient.Age.ToString();

textCallMonitoringReason.Text = \_calls[this.callDataGrid.SelectedIndex].Diagnosis;

textCallMonitoringWho.Text = \_calls[this.callDataGrid.SelectedIndex].WhoCalled;

textCallMonitoringReceptionTime.Text = \_calls[this.callDataGrid.SelectedIndex].DateTimeReception.ToString("hh:mm");

textCallMonitoringTransmissionTime.Text = \_calls[this.callDataGrid.SelectedIndex].TransferDateTime.ToString("hh:mm");

textCallMonitoringInformation.Text = \_calls[this.callDataGrid.SelectedIndex].CallNotes;

}

}

public void CarRow\_DoubleClick(object sender, MouseButtonEventArgs e)

{

try

{

var indexOfselectedCar = carsDataGrid.SelectedIndex;

var brigade = MainWindow.Brigade[indexOfselectedCar];

if (brigade.IsOnCall)

{

var call = brigade.AmbulanceCalls.Last();

var callInformation = new CallInformation(call);

callInformation.Owner = this;

callInformation.Show();

}

}

catch

{

MessageBox.Show("");

}

}

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

{

if (callDataGrid.SelectedIndex >= 0)

{

try

{

var brigadeNumber = int.Parse(comboBrigadeNumbers.Text);

var brigade = MainWindow.Brigade.First(e => e.Number == brigadeNumber);

if (!brigade.IsOnCall)

{

var call = \_calls[this.callDataGrid.SelectedIndex];

call.TransferDateTime = DateTime.Now;

call.TransferringDispatcher = \_dispatcher.ToString();

brigade.SendCall(call);

brigade.IsOnCall = true;

\_calls.Remove(call);

UpdateCarsGrid();

}

else

{

MessageBox.Show("Бригаде уже передан вызов.", "Ошибка передачи вызова");

}

}

catch

{

MessageBox.Show("Не верный номер бригады");

}

}

}

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

{

try

{

var brigade = MainWindow.Brigade[carsDataGrid.SelectedIndex];

if (brigade.IsOnCall)

{

var returnedCall = brigade.AmbulanceCalls.LastOrDefault();

if (returnedCall != null)

{

\_calls.Add(returnedCall);

brigade.AmbulanceCalls.Remove(returnedCall);

brigade.IsOnCall = false;

brigade.AtTheStation();

UpdateCarsGrid();

}

}

}

catch (IndexOutOfRangeException)

{

}

}

private void UpdateCarsGrid()

{

carsDataGrid.ItemsSource = null;

carsDataGrid.ItemsSource = MainWindow.Brigade;

}

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

{

var carIndex = carsDataGrid.SelectedIndex;

if (carIndex >= 0)

{

var brigade = MainWindow.Brigade[carIndex];

if (brigade.IsOnCall)

{

brigade.AtTheStation();

brigade.IsOnCall = false;

brigade.AmbulanceCalls.Last().Department = brigade.StationName;

var ambulanceCalls = brigade.AmbulanceCalls.Last();

var processedCall = new ProcessedCall

{

CallNumber = ambulanceCalls.CallNumber, PatientId = ambulanceCalls.Patient.Id,

Patient = ambulanceCalls.Patient, AmbulanceBrigade = brigade, DateTimeReception = ambulanceCalls.DateTimeReception,

TransferDateTime = ambulanceCalls.TransferDateTime, CallType = ambulanceCalls.Type, TransferringDispatcher = ambulanceCalls.TransferringDispatcher,

CallNotes = ambulanceCalls.CallNotes

};

MainWindow.AmbulanceCallsForProcessing.Add(processedCall);

UpdateCarsGrid();

}

}

}

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

{

var carIndex = carsDataGrid.SelectedIndex;

if (carIndex >= 0)

{

var direction = comboDirection.Text;

if (!string.IsNullOrEmpty(direction))

{

MainWindow.Brigade[carIndex].StreetOfLastCall = direction;

MainWindow.Brigade[carIndex].ActionTime = DateTime.Now.ToString("hh:mm");

MainWindow.Brigade[carIndex].IsOnCall = true;

UpdateCarsGrid();

}

}

}

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

{

if (carsDataGrid.SelectedIndex >= 0)

{

var brigade = MainWindow.Brigade[carsDataGrid.SelectedIndex];

if (brigade.IsOnCall)

{

brigade.AtTheStation();

brigade.IsOnCall = false;

UpdateCarsGrid();

}

}

}

private void CarsDataGrid\_MouseRightButtonUp(object sender, MouseButtonEventArgs e)

{

var carIndex = carsDataGrid.SelectedIndex;

if (carIndex >= 0)

{

var carInformationWindow = new CarsInfrmationWindow(MainWindow.Brigade[carIndex]);

carInformationWindow.Owner = this;

carInformationWindow.ShowDialog();

}

}

}

}

**CarsInfrmationWindow.xaml**

using System.Windows;

using Ambulance.Classes;

namespace Ambulance

{

/// <summary>

/// Interaction logic for CarsInfrmationWindow.xaml

/// </summary>

public partial class CarsInfrmationWindow : Window

{

public CarsInfrmationWindow(AmbulanceBrigade brigade)

{

InitializeComponent();

gridCalls.ItemsSource = brigade.AmbulanceCalls;

lableDoktor.Content = brigade.Doktor.ToString();

lableMedAssistant1.Content = brigade.MedicalAssistants1.ToString();

lableMedAssistant2.Content = brigade.MedicalAssistants2.ToString();

lableOrderly.Content = brigade.Orderly.ToString();

lableDriver.Content = brigade.Driver.ToString();

lableStationName.Content = brigade.StationName;

lableType.Content = brigade.BrigadeType;

lableNumberOfCalls.Content = brigade.AmbulanceCalls.Count;

}

}

}

**ChronicPatientWindow.xaml**

using System;

using System.Windows;

using System.Linq;

using Ambulance.DBAccess;

using Ambulance.Classes;

namespace Ambulance

{

/// <summary>

/// Interaction logic for ChronicPatientWindow.xaml

/// </summary>

public partial class ChronicPatientWindow : Window

{

private AmbulanceDBContext \_context = new AmbulanceDBContext();

public ChronicPatientWindow()

{

InitializeComponent();

comboStreet.ItemsSource = ServerService.GetStreets();

comboDiagnosis.ItemsSource = ServerService.GetDiagnoses();

gridChrnicPatient.ItemsSource = \_context.ChronicPatients.ToList();

}

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

{

var fio = textFIO.Text;

var diagnosis = comboDiagnosis.Text;

var street = comboStreet.Text;

var houseNumber = textHouseNumber.Text;

var flatNumber = textFlatNumber.Text;

try

{

var listOfChronicPatient = \_context.ChronicPatients.ToList();

if (!string.IsNullOrEmpty(fio) && listOfChronicPatient.Any())

{

listOfChronicPatient = listOfChronicPatient.Where(

e => e.FIO.ToUpper().Contains(fio.ToUpper())

).ToList();

}

if (!string.IsNullOrEmpty(diagnosis) && listOfChronicPatient.Any())

{

listOfChronicPatient = listOfChronicPatient.Where(

e => e.Diagnosis.ToUpper().Contains(diagnosis.ToUpper())

).ToList();

}

if (!string.IsNullOrEmpty(street) && listOfChronicPatient.Any())

{

listOfChronicPatient = listOfChronicPatient.Where(

e => e.Street.ToUpper().Contains(street.ToUpper())

).ToList();

}

if (!string.IsNullOrEmpty(houseNumber) && listOfChronicPatient.Any())

{

listOfChronicPatient = listOfChronicPatient.Where(

e => e.HouseNumber.ToUpper().Contains(houseNumber.ToUpper())

).ToList();

}

if (!string.IsNullOrEmpty(flatNumber) && listOfChronicPatient.Any())

{

listOfChronicPatient = listOfChronicPatient.Where(

e => e.FlatNumber.ToUpper().Contains(flatNumber.ToUpper())

).ToList();

}

gridChrnicPatient.ItemsSource = listOfChronicPatient;

}

catch(ArgumentException)

{

MessageBox.Show("Данные введены не верно");

}

}

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

{

try

{

var selectedPatient = (ChronicPatient)gridChrnicPatient.SelectedItem;

if (selectedPatient != null)

{

var patient = new Patient

{

FIO = selectedPatient.FIO,

Age = selectedPatient.Age,

Street = selectedPatient.Street,

HouseNumber = selectedPatient.HouseNumber,

FlatNumber = selectedPatient.FlatNumber,

Gender = selectedPatient.Gender

};

var ambulanceCall = new AmbulanceCall(patient, DateTime.Now, selectedPatient.Diagnosis, selectedPatient.Notes,

selectedPatient.WhoCalled, selectedPatient.Urgency);

MainWindow.AddAmbulanceCall(ambulanceCall);

}

}

catch

{

}

}

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

{

\_context.DisposeAsync();

this.Close();

}

}

}

**DataWindow.xaml**

using System;

using System.Collections.ObjectModel;

using System.Windows;

using System.Windows.Input;

using Ambulance.Classes;

using System.Linq;

using Ambulance.DBAccess;

using System.Text.RegularExpressions;

namespace Ambulance

{

/// <summary>

/// Interaction logic for DataWindow.xaml

/// </summary>

public partial class DataWindow : Window

{

private ObservableCollection<AmbulanceBrigade> \_brigades;

private static AmbulanceDBContext \_context;

public DataWindow(ObservableCollection<AmbulanceBrigade> brigades)

{

InitializeComponent();

\_context = new AmbulanceDBContext();

\_brigades = brigades;

dateAutoBrigade.SelectedDate = DateTime.Now;

comboBrigadeType.ItemsSource = ServerService.GetBrigadeTypes();

comboDoktor.ItemsSource = \_context.Doktors.ToList();

comboMedcalAssistant1.ItemsSource = \_context.MedicalAssistants.ToList();

comboMedcalAssistant2.ItemsSource = comboMedcalAssistant1.ItemsSource;

comboOrderly.ItemsSource = \_context.Orderlies.ToList();

comboDriver.ItemsSource = \_context.Drivers.ToList();

comboDepartment.ItemsSource = ServerService.GetDepartments();

UpdateGridChronicPatients();

comboStreet.ItemsSource = ServerService.GetStreets();

comboReason.ItemsSource = ServerService.GetDiagnoses();

comboWho.ItemsSource = ServerService.GetCallers();

}

private void NumberValidationTextBox(object sender, TextCompositionEventArgs e)

{

Regex regex = new Regex("[^0-9]+");

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

}

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

{

var brigadeNumber = uint.Parse(textBrigadeNumber.Text);

if (!\_brigades.Where(e => e.Number == brigadeNumber).Any())

{

var medicalAssistans = new string[2];

var brgadeType = comboBrigadeType.Text;

var doktor = comboDoktor.Text;

medicalAssistans[0] = comboMedcalAssistant1.Text;

medicalAssistans[1] = comboMedcalAssistant2.Text;

var orderly = comboOrderly.Text;

var driver = comboDriver.Text;

var department = comboDepartment.Text;

var brigade = new AmbulanceBrigade

{

Number = brigadeNumber,

Doktor = doktor,

MedicalAssistants1 = medicalAssistans[0],

MedicalAssistants2 = medicalAssistans[1],

Orderly = orderly,

Driver = driver,

BrigadeType = brgadeType,

StationName = department,

ActionTime = DateTime.Now.ToString("hh:mm"),

ActionDate = DateTime.Now.ToString("dd.MM.yyyy")

};

\_brigades.Add(brigade);

ClearFields();

}

else

{

MessageBox.Show("Бригада с таким номером уже принята к работе", "Ошибка");

}

}

private void ClearFields()

{

textBrigadeNumber.Text = "";

comboBrigadeType.Text = "";

comboDoktor.Text = "";

comboMedcalAssistant1.Text = "";

comboMedcalAssistant2.Text = "";

comboOrderly.Text = "";

comboDriver.Text = "";

comboDepartment.Text = "";

}

private void RemoveBrigade\_Cliick(object sender, RoutedEventArgs e)

{

try

{

if (MessageBox.Show("Подтвердите ваш выбор", "Удаление бригады", MessageBoxButton.YesNo) == MessageBoxResult.Yes)

{

var brigadeNumber = textBrigadeNumber.Text;

if (string.IsNullOrEmpty(brigadeNumber))

{

throw new ArgumentException();

}

var brigadeForRemove = \_brigades.First(e => e.Number == uint.Parse(brigadeNumber));

\_brigades.Remove(brigadeForRemove);

ClearFields();

}

}

catch

{

MessageBox.Show("Не верно введен номер бригады", "Ошибка");

}

}

private void TextBrigadeNumber\_LostFocus(object sender, RoutedEventArgs e)

{

var brigadeNumber = textBrigadeNumber.Text;

if (!string.IsNullOrEmpty(brigadeNumber))

{

var selectedBrigade = \_brigades.FirstOrDefault(e => e.Number == uint.Parse(brigadeNumber));

if (selectedBrigade != null)

{

comboBrigadeType.Text = selectedBrigade.BrigadeType;

comboDoktor.Text = selectedBrigade.Doktor;

comboMedcalAssistant1.Text = selectedBrigade.MedicalAssistants1.ToString();

comboMedcalAssistant2.Text = selectedBrigade.MedicalAssistants2.ToString();

comboOrderly.Text = selectedBrigade.Orderly.ToString();

comboDriver.Text = selectedBrigade.Driver.ToString();

comboDepartment.Text = selectedBrigade.StationName;

}

}

}

private void Window\_Closed(object sender, EventArgs e)

{

\_context.DisposeAsync();

}

private void UpdateGridChronicPatients()

{

var chronicPatients = \_context.ChronicPatients.ToList();

gridChronicPatients.ItemsSource = null;

gridChronicPatients.ItemsSource = chronicPatients;

}

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

{

try

{

var fio = textFio.Text;

var street = comboStreet.Text;

var houseNumber = textHouse.Text;

var flatNumber = textFlat.Text;

var age = double.Parse(textAge.Text);

var reason = comboReason.Text;

var gender = comboGender.Text;

var urgency = comboUrgency.Text;

var whoCalled = comboWho.Text;

var notes = textNotes.Text;

var chronicPatient = new ChronicPatient

{

FIO = fio,

Street = street,

HouseNumber = houseNumber,

FlatNumber = flatNumber,

Age = age,

Diagnosis = reason,

Gender = gender,

Urgency = urgency,

WhoCalled = whoCalled,

Notes = notes

};

\_context.ChronicPatients.AddAsync(chronicPatient);

\_context.SaveChanges();

UpdateGridChronicPatients();

}

catch

{

}

}

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

{

textFio.Text = "";

comboStreet.Text = "";

textHouse.Text = "";

textFlat.Text = "";

textAge.Text = "";

comboReason.Text = "";

comboGender.Text = "";

comboWho.Text = "";

textNotes.Text = "";

}

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

{

var index = gridChronicPatients.SelectedIndex;

if (index >= 0)

{

var removedPatient = \_context.ChronicPatients.Skip(index).First();

\_context.ChronicPatients.Remove(removedPatient);

\_context.SaveChanges();

UpdateGridChronicPatients();

}

}

}

}

**DispatcherRegistrationWindow.xaml**

using System;

using System.Windows;

using System.Linq;

using Ambulance.DBAccess;

namespace Ambulance

{

/// <summary>

/// Interaction logic for DispatcherRegistrationWindow.xaml

/// </summary>

public partial class DispatcherRegistrationWindow : Window

{

public string AmbDispatcher;

private AmbulanceDBContext \_context;

public DispatcherRegistrationWindow()

{

InitializeComponent();

\_context = new AmbulanceDBContext();

}

public void Button\_Click(object sender, RoutedEventArgs e)

{

try

{

if (AmbDispatcher != null)

{

this.Owner.Title = "Скорая помощь " + $"[{AmbDispatcher}]";

this.Close();

}

else

{

throw new ArgumentException("Не верно введен диспетчер");

}

}

catch (ArgumentException argException)

{

MessageBox.Show(argException.Message);

}

}

private void DispatcherNumber\_LostFocus(object sender, RoutedEventArgs e)

{

try

{

var dispatcherNumber = int.Parse(textDispatcherNumber.Text);

var dispatcher = \_context.Dispatchers.Where(e => e.Id == dispatcherNumber).FirstOrDefault();

if (dispatcher != null)

{

textDispatcherFIO.Text = dispatcher.ToString();

AmbDispatcher = dispatcher.ToString();

}

}

catch (ArgumentException argException)

{

MessageBox.Show(argException.Message);

}

catch (FormatException formatException)

{

;

}

}

}

}

**EditingCalls.xaml**

using Ambulance.Classes;

using System;

using System.Windows;

using System.Windows.Input;

using System.Linq;

using System.Text.RegularExpressions;

using Ambulance.DBAccess;

using System.Windows.Controls;

namespace Ambulance

{

/// <summary>

/// Interaction logic for EditingCalls.xaml

/// </summary>

public partial class EditingCalls : Window

{

private readonly string \_dispatcherFIO;

private int \_selectedCall = -1;

private AmbulanceDBContext \_context = new AmbulanceDBContext();

public EditingCalls(string dispatcherFIO)

{

InitializeComponent();

\_dispatcherFIO = dispatcherFIO;

gridEditCalls.ItemsSource = MainWindow.AmbulanceCallsForProcessing;

comboStreet.ItemsSource = ServerService.GetStreets();

comboDiagnosis.ItemsSource = ServerService.GetDiagnoses();

comboPlace.ItemsSource = ServerService.GetPlaces();

comboRezault.ItemsSource = ServerService.GetResults();

UpdateNumberOfCalls();

}

private void UpdateNumberOfCalls()

{

lableCountOfCalls.Content = MainWindow.AmbulanceCallsForProcessing.Count;

}

private void NumberValidationTextBox(object sender, TextCompositionEventArgs e)

{

Regex regex = new Regex("[^0-9]+");

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

}

private void SetCall(ProcessedCall processedCall)

{

textCallNumber.Text = processedCall.CallNumber.ToString();

textDateComeback.Text = processedCall.AmbulanceBrigade.ActionDate;

textTimeComeback.Text = processedCall.AmbulanceBrigade.ActionTime;

textDateTransfer.Text = processedCall.TransferDateTime.ToString("dd.MM.yyyy");

textTimeTransfer.Text = processedCall.TransferDateTime.ToString("hh:mm");

textBrigadeNumber.Text = processedCall.AmbulanceBrigade.Number.ToString();

comboDiagnosis.Text = processedCall.Diagnosis;

textDoktor.Text = processedCall.AmbulanceBrigade.Doktor;

textMedAsstant1.Text = processedCall.AmbulanceBrigade.MedicalAssistants1;

textMedAsstant2.Text = processedCall.AmbulanceBrigade.MedicalAssistants2;

textOrderly.Text = processedCall.AmbulanceBrigade.Orderly;

textDriver.Text = processedCall.AmbulanceBrigade.Driver;

comboStreet.Text = processedCall.Patient.Street;

textHouseNumber.Text = processedCall.Patient.HouseNumber.ToString();

textFlatNumber.Text = processedCall.Patient.FlatNumber.ToString();

textFIO.Text = processedCall.Patient.FIO;

textAge.Text = processedCall.Patient.Age.ToString();

textNotes.Text = processedCall.CallNotes;

}

private void TextCallNumber\_LostFocus(object sender, RoutedEventArgs e)

{

try

{

var callNumber = double.Parse(textCallNumber.Text);

var call = MainWindow.AmbulanceCallsForProcessing.Where(e => e.CallNumber == callNumber).FirstOrDefault();

if (call != null)

{

\_selectedCall = MainWindow.AmbulanceCallsForProcessing.IndexOf(call);

SetCall(call);

}

}

catch

{

}

}

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

{

if (checkGender.IsChecked == true)

{

checkMale.IsEnabled = true;

chekFamale.IsEnabled = true;

}

else

{

checkMale.IsEnabled = false;

chekFamale.IsEnabled = false;

}

}

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

{

if (checkMale.IsChecked == true)

{

chekFamale.IsChecked = false;

}

}

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

{

if (chekFamale.IsChecked == true)

{

checkMale.IsChecked = false;

}

}

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

{

textDateComeback.Text = "\_\_.\_\_.\_\_";

textTimeComeback.Text = "\_\_:\_\_";

textDateTransfer.Text = "\_\_.\_\_.\_\_";

textTimeTransfer.Text = "\_\_:\_\_";

textDateArival.Text = "\_\_.\_\_.\_\_";

textTimeArival.Text = "\_\_:\_\_";

textDateDeparture.Text = "\_\_.\_\_.\_\_";

textTimeDeparture.Text = "\_\_:\_\_";

comboDiagnosis.Text = "";

comboRezault.Text = "";

textCallNumber.Text = "";

textBrigadeNumber.Text = "";

textKMBefore.Text = "";

textKMAfter.Text = "";

textDoktor.Text = "-";

checkGender.IsChecked = false;

checkMale.IsEnabled = false;

chekFamale.IsEnabled = false;

comboPlace.Text = "";

textMedAsstant1.Text = "-";

textMedAsstant2.Text = "-";

textOrderly.Text = "-";

textDriver.Text = "-";

comboStreet.Text = "";

textHouseNumber.Text = "";

textFlatNumber.Text = "";

textFIO.Text = "";

textAge.Text = "";

textNotes.Text = "";

textTreatment.Text = "";

\_selectedCall = -1;

}

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

{

try

{

var callNumber = double.Parse(textCallNumber.Text);

var dateOfComeback = textDateComeback.Text;

var TimeOfComeback = textTimeComeback.Text;

var dateOfTransfer = textDateTransfer.Text;

var timeOfTransfer = textTimeTransfer.Text;

var dateOfArival = textDateArival.Text;

var timeOfArival = textTimeArival.Text;

var dateOfDeparture = textDateDeparture.Text;

var timeOfDeparture = textTimeDeparture.Text;

var diagnosis = comboDiagnosis.Text;

var result = comboRezault.Text;

var brigadeNumber = textBrigadeNumber.Text;

int kmBefore = 0;

int kmAfter = 0;

if (!string.IsNullOrEmpty(textKMBefore.Text) && !string.IsNullOrEmpty(textKMAfter.Text))

{

try

{

kmBefore = int.Parse(textKMBefore.Text);

kmAfter = int.Parse(textKMAfter.Text);

}

catch

{

throw new ArgumentException("Не верно введен километраж");

}

}

else

{

MessageBox.Show("Не введен километраж бригады");

}

string gender = "";

if (checkGender.IsChecked == true)

{

if (checkMale.IsChecked == true)

{

gender = "МУЖСКОЙ";

}

else

{

if (chekFamale.IsChecked == true)

{

gender = "ЖЕНСКИЙ";

}

else

{

throw new ArgumentException("Не выбран пол пациента");

}

}

}

else

{

throw new ArgumentException("Не выбран пол пациента");

}

var place = comboPlace.Text;

var doktor = textDoktor.Text;

var medAsstant1 = textMedAsstant1.Text;

var medAsstant2 = textMedAsstant2.Text;

var orderly = textOrderly.Text;

var driver = textDriver.Text;

var street = comboStreet.Text;

var houseNumber = textHouseNumber.Text;

var flatNumber = textFlatNumber.Text;

var fio = textFIO.Text;

double age = 0;

if (!string.IsNullOrEmpty(textAge.Text))

{

try

{

age = double.Parse(textAge.Text);

}

catch

{

throw new ArgumentException("Возраст введен не верно");

}

}

var notes = textNotes.Text;

var processedCall = MainWindow.AmbulanceCallsForProcessing.Where(e => e.CallNumber == callNumber).FirstOrDefault();

if (processedCall != null)

{

var samePatient = \_context.Patients.Where(

e => e.FIO == fio && e.Age == age && e.Street == street && e.HouseNumber == houseNumber && e.FlatNumber == flatNumber && e.Gender == gender

).ToList();

if (samePatient.Any())

{

processedCall.Patient = samePatient.First();

}

else

{

processedCall.Patient = new Patient { FIO = fio, Age = age, Street = street, HouseNumber = houseNumber, FlatNumber = flatNumber, Gender = gender };

}

processedCall.Diagnosis = diagnosis;

processedCall.AmbulanceBrigade.Doktor = doktor;

processedCall.AmbulanceBrigade.MedicalAssistants1 = medAsstant1;

processedCall.AmbulanceBrigade.MedicalAssistants2 = medAsstant2;

processedCall.AmbulanceBrigade.Orderly = orderly;

processedCall.AmbulanceBrigade.Driver = driver;

processedCall.Results = result;

processedCall.TransferDateTime = DateTime.Parse(dateOfTransfer + " " + timeOfTransfer);

processedCall.ArrivalDateTime = DateTime.Parse(dateOfArival + " " + timeOfArival);

processedCall.DepartureDateTime = DateTime.Parse(dateOfDeparture + " " + timeOfDeparture);

processedCall.ComeBackDateTime = DateTime.Parse(dateOfComeback + " " + TimeOfComeback);

processedCall.ProcessingDispatcher = \_dispatcherFIO;

processedCall.KilometrageBefor = kmBefore;

processedCall.KilometrageAfter = kmAfter;

processedCall.Place = place;

processedCall.CallNotes = notes;

processedCall.Treatment = textTreatment.Text;

}

ClearButton\_Click(null,null);

MessageBox.Show("Вызов обработан");

\_context.ProcessedCalls.AddAsync(processedCall);

\_context.SaveChangesAsync();

MainWindow.AmbulanceCallsForProcessing.RemoveAt(\_selectedCall);

UpdateNumberOfCalls();

}

catch (ArgumentException argException)

{

MessageBox.Show(argException.Message);

}

}

private void TreatmentTextBox\_GotFocus(object sender, RoutedEventArgs e)

{

var treatmentWindow = new TreatmentWindow();

treatmentWindow.Owner = this;

treatmentWindow.ShowDialog();

textTreatment.Text = treatmentWindow.textTreatment.Text;

}

private void TextBox\_GotFocus(object sender, RoutedEventArgs e)

{

((TextBox)sender).Text = "";

}

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

{

try

{

if (!string.IsNullOrEmpty(textCallNumber.Text))

{

var InputCallWindow = new InputCall(WindowTypes.EnterIncidentalCall, \_dispatcherFIO, int.Parse(textCallNumber.Text));

InputCallWindow.Owner = this;

InputCallWindow.ShowDialog();

}

else

{

MessageBox.Show("Номер вызова не введен.");

}

}

catch

{

}

}

private void gridEditCalls\_MouseDoubleClick(object sender, MouseButtonEventArgs e)

{

var indexOfSelectedSell = gridEditCalls.SelectedIndex;

if (indexOfSelectedSell >= 0)

{

SetCall(MainWindow.AmbulanceCallsForProcessing[indexOfSelectedSell]);

\_selectedCall = gridEditCalls.SelectedIndex;

}

}

}

}

**FullCallInformationWindow.xaml**

using System.Windows;

using Ambulance.Classes;

namespace Ambulance

{

/// <summary>

/// Interaction logic for FullCallInformationWindow.xaml

/// </summary>

public partial class FullCallInformationWindow : Window

{

public FullCallInformationWindow(Call call)

{

InitializeComponent();

lableFIO.Content = call.Patient.FIO.ToUpper();

lableAge.Content = call.Patient.Age;

lableStreet.Content = call.Patient.Street;

lableHouseNumber.Content = call.Patient.HouseNumber;

lableFlatNumber.Content = call.Patient.FlatNumber;

lableGender.Content = call.Patient.Gender;

lableDiagnosis.Content = call.Diagnosis;

lableResult.Content = call.Results;

textTreatment.Text = call.Treatment;

textNotes.Text = call.CallNotes;

lableReception.Content = call.DateTimeReception.GetDateTimeFormats('F')[1];

lableTransfer.Content = call.TransferDateTime.GetDateTimeFormats('F')[1];

lableArival.Content = call.ArrivalDateTime.GetDateTimeFormats('F')[1];

lableDeparture.Content = call.DepartureDateTime.GetDateTimeFormats('F')[1];

lableComeback.Content = call.ComeBackDateTime.GetDateTimeFormats('F')[1];

var time = (call.TransferDateTime - call.DateTimeReception);

lableTransferingTime.Content = time.ToString(@"hh\:mm\:ss");

time = (call.ArrivalDateTime - call.TransferDateTime);

lableDrivingTime.Content = time.ToString(@"hh\:mm\:ss");

time = (call.DepartureDateTime - call.ArrivalDateTime);

lableServiceTime.Content = time.ToString(@"hh\:mm\:ss");

lablUrgency.Content = call.CallType;

lableDispatcherTransfered.Content = call.TransferringDispatcher;

lableDispatcherProcessed.Content = call.ProcessingDispatcher;

lableBrigadeNumber.Content = call.AmbulanceBrigade.Number;

lableBrigadeType.Content = call.AmbulanceBrigade.BrigadeType;

lableDoktor.Content = call.AmbulanceBrigade.Doktor;

lableMedAssistant1.Content = call.AmbulanceBrigade.MedicalAssistants1;

lableMedAssistant2.Content = call.AmbulanceBrigade.MedicalAssistants2;

lableOrderly.Content = call.AmbulanceBrigade.Orderly;

lableDriver.Content = call.AmbulanceBrigade.Driver;

lableKMBefore.Content = call.KilometrageBefor;

lableKMAfter.Content = call.KilometrageAfter;

}

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

{

this.Close();

}

}

}

**InputCall.xaml**

using System;

using System.Text.RegularExpressions;

using System.Windows;

using System.Windows.Controls;

using System.Windows.Input;

using System.Linq;

using Ambulance.Classes;

using Ambulance.DBAccess;

namespace Ambulance

{

public enum WindowTypes

{

EnterCall = 1,

EnterIncidentalCall

}

/// <summary>

/// Interaction logic for InputCall.xaml

/// </summary>

public partial class InputCall : Window

{

private string \_dispatcherFIO;

private AmbulanceDBContext \_context = new AmbulanceDBContext();

private WindowTypes \_typeOfWindow;

public InputCall(WindowTypes types, string dispatcher, int processedCallNumber = 0)

{

InitializeComponent();

\_dispatcherFIO = dispatcher;

\_typeOfWindow = types;

if (types != WindowTypes.EnterCall)

{

textReceivingDate.Visibility = Visibility.Hidden;

textReceivingTime.Visibility = Visibility.Hidden;

textIncidentalCallnumber.Text = processedCallNumber.ToString() + ",";

}

ComboStreets.ItemsSource = ServerService.GetStreets();

ComboDepartment.ItemsSource = ServerService.GetDepartments();

ComboReason.ItemsSource = ServerService.GetDiagnoses();

ComboCaller.ItemsSource = ServerService.GetCallers();

comboPlace.ItemsSource = ServerService.GetPlaces();

comboDiagnosis.ItemsSource = ServerService.GetDiagnoses();

comboResult.ItemsSource = ServerService.GetResults();

comboBrigadeType.ItemsSource = ServerService.GetBrigadeTypes();

comboDoktor.ItemsSource = \_context.Doktors.ToList();

comboMedAssistant1.ItemsSource = \_context.MedicalAssistants.ToList();

comboMedAssistant2.ItemsSource = \_context.MedicalAssistants.ToList();

comboOrderly.ItemsSource = \_context.Orderlies.ToList();

comboDriver.ItemsSource = \_context.Drivers.ToList();

}

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

{

this.Close();

}

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

{

try

{

var dateOfReceiv = textReceivingDate.Text;

var TimeOfReceiv = textReceivingTime.Text;

var dateOfComeback = textComebackDate.Text;

var TimeOfComeback = textComebackTime.Text;

var dateOfTransfer = textTransferDate.Text;

var timeOfTransfer = textTransferTime.Text;

var dateOfArival = textArrivalDate.Text;

var timeOfArival = textArrivalTime.Text;

var dateOfDeparture = textDepartureDate.Text;

var timeOfDeparture = textDepartureTime.Text;

var diagnosis = comboDiagnosis.Text;

var result = comboResult.Text;

uint brigadeNumber = 0;

if (!string.IsNullOrEmpty(textBrigadeNumber.Text))

{

try

{

brigadeNumber = uint.Parse(textBrigadeNumber.Text);

}

catch

{

throw new ArgumentException("Не верно введен номер бригады");

}

}

else

{

}

int kmBefore = 0;

int kmAfter = 0;

if (!string.IsNullOrEmpty(textKMBefore.Text) && !string.IsNullOrEmpty(textKMAfter.Text))

{

try

{

kmBefore = int.Parse(textKMBefore.Text);

kmAfter = int.Parse(textKMAfter.Text);

}

catch

{

throw new ArgumentException("Не верно введен километраж");

}

}

else

{

MessageBox.Show("Не введен километраж бригады");

}

string gender = "";

if (checkGender.IsChecked == true)

{

if (checkMale.IsChecked == true)

{

gender = "МУЖСКОЙ";

}

else

{

if (chekFamale.IsChecked == true)

{

gender = "ЖЕНСКИЙ";

}

else

{

throw new ArgumentException("Не выбран пол пациента");

}

}

}

else

{

throw new ArgumentException("Не выбран пол пациента");

}

var place = comboPlace.Text;

var brigadeType = comboBrigadeType.Text;

var doktor = comboDoktor.Text;

var medAsstant1 = comboMedAssistant1.Text;

var medAsstant2 = comboMedAssistant2.Text;

var orderly = comboOrderly.Text;

var driver = comboDriver.Text;

var street = ComboStreets.Text;

var houseNumber = textHouseNumber.Text;

var flatNumber = textFlatNumber.Text;

var station = ComboDepartment.Text;

var fio = textFIO.Text;

var notes = textBoxNote.Text;

double age = 0;

if (!string.IsNullOrEmpty(textAge.Text))

{

try

{

age = double.Parse(textAge.Text);

}

catch

{

throw new ArgumentException("Возраст введен не верно");

}

}

DateTime receptionDateTime;

DateTime transferDateTime;

DateTime arivalDateTime;

DateTime departureDateTime;

DateTime combebackDateTime;

try

{

receptionDateTime = DateTime.Parse(dateOfReceiv + " " + TimeOfReceiv);

transferDateTime = DateTime.Parse(dateOfTransfer + " " + timeOfTransfer);

arivalDateTime = DateTime.Parse(dateOfArival + " " + timeOfArival);

departureDateTime = DateTime.Parse(dateOfDeparture + " " + timeOfDeparture);

combebackDateTime = DateTime.Parse(dateOfComeback + " " + TimeOfComeback);

}

catch

{

throw new ArgumentException("Дата и время введены не верно");

}

if (\_typeOfWindow == WindowTypes.EnterCall)

{

double incidentalCallNumber;

if (!string.IsNullOrEmpty(textIncidentalCallnumber.Text))

{

incidentalCallNumber = double.Parse(textIncidentalCallnumber.Text);

}

else

throw new ArgumentException("Не введен номер вызова");

var processedCall = new ProcessedCall();

processedCall.Patient = new Patient();

processedCall.AmbulanceBrigade = new AmbulanceBrigade();

processedCall.Patient.FIO = fio;

processedCall.Patient.Age = age;

processedCall.Diagnosis = diagnosis;

processedCall.Patient.Street = street;

processedCall.Patient.HouseNumber = houseNumber;

processedCall.Patient.FlatNumber = flatNumber;

processedCall.Patient.Gender = gender;

processedCall.Results = result;

processedCall.AmbulanceBrigade.Number = brigadeNumber;

processedCall.AmbulanceBrigade.Doktor = doktor;

processedCall.AmbulanceBrigade.MedicalAssistants1 = medAsstant1;

processedCall.AmbulanceBrigade.MedicalAssistants2 = medAsstant2;

processedCall.AmbulanceBrigade.Orderly = orderly;

processedCall.AmbulanceBrigade.Driver = driver;

processedCall.AmbulanceBrigade.BrigadeType = brigadeType;

processedCall.AmbulanceBrigade.StationName = station;

processedCall.DateTimeReception = receptionDateTime;

processedCall.TransferDateTime = transferDateTime;

processedCall.ArrivalDateTime = arivalDateTime;

processedCall.DepartureDateTime = departureDateTime;

processedCall.ComeBackDateTime = combebackDateTime;

processedCall.TransferringDispatcher = \_dispatcherFIO;

processedCall.ProcessingDispatcher = \_dispatcherFIO;

processedCall.KilometrageBefor = kmBefore;

processedCall.KilometrageAfter = kmAfter;

processedCall.Place = place;

processedCall.CallNotes = notes;

processedCall.Treatment = textTreatment.Text;

\_context.ProcessedCalls.AddAsync(processedCall);

}

else

{

double incidentalCallNumber;

if (!string.IsNullOrEmpty(textIncidentalCallnumber.Text))

{

incidentalCallNumber = double.Parse(textIncidentalCallnumber.Text);

}

else

throw new ArgumentException("Не введен номер вызова");

var incidentalCall = new IncidentalCall()

{

CallNumber = incidentalCallNumber,

ProcessedCallId = (uint)incidentalCallNumber

};

incidentalCall.Patient = new Patient();

incidentalCall.AmbulanceBrigade = new AmbulanceBrigade();

incidentalCall.Patient.FIO = fio;

incidentalCall.Patient.Age = age;

incidentalCall.Diagnosis = diagnosis;

incidentalCall.Patient.Street = street;

incidentalCall.Patient.HouseNumber = houseNumber;

incidentalCall.Patient.FlatNumber = flatNumber;

incidentalCall.Patient.Gender = gender;

incidentalCall.Results = result;

incidentalCall.AmbulanceBrigade.Number = brigadeNumber;

incidentalCall.AmbulanceBrigade.Doktor = doktor;

incidentalCall.AmbulanceBrigade.MedicalAssistants1 = medAsstant1;

incidentalCall.AmbulanceBrigade.MedicalAssistants2 = medAsstant2;

incidentalCall.AmbulanceBrigade.Orderly = orderly;

incidentalCall.AmbulanceBrigade.Driver = driver;

incidentalCall.AmbulanceBrigade.BrigadeType = brigadeType;

incidentalCall.AmbulanceBrigade.StationName = station;

var processedCall = MainWindow.AmbulanceCallsForProcessing.Where(e => e.CallNumber == ((uint)incidentalCall.CallNumber)).FirstOrDefault();

if (processedCall != null)

{

incidentalCall.DateTimeReception = processedCall.DateTimeReception;

}

else

throw new ArgumentException();

incidentalCall.TransferDateTime = transferDateTime;

incidentalCall.ArrivalDateTime = arivalDateTime;

incidentalCall.DepartureDateTime = departureDateTime;

incidentalCall.ComeBackDateTime = combebackDateTime;

incidentalCall.TransferringDispatcher = \_dispatcherFIO;

incidentalCall.ProcessingDispatcher = \_dispatcherFIO;

incidentalCall.KilometrageBefor = kmBefore;

incidentalCall.KilometrageAfter = kmAfter;

incidentalCall.Place = place;

incidentalCall.CallNotes = notes;

incidentalCall.Treatment = textTreatment.Text;

\_context.IncidentalCalls.AddAsync(incidentalCall);

}

ClearButton\_Click(null, null);

MessageBox.Show("Вызов обработан");

\_context.SaveChanges();

}

catch (ArgumentException argException)

{

MessageBox.Show(argException.Message);

}

catch (FormatException ed)

{

MessageBox.Show(ed.Message);

}

}

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

{

textIncidentalCallnumber.Clear();

ComboStreets.Text = "";

textHouseNumber.Clear();

textFlatNumber.Clear();

textFIO.Clear();

textAge.Clear();

ComboDepartment.Text = "";

ComboReason.Text = "";

ComboCaller.Text = "";

ComboUrgency.Text = "";

textBoxNote.Clear();

comboDiagnosis.Text = "";

textTransferDate.Text = "\_\_.\_\_.\_\_";

textTransferTime.Text = "\_\_:\_\_";

textArrivalDate.Text = "\_\_.\_\_.\_\_";

textArrivalTime.Text = "\_\_:\_\_";

textDepartureDate.Text = "\_\_.\_\_.\_\_";

textDepartureTime.Text = "\_\_:\_\_";

textComebackDate.Text = "\_\_.\_\_.\_\_";

textComebackTime.Text = "\_\_:\_\_";

checkGender.IsChecked = false;

checkMale.IsEnabled = false;

chekFamale.IsEnabled = false;

comboResult.Text = "";

comboBrigadeType.Text = "";

comboDoktor.Text = "";

comboMedAssistant1.Text = "";

comboMedAssistant2.Text = "";

comboOrderly.Text = "";

comboDriver.Text = "";

textBrigadeNumber.Clear();

textKMBefore.Clear();

textKMAfter.Clear();

textTreatment.Clear();

}

private void TextBox\_GotFocus(object sender, RoutedEventArgs e)

{

((TextBox)sender).Text = "";

}

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

{

if (checkGender.IsChecked == true)

{

checkMale.IsEnabled = true;

chekFamale.IsEnabled = true;

}

else

{

checkMale.IsEnabled = false;

chekFamale.IsEnabled = false;

}

}

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

{

if (checkMale.IsChecked == true)

{

chekFamale.IsChecked = false;

}

}

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

{

if (chekFamale.IsChecked == true)

{

checkMale.IsChecked = false;

}

}

private void NumberValidationTextBox(object sender, TextCompositionEventArgs e)

{

Regex regex = new Regex("[^0-9]+");

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

}

private void TreatmentTextBox\_GotFocus(object sender, RoutedEventArgs e)

{

var treatmentWindow = new TreatmentWindow();

treatmentWindow.Owner = this;

treatmentWindow.ShowDialog();

textTreatment.Text = treatmentWindow.textTreatment.Text;

}

}

}

**ReportWindow.xaml**

using System.Windows;

using System.Linq;

using Ambulance.Classes;

using Ambulance.DBAccess;

namespace Ambulance

{

/// <summary>

/// Interaction logic for ReportWindow.xaml

/// </summary>

public partial class ReportWindow : Window

{

private AmbulanceDBContext \_context = new AmbulanceDBContext();

public ReportWindow()

{

InitializeComponent();

comboDiagnosis.ItemsSource = ServerService.GetDiagnoses();

comboResult.ItemsSource = ServerService.GetResults();

}

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

{

var count1 = \_context.ProcessedCalls.Count();

count1 = \_context.IncidentalCalls.Count();

lable1.Content = count1.ToString();

var count2 = \_context.Patients.Count();

lable2.Content = count2.ToString();

var count3 = \_context.ChronicPatients.Count();

lable3.Content = count3.ToString();

var count4 = \_context.ProcessedCalls.Where(e => e.Results == "Безрезультатный").Count();

count4 = \_context.IncidentalCalls.Where(e => e.Results == "Безрезультатный").Count();

lable4.Content = count4.ToString();

var count5 = \_context.ProcessedCalls.Where(e => e.Results.Contains("Смерть")).Count();

count5 += \_context.IncidentalCalls.Where(e => e.Results.Contains("Смерть")).Count();

lable5.Content = count5.ToString();

var count6 = \_context.ProcessedCalls.Where(e => e.Results == "Отказ от госпитализации").Count();

count6 = \_context.IncidentalCalls.Where(e => e.Results == "Отказ от госпитализации").Count();

lable6.Content = count6.ToString();

var count7 = \_context.Doktors.Count();

count7 = \_context.MedicalAssistants.Count();

count7 = \_context.Orderlies.Count();

count7 = \_context.Drivers.Count();

count7 = \_context.Dispatchers.Count();

lable7.Content = count7.ToString();

var count8 = \_context.Patients.Where(e => e.Gender.Contains("МУЖ")).Count();

lable8.Content = count8.ToString();

var count9 = \_context.Patients.Where(e => e.Gender.Contains("ЖЕН")).Count();

lable9.Content = count9.ToString();

var count10 = \_context.IncidentalCalls.Count();

lable10.Content = count10.ToString();

var count11 = \_context.Patients.Where(e => e.Age < 18).Count();

lable11.Content = count11.ToString();

var count12 = \_context.ProcessedCalls.Where(e => e.CallType == "Экстренный").Count();

lable12.Content = count12.ToString();

}

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

{

this.Close();

}

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

{

var diagnosis = comboDiagnosis.Text;

lableDiagnosis.Content = \_context.ProcessedCalls.Where(e => e.Diagnosis == diagnosis).Count();

}

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

{

var result = comboResult.Text;

lableResult.Content = \_context.ProcessedCalls.Where(e => e.Results == result).Count();

}

}

}

**ReprocessingCallsWindow.xaml**

using System.Windows;

using System.Linq;

using Ambulance.DBAccess;

using Ambulance.Classes;

using System.Windows.Controls;

using System.Text.RegularExpressions;

using System.Windows.Input;

using System;

namespace Ambulance

{

/// <summary>

/// Interaction logic for ReprocessingCallsWindow.xaml

/// </summary>

public partial class ReprocessingCallsWindow : Window

{

private AmbulanceDBContext \_context = new AmbulanceDBContext();

public ReprocessingCallsWindow()

{

InitializeComponent();

}

private void TextCallNumber\_LostFocus(object sender, RoutedEventArgs e)

{

try

{

var callNumber = int.Parse(textCallNumber.Text);

var call = \_context.ProcessedCalls.Where(e => e.CallNumber == callNumber).FirstOrDefault();

if (call != null)

{

SetCall(call);

}

}

catch

{

}

}

private void SetCall(ProcessedCall processedCall)

{

textDateComeback.Text = processedCall.AmbulanceBrigade.ActionDate;

textTimeComeback.Text = processedCall.AmbulanceBrigade.ActionTime;

textDateTransfer.Text = processedCall.TransferDateTime.Date.ToString("dd.mm.gggg");

textTimeTransfer.Text = processedCall.TransferDateTime.ToString("hh:mm");

lableBrigadeNumber.Content = processedCall.AmbulanceBrigade.Number.ToString();

comboDiagnosis.Text = processedCall.Diagnosis;

textDoktor.Text = processedCall.AmbulanceBrigade.Doktor;

textMedAsstant1.Text = processedCall.AmbulanceBrigade.MedicalAssistants1;

textMedAsstant2.Text = processedCall.AmbulanceBrigade.MedicalAssistants2;

textOrderly.Text = processedCall.AmbulanceBrigade.Orderly;

textDriver.Text = processedCall.AmbulanceBrigade.Driver;

comboStreet.Text = processedCall.Patient.Street;

textHouseNumber.Text = processedCall.Patient.HouseNumber.ToString();

textFlatNumber.Text = processedCall.Patient.FlatNumber.ToString();

textFIO.Text = processedCall.Patient.FIO;

textAge.Text = processedCall.Patient.Age.ToString();

textNotes.Text = processedCall.CallNotes;

}

private void TextBox\_GotFocus(object sender, RoutedEventArgs e)

{

((TextBox)sender).Text = "";

}

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

{

if (checkGender.IsChecked == true)

{

checkMale.IsEnabled = true;

chekFamale.IsEnabled = true;

}

else

{

checkMale.IsEnabled = false;

chekFamale.IsEnabled = false;

}

}

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

{

if (checkMale.IsChecked == true)

{

chekFamale.IsChecked = false;

}

}

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

{

if (chekFamale.IsChecked == true)

{

checkMale.IsChecked = false;

}

}

private void NumberValidationTextBox(object sender, TextCompositionEventArgs e)

{

Regex regex = new Regex("[^0-9]+");

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

}

private void TreatmentTextBox\_GotFocus(object sender, RoutedEventArgs e)

{

var treatmentWindow = new TreatmentWindow();

treatmentWindow.Owner = this;

treatmentWindow.ShowDialog();

textTreatment.Text = treatmentWindow.textTreatment.Text;

}

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

{

textDateComeback.Text = "\_\_.\_\_.\_\_";

textTimeComeback.Text = "\_\_:\_\_";

textDateTransfer.Text = "\_\_.\_\_.\_\_";

textTimeTransfer.Text = "\_\_:\_\_";

textDateArival.Text = "\_\_.\_\_.\_\_";

textTimeArival.Text = "\_\_:\_\_";

textDateDeparture.Text = "\_\_.\_\_.\_\_";

textTimeDeparture.Text = "\_\_:\_\_";

comboDiagnosis.Text = "";

comboRezault.Text = "";

textCallNumber.Text = "";

lableBrigadeNumber.Content= "";

textKMBefore.Text = "";

textKMAfter.Text = "";

textDoktor.Text = "-";

checkGender.IsChecked = false;

checkMale.IsEnabled = false;

chekFamale.IsEnabled = false;

comboPlace.Text = "";

textMedAsstant1.Text = "-";

textMedAsstant2.Text = "-";

textOrderly.Text = "-";

textDriver.Text = "-";

comboStreet.Text = "";

textHouseNumber.Text = "";

textFlatNumber.Text = "";

textFIO.Text = "";

textAge.Text = "";

textNotes.Text = "";

}

private ProcessedCall GetReprocessedCall()

{

try

{

var callNumber = uint.Parse(textCallNumber.Text);

var processedCall = \_context.ProcessedCalls.Where(e => e.CallNumber == callNumber).FirstOrDefault();

if (processedCall != null)

{

var dateOfComeback = textDateComeback.Text;

var TimeOfComeback = textTimeComeback.Text;

var dateOfTransfer = textDateTransfer.Text;

var timeOfTransfer = textTimeTransfer.Text;

var dateOfArival = textDateArival.Text;

var timeOfArival = textTimeArival.Text;

var dateOfDeparture = textDateDeparture.Text;

var timeOfDeparture = textTimeDeparture.Text;

var diagnosis = comboDiagnosis.Text;

var result = comboRezault.Text;

var kmBefore = int.Parse(textKMBefore.Text);

var kmAfter = int.Parse(textKMAfter.Text);

string gender = "";

if (checkGender.IsChecked == true)

{

if (checkMale.IsChecked == true)

{

gender = "МУЖСКОЙ";

}

else

{

if (chekFamale.IsChecked == true)

{

gender = "ЖЕНСКИЙ";

}

else

{

throw new ArgumentException("Не выбран пол пациента");

}

}

}

else

{

throw new ArgumentException("Не выбран пол пациента");

}

var place = comboPlace.Text;

var doktor = textDoktor.Text;

var medAsstant1 = textMedAsstant1.Text;

var medAsstant2 = textMedAsstant2.Text;

var orderly = textOrderly.Text;

var driver = textDriver.Text;

var street = comboStreet.Text;

var houseNumber = textHouseNumber.Text;

var flatNumber = textFlatNumber.Text;

var fio = textFIO.Text;

var age = double.Parse(textAge.Text);

var notes = textNotes.Text;

processedCall.Patient.FIO = fio;

processedCall.Patient.Age = age;

processedCall.Diagnosis = diagnosis;

processedCall.Patient.Street = street;

processedCall.Patient.HouseNumber = houseNumber;

processedCall.Patient.FlatNumber = flatNumber;

processedCall.Patient.Gender = gender;

processedCall.AmbulanceBrigade.Doktor = doktor;

processedCall.AmbulanceBrigade.MedicalAssistants1 = medAsstant1;

processedCall.AmbulanceBrigade.MedicalAssistants2 = medAsstant2;

processedCall.AmbulanceBrigade.Orderly = orderly;

processedCall.AmbulanceBrigade.Driver = driver;

processedCall.Results = result;

processedCall.TransferDateTime = DateTime.Parse(dateOfTransfer + " " + timeOfTransfer);

processedCall.ArrivalDateTime = DateTime.Parse(dateOfArival + " " + timeOfArival);

processedCall.DepartureDateTime = DateTime.Parse(dateOfDeparture + " " + timeOfDeparture);

processedCall.ComeBackDateTime = DateTime.Parse(dateOfComeback + " " + TimeOfComeback);

processedCall.KilometrageBefor = kmBefore;

processedCall.KilometrageAfter = kmAfter;

processedCall.Place = place;

processedCall.CallNotes = notes;

processedCall.Treatment = textTreatment.Text;

ClearButton\_Click(null, null);

return processedCall;

}

else

{

throw new ArgumentException("Нет вызова с таким номером.");

}

}

catch (ArgumentException argException)

{

MessageBox.Show(argException.Message);

return null;

}

}

private IncidentalCall GetIncidentalCall()

{

try

{

var incidentalCallNumber = double.Parse(textCallNumber.Text);

var incidentalCall = \_context.IncidentalCalls.Where(e => e.CallNumber == incidentalCallNumber).FirstOrDefault();

if (incidentalCall != null)

{

var dateOfComeback = textDateComeback.Text;

var TimeOfComeback = textTimeComeback.Text;

var dateOfTransfer = textDateTransfer.Text;

var timeOfTransfer = textTimeTransfer.Text;

var dateOfArival = textDateArival.Text;

var timeOfArival = textTimeArival.Text;

var dateOfDeparture = textDateDeparture.Text;

var timeOfDeparture = textTimeDeparture.Text;

var diagnosis = comboDiagnosis.Text;

var result = comboRezault.Text;

var kmBefore = int.Parse(textKMBefore.Text);

var kmAfter = int.Parse(textKMAfter.Text);

string gender = "";

if (checkGender.IsChecked == true)

{

if (checkMale.IsChecked == true)

{

gender = "МУЖСКОЙ";

}

else

{

if (chekFamale.IsChecked == true)

{

gender = "ЖЕНСКИЙ";

}

else

{

throw new ArgumentException("Не выбран пол пациента");

}

}

}

else

{

throw new ArgumentException("Не выбран пол пациента");

}

var place = comboPlace.Text;

var doktor = textDoktor.Text;

var medAsstant1 = textMedAsstant1.Text;

var medAsstant2 = textMedAsstant2.Text;

var orderly = textOrderly.Text;

var driver = textDriver.Text;

var street = comboStreet.Text;

var houseNumber = textHouseNumber.Text;

var flatNumber = textFlatNumber.Text;

var fio = textFIO.Text;

var age = double.Parse(textAge.Text);

var notes = textNotes.Text;

incidentalCall.Patient.FIO = fio;

incidentalCall.Patient.Age = age;

incidentalCall.Diagnosis = diagnosis;

incidentalCall.Patient.Street = street;

incidentalCall.Patient.HouseNumber = houseNumber;

incidentalCall.Patient.FlatNumber = flatNumber;

incidentalCall.Patient.Gender = gender;

incidentalCall.AmbulanceBrigade.Doktor = doktor;

incidentalCall.AmbulanceBrigade.MedicalAssistants1 = medAsstant1;

incidentalCall.AmbulanceBrigade.MedicalAssistants2 = medAsstant2;

incidentalCall.AmbulanceBrigade.Orderly = orderly;

incidentalCall.AmbulanceBrigade.Driver = driver;

incidentalCall.Results = result;

incidentalCall.TransferDateTime = DateTime.Parse(dateOfTransfer + " " + timeOfTransfer);

incidentalCall.ArrivalDateTime = DateTime.Parse(dateOfArival + " " + timeOfArival);

incidentalCall.DepartureDateTime = DateTime.Parse(dateOfDeparture + " " + timeOfDeparture);

incidentalCall.ComeBackDateTime = DateTime.Parse(dateOfComeback + " " + TimeOfComeback);

incidentalCall.KilometrageBefor = kmBefore;

incidentalCall.KilometrageAfter = kmAfter;

incidentalCall.Place = place;

incidentalCall.CallNotes = notes;

incidentalCall.Treatment = textTreatment.Text;

ClearButton\_Click(null, null);

return incidentalCall;

}

else

{

throw new ArgumentException("Попутного вызова с таким номером не существует.");

}

}

catch (ArgumentException argException)

{

MessageBox.Show(argException.Message);

return null;

}

catch (FormatException)

{

MessageBox.Show("Не верный формат номера.");

return null;

}

}

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

{

var callNumber = textCallNumber.Text;

if (!string.IsNullOrEmpty(callNumber))

{

if (!callNumber.Contains(','))

{

var reprocessedCall = GetReprocessedCall();

if (reprocessedCall != null)

{

\_context.Update(reprocessedCall);

}

}

else

{

var reprocessedIncidentalCall = GetIncidentalCall();

if (reprocessedIncidentalCall != null)

{

\_context.Update(reprocessedIncidentalCall);

}

}

\_context.SaveChangesAsync();

MessageBox.Show("Запись о вызове изменена.");

}

}

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

{

var callNumber = textCallNumber.Text;

if (!string.IsNullOrEmpty(callNumber))

{

if (!callNumber.Contains(','))

{

var processedCall = \_context.ProcessedCalls.Where(e => e.CallNumber == uint.Parse(callNumber)).FirstOrDefault();

if (processedCall != null)

{

\_context.Remove(processedCall);

}

}

else

{

var incidentalCall = \_context.IncidentalCalls.Where(e => e.CallNumber == double.Parse(callNumber)).FirstOrDefault();

if (incidentalCall != null)

{

\_context.Remove(incidentalCall);

}

}

\_context.SaveChangesAsync();

MessageBox.Show("Запись о вызове изменена.");

}

}

}

}

**SearchForCallInformation.xaml**

using Ambulance.Classes;

using System;

using System.Collections.Generic;

using System.Windows;

using System.Linq;

using Ambulance.DBAccess;

namespace Ambulance

{

/// <summary>

/// Interaction logic for SearchInformationAboutCall.xaml

/// </summary>

public partial class SearchInformationAboutCall : Window

{

private AmbulanceDBContext \_context = new AmbulanceDBContext();

public SearchInformationAboutCall()

{

InitializeComponent();

dateFrom.SelectedDate = DateTime.Parse("1.1." + DateTime.Now.Year);

dateTo.SelectedDate = DateTime.Now;

isAllTime.IsChecked = true;

dateFrom.IsEnabled = false;

dateTo.IsEnabled = false;

}

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

{

if (isAllTime.IsChecked == false)

{

dateFrom.IsEnabled = true;

dateTo.IsEnabled = true;

}

else

{

dateFrom.IsEnabled = false;

dateTo.IsEnabled = false;

}

}

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

{

try

{

var callNumber = textCallnumber.Text;

var fio = textFIO.Text;

var street = comboStreet.Text;

var houseNumber = textHouseNumber.Text;

var flatNumber = textFlatNumber.Text;

var diagnosis = comboDiagnosis.Text;

var result = comboResult.Text;

var startDate = dateFrom.SelectedDate;

var endDate = dateTo.SelectedDate;

List<Call> listOfCalls;

if (isAllTime.IsChecked == true)

{

listOfCalls = \_context.ProcessedCalls.ToList<Call>();

var listOfIncidentalCalls = \_context.IncidentalCalls.ToList<Call>();

listOfCalls.AddRange(listOfIncidentalCalls);

}

else

{

listOfCalls = \_context.ProcessedCalls.Where(

e => e.DateTimeReception.Date >= startDate.Value.Date && e.DateTimeReception <= endDate.Value.Date

).ToList<Call>();

var listOfIncidentalCalls = \_context.IncidentalCalls.Where(

e => e.DateTimeReception.Date >= startDate.Value.Date && e.DateTimeReception <= endDate.Value.Date

).ToList<Call>();

if (listOfIncidentalCalls.Any())

{

listOfCalls.AddRange(listOfIncidentalCalls);

}

}

if (!string.IsNullOrEmpty(callNumber) && listOfCalls.Any())

{

listOfCalls = listOfCalls.Where(e => e.CallNumber == double.Parse(callNumber)).ToList();

}

if (!string.IsNullOrEmpty(fio) && listOfCalls.Any())

{

listOfCalls = listOfCalls.Where(e => e.Patient.FIO.ToUpper().Contains(fio.ToUpper())).ToList();

}

if (!string.IsNullOrEmpty(street) && listOfCalls.Any())

{

listOfCalls = listOfCalls.Where(e => e.Patient.Street.ToUpper().Contains(street.ToUpper())).ToList();

}

if (!string.IsNullOrEmpty(houseNumber) && listOfCalls.Any())

{

listOfCalls = listOfCalls.Where(e => e.Patient.HouseNumber.ToUpper().Contains(houseNumber.ToUpper())).ToList();

}

if (!string.IsNullOrEmpty(flatNumber) && listOfCalls.Any())

{

listOfCalls.Where(e => e.Patient.FlatNumber.ToUpper().Contains(flatNumber.ToUpper())).ToList();

}

if (!string.IsNullOrEmpty(diagnosis) && listOfCalls.Any())

{

listOfCalls.Where(e => e.Diagnosis.ToUpper().Contains(diagnosis.ToUpper())).ToList();

}

if (!string.IsNullOrEmpty(result) && listOfCalls.Any())

{

listOfCalls = listOfCalls.Where(e => e.Results.ToUpper().Contains(result.ToUpper())).ToList();

}

if (listOfCalls.Any())

{

listOfCalls.ForEach(e => e.Patient = \_context.Patients.Where(p => p.Id == e.PatientId).First());

gridCalls.ItemsSource = listOfCalls;

lableFoundRecordNumber.Content = listOfCalls.Count;

}

}

catch (Exception ex)

{

MessageBox.Show(ex.Message);

}

}

private void GridCalls\_MouseDoubleClick(object sender, System.Windows.Input.MouseButtonEventArgs e)

{

ShowFullCallInformation();

}

private void ShowFullCallInformation()

{

var selectedCalls = gridCalls.SelectedItem;

if (selectedCalls != null)

{

var fullCallInformation = new FullCallInformationWindow((Call)selectedCalls);

fullCallInformation.Owner = this;

fullCallInformation.ShowDialog();

}

}

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

{

ShowFullCallInformation();

}

}

}

**TreatmentWindow.xaml**

using System.Text;

using System.Windows;

using Ambulance.Classes;

namespace Ambulance

{

/// <summary>

/// Interaction logic for TreatmentWindow.xaml

/// </summary>

public partial class TreatmentWindow : Window

{

private string \_treatment;

private StringBuilder stringBuilder = new StringBuilder();

public TreatmentWindow()

{

InitializeComponent();

comboTreatment.ItemsSource = ServerService.GetTreatment();

}

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

{

textTreatment.Text = "";

stringBuilder.Clear();

}

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

{

stringBuilder.Append(comboTreatment.Text + "\n");

\_treatment = stringBuilder.ToString();

textTreatment.Text = stringBuilder.ToString();

}

}

}

**WorkerWindow.xaml**

using System;

using System.Collections.Generic;

using System.Windows;

using System.Linq;

using Ambulance.Classes.Staff;

using Ambulance.DBAccess;

namespace Ambulance

{

/// <summary>

/// Interaction logic for WorkerWindow.xaml

/// </summary>

public partial class WorkerWindow : Window

{

private static AmbulanceDBContext context;

public WorkerWindow()

{

InitializeComponent();

context = new AmbulanceDBContext();

}

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

{

var listOfWorkers = new List<Worker>();

string fio = "";

if (checkAllWorkers.IsChecked == true)

{

fio = " ";

}

else

{

fio = textFIO.Text;

}

if (!string.IsNullOrEmpty(fio) || checkAllWorkers.IsChecked == true)

{

var workers = new List<Worker>();

workers = context.Doktors.Where(e => (e.Surname + " " + e.Name + " " + e.MiddleName).ToUpper().Contains(fio.ToUpper())).ToList<Worker>();

if (workers.Any())

{

listOfWorkers.AddRange(workers);

}

workers = context.Dispatchers.Where(e => (e.Surname + " " + e.Name + " " + e.MiddleName).ToUpper().Contains(fio.ToUpper())).ToList<Worker>();

if (workers.Any())

{

listOfWorkers.AddRange(workers);

}

workers = context.MedicalAssistants.Where(e => (e.Surname + " " + e.Name + " " + e.MiddleName).ToUpper().Contains(fio.ToUpper())).ToList<Worker>();

if (workers.Any())

{

listOfWorkers.AddRange(workers);

}

workers = context.Orderlies.Where(e => (e.Surname + " " + e.Name + " " + e.MiddleName).ToUpper().Contains(fio.ToUpper())).ToList<Worker>();

if (workers.Any())

{

listOfWorkers.AddRange(workers);

}

workers = context.Drivers.Where(e => (e.Surname + " " + e.Name + " " + e.MiddleName).ToUpper().Contains(fio.ToUpper())).ToList<Worker>();

if (workers.Any())

{

listOfWorkers.AddRange(workers);

}

workers = context.Statisticians.Where(e => (e.Surname + " " + e.Name + " " + e.MiddleName).ToUpper().Contains(fio.ToUpper())).ToList<Worker>();

if (workers.Any())

{

listOfWorkers.AddRange(workers);

}

UpdateWorkerGrid(listOfWorkers);

ClearFilds();

}

else

{

MessageBox.Show("Введите Ф.И.О или выберите 'Все работники'","Ошибка!");

}

}

private void UpdateWorkerGrid(List<Worker> listOfWorkers)

{

gridWorkers.ItemsSource = null;

gridWorkers.ItemsSource = listOfWorkers;

}

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

{

this.Close();

}

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

{

var selectedWorker = gridWorkers.SelectedItem;

if (selectedWorker != null && MessageBox.Show("Удалить эту запись?", "Удаление сотрудника", MessageBoxButton.YesNo) == MessageBoxResult.Yes)

{

if (selectedWorker is Doktor)

{

context.Doktors.Remove((Doktor)selectedWorker);

}

if (selectedWorker is Dispatcher)

{

context.Dispatchers.Remove((Classes.Staff.Dispatcher)selectedWorker);

}

if (selectedWorker is MedicalAssistant)

{

context.MedicalAssistants.Remove((MedicalAssistant)selectedWorker);

}

if (selectedWorker is Orderly)

{

context.Orderlies.Remove((Orderly)selectedWorker);

}

if (selectedWorker is Driver)

{

context.Drivers.Remove((Driver)selectedWorker);

}

if (selectedWorker is Statistician)

{

context.Statisticians.Remove((Statistician)selectedWorker);

}

}

context.SaveChanges();

MessageBox.Show($"Сотрудник {((Worker)selectedWorker).ToString()} удален");

Search\_Click(null, null);

ClearFilds();

}

private void ClearFilds()

{

textFIO.Text = "";

comboPosition.Text = "";

}

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

{

try

{

bool isAdd = false;

var fio = textFIO.Text.Split(" ");

var position = comboPosition.Text;

if (string.IsNullOrEmpty(position) || fio.Length != 3)

{

throw new ArgumentException("Не правильно введено имя или должность сотрудника.");

}

switch (position)

{

case "Врач":

if (!context.Doktors.Where(e => e.Surname == fio[0] && e.Name == fio[1] && e.MiddleName == fio[2]).Any())

{

var doctor = new Doktor {Surname = fio[0], Name = fio[1], MiddleName = fio[2]};

context.Doktors.Add(doctor);

isAdd = true;

}

break;

case "Диспетчер":

var dispatcher = new Dispatcher { Surname = fio[0], Name = fio[1], MiddleName = fio[2] };

if (!context.Dispatchers.Where(e => e.Surname == fio[0] && e.Name == fio[1] && e.MiddleName == fio[2]).Any())

{

context.Dispatchers.Add(dispatcher);

isAdd = true;

}

break;

case "Фельдшер":

var assistant = new MedicalAssistant { Surname = fio[0], Name = fio[1], MiddleName = fio[2] };

if (!context.MedicalAssistants.Where(e => e.Surname == fio[0] && e.Name == fio[1] && e.MiddleName == fio[2]).Any())

{

context.MedicalAssistants.Add(assistant);

isAdd = true;

}

break;

case "Санитар":

var orderly = new Orderly { Surname = fio[0], Name = fio[1], MiddleName = fio[2] };

if (!context.Orderlies.Where(e => e.Surname == fio[0] && e.Name == fio[1] && e.MiddleName == fio[2]).Any())

{

context.Orderlies.Add(orderly);

isAdd = true;

}

break;

case "Водитель":

var driver = new Driver { Surname = fio[0], Name = fio[1], MiddleName = fio[2] };

if (!context.Drivers.Where(e => e.Surname == fio[0] && e.Name == fio[1] && e.MiddleName == fio[2]).Any())

{

context.Drivers.Add(driver);

isAdd = true;

}

break;

case "Статистик":

var statistician = new Statistician { Surname = fio[0], Name = fio[1], MiddleName = fio[2] };

if (!context.Statisticians.Where(e => e.Surname == fio[0] && e.Name == fio[1] && e.MiddleName == fio[2]).Any())

{

context.Statisticians.Add(statistician);

isAdd = true;

}

break;

default:

throw new ArgumentException();

}

if (isAdd)

{

context.SaveChanges();

ClearFilds();

MessageBox.Show("Сотрудник добавлен.");

}

else

{

MessageBox.Show($"Сотрудник {fio[0] + " " + fio[1] + " " + fio[2]} уже добавлен.");

}

}

catch(ArgumentException argException)

{

MessageBox.Show(argException.Message);

}

}

private void Window\_Closed(object sender, EventArgs e)

{

context.Dispose();

}

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

{

if (checkAllWorkers.IsChecked == true)

{

textFIO.IsEnabled = false;

comboPosition.IsEnabled = false;

}

else

{

textFIO.IsEnabled = true;

comboPosition.IsEnabled = true;

}

}

}

}