содержание

Изм.

Лист

№ докум.

Подпись

Дата

Лист

2

Дипломная работа

[ВВЕДЕНИЕ 3](#_Toc100779585)

[1 Аналитический обзор существующих методов и средств решения поставленной задачи 5](#_Toc100779586)

[1.1 Пример решения аналогичной задачи 5](#_Toc100779587)

[1.2 Язык и среда программирования 7](#_Toc100779589)

[1.3 Основы графических интерфейсов 7](#_Toc100779590)

[1.4 Основы Web сервисов 10](#_Toc100779591)

[1.5 Основы проектирования базы данных 11](#_Toc100779592)

[1.6 Постановка задачи 13](#_Toc100779594)

[2 Архитектура программного обеспечения 15](#_Toc100779595)

[2.1 Анализ задания на проектирование 15](#_Toc100779596)

[2.1.1 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. 18](#_Toc100779598)

[2.1.2 После выполнения вызова, он поступает в список вызовов для обработки. Диспетчер должен заменить неточную информацию о вызове на информацию, уточненную у пациента. 19](#_Toc100779599)

[2.2 Доменные классы 19](#_Toc100779600)

[Список использованных источников 26](#_Toc100779648)

# ВВЕДЕНИЕ

Изм.

Лист

№ докум.

Подпись

Дата

Лист

3

Дипломная работа

Уровень внедрения комплексных медицинских информационных систем в белорусском здравоохранении, по сдержанным оценкам, не превышает в настоящее время 30% 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.

Создание Web сервисов позволит унифицировать работу настольных и Web приложений. Это позволит переиспользовать функционал на разных платформах.

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.

Изм.

Лист

№ докум.

Подпись

Дата

Лист

4

Дипломная работа

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.

Цель работы:

Создать приложение позволяющее облегчить и автоматизировать работу станций скорой медицинской помощи, а также собирать и анализировать статистические данные. Программа должна осуществлять передачу информации о вызовах между подстанциями и вести учет больных. Отслеживать работу бригад скорой помощи и вести учет вызовов, на которые реагировала бригада. Выводить статистические данные по разным критериям для пользователей с определенным доступом, для анализа работы станций.

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. Собирать и выводить статистические данные, позволять анализировать работу станций. Также в задачи работы включается создание Web приложения, которое позволит ускорить и облегчить работу сотрудников.

# **Аналитический обзор существующих методов и средств решения поставленной задачи**

Изм.

Лист

№ докум.

Подпись

Дата

Лист

5

Дипломная работа

## **Пример решения аналогичной задачи**

Готовые программные решения для станций скорой медицинской помощи и мониторинга их работы являются узкоспециализированными, именно поэтому список возможных аналогов представлен несколькими системами. Также анализ существующих решений усложняется тем, что данные приложения не находятся в свободном доступе, даже для ознакомления.

Специализированная программная система ISIDA 103 обеспечивает управление деятельностью станции скорой медицинской помощи (ССМП) – как централизованной, так и децентрализованной, имеющей развитую сеть филиалов (подстанций).

Основные функции системы:

1. прием и регистрация вызовов от населения:

* получение номера телефона входящего вызова;
* определение повода к вызову скорой медицинской помощи, соответствующего ему профиля выездной бригады и номера подстанции, с которой будет осуществлен выезд;
* определение места вызова с использованием электронной карты местности;
* регистрация вызова.

1. диспетчеризация вызовов:

* передача электронной карты вызова на соответствующую подстанцию, в соответствии с номером подстанции в карте вызова;
* назначение выездных бригад на вызов, в том числе с использованием электронной карты местности;
* передача электронных карт вызова выездным бригадам, в том числе с возможностью получения карты вызова на экран мобильного устройства (смартфона или планшетного компьютера);
* закрытие карт вызова.

1. ввод данных о результатах вызовов:

* ввод уточненных данных о пациенте;
* ввод результата вызова и диагнозов по МКБ-10;
* ввод данных о выполненных специфических мероприятиях при оказании медицинской помощи;
* ввод данных об израсходованных медикаментах.

1. формирование оперативной отчетности;
2. формирование статистической отчетности на основании закрытых карт вызовов.

Изм.

Лист

№ докум.

Подпись

Дата

Лист

6

Дипломная работа

Кроме того, система снабжена развитыми поисковыми возможностями, набором настраиваемых отчётных срезов (признаки, по которым можно искать и производить группировку данных), системой адресной привязки зон обслуживания подстанций и поликлиник, возможностью формирования графиков смен выездных бригад и оперативного изменения профилей бригад.

Функциональность системы может быть расширена за счет подключения дополнительных модулей и организации взаимодействия с информационными системами региона

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.

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.

Существующая программа позволяет производить поиск вызовов и пациентов в системе, включает в себя список возможных адресов, болезней, бригад, сотрудников, адреса поликлиник, результатов вызова. Данная система также обладает возможностью наблюдения за бригадами скорой помощи с помощью GPS трекеров.

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.

Недостатками являются:

1. ручной ввод вызовов;
2. ручная обработка выполненных вызовов;
3. неполностью продуманный интерфейс программы.

## **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.

Изм.

Лист

№ докум.

Подпись

Дата

Лист

7

Дипломная работа

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.

## **Основы графических интерфейсов**

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.

Изм.

Лист

№ докум.

Подпись

Дата

Лист

8

Дипломная работа

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 Web-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, Web-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.

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, Web-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.

Изм.

Лист

№ докум.

Подпись

Дата

Лист

9

Дипломная работа

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.

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;

Изм.

Лист

№ докум.

Подпись

Дата

Лист

10

Дипломная работа

1. 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.

## **Основы Web сервисов**

Web API представляет способ построения приложения ASP.NET, который специально заточен для работы в стиле REST.

Зачастую REST-стиль особенно удобен при создании всякого рода Single Page Application, которые нередко используют специальные javascript фреймворки типа Angular, React или Vue.js. По сути Web API представляет собой Web-службу, к которой могут обращаться другие приложения. Причем эти приложения могут представлять любую технологию и платформу - это могут быть Web-приложения, мобильные или десктопные клиенты.

REST (Representational state transfer) – это стиль архитектуры программного обеспечения для распределенных систем, таких как World Wide Web, который, как правило, используется для построения Web-служб. Термин REST был введен в 2000 году Роем Филдингом, одним из авторов HTTP протокола. Системы, поддерживающие REST, называются RESTful-системами. В общем случае REST является очень простым интерфейсом управления информацией без использования каких-то дополнительных внутренних прослоек. Каждая единица информации однозначно определяется глобальным идентификатором, таким как URL. Каждая URL в свою очередь имеет строго заданный формат.

Особенности архитектурного стиля:

* каждая сущность должна иметь уникальный идентификатор – URI;
* сущности должны быть связаны между собой;
* для чтения и изменения данных должны использоваться;
* стандартные методы;
* должна быть поддержка нескольких типов ресурсов;
* взаимодействие должно осуществляться без состояния.

Изм.

Лист

№ докум.

Подпись

Дата

Лист

11

Дипломная работа

Стандартные методы таковы:

* GET – получение данных без их изменения. Это наиболее популярный и легкий метод. Он только возвращает данные, а не изменяет их, поэтому на клиенте вам не нужно заботиться о том, что вы можете повредить данные;
* POST – метод, подразумевающий вставку новых записей;
* PUT – метод, подразумевающий изменение существующих записей;
* PATCH – метод, подразумевающий изменение идентификатора существующих записей;
* DELETE – метод, подразумевающий удаление записей.

## **Основы проектирования базы данных**

Проектирование базы данных для приложения осуществлялась с помощью MsSQL Server используя Sql management studio для создания БД.

Microsoft SQL Server — система управления реляционными базами данных (РСУБД), разработанная корпорацией Microsoft. Основной используемый язык запросов — Transact-SQL, создан совместно Microsoft и Sybase. Transact-SQL является реализацией стандарта ANSI/ISO по структурированному языку запросов (SQL) с расширениями. Используется для работы с базами данных размером от персональных до крупных баз данных масштаба предприятия; конкурирует с другими СУБД в этом сегменте рынка.

SQL Server — это основа платформы обработки данных Майкрософт, которая предоставляет надежную и устойчивую производительность (в том числе благодаря технологиям обработки данных в памяти) и помогает быстрее извлечь ценную информацию из любых данных, расположенных как в локальной среде, так и в облаке.

SQL Server характеризуется такими особенностями как:

* производительность;
* надежность и безопасность;
* простота.

Центральным аспектом в MS SQL Server, как и в любой СУБД, является база данных. База данных представляет хранилище данных, организованных определенным способом. Нередко физически база данных представляет файл на жестком диске, хотя такое соответствие необязательно. Для хранения и администрирования баз данных применяются системы управления базами данных (database management system) или СУБД (DBMS). И как раз MS SQL Server является одной из такой СУБД.

Для организации баз данных MS SQL Server использует реляционную модель. Эта модель баз данных была разработана еще в 1970 году Эдгаром Коддом. А на сегодняшний день она фактически является стандартом для организации баз данных.

Изм.

Лист

№ докум.

Подпись

Дата

Лист

12

Дипломная работа

Реляционная модель предполагает хранение данных в виде таблиц, каждая из которых состоит из строк и столбцов. Каждая строка хранит отдельный объект, а в столбцах размещаются атрибуты этого объекта.

Для идентификации каждой строки в рамках таблицы применяется первичный ключ (primary key). В качестве первичного ключа может выступать один или несколько столбцов. Используя первичный ключ, мы можем ссылаться на определенную строку в таблице. Соответственно две строки не могут иметь один и тот же первичный ключ.

Через ключи одна таблица может быть связана с другой, то есть между двумя таблицами могут быть организованы связи. А сама таблица может быть представлена в виде отношения ("relation").

Для взаимодействия с базой данных применяется язык SQL (Structured Query Language). Клиент (например, внешняя программа) отправляет запрос на языке SQL посредством специального API. СУБД должным образом интерпретирует и выполняет запрос, а затем посылает клиенту результат выполнения.

Изначально язык SQL был разработан в компании IBM для системы баз данных, которая называлась System/R. При этом сам язык назывался SEQUEL

(Structured English Query Language). Хотя в итоге ни база данных, ни сам язык не были впоследствии официально опубликованы, по традиции сам термин SQL нередко произносят как "сиквел".

В 1979 году компания Relational Software Inc. разработала первую систему управления баз данных, которая называлась Oracle и которая использовала язык SQL. В связи с успехом данного продукта компания была переименована в Oracle.

Впоследствии стали появляться другие системы баз данных, которые использовали SQL. В итоге в 1989 году Американский Национальный Институт Стандартов (ANSI) кодифицировал язык и опубликовал его первый стандарт. После этого стандарт периодически обновлялся и дополнялся. Последнее его обновление состоялось в 2011 году. Но несмотря на наличие стандарта нередко производители СУБД используют свои собственные реализации языка SQL, которые немного отличаются друг от друга.

Выделяются две разновидности языка SQL: PL-SQL и T-SQL. PL-SQL используется в таких СУБД как Oracle и MySQL. T-SQL (Transact-SQL) применяется в SQL Server.

Таким образом язык программирования C# позволит создать быстрое и удобное настольное приложение для работы в офисе, которое позволит использовать Web сервисы для оптимизации и обобщения работы между несколькими частями системы. Web сервис на базе WebAPI позволит создать Web приложение с помощью которого несколько программных решений могут иметь одинаковый функционал, что позволит переложить ответственность выполнения конкретной задачи на других участников процесса.

Изм.

Лист

№ докум.

Подпись

Дата

Лист

13

Дипломная работа

База данных построенная с использованием реляционного подхода, позволит создать нормализированные сущности, что позволяет удобно хранить и обрабатывать данные. Работать с такой базой намного проще так как между всеми сущностями имеются различные связи, которые описывают взаимоотношения между ними.

## **Основы контейнеризации**

Docker — программное обеспечение для автоматизации развёртывания и управления приложениями в средах с поддержкой [контейнеризации](https://ru.wikipedia.org/wiki/%D0%9A%D0%BE%D0%BD%D1%82%D0%B5%D0%B9%D0%BD%D0%B5%D1%80%D0%B8%D0%B7%D0%B0%D1%86%D0%B8%D1%8F), контейнеризатор приложений. Позволяет «упаковать» приложение со всем его [окружением](https://ru.wikipedia.org/w/index.php?title=%D0%9E%D0%BF%D0%B5%D1%80%D0%B0%D1%86%D0%B8%D0%BE%D0%BD%D0%BD%D0%BE%D0%B5_%D0%BE%D0%BA%D1%80%D1%83%D0%B6%D0%B5%D0%BD%D0%B8%D0%B5&action=edit&redlink=1) и зависимостями в контейнер, который может быть развёрнут на любой [Linux](https://ru.wikipedia.org/wiki/Linux)-системе с поддержкой [контрольных групп](https://ru.wikipedia.org/wiki/Cgroups) в [ядре](https://ru.wikipedia.org/wiki/%D0%AF%D0%B4%D1%80%D0%BE_Linux), а также предоставляет набор команд для управления этими контейнерами. Изначально использовал возможности [LXC](https://ru.wikipedia.org/wiki/LXC), с 2015 года начал использовать собственную библиотеку, абстрагирующую виртуализационные возможности ядра Linux — libcontainer. С появлением Open Container Initiative начался переход от монолитной к модульной архитектуре.

## **Постановка задачи**

Целью дипломной работы является спроектировать приложение позволяющее собирать и обрабатывать статистику межрегиональных станций скорой медицинской помощи.

Программа должна моделировать клиент – серверное приложение и должно быть построено на базе микросервисной архитектуры. Набор создаваемых приложений должен предоставлять собой распределенный узел работы. 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.

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

Web приложение должно осуществлять регистрацию вызова, передачу его бригадам, также должно обладать функциональностью по обработке завершенного вызова и передаче вызова освободившейся бригаде.

При регистрации вызова выполняется запрос к микросервису на создание записи в базе данных. 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, либо бригада обрабатывает его сама с помощью Web приложения, после чего сможет взять следующий доступный вызов.

Обработанный вызов передается микросервису, 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.

Изм.

Лист

№ докум.

Подпись

Дата

Лист

14

Дипломная работа

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

Просмотр вызовов выполненные каждой бригадой за смену.

# **АРХИТЕКТУРА ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ**

Изм.

Лист

№ докум.

Подпись

Дата

Лист

15

Дипломная работа

## **Анализ задания на проектирование**

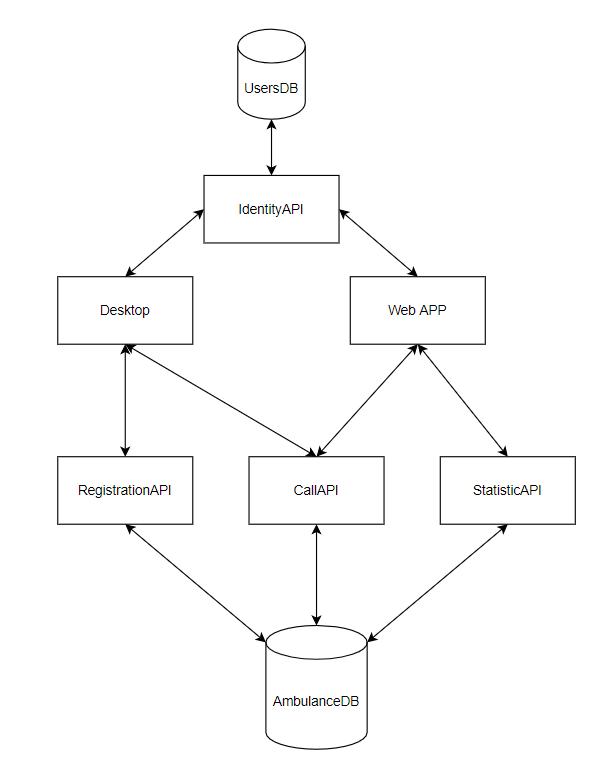


Рисунок 2.1 – Схема взаимодействия модулей

Согласно рисункам 2.1 и 2.2, настольное приложение обращается к CallAPI для реализации обработки вызовов, а именно регистрации вызова, обработка вызова, получение вызова или списка вызовов.

Diagram

Description automatically generated

Изм.

Лист

№ докум.

Подпись

Дата

Лист

16

Дипломная работа

Рисунок 2.2 – Схема взаимодействия диспетчера с системой

Также обращение к RegistrationAPI позволяет регистрировать для работы сотрудников, бригады скорой помощи и данные нужные для работы приложения.

Web приложение осуществляет похожую работу с вызовами как и настольное приложение, но также позволяет просматривать статистические данные по работе скорой помощи. Для работы обоих приложений, настольного и Web, требуется авторизоваться.

Diagram

Description automatically generated

Рисунок 2.3 – Схема взаимодействия фельдшера с Web приложением

Согласно рисунку 2.3, авторизация осуществляется с помощью IdentityAPI, к которой обращаются оба приложения. Не авторизированные пользователи не могут работать с настольным и Web приложениями.

Изм.

Лист

№ докум.

Подпись

Дата

Лист

17

Дипломная работа

Diagram

Description automatically generated

Рисунок 2.4 – Действия диспетчера

В обязанности диспетчера скорой помощи входит регистрация вызовов, их передача, финальная обработка ввод бригад скорой помощи. Подробнее на рисунке 2.4.

В результате, данное приложение должно помочь отказаться от ручного труда, благодаря Web решению и создания микросервисов. Заполнением дополнительной информации, согласно рисунку 2.5, будет заниматься сама бригада, которая выполняет конкретный вызов, данное заполнение будет происходить через браузер.

Diagram

Description automatically generated

Рисунок 2.5 – Заполнение дополнительной информации по вызову

### Diagram Description automatically generatedВызов 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. Подробнее на рисунке 2.6.

Изм.

Лист

№ докум.

Подпись

Дата

Лист

18

Дипломная работа

Рисунок 2.6 – Алгоритм регистрации вызова

В случае ошибочного заполнения, доступна функция очистки формы.

### После выполнения вызова, он поступает в список вызовов для обработки. Диспетчер должен заменить неточную информацию о вызове на информацию, уточненную у пациента. Подробнее на рисунке 2.7.

Изм.

Лист

№ докум.

Подпись

Дата

Лист

19

Дипломная работа

Chart, diagram, box and whisker chart

Description automatically generated

Рисунок 2.7 – Алгоритм обработки вызова

## **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;
* (Вызов) Call;
* (Бригада) AmbulanceBrigade;
* (Глобавльные настройки) GlobalSettings.

Доменная область не подразумевает удаление записей.

Согласно рисунку 2.8, все сущности в схеме имеют отношение один ко многим между несколькими связанными сущностями, кроме сущности GlobalSettings так как эта таблица создана для хранения настроек.

Изм.

Лист

№ докум.

Подпись

Дата

Лист

20

Дипломная работа

A screenshot of a computer

Description automatically generated with low confidence

Рисунок 2.8 – Диаграмма доменных классов

Сущность Patient.

Описывает сущность «Пациент», которого обслуживает бригада скорой помощи. Содержит поля, которые содержат всю нужную информацию о пациенте для обработки. Подробнее в таблице 2.1.

Таблица 2.1 – Сущность Patient.

|  |  |  |  |
| --- | --- | --- | --- |
| **Название** | **Содержание** | **Тип** | **Комментарий** |
| Id | ID | int | Идентификационный номер |
| FIO | Фамилия Имя Отчество | varchar | Фамилия Имя Отчество пациента |
| Age | Возраст | Float | Возраст пациента |
| Diagnosis | Диагноз | varchar | Диагноз поставленный бригадой |

Продолжение таблицы 2.1

|  |  |  |  |
| --- | --- | --- | --- |
| Street | Улица адреса прописки | varchar | Улица на которой прописан пациент |
| HouseNumber | Номер дома адреса прописки | varchar | Номер дома в котором прописан |
| FlatNumber | Номер квартиры адреса прописки | varchar | Номер квартиры в которой прописан |
| Gender | Пол пациента | varchar | Пол пациента |

Изм.

Лист

№ докум.

Подпись

Дата

Лист

21

Дипломная работа

Сущность Dispatcher.

Описывает сущность «Диспетчер скорой помощи», который работает с настольной программой. Подробнее в таблице 2.2.

Таблица 2.2 – Сущность Dispatcher

|  |  |  |  |
| --- | --- | --- | --- |
| **Название** | **Содержание** | **Тип** | **Комментарий** |
| Id | ID | int | Идентификационный номер |
| Name | Имя | varchar | Имя сотрудника |
| Surname | Фамилия | varchar | Фамилия сотрудника |
| MiddleName | Отчество | varchar | Отчество сотрудника |

Сущность Doktor

Описывает сущность «Доктор выездной бригады» скорой помощи. Подробнее в таблице 2.3.

Таблица 2.3 – Сущность Doktor

|  |  |  |  |
| --- | --- | --- | --- |
| **Название** | **Содержание** | **Тип** | **Комментарий** |
| Id | ID | int | Идентификационный номер |
| Name | Имя | varchar | Имя сотрудника |

Продолжение таблицы 2.3

|  |  |  |  |
| --- | --- | --- | --- |
| Surname | Фамилия | varchar | Фамилия сотрудника |
| MiddleName | Отчество | varchar | Отчество сотрудника |

Изм.

Лист

№ докум.

Подпись

Дата

Лист

22

Дипломная работа

Сущность MedicalAssistant.

Описывает сущность «Фельдшер выездной бригады» скорой помощи. Подробнее в таблице 2.4.

Таблица 2.4 – Сущность MedicalAssistant

|  |  |  |  |
| --- | --- | --- | --- |
| **Название** | **Содержание** | **Тип** | **Комментарий** |
| Id | ID | int | Идентификационный номер |
| Name | Имя | varchar | Имя сотрудника |
| Surname | Фамилия | varchar | Фамилия сотрудника |
| MiddleName | Отчество | varchar | Отчество сотрудника |

Сущность Orderly.

Описывает сущность «Санитар выездной бригады» скорой помощи. Подробнее в таблице 2.5.

Таблица 2.5 – Сущность Orderly

|  |  |  |  |
| --- | --- | --- | --- |
| **Название** | **Содержание** | **Тип** | **Комментарий** |
| Id | ID | int | Идентификационный номер |
| Name | Имя | varchar | Имя сотрудника |
| Surname | Фамилия | varchar | Фамилия сотрудника |
| MiddleName | Отчество | varchar | Отчество сотрудника |

Сущность Driver.

Описывает сущность «Водитель машины выездной бригады» скорой помощи. Подробнее в таблице 2.6.

Таблица 2.6 – Сущность Driver

|  |  |  |  |
| --- | --- | --- | --- |
| **Название** | **Содержание** | **Тип** | **Комментарий** |
| Id | ID | int | Идентификационный номер |
| Name | Имя | varchar | Имя сотрудника |
| Surname | Фамилия | varchar | Фамилия сотрудника |
| MiddleName | Отчество | varchar | Отчество сотрудника |

Изм.

Лист

№ докум.

Подпись

Дата

Лист

23

Дипломная работа

Сущность Call.

Описывает сущность «Вызов», содержит в себе информацию о пациенте, о бригаде, которая выполняла вызов, о лечении и результате лечения. Подробнее в таблице 2.7.

Таблица 2.7 – Сущность Call

|  |  |  |  |
| --- | --- | --- | --- |
| **Название** | **Содержание** | **Тип** | **Комментарий** |
| Id | ID | int | Идентификационный номер |
| AmbulanceBrigadeId | Идентификатор бригады | Int | Идентификатор бригады |
| DispatcherId | Идентификатор диспетчера | Int | Идентификатор диспетчера принявшего вызов |
| MainCallNumber | Номер вызова с которым связан текущий | Int | Номер вызова с которым связан текущий |
| IsIncidential | Флаг показывающий тип вызова | Bit | Вызов может быть основным(false) или попутным(true) |
| CallNumber | Номер вызова | uint | Уникальный номер вызова |
| PatientId | Номер пациента | int | Номер пациента |
| Results | Результат | varchar | Результата вызова |
| DateTimeReception | Дата и время получения | DateTime | Дата и время получения вызова |
| TransferDateTime | Дата и время передачи | DateTime | Дата и время передачи вызова |
| ArrivalDateTime | Дата и время прибытия | DateTime | Дата и время прибытия вызова |
| DepartureDateTime | Дата и время отправления | DateTime | Дата и время выезда с вызова |

Продолжение таблицы 2.7

Изм.

Лист

№ докум.

Подпись

Дата

Лист

24

Дипломная работа

|  |  |  |  |
| --- | --- | --- | --- |
| ComeBackDateTime | Дата и время возвращения | DateTime | Дата и время возвращения на подстанцию |
| TransferringDispatcherId | Передающий диспетчер | int | Диспетчер передавший вызов |
| ProcessingDispatcher | Обрабатывающий диспетчер | varchar | Диспетчер обработавший вызов |
| KilometrageBefor | Километраж до | int | Километраж машины до вызова |
| KilometrageAfter | Километраж после | int | Километраж машины после вызова |
| Place | Место | varchar | Место обслуживания пациента |
| CallNotes | Заметки | varchar | Заметки к вызову |
| Treatment | Лечение | varchar | Лечение |
| CallType | Тип вызова | varchar | Тип вызова |
| IsProcessed | Флаг состояния обработки вызова | bit | Показывает обработан вызов или нет |

Сущность AmbulanceBrigade.

Данная сущность базы данных описывает бригаду скорой помощи которая занимается выполнением вызова. Подробнее в таблице 2.8.

Таблица 2.8 – Сущность AmbulanceBrigade

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Название** | **Содержание** | | **Тип** | | **Комментарий** | |
| Id | Уникальный идентификатор | | Int | | Уникальный идентификатор | |
| DoktorId | Идентификатор врача | | Int | | Идентификатор врача | |
| FirstMedicalAssistantId | Идентификатор фельдшера | | Int | | Идентификатор фельдшера | |
| SecondMedicalAssistantId | | Идентификатор фельдшера | | Int | | Идентификатор фельдшера |
| DriverId | | Идентификатор водителя | | Int | | Идентификатор водителя |
| OrderlyId | | Идентификатор санитара | | Int | | Идентификатор санитара |

Продолжение таблицы 2.8

Изм.

Лист

№ докум.

Подпись

Дата

Лист

25

Дипломная работа

|  |  |  |  |
| --- | --- | --- | --- |
| Number | Номер бригады | Int | Номер бригады |
| BrigadeType | Тип бригады | Byte | Тип бригады |
| DateTimeStart | Время и дата начала работы | Datetime | Время и дата начала работы |
| DateTimeEnd | Время и дата конца работы | Datetime | Время и дата конца работы |
| StationName | Название станции за которой закреплена бригада | varchar | Название станции за которой закреплена бригада |

# **Разработка программного обеспечени**

## **Уровень доступа к данным**

Для доступа к базе данных используется подключаемая библиотека EntityFrameworkCore. Она построена с помощью таких паттернов как репозиторий и единица работы, что позволяет получить многофункциональный доступ к базе и строить сложные запросы в самом коде, вмето сложных запрососв SQL.

Для того чтобы осуществлять базовые CRUD операции над сущностями создается класс контекста базы данных CallContext, который представлен на рисунке 3.1.

Graphical user interface, text

Description automatically generated

Рисунок 3.1 – Класс доступа к базе данных

Также, для получения данных создаются сущности соответсвующие таблицам базы. В классе контекста данные таблица представляются в ввиде коллекций сущностей DbSet<T>, где T конкретная сущность доменной области. Список сущностей представлен на рисунке 3.2.

Text

Description automatically generated with medium confidence

Рисунок 3.2 – Сущности базы данных

Для обеспечения целостности и согласованности базы, создается поставщик контекста базы, рисунок 3.3, который оборачивает любое обращение к базе в транзакцию, которая подтверждается при успешном завершении запроса или отменяет все внесенные изменения, что позволяет базе находиться в состоянии согласованности данных.

Text

Description automatically generated

Рисунок 3.3 – Поставщик контекста базы DatabaseProvider

## **WEB-сервис CallAPI**

По требованию REST спецификации приложения должны быть разработаны на основе клиент-серверной архитектуры. В роли серверов выступают WEB-сервисы CallAPI, ServiceAPI, StatisticsAPI.

Данные сервисы написаны с использованием паттерна проектирования Mediatr. Данный подход позволяет реализовать принцип единственной ответсвенности у контроллеров и уменьшить количество зависимостей на другие внутренние сервисы, которые используются для выполнения команды. Пример использования медиатора можно увидеть на рисунке 3.1

Text

Description automatically generated

Рисунок 3.1 – Пример использования паттерна Mediatr

Как видно из риунка, передача команд на выполнение запросов происходит с помощью одного метода Send. Для этого передаваемая команда должна реализовывать интерфейс IRequest<T>, где T возвращаемый тип данных для соответсвующей команды, рисунок 3.2.

Text

Description automatically generated

Рисунок 3.2 – Пример команды реализующей интерфейс

Также для обработки команды должен быть написан обработчик этой команды, под названием Handler. Данный обработчик релизовывает интерфейс IRequestHandler<TRequest, TResponce>, где TRequest это соответсвующая команда для выполнения, а TResponce тип данных который будет возвращать данный обработчик и который совпадает с запрашиваемым типом данных в реализованном интерфейсе у команды, рисунок 3.3.

Text

Description automatically generated

Рисунок 3.3 – Пример обработчика команды.

В роли клиентов, тоесть потребителей данных сервисов, выступают настольное приложение Ambulance.Desktop и WEB приложение Ambulance.Web.

Для данных потребителей предоставляются клиенты соответсвующих WEB-сервисов, для вызова соответсвующих точек доступа к данным. Соответсвующие клиенты можно упаковать в подключаемую библиотеку и подключать в разные приложения с помощью управления NuGet пакетами.

Каждый набор приложений, состоящий из Ambulance.Desktop и Ambulance.WEB, представляет собой тенант со своим уникальным названием. Данное название передается с каждым запросов в адресной строке в качестве запрашиваемого параметра. Имя тенанта используется WEB-сервисами для получения адреса соответсвующей базы данных.

Данный WEB-сервис позволяет работать с сущностями вызов, бригада скорой помощи и пациент, CallEntity, AmbulanceBrigadeEntity и PatientEntity, соответсвенно. Доступ к данным осуществляется через контроллеры, рисунок 3.4.

Text

Description automatically generated

Рисунок 3.4 – Структура WEB-сервиса CallAPI

Контроллеры микросервисов написаны с помощью паттерна Mediatr. Для каждой команды написаны соответсвующие обработчики. Для работы с бригадами скорой помощи используется контроллер BrigadeController, рисунок 3.5.

Text

Description automatically generated

Рисунок 3.5 – Список методов BrigadeController

Метод Get, с параметом int, принимает идентификатор бригады и возвращает данные по ней в моделе AmbulanceBrigade, рисунок 3.6, которая включает в себя номер бригады, фамилия, имя и отчество доктора, обоих фельдшеров, санитара и водителя, также возвращается тип бригады и название станции за которой она закреплена.

Text

Description automatically generated

Рисунок 3.6 – Модель AmbulanceBrigade

Метод GetBrigades возвращает коллекцию состоящая из моделей, представленной на рисунке 3.6.

Метод GetBrigadesMonitoringInfo возвращает коллекцию сущностей которые описывают состояние бригад скорой помощи на данный момент. Эта модель содержит в себе номер бригады, ее тип, улицу на которой находится бригада в данный момент, номер дома, уникальный идентификатор и номер вызова, дата и время последнего обновления состояния бригады и название станции к которой она привязана, модель представлена на рисунке 3.7.

Text

Description automatically generated

Рисунок 3.7 – Модель BrigadeMonitoringInfo

Метод GetCalls, с параметорм int, принимает уникальный идентификатор бригады и возвращает коллекцию моделей описывающие вызова которые обслужила конкретная бригада. Данные модели включают в себя номер вызова фамилию, имя и отчество пациента, улицу пациента, номер его дома и квартиры, возраст пациента, первичная причина вызова в виде предварительного диагноза, кто вызывает, дата и время получения вызова, дата и время передачи вызова конкретной бригаде, идентификаторы диспетчера принявший вызов и диспетчера передавший вызов, заметки к вызову и тип вызова, модель представлена на рисунке 3.8.

Graphical user interface, text

Description automatically generated

Рисунок 3.8 – Модель CallOfficeInfo

Метод CreateBrigade, принимающий в параметрах модель запроса на создание сущности бригады, которая представлена на рисунке 3.9 и содержит в себе идентификаторы доктора, обоих фельдшеров, санитара, водителя, номер бригады, ее тип и название станции к кторой привязана бригада, возвращает уникальный идентификатор созданной бригады скорой попомщи.

Text

Description automatically generated

Рисунок 3.9 – Модель CreateBrigadeRequest

Метод DeleteBrigade, принимает в параметрах модель для удаления, а именно изменения статуса бригады скорой попомщи, которая представлена на рисунке 3.10. Данная модель включает в себя идентификатор бригады и время удаления бригады.

Text

Description automatically generated

Рисунок 3.10 – Модель DeleteBrigadeRequest

Метод AssignCallToBrigade, принимает в параметрах модель котрая содержит данные для передачи вызова на выполнение бригаде, модель представлена на рисунке 3.11 и включает в себя идентификатор бригады и вызова, Дата и время передачи вызова бригаде и идентификатор диспетчера, который передал вызов.

A screenshot of a computer

Description automatically generated with medium confidence

Рисунок 3.11 – Модель AssignCallRequest

Метод ReleaseBrigade, принимает в параметрах идентификатор бригады и возвращает статус код успешно выполненной операции. Этот метод используется для завершения выполнения вызова бригадой и перевод статсуса вызова в ожидающий обработки.

Метод ReturnCall, также как и метод ReleaseBrigade, принимает идентификатор бригады и возвращает статсу код успешно выполненной операции. Метод переводит вызов в статус ожидающий передачи бригаде.

Список команд и их обработчиков можно увидеть на рисунке 3.12. Все обработчики созданы на основе паттерна Mediatr и каждый обрабатывают свою уникальную команду.

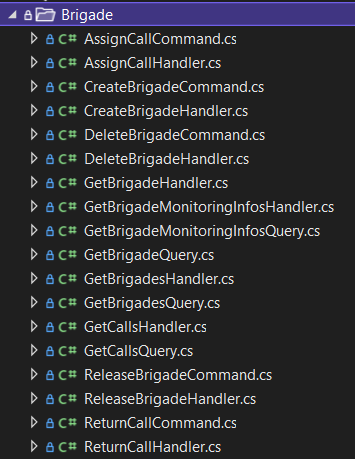


Рисунок 3.12 – Примеры команд и их обработчиков для BrigadeController

Для работы с вызовами используется контроллер CallController, который представлен на рисунке 3.13.

Text

Description automatically generated

Рисунок 3.13 – Список методов контроллера CallController

Метод Get, принимающий в параметрах int, используется для получения вызова по его уникальному идентификатору, содержит номер вызова, фамилию, имя и отчество пациента, улицу пациента, номер его дома и квартиры, возраст пациента, первичная причина вызова в виде предварительного диагноза, кто вызывает, дата и время получения вызова, дата и время передачи вызова конкретной бригаде, идентификаторы диспетчера принявший вызов и диспетчера передавший вызов, заметки к вызову и тип вызова, модель представлена на рисунке 3.8.

Метод GetAccepted, возвращает коллекцию вызовов принятых для обслуживания но еще не переданные пригаде, данные вызыва имеют статус Accepted. Модель из которой состоит коллекция представлена на рисунке 3.8.

Метод GetPending, возвращает коллекцию обслуженных вызовов, данные вызыва имеют статус PendingForProcessing. Данная модель включает в себя полную информацию о вызове и представлена на рисунке 3.14 и 3.15.

Graphical user interface, text, application

Description automatically generated

Рисунок 3.14 – Модель CallFullOfficeInfo

Text

Description automatically generated

Рисунок 3.15 – Продолжение модели CallFullOfficeInfo

Метод Create, принимает в параметрах модель для создания вызова, включающую в себя поля фамилия, имя, отчество, улица, номер дома и квартиры париента, его возраст, идентификатор вызывающего, диспетчера принявшего вызов и даигноза, дата и время принятия, заметки по вызову, и тип вызова. Модель представлена на рисунке 3.16.

Text

Description automatically generated

Рисунок 3.16 – Модель CreateCallRequest

Метод ProcessCall, принимает в параметрах модель содержащую в себя полные данных о вызове для его финальной обработки. Данные модели представлены на рисунке 3.17 и 3.18.

Text

Description automatically generated

Рисунок 3.17 – Модель ProcessCallRequest

A screenshot of a computer

Description automatically generated with medium confidence

Рисунок 3.18 – Продолжение модели ProcessCallRequest

Список команд и их обработчиков можно увидеть на рисунке 3.19. Данные обработчики также построены с использованием паттерна Mediatr.

Text

Description automatically generated

Рисунок 3.19 – Список команд и обработчиков для CallController

Для работы с пациентами используется PatientController, рисунок 3.20.

Text

Description automatically generated

Рисунок 3.20 – Список методов контроллера PatienController

Метод GetPatien, принимает в параметрах идентификатор пациента и возвращает информацию о нем, а именно фамилию, имя, отчество, возраст пациента, идентификатор улицы, номер дома и квартиры пациента, а также его пол. Модель представлена на рисунке 3.21.

Text

Description automatically generated

Рисунок 3.21 – Модель Patient

Метод Update, принимает в параметрах модель содержащую данные о пациенте для обновления. Данная модель представлена на рисунке 3.22.

Text

Description automatically generated

Рисунок 3.22 – Модель UpdatePatienRequest

Cписок команд и обработчиков представлен на рисунке 3.23.

Text

Description automatically generated

Рисунок 3.23 – Список команд и обработчиков для PatienController

Для возможности маштабирования и для того, чтобы потребителями данного сервиса могли быть приложения установленные в разных городах или регионах создана логика позволяющая получить адрес базы данных соответсвующего города или региона. Так как каждый город или регион имеет свой уникальный идентификатор под названием Tenant, создана соответсвующая логика, представленная на рисунке 3.24, позволяющая получить из строки запроса параметр с названием Tenant. Данный класс получает дарес базы данных, соответсвующий полученному идентификатору приложения, и передает ее в контекст базы данных для установки соединения.

Text

Description automatically generated

Рисунок 3.24 – Получение строки подключения к базе

## **WEB-сервис ServiceAPI**

## Данный WEB-сервис используется для получения и обработки сервисной информации такой как получения списков болезней, адресов, вызывающих людей, мест вызова, результатов, лекарств. Также, данный сервис используется для получения списка сотрудников, а именно список докторов, фельдшеров, санитаров, водителей и для получения данных диспетчера.

Для получения данных диспетчера используется контроллер DispatcherController, рисунок 3.24.

Graphical user interface, text, application

Description automatically generated

Рисунок 3.24 – Методы контроллера DispatcherController

Метод GetDispatcher принимает идентификатор диспетчера в параметрах и возвращает модель с полной информацией о диспетчере. Эта информация включает в себя фамилию, имя, отчество диспетчера и его идентификатор. Возвращаемая модель представлена на рисунке 3.25.

Text

Description automatically generated

Рисунок 3.25 – Модель Dispatcher

Медиатр данного контроллера использует команды и их обработчики указанные на рисунке 3.26.

Text

Description automatically generated

Рисунок 3.26 – Команды и обработчики DispatcherController

Для получения данных о сотрудниках используется контроллер EmployeeController, показанный на рисунке 3.27.

Text

Description automatically generated

Рисунок 3.27 – Методы контроллера EmployeeController

Метод GetDoktors, возвращает список моделей докторов, которые включают в себе информацию о сотруднике. Информация состоит из уникального идентификатора, фамилии, имени и отчества. Модель представлена на рисунке 3.28.

Text

Description automatically generated

Рисунок 3.28 – Модель сотрудника скорой помощи

Метод GetMedicalAssistants, возвращает список моделей фельдшеров, которые включают в себе информацию о сотруднике. Информация состоит из уникального идентификатора, фамилии, имени и отчества. Модель представлена на рисунке 3.28.

Метод GetOrderlies, возвращает список моделей санитаров, которые включают в себе информацию о сотруднике. Информация состоит из уникального идентификатора, фамилии, имени и отчества. Модель представлена на рисунке 3.28.

Метод GetDrivers, возвращает список моделей водителей, которые включают в себе информацию о сотруднике. Информация состоит из уникального идентификатора, фамилии, имени и отчества. Модель представлена на рисунке 3.28.

Медиатр данного контроллера использует команды и их обработчики указанные на рисунке 3.29.

Graphical user interface, text, chat or text message

Description automatically generated

Рисунок 3.29 – Команды и их обработчики для EmployeeController

Для получения сервисной информации, такой как список вызывающих, диагнозов, улиц, мест вызова, результатов вызова и лекарств используется контроллер ServiceController, представленный на рисунке 3.30.

Text

Description automatically generated

Рисунок 3.30 – Методы контроллера ServiceController

Метод GetCallers, возвращает список моделей вызывающих, тоесть тип людей, которые сообщают о вызове. Модель содержит в себе уникальный идентификатор и название, подробнее на рисунке 3.31.

Text

Description automatically generated

Рисунок 3.31 – Модель Caller

Метод GetDiagnosis, возвращает список моделей диагнозов. Модель содержит в себе уникальный идентификатор и название, подробнее на рисунке 3.32.

Text

Description automatically generated

Рисунок 3.32 – Модель Diagnosis

Метод GetStreets, возвращает список моделей улиц. Модель содержит в себе уникальный идентификатор и название, подробнее на рисунке 3.33.

Text

Description automatically generated

Рисунок 3.33 – Модель Street

Метод GetPlaces, возвращает список моделей мест вызова. Модель содержит в себе уникальный идентификатор и название, подробнее на рисунке 3.34.

Text

Description automatically generated

Рисунок 3.34 – Модель Place

Метод GetResults, возвращает список моделей результатов вызова. Модель содержит в себе уникальный идентификатор и название, подробнее на рисунке 3.35.

Text

Description automatically generated

Рисунок 3.35 – Модель Result

Метод GetDrugs, возвращает список моделей лекарств. Модель содержит в себе уникальный идентификатор и название и дозу лекарства, подробнее на рисунке 3.36.

A screenshot of a computer

Description automatically generated with medium confidence

Рисунок 3.36 – Модель Drug

Медиатр данного контроллера использует команды и их обработчики указанные на рисунке 3.37.

Text

Description automatically generated

Рисунок 3.37 – Команды и их обработчики контроллера ServiceController

## **Клиенты микросервисов**

Клиенты для микросервисов создаются с целью обеспечения удобного и понятного API для разработки приложения-потребителя соответствующего микросервиса. Каждый клиент можно превратить в подключаемую библиотеку и поставлять отдельно, что позволяет получить доступ к соответствующему WEB-сервису независимо от другого.

В клиенте, который позволяет работать с WEB-сервисом CallAPI, создан класс-кллиент, который содержит классы-ресурсы, с помощью который осуществляется доступ к соответсвующим методам сервиса, рисунок 3.38.

Text

Description automatically generated

Рисунок 3.38 – Класс-клиент для CalAPI

Для каждого контроллера написан интерфейс который описывает адрес метода микросервиса, тип данных и как они будут передаваться. Для контроллера CallController написан ICallEndpoint, для контроллера BrigadeController написан IBrigadeEndpoint, для контроллера PatientController написан IPatientEndpoint. Интерфейсы представлены на рисунках 3.39, 3.40 и 3.41 соответсвенно.

Text

Description automatically generated

Рисунок 3.39 – Методы доступа CallController

Text

Description automatically generated

Рисунок 3.40 – Методы доступа BrigadeController

Text

Description automatically generated

Рисунок 3.41 – Методы доступа PatientController

Все вышеперечисленные интерфейсы при вызове методы, передают уникальный идентификатор приложения в строке запроса, рисунок 3.42, с помощью которого микросервисы будут получать соответсвующую строку подключения к базе данных.

A screenshot of a computer

Description automatically generated with medium confidence

Рисунок 3.42 – Базовый контракт для интерфейсов доступа.

Также соответсвующий класс-клиент, для получения данных, создан для ServiceAPI, рисунок 3.43.

Text

Description automatically generated

Рисунок 3.43 – Класс-клиент для ServiceAPI

Для получения доступа к методам контроллеров созданы интерфейсы, опиывающие адрес и модель запроса к каждому методу контроллеров ServiceController, DispatcherController, EmployeeController, рисунки 3.44, 3.45, 3.46 соответсвенно.

Text

Description automatically generated

Рисунок 3.44 – Методы доступа ServiceController

Text

Description automatically generated

Рисунок 3.45 – Методы доступа DispatcherController

A screenshot of a computer

Description automatically generated with medium confidence

Рисунок 3.46 – Методы доступа EmployeeController

## **Настольное приложение для диспетчеров**

При запуске приложения диспетчера встречает главное меню, с помощью которого можно ввести вызов, авторизоваться как диспетчер в системе, просмотреть принятые вызова и сисок бригад, зарегистрировать бригаду или обработать вызов. Главное меню представлено на рисунке 3.47.

Graphical user interface

Description automatically generated

Рисунок 3.47 – Главное окно приложения

Для авторизации выбирается соответсвующая кнопка и производиться ввод уникального идентификатора диспетчера, рисунок 3.48. После авторизации становятся доступны все пункты главного окна приложения. В противном случае, пользователь не сможет работать с приложением.

Graphical user interface, application

Description automatically generated

Рисунок 3.48 – Авторризация диспетчера

Окно мониторинга работы скорой помощи, рисунок 3.49, позволяет просмотривать принятые вызова, подробные данные о них и списокок бригад которые доступны или заняты обслуживанием вызова. Данное окно позволяет управлять работой бригады, а именно, передавать вызов бригаде, отменять вызов, завершать вызов и просматривать все вызова совершенные бригадой за смену, рисунок 3.50, просматривать данные о вызове на котором находится бригада, рисунок 3.51.

Graphical user interface

Description automatically generated

Рисунок 3.49 – Окно мониторинга работы бригад.

Table

Description automatically generated

Рисунок 3.50 – Список вызовов выполненные бригадой

Graphical user interface, text, application

Description automatically generated

Рисунок 3.51 – Данные вызова выполняемого бригадой

С главного окна можно получить доступ к окну регистрации бригад, рисунок 3.52. На нем нужно ввести ФИО доктора бригады, двух фельдшеров, санитара и водителя бригады. Также через это же окно происходит завершение работы бригады в конце рабочей смены.

Graphical user interface, application

Description automatically generated

Рисунок 3.52 – Регистрация бригад скорой помощи

Окно обработки вызовов позволяет выбрать вызов ожидающий обработки, ввести уточненные данные пациента и заполнить списко лекарств которыми производилось лечение,рисунок 3.53.

Graphical user interface, application

Description automatically generated

Рисунок 3.53 – Окно обработки вызовов

## **WEB–приложение для бригад и персонала**

WEB-приложение должно позволить избавить бригады от бумажной работы. Также приложение позволяет просматривать вызыва и статистику по ним. Для просмотра вызовов выбирается соответсвующий пункт меню и вводтся данные в поля для фильтрации, рисунок 3.54. Осуществлять поиск вызовов можно по номеру вызова, фимилии пациента, диагнозу, периоду времени в которое был принят вызов, цлица, номер дома, номер квартиры пациента и результату. Для удобства просмотра реализована пагинация.

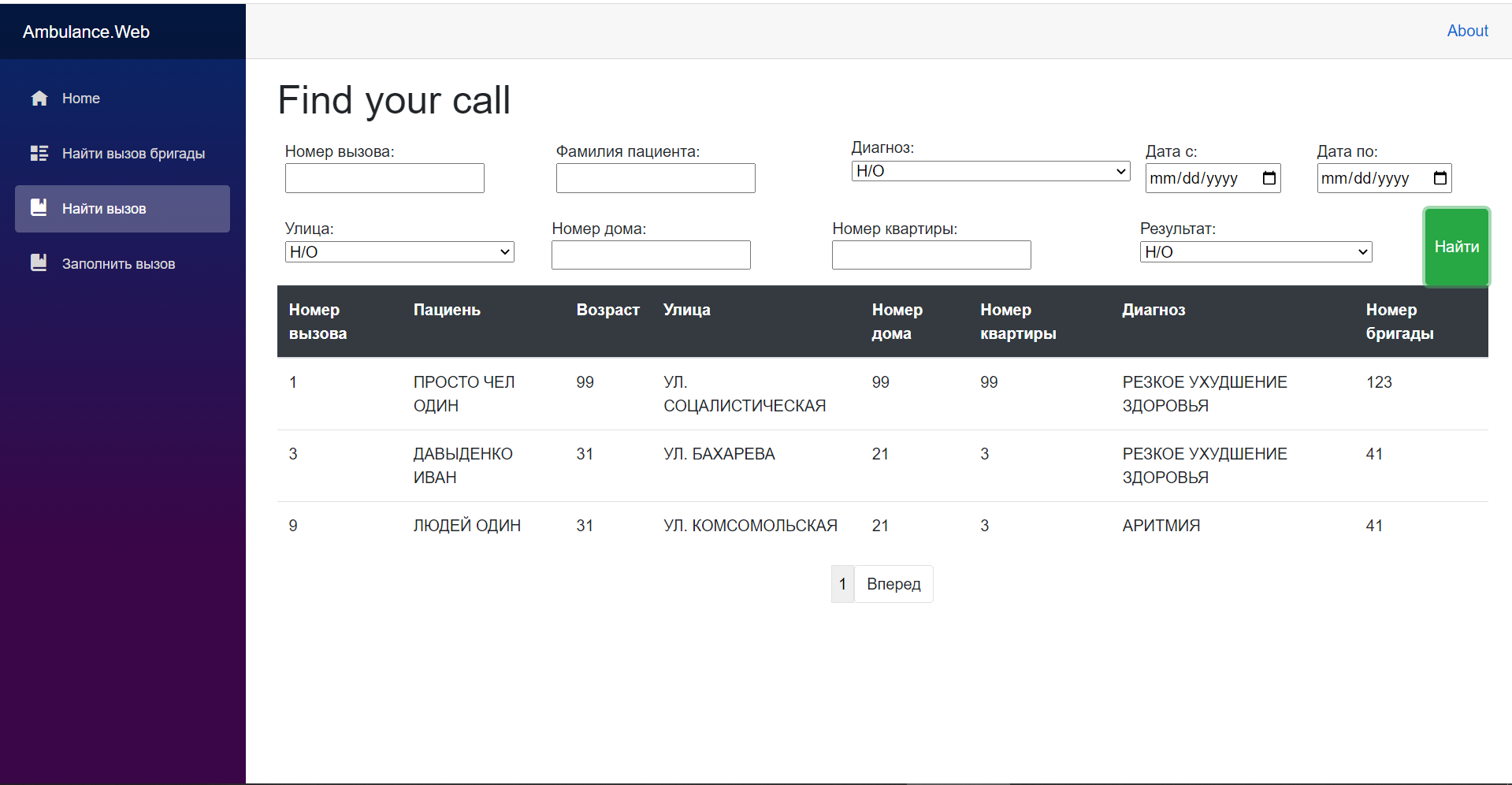


Рисунок 3.54 – Просмотр обработанных вызовов

Для сотрудников бригады реализован пункт меню в котором можно занести всю дополнительную подробную информацию по вызову который обслуживает бригада. Данный пункт меню содержит информацию о состоянии пациента на момент вызова, при транспортировке, после транспортировки, состояние его органов и самочувствия. Для удобства выводится краткая информацию по вызову в верхней части экрана. Окно представлено на рисунке 3.55.

A screenshot of a computer

Description automatically generated

Рисунок 3.55 – Пример заполнения дополнительной инормации по вызову

# **ТЕСТИРОВАНИЕ, ВЕРИФИКАЦИЯ И ВАЛИДАЦИЯ**

Тестирование программного продукта является важной частью разработка программного обеспечения. Программа которая не функционирует согласно требованиям или функционирует но основной процесс, ради которого разрабатывалось приложение, не работает, не может конкурировать и быть востребованным на рынке.

Для проверки работоспособности программы создаются тест-кейсы с описанием шагов по воспроизведению теста и описанием ожидаемого поведения программы на данные шаги. Особенно важно создать тест-кейсы для критических путей программы, тоесть для тех функций которые несут в себе основной смысл разрабатываемого продукта.

Верификация – это процесс оценки того, насколько система (программа, устройство) по итогам некоторого этапа ее разработки соответствует условиям, заданным в начале этапа.

Валидация – процесс оценки того, насколько система (программа, устройство) соответствует требованиям по ее назначению.

Основное правило разрабатываемого настольного приложения – с приложением должен работать диспетчер. Тоесть для работы с приложением нужно авторизоваться как диспетчер, в противном случае пункты меню не будут доступны. Тест-кейсы описаны в таблице 4.1 и 4.2.

Таблица 4.1 – Негативные тейст-кейсы для главного окна Desktop приложения.

|  |  |  |
| --- | --- | --- |
| **Описание** | **Шаги вопроизведения** | **Ожидаемый результат** |
| Попытка ввести вызов | 1. Загрузка Desktop приложения. 2. Заполнение полей вызова. 3. Нажатие на кнопку «Пинять». | Появляется новое окно с сообщением «Вы не зарегистрировались как диспетчер» |
| Попытка открытия окна с принятыми вызывами | 1. Загрузка Desktop приложения. 2. Заполнение полей вызова. 3. Нажатие на кнопку «Передача вызова». |
| Попытка открытия окна обработки вызовов | 1. Загрузка Desktop приложения. 2. Заполнение полей вызова. 3. Нажатие на кнопку «Обработка». |
| Попытка открытия окна для регистрации бригад | 1. Загрузка Desktop приложения. 2. Заполнение полей вызова. 3. Нажатие на кнопку «Данные». |

На рисунке 4.1 представлен результат выполнения тесткейсов описанных в таблице 4.1.

Graphical user interface

Description automatically generated

Рисунок 4.1 – Сообщение для неавторизованоого пользователя

Таблица 4.2 – Положительные тест-кейсы для главного окна Desktop приложения

|  |  |  |
| --- | --- | --- |
| **Описание** | **Шаги вопроизведения** | **Ожидаемый результат** |
| Попытка ввести вызов | 1. Загрузка Desktop приложения. 2. Авторизация диспетчером. 3. Заполнение полей вызова. 4. Нажатие на кнопку «Пинять». | Появляется новое окно с надписью о принятии вызова и его номером. |
| Попытка открытия окна с принятыми вызывами | 1. Загрузка Desktop приложения. 2. Авторизация диспетчером. 3. Заполнение полей вызова. 4. Нажатие на кнопку «Передача вызова» | Появляется новое окно с принятыми вызывами и списком бригад |
| Попытка открытия окна обработки вызовов | 1. Загрузка Desktop приложения. 2. Авторизация диспетчером. 3. Заполнение полей вызова. 4. Нажатие на кнопку «Обработка» | Появляется окно для обработки вызова со списком вызовов ожидающие обработки |
| Попытка открытия окна для регистрации бригад | 1. Загрузка Desktop приложения. 2. Авторизация диспетчером. 3. Заполнение полей вызова. 4. Нажатие на кнопку «Данные» | Появляется новое окно для ввобда и удаления бригад. |

Пример авторизации и результат одного из тест кейсов представлены на рисунках 4.2 и 4.3 соответсвенно.

Graphical user interface

Description automatically generated

Рисунок 4.2 – Авторизация диспетчера

A screenshot of a computer

Description automatically generated

Рисунок 4.3 – Открытие окна регистрации бригад авторизованным пользователемСписок использованных источников

Изм.

Лист

№ докум.

Подпись

Дата

Лист

26

Дипломная работа

1. Фриман, Адам Pro ASP.NET Core MVC 2 / Адам Фриман. - Лондон: Apress. 2017. – 1013 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. Обзор Windows Forms.: Официальная документация. – Электрон. данные. – Режим доступа: <https://docs.microsoft.com/ru-ru/dotnet/framework/winforms/windows-forms-overview>. – Дата доступа: 05.04.2022.
5. Общие сведения о WPF: Официальная документация. – Электрон. данные. – Режим доступа: <https://docs.microsoft.com/ru-ru/dotnet/framework/wpf/introduction-to-wpf>. – Дата доступа: 05.04.2022.