ИИТ БГУИР

Факультет повышения квалификации и переподготовки

Кафедра МПСС

Пояснительная записка к курсовому проекту

по дисциплине

«Средства визуального программирования приложений»

на тему: «Программное средство «Информационная система управления услугами поликлиники»

**Выполнил:**

слушатель гр. 80321

Денисов А.Ю.

|  |  |  |
| --- | --- | --- |
| **Руководитель:** | **Проверил:** | |
| Прибыльская Н.М. | к.т.н., доцент Боброва Н. Л. | |
|  | Оценка: |  |
|  |  |  |
|  |  | (подпись) |
|  |  |  |
|  |  | (дата) |

Минск 2019

**Содержание**

[Введение 3](#_Toc24414263)

[1. Описание функциональных требований 5](#_Toc24414266)

[2. Моделирование структуры приложения 7](#_Toc24414267)

[2.1. Описание структуры базы данных 7](#_Toc24414268)

[2.2. UML диаграмма классов 8](#_Toc24414269)

[3. Описание реализации приложения 12](#_Toc24414270)

[3.1. Описание кода программы 12](#_Toc24414271)

[3.2. Схемы основных алгоритмов, реализующих бизнес-логику программной системы 13](#_Toc24414272)

[4. Тестирование программы 17](#_Toc24414273)

[5. Исходный код программы 20](#_Toc24414274)

[Заключение 28](#_Toc24414275)

[Список использованной литературы 29](#_Toc24414276)

# **Введение**

## **Область применения приложения**

*Постановка задачи:* разработать информационную систему управления услугами поликлиники (далее – программная система). Посредством программной системы планируется вести учет предоставляемых медицинских услуг, а также хранить персональные данные врачей и пациентов организации.

*Цель приложения*: учет услуг поликлиники.

*Дополнительная цель*: хранение контактных данных врачей и пациентов поликлиники.

*Область применения*: медицинские учреждения.

*Пользователь*: администратор программной системы.

*Основные виды работ*, проводимые пользователем посредством разрабатываемой программной системы:

1. редактирование (добавление/изменение/удаление) хранимых в программной системе персональных данных врачей;
2. редактирование(добавление/изменение/удаление) хранимой в программной системе информации об оказываемых услугах;
3. редактирование (добавление/изменение/удаление) хранимых в программной системе персональных данных пациентов;
4. формирование листа посещения пациента;
5. просмотр аналитической информации о предоставленных услугах по медицинским направлениям в течение отчетного временного интервала в форме столбчатых и круговых диаграмм;
6. экспорт и импорт данных в формате xml.

## **Особенности средств разработки**

Разработанное приложение относится к типу клиентского десктопного программного обеспечения, работающего под управлением Windows не ниже версии Vista SP2 с поддержкой платформы .NET 4.5.

В качестве программной платформы разработки приложения в курсовом проекте использован Microsoft Visual Studio Community 2017.

Приложение разработано на основе технологии WPF с использованием языка C# 5.0. Выбор данной технологии предоставляет возможность использования в качестве базовой графической технологии DirectX вместо GDI, благодаря чему достигается более высокая производительность приложений WPF, чем WinForms, а также возможность реализации некоторых дополнительных визуальных эффектов (например, эффект прозрачности элементов управления).

При проектировании приложения использовался подход Database First. В качестве системы управления базами данных использовалась Microsoft SQL Server. Взаимодействие базы данных и разработанного приложения осуществляется с использованием технологии Entity Framework (версия 6.2.0).

При работе с информацией базы данных приложения для доступа к данным использовалась технология ORM LINQ to Entities, для обеспечения импорта и экспорта данных - технология LINQ to XML.

# **1. Описание функциональных требований**

Разрабатываемое приложение будет использоваться одним пользователем - работником регистратуры поликлиники (далее - «администратор программной системы»), который должен выполнять следующие функции:

1. работать с персональными данными пациентов (далее - анкетные данные пациента):

* вносить в базу данных информацию о новом пациенте;
* изменять анкетные данные пациентов;
* удалять анкетные данные пациентов;

2. работать с листами посещений пациентов:

* вносить в базу данных лист посещения, содержащий информацию о заказанной(ых) пациентом услуги(ах);
* изменять данные листа посещений;
* удалять лист посещений;

3. работать с информацией о предоставляемых медицинских услугах:

* вносить в базу данных информацию о новых медицинских услугах;
* изменять данные о предоставляемых медицинских услугах;
* удалять записи о медицинских услугах;

4. работать с персональными данными врачей (далее - анкетные данные врача):

* вносить в базу данных информацию о новом враче;
* изменять анкетные данные врача;
* удалять анкетные данные врача;

5. просматривать агрегированную и представленную в графической форме информацию о предоставленных услугах по медицинским направлениям в течение отчетного временного интервала;

6. экспорт / импорт информации программной системы в формате XML

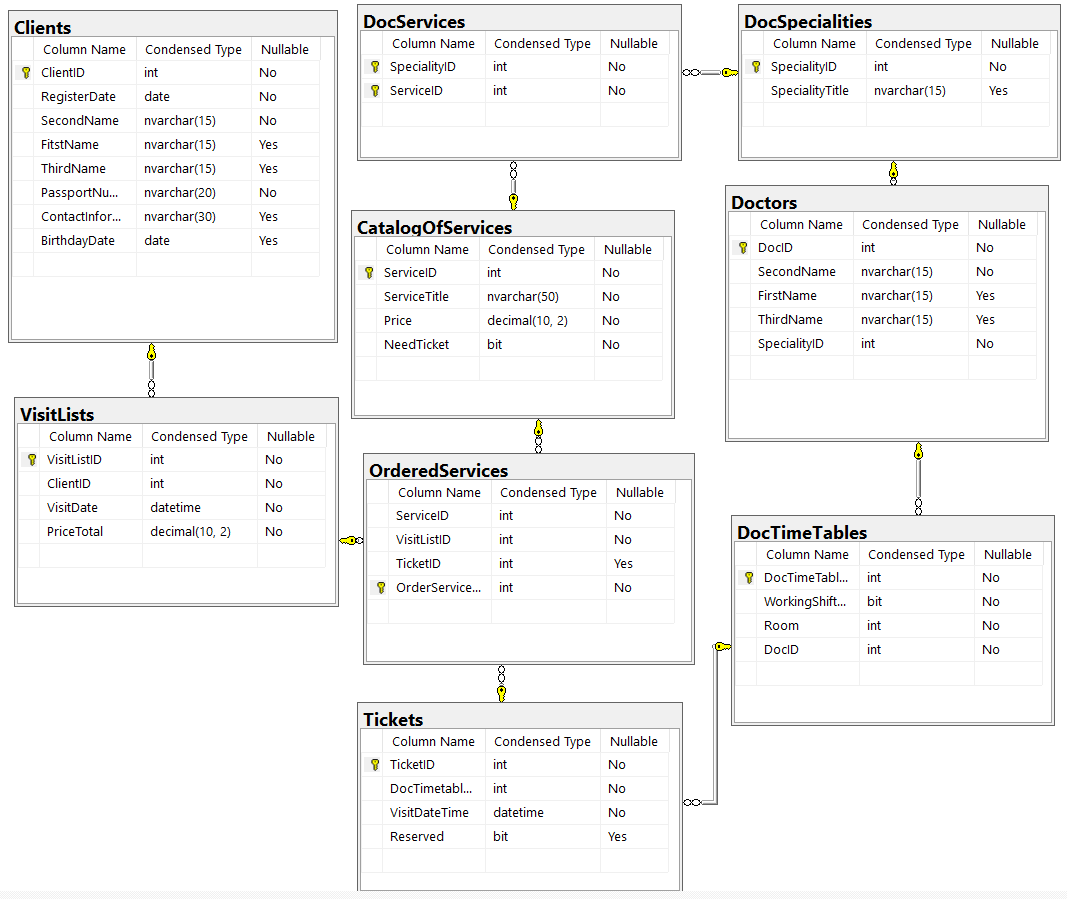
Описание функциональных требований к программе в формате UML 2.0 представлены на рисунке 1.



**Рисунок 1.** Диаграмма вариантов использования программной системы актором «администратор»

# **2. Моделирование структуры приложения**

## **2.1. Описание структуры базы данных**



**Рисунок 2.** Структура базы данных приложения

Результатом создания датологической модели данных стало выделение девяти отношений, приведенных к третьей нормальной форме. При построении модели выделены два отношения для реализации связей *многие-ко-многим*:

- отношение DocServices, которое связывает отношения CatalogOfServices и DocSpecialities;

- отношение OrderedService (с суррогатным первичным ключом OrderedServiceID), которое связывает три отношения: VisitLists, CatalogOfServices и Tickets.

## **2.2. UML диаграмма классов**

С учетом поставленных задач в качестве архитектуры системы выбрана модель двухслойной архитектуры: все поведение программной системы вынесено в слой сервисов (реализованных как набор пользовательских элементов (User Controls)), который отделен от модели предметной области.

Отказ от реализации отдельных шаблонов проектирования, изученных в рамках лабораторной работы № 9, обусловлен следующими причинами:

- отсутствие соответствующей задачи в рамках задания на выполнение курсового проекта;

- неоправданное усложнение процесса написания и сопровождения программы. Так, при реализации изученных шаблонов помимо выделенных восьми классов предметной области (см. рисунок 4 и РДКП) общее количество вспомогательных классов увеличивается минимум на 16 (на каждый класс предметной области – по одному классу-репозиторию, а также классу, отвечающему за бизнес логику), без учета дополнительных специальных интерфейсов и специальных классов-сервисов.

Следуя выбранного подходу выделено четыре группы классов:

- классы предметной области имеют только конструкторы и свойства для доступа к данным. Таким образом, данные классы реализуют только отношения (связи) между сущностями предметной области, но собственным поведением (т. е. методами для работы с данными сущностями) практически не обладают;

- классы пользовательского интерфейса (в рамках оконных классов (основные и диалоговые окна), а также классов пользовательских управляющих элементов (user controls));

- служебный класс, отвечающие за формирование журнала использования приложения и выявление неперехваченных ошибок LogSaver;

- три служебных класса, которые инкапсулируют рисования диаграмм.

Логика работа с классами предметной области вынесена преимущественно в обработчики событий пользовательского интерфейса.

В связи с этим можно представить три диаграммы классов программной системы:

сокращенная диаграмма классов пользовательского интерфейса и служебных классов (см. рисунок 3);

сокращенная диаграмма классов предметной области по результатам подхода Database First с использованием EntityFrameWork (см. рисунок 4);

полная диаграмма классов классов пользовательского интерфейса и служебных классов (см. РДКП 80321.1);

полная диаграмма классов предметной области (см. РДКП 80321.2).



**Рисунок 3.** Диаграмма классов пользовательского интерфейса и служебных классов (сокращенная)

На рисунке 3 изображены отношения между 17 классами пользовательского интерфейса и служебными классами программной системы.

*Класс App* (наследует от класса Application), в котором определены глобальные стили оформления оконного интерфейса, который передает управление главного окну приложения (класс MainWindow), а также в котором осуществлена подписка на ряд событий с целью ведения журнала использования программы.

*Класс LogSaver* инкапсулирует логику обработчиков событий уровня приложения: записи в журнал факта и времения открытия и закрытия приложения, а также описание генерируемого неперехваченного исключения (DispatcherUnhandledException event).

*Класс MainWindow* – предлагает пользователю визуальный интерфейс с выбором тематического модуля (UserControl) для последующей работы: получение и редактирование данных о клиентах (ClientsUserControl), заказах (VisiListsUserControl), врачах (DoctorsUserControl) и оказываемых услугах (DoctorUserControl). Также данный класс позволяет вызвать диалоговое окно со сведениями о программе (класс AboutWindow) и модуль рисования диаграмм для получения обобщенной статики оказанных услуг. Так как MainWindow является родительским окном, то с указанными модулями оно объединено таким видом связи как *композиция* – то есть при закрытии главного окна дочерние окна также прекратят свою работу.

*Наследники класса UserControl* инкапсулируют логику обработчиков событий при нажатии пользователем на конкретные элементы пользовательского интерфейса и напрямую обрабатывают пользовательские запросы по отображению, сохранению, загрузке, изменению и удалению данных предметной области. Каждый наследник класса UserControl также находится в отношении *композии* с диалоговыми окнами – вспомогательными окнами, которые упрощают процедуру редактирования пользователем сведений о каком-либо объекте.

Исключением является *класс ChartsUserControl*, который с использованием специально разработанных классов Bar, BarChart и PieChart позволяет визуализировать сведения из базы данных с использованием столбчатых и круговых диаграмм.

Также на рисунке 4 представлена сокращенная диаграмма классов предметной области (подробная диаграмма классов – см. РДКП). Всего выделено восемь таких классов. Для сохранения возможности эффективно взаимодействовать между собой в каждом классе есть так называемые *навигационные свойства*, которые представляют из себя generic-коллекции ICollection. В результате все ассоциации между классами предметной области являются *двунаправленными ассоциациями*.



**Рисунок 4\*.** Диаграмма классов предметной области (сокращенная)

*\* - полная диаграмма классов предметном области изображена в РДКП*

# **3. Описание реализации приложения**

## **3.1. Описание кода программы**

При разработке программной системы использовался подход «database first». База данных включает 9 таблиц (см. рисунок 2), в программной системе на основе LINQ to Entities реализована возможность просмотра и редактирования содержимого данной базы. Также реализована возможность экспорта / импорта таблиц базы данных в XML-файлы посредством технологии LINQ to XML.

В программной системе реализована многослойная архитектура, в рамках которой выделены

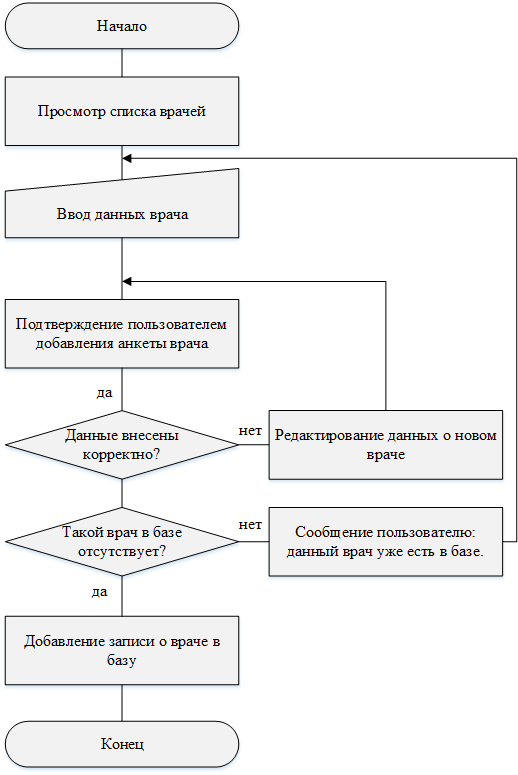
1. уровень доступа к базе данных;
2. уровень бизнес-логики.

В соответствии с поставленным заданием курсового проекта, программная система содержит:

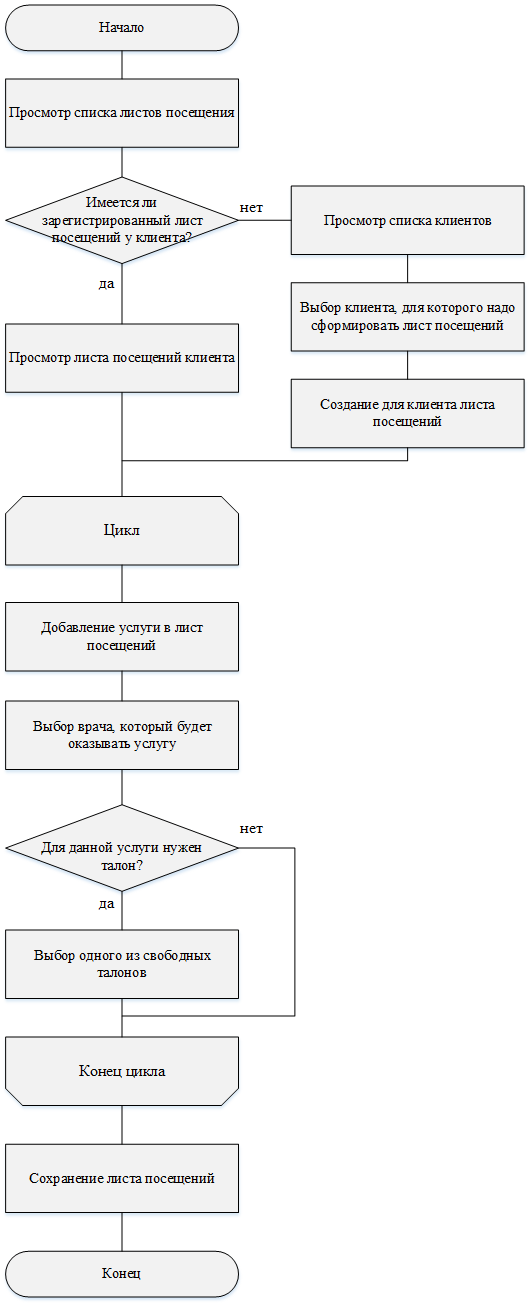
1. меню (включает 3 раздела «На главную», «Аналитическая информация» и «О программе»);
2. панель инструментов (реализована в классе ChartsUserControl для изменения внешнего вида круговой и столбчатой диаграммы);
3. строка статуса (используется для отображения клиента, для которого формируется лист посещений);
4. пользовательских элементов и 6 диалоговых окон, реализующих необходимый функционал для работы с базой данных;
5. реализован пользовательский элемент, позволяющий строить как столбчатые, так и круговые диаграммы.

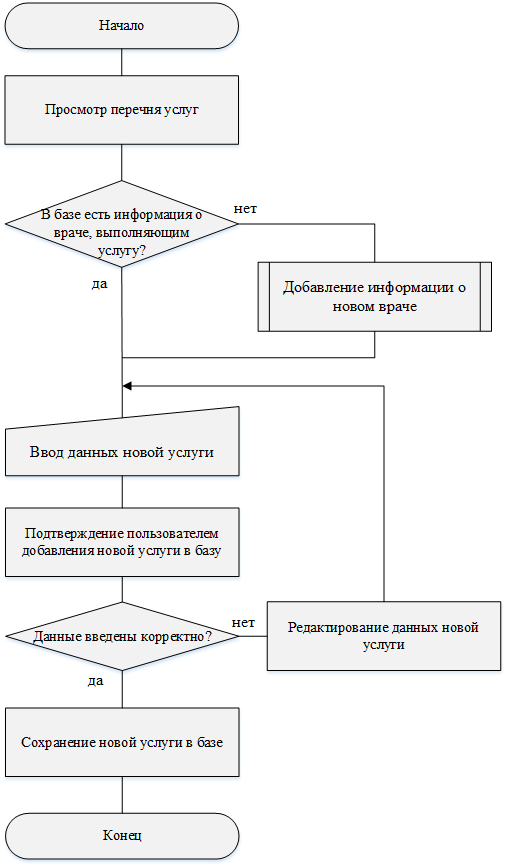
## **3.2. Схемы основных алгоритмов, реализующих бизнес-логику программной системы**

**Рисунок 4.** Добавление сведений о новом клиенте в базу данных



**Рисунок 5.** Добавление сведений о новом враче в базу данных

**Рисунок 6.** Оформление листа посещений клиента



**Рисунок 7.** Добавление сведений о новой медицинской услуге в базу данных.

# **4. Тестирование программы**

|  |  |
| --- | --- |
| **5.1 Регистрация нового клиента.** | |
|  | Обработка ситуации ввода некорректной информации (номер паспорта) |
| **5.2 Регистрация нового врача.** | |
|  |  |
| **5.3 Формирование листа посещений при неуказанном клиенте** | |
|  | Сообщение о необходимости выбрать клиента из списка клиентов |
| **5.4 Формирование листа посещений при указанном клиенте** | |
|  |  |
| **5.5 Экспорт и импорт данных клиентов в Xml** | |
|  | Экспорт данных на рабочий стол |
|  | Все клиенты были удалены. Импортируется файл с данными клиентов |
|  | Сведения обо всех клиентов посредством импорта файла XML вновь внесены с базу данных |
| **5.6 Добавление новой услуги** | |
|  |  |
| **5.7 Создание круговой диаграммы** | |
|  |  |
| **5.8 Создание столбчатой диаграммы** | |
|  |  |

# **5. Исходный код программы**

**(на примере реализации класса VisitListUserControl)**

***Реализация интерфейса на языке XAML***

<UserControl x:Class="DenisovArt\_Kurs.UserControls.VisitListsUserControl"

xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"

xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"

xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"

xmlns:d="http://schemas.microsoft.com/expression/blend/2008"

xmlns:local="clr-namespace:DenisovArt\_Kurs.UserControls"

mc:Ignorable="d"

Background="White"

Style="{StaticResource StandartUseControl}">

<Grid Height="610" Width="800">

<Grid.RowDefinitions>

<RowDefinition Height="25"></RowDefinition>

<RowDefinition Height="5"></RowDefinition>

<RowDefinition></RowDefinition>

</Grid.RowDefinitions>

<Grid.ColumnDefinitions>

<ColumnDefinition Width="250"></ColumnDefinition>

<ColumnDefinition></ColumnDefinition>

</Grid.ColumnDefinitions>

<Rectangle Grid.Row="0" Grid.RowSpan="2" Grid.Column="0" Grid.ColumnSpan="2"

Fill="MidnightBlue" Stretch="Fill" />

<Image Grid.Row="2" Grid.Column="0" Grid.ColumnSpan="2"

Source="/Images/UC\_visit.jpg" Stretch="Fill"

Opacity="0.6"/>

<TextBlock Grid.Row="0" Grid.Column="0" Grid.ColumnSpan="2"

Foreground="White" Style="{StaticResource HeaderUseControl}"

Margin="0,0,0,0">Листы посещений</TextBlock>

<Separator Grid.Row="1" Grid.ColumnSpan="2" BorderBrush="Gray"></Separator>

<StackPanel Grid.Row="2" Grid.Column="0" >

<GroupBox Header=" П О И С К " Margin="0,5,0,10">

<StackPanel>

<TextBlock>(по любому полю)</TextBlock>

<Label FontWeight="Bold">Номер заказа:</Label>

<Border Style="{StaticResource OvalBorder}">

<TextBox Name="tbxOrderId" Style ="{StaticResource ValidationFailed}">

<TextBox.Text>

<Binding Path="MyVisitListID" ValidatesOnExceptions="True"

UpdateSourceTrigger="PropertyChanged"/>

</TextBox.Text>

</TextBox>

</Border>

<Label FontWeight="Bold">Номер карточки клиента:</Label>

<Border Style="{StaticResource OvalBorder}">

<TextBox Name="tbxClientId" Style ="{StaticResource ValidationFailed}">

<TextBox.Text>

<Binding Path="MyClientID" ValidatesOnExceptions="True"

UpdateSourceTrigger="PropertyChanged"/>

</TextBox.Text>

</TextBox>

</Border>

<Label FontWeight="Bold">Фамилия клиента:</Label>

<Border Style="{StaticResource OvalBorder}">

<TextBox Name="tbxClientName" Style ="{StaticResource ValidationFailed}">

<TextBox.Text>

<Binding Path="MYSecondName" ValidatesOnExceptions="True"

UpdateSourceTrigger="PropertyChanged"/>

</TextBox.Text>

</TextBox>

</Border>

<Button Name="Find\_Button" Content="Найти" Click="Find\_Button\_Click"/>

<Button Name="Reboot\_Button" Content="Сброс" Click="Reboot\_Button\_Click"/>

</StackPanel>

</GroupBox>

<GroupBox Header="Ф У Н К Ц И И " Margin="0,5,0,10">

<StackPanel>

<Button Name="Add\_Button" Content="Добавить" Click="Add\_Button\_Click"/>

<Button Name="Edit\_Button" Content="Редактировать" Click="Edit\_Button\_Click"/>

<Button Name="Delete\_Button" Content="Удалить" Click="Delete\_Button\_Click"/>

</StackPanel>

</GroupBox>

<GroupBox Header="X M L " Margin="0,5,0,10">

<StackPanel>

<Button Name="Export" Content="Экспорт" Click="Export\_Click"/>

<Button Name="Import" Content="Импорт" Click="Import\_Click"/>

</StackPanel>

</GroupBox>

</StackPanel>

<DataGrid

Name="dgrid"

ItemsSource="{Binding}"

Grid.Row="2" Grid.Column="1"

AutoGenerateColumns="False"

Margin="15,15,15,35"

IsReadOnly="True">

<DataGrid.Columns>

<DataGridTextColumn Header="ID заказа" Binding="{Binding VisitListID}"/>

<DataGridTextColumn Header="ID клиента" Binding="{Binding ClientID}"/>

<DataGridTextColumn Header="Фамилия клиента" Binding="{Binding ClintName}"/>

<DataGridTextColumn Header="Дата заказа" Binding="{Binding VisitDate}"/>

<DataGridTextColumn Header="Стоимость" Binding="{Binding PriceTotal}"/>

</DataGrid.Columns>

</DataGrid>

</Grid>

</UserControl>

***Реализация класса на языке С#***

using System;

using System.Collections.Generic;

using System.Linq;

using System.Windows;

using System.Windows.Media;

using System.Windows.Controls;

using System.Collections.ObjectModel;

using DenisovArt\_Kurs.DialogWindows;

using System.Xml.Linq;

namespace DenisovArt\_Kurs.UserControls

{

public partial class VisitListsUserControl : UserControl

{

ProjectEntitiesContext context;

MainWindow owner;

IQueryable<VisitList> visitsDB;

ObservableCollection<VisitList> visitObsColl;

ObservableCollection<VisitList> visitSearchColl;

public VisitListsUserControl(MainWindow mW)

{

owner = mW;

InitializeComponent();

dgrid.DataContext = ContextInitialization();

}

private ObservableCollection<VisitList> ContextInitialization()

{

context = new ProjectEntitiesContext();

visitsDB = from c in context.VisitLists

select c;

visitObsColl = new ObservableCollection<VisitList>();

foreach (VisitList c in visitsDB)

{

visitObsColl.Add(c);

}

return visitObsColl;

}

private void Find\_Button\_Click(object sender, RoutedEventArgs e)

{

try

{

List<VisitList> vList = new List<VisitList>();

if (String.IsNullOrEmpty(tbxOrderId.Text) == false)

{

foreach (VisitList c in visitObsColl)

{

if (c.VisitListID.ToString() == tbxOrderId.Text)

{

vList.Add(c);

}

}

FindAndSelect(vList);

return;

}

if (String.IsNullOrEmpty(tbxClientId.Text) == false)

{

foreach (VisitList c in visitObsColl)

{

if (c.ClientID.ToString() == tbxClientId.Text)

{

vList.Add(c);

}

}

FindAndSelect(vList);

return;

}

if (String.IsNullOrEmpty(tbxClientName.Text) == false)

{

foreach (VisitList c in visitObsColl)

{

if (c.ClintName == tbxClientName.Text)

{

vList.Add(c);

}

}

FindAndSelect(vList);

return;

}

}

catch

{ }

}

private void FindAndSelect(List<VisitList> vList)

{

visitSearchColl = new ObservableCollection<VisitList>();

foreach (VisitList c in vList)

{

visitSearchColl.Add(c);

}

dgrid.DataContext = visitSearchColl;

}

private void Reboot\_Button\_Click(object sender, RoutedEventArgs e)

{

if (visitSearchColl != null)

{

visitSearchColl.Clear();

dgrid.DataContext = visitObsColl;

tbxClientId.Text = tbxClientName.Text = tbxOrderId.Text = String.Empty;

}

}

private void Edit\_Button\_Click(object sender, RoutedEventArgs e)

{

//Check if visit list selected

if(dgrid.SelectedItem == null) { MessageBox.Show("Для редактирования надо указать лист посещения."); return; }

VisitList v = (VisitList)dgrid.SelectedItem;

//Set a client info in the status bar

owner.StatusBar\_ClientID.Text = v.Client.ClientID.ToString();

owner.StatusBar\_ClientFullName.Text = v.Client.SecondName + " " + v.Client.FitstName + " " + v.Client.ThirdName;

owner.StatusBar\_ClientID.Style = owner.StatusBar\_ClientFullName.Style = (Style)FindResource("StatusBar\_ACTIVE");

//Create DialogWindow for visitList edition

VisitListsEditWindow vW = new VisitListsEditWindow(v.Client, v, context);

if (vW.ShowDialog() == false) return;

context.SaveChanges();

dgrid.DataContext = null;

dgrid.DataContext = visitObsColl;

}

private void Add\_Button\_Click(object sender, RoutedEventArgs e)

{

//Check is a client selected

try { Convert.ToInt32(owner.StatusBar\_ClientID.Text); }

catch { MessageBox.Show("Для оформления заказа выберите клиента из базы данных КЛИЕНТЫ."); return; }

Int32 check = Convert.ToInt32(owner.StatusBar\_ClientID.Text);

Client cSelected = (from c in context.Clients

where c.ClientID == check

select c).First();

VisitList vl = new VisitList();

VisitListsEditWindow vW = new VisitListsEditWindow(cSelected, vl, context);

if (vW.ShowDialog() == false) return;

context.VisitLists.Add(vl);

context.SaveChanges();

visitObsColl.Add(vl);

}

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

{

if (dgrid.SelectedItem == null) { MessageBox.Show("Выберите лист для удаления"); return; }

List<VisitList> vList = dgrid.SelectedItems.Cast<VisitList>().ToList();

foreach (VisitList vl in vList)

{

foreach (OrderedService o in vl.OrderedServices)

{

if(o.TicketID != null) o.Ticket.Reserved = false;

}

context.VisitLists.Remove(vl);

visitObsColl.Remove(vl);

}

context.SaveChanges();

}

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

{

Microsoft.Win32.SaveFileDialog dlg = new Microsoft.Win32.SaveFileDialog();

dlg.FileName = "Orders"; // Default file name

dlg.DefaultExt = ".xml"; // Default file extension

dlg.Filter = "XML documents (.xml)|\*.xml"; // Filter files by extension

if (dlg.ShowDialog() == false) return;

try

{

//check for correct file name

XDocument.Load(dlg.FileName);

}

catch { }

IEnumerable<VisitList> vlColl = context.VisitLists;

IEnumerable<OrderedService> osTT = context.OrderedServices;

IEnumerable<Ticket> cTick = context.Tickets;

XDocument doc = new XDocument(

new XElement("Orders",

new XElement("VisitLists",

from c in vlColl

select c.VisitListPassportXml()),

new XElement("OrderedServices",

from tt in osTT

select tt.OrderedServPassportXml()),

new XElement("Tickets",

from t in cTick

where t.Reserved == true

select t.TickPassportXml())));

doc.Save(dlg.FileName);

}

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

{

Microsoft.Win32.OpenFileDialog dlg = new Microsoft.Win32.OpenFileDialog();

dlg.FileName = "Orders"; // Default file name

dlg.DefaultExt = ".xml"; // Default file extension

dlg.Filter = "XML documents (.xml)|\*.xml"; // Filter files by extension

if (dlg.ShowDialog() == false) return;

try

{

//check for file existence

XDocument test = XDocument.Load(dlg.FileName);

}

catch { }

XDocument xDoc = XDocument.Load(dlg.FileName);

IEnumerable<XElement> mainContainer = xDoc.Descendants();

List<XElement> elementContainer = new List<XElement>();

//Export collection from mainContainer to elemntContainer (can routing by index)

foreach (XElement elem in mainContainer)

{

elementContainer.Add(elem);

}

List<VisitList> vlColl = new List<VisitList>();

List<OrderedService> osColl = new List<OrderedService>();

List<Ticket> tickColl = new List<Ticket>();

//Find and create new VisitLists, OrderedServices and Tickets from XML-file

for (Int32 i = 0; i < elementContainer.Count; i++)

{

VisitList vNew;

OrderedService osNew;

Ticket tick;

if (elementContainer[i].Name.ToString() == "VisitList")

{

vNew = new VisitList

{

VisitListID = Convert.ToInt32(elementContainer[i + 1].Value.ToString()),

ClientID = Convert.ToInt32(elementContainer[i + 2].Value.ToString()),

VisitDate = Convert.ToDateTime(elementContainer[i + 3].Value.ToString()),

PriceTotal = Convert.ToDecimal(elementContainer[i + 4].Value.ToString()),

};

vlColl.Add(vNew);

}

if (elementContainer[i].Name.ToString() == "OrderedService")

{

string x = elementContainer[i + 3].Value.ToString();

//If ticket != null

if (String.IsNullOrEmpty(x) == false)

{

osNew = new OrderedService

{

ServiceID = Convert.ToInt32(elementContainer[i + 1].Value),

VisitListID = Convert.ToInt32(elementContainer[i + 2].Value),

TicketID = Convert.ToInt32(elementContainer[i + 3].Value)

};

osColl.Add(osNew);

}

else

{

osNew = new OrderedService

{

ServiceID = Convert.ToInt32(elementContainer[i + 1].Value),

VisitListID = Convert.ToInt32(elementContainer[i + 2].Value),

TicketID = null

};

osColl.Add(osNew);

osColl.Add(osNew);

}

}

if (elementContainer[i].Name.ToString() == "Ticket")

{

tick = new Ticket

{

TicketID = Convert.ToInt32(elementContainer[i + 1].Value),

DocTimetableID = Convert.ToInt32(elementContainer[i + 2].Value),

VisitDateTime = Convert.ToDateTime(elementContainer[i + 3].Value),

Reserved = Convert.ToBoolean(elementContainer[i + 4].Value)

};

tickColl.Add(tick);

}

}

//Add Tickets to OrderedServices

for (Int32 i = 0; i < osColl.Count; i++)

{

foreach (Ticket tick in tickColl)

{

if (tick.TicketID == osColl[i].TicketID)

{

osColl[i].Ticket = tick;

}

}

}

//Add OrderedServices to VisitLists

for (Int32 i = 0; i < vlColl.Count; i++)

{

foreach (OrderedService tt in osColl)

{

if (tt.VisitListID == vlColl[i].VisitListID)

{

vlColl[i].OrderedServices.Add(tt);

}

}

}

//Add VisitLists to the Context

foreach (VisitList v in vlColl)

{

if (context.VisitLists.Where(d => d.VisitListID == v.VisitListID).Any<VisitList>() == false)

{

context.VisitLists.Add(v);

context.SaveChanges();

visitObsColl.Add(v);

}

}

}

}

}

# **Заключение**

В соответствии с поставленным заданием курсового проекта в программной системе были реализованы следующие пункты:

1. Создана база данных, состоящая из 9 таблиц, работа с ней реализована на базе технологии LINQ to Entities.
2. Реализована возможность экспорта и импорта строк таблиц базы данных в XML-файлы посредством технологии LINQ to XML.
3. Графический интерфейс приложения содержит меню (3 раздела - «На главную», «Аналитическая информация» и «О программе»); панель инструментов (редактирование внешнего вида круговой / столбчатой диаграммы), строка статуса (содержит ID и ФИО клиента, для которого формируется лист посещений).
4. Ввод и редактирование данных осуществляется посредством 6 диалоговых окон 4 пользовательских интерфейсов (представляют собой логически обособленные функциональные модули – редактирование данных клиентов; редактирование данных врачей; редактирование услуг; формирование листов посещений клиентов).
5. Реализован пользовательский элемент, позволяющий строить как столбчатые, так и круговые диаграммы, отражающий объем оказанных услуг за отчетный период (в кварталах) по медицинским направлениям поликлиники.
6. Реализована проверка допустимости введенных значений (валидация посредством триггеров WPF) для обеспечения стабильной и комфортной работы программной системы.

Таким образом, можно утверждать, что поставленное задание реализовано в полном объеме.

# **Список использованной литературы**

1. Арлоу, Д., Нейштадт, А. UML 2 и Унифицированный процесс. Практический объектно-ориентированный анализ и проектирование/ Д. Арлоу, А. Нейштадт - М.: 2007. – 620 с.
2. Бен-Ган, И. Microsoft SQL Server 2012. Основы T-SQL/ И. Бен-Ган - М.: Эксмо, 2015 – 400 с.
3. Макдональд, М. WPF: Windows Presentation Foundation в .NET 4.5 с примерами на C# 5.0 для профессионалов/ М. Макдональд – М.: 2013 г. – 1024 с.
4. Натан, А. WPF 4. Подробное руководство/ А. Натан. – М.: 2011 – 880 с.