**МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ**

**УЧРЕЖДЕНИЕ ОБРАЗОВАНИЯ**

**ГОМЕЛЬСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ**

**УНИВЕРСИТЕТ ИМЕНИ П. О. СУХОГО**

Факультет автоматизированных и информационных систем

Кафедра «Информационные технологии»

ОТЧЁТ ПО ЛАБОРАТОРНОЙ РАБОТЕ №6-7

по дисциплине: «Объектно-ориентированное проектирование и   
программирование»

на тему: **«**Коллекции. Обобщенные типы. Технология доступа к   
данным *ADO.NET***»**

Выполнил: студент гр. ИТИ-21

Мусафиров А.Ю.

Принял: преподаватель

Башаримов Ю. С.

Гомель 2024

**Цель работы**: изучение обобщений, итераторов, коллекций и технологии доступа к данным *ADO.NET*.

**Задание:**

* На основании предметной области разработать БД, состоящую из 3-5 таблиц (кол-во таблиц и схему данных предварительно согласовать с преподавателем). Заполнить каждую таблицу 20-30 записями.
* Реализовать *CRUD* операции используя технологию *ADO.NET*. В этой лабораторной работе запрещено использовать компоненты, которые скрывают работу с *ADO.NET*.
* При реализации обязательно использовать обобщенные интерфейсы методы и классы (в рамках темы 6ой лабораторной работы)
* Разработать *GUI*. Приложение должно быть простым в использовании и включать в себя полную обработку исключений. Обязательно использовать подгрузку данных в элементы управления из справочных таблиц (Например: выпадающие списки).
* Написать *unit*-тесты для тестирования разработанных библиотечных классов, тестирование должно покрывать более 80% библиотечного кода.
* При написании и оформлении кода обязательно руководствоваться *Code Convention*, принципами ООП, *SOLID* и использовать элементы авто документирования с генерацией соответствующих файлов.

*Примечание: при реализации 7 и 8 работ приветствуется использование DAO слоя и паттернов проектирования.*

Таблица 1 – Вариант условия задач

|  |  |
| --- | --- |
| **Вариант** | **Условие задачи** |
| 14 | Медицинская карта: Ф.И.О. врача, Ф.И.О. пациента, год рождения, рост, вес, давление, диагноз, дата осмотра |

**Ход работы**

На основании предметной области разработаем базу данных *OOPaP\_67* в СУБД *MS SQL*, состоящую из 3 таблиц (*Doctors, MedicalRecords* и *Patients*). Диаграмма базы данных представлена на рисунке 1.

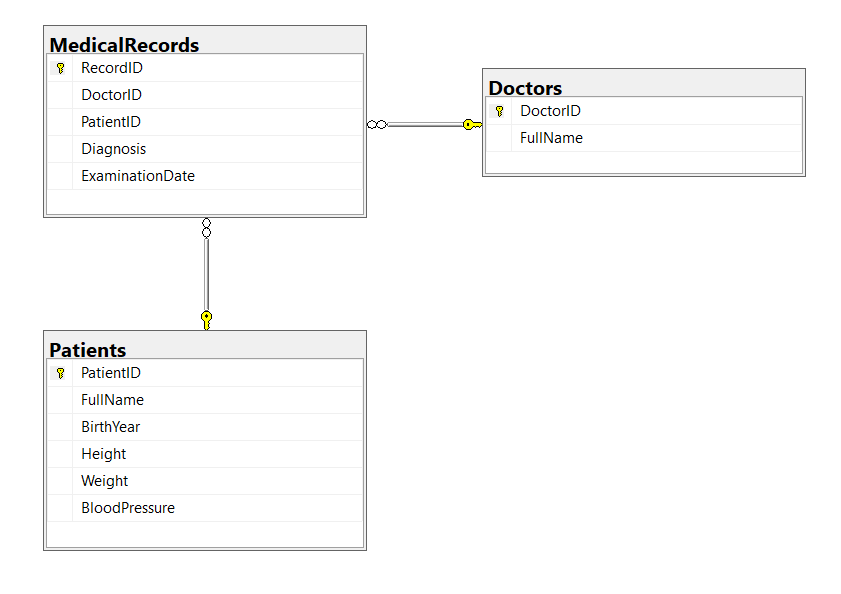


Рисунок 1 – Диаграмма базы данных

Далее разработаем *WPF*-приложение для работы с созданной базой данных *OOPaP\_67,* реализуя *CRUD* операции используя технологию *ADO.NET.*

На рисунке 2 показана работа графического приложения *WPF* при нажатии кнопки *«Загрузить».*

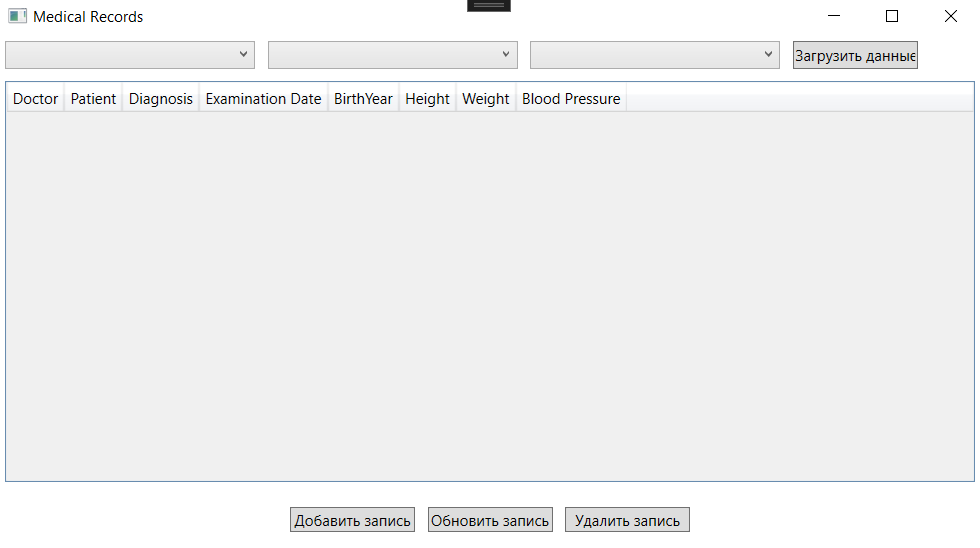


Рисунок 2 – Работа *WPF*-приложения при нажатии   
кнопки *«Загрузить»*

На рисунке 3 показана работа графического приложения *WPF* при нажатии кнопки *«Добавить».*

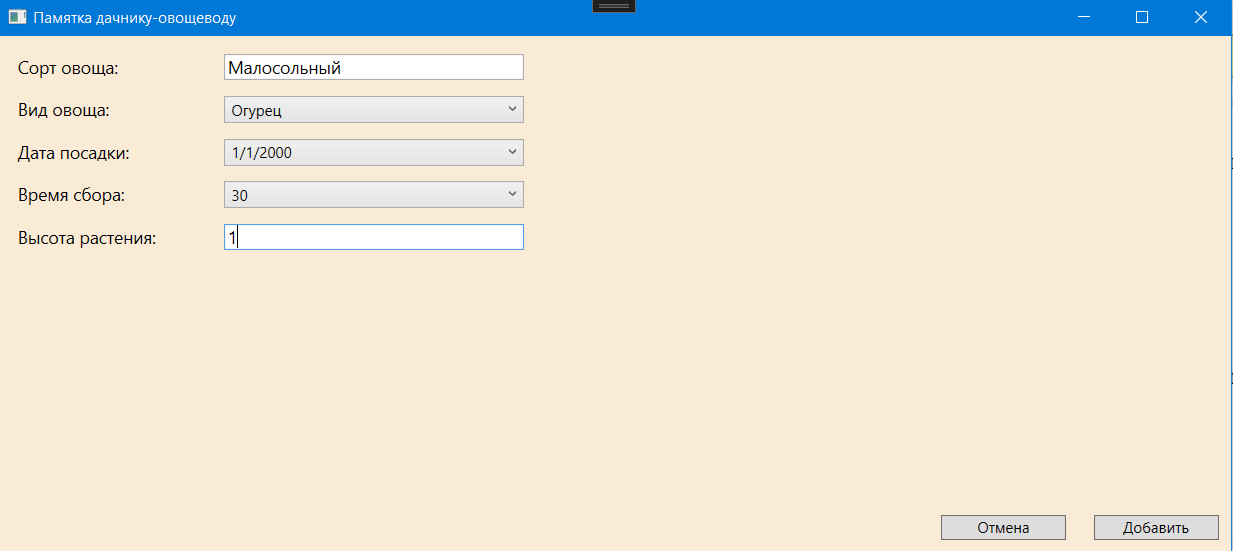


Рисунок 3 – Работа *WPF*-приложения при нажатии   
кнопки *«Добавить»*

На рисунке 4 показана работа графического приложения *WPF* при нажатии кнопки *«Удалить».*

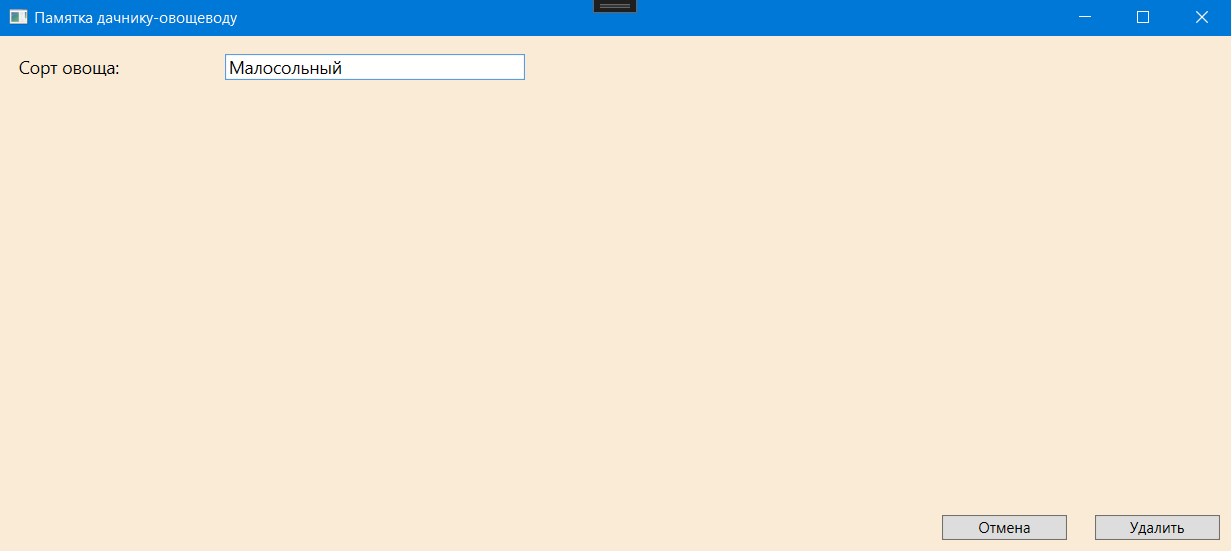


Рисунок 4 – Работа *WPF*-приложения при нажатии   
кнопки *«Удалить»*

На рисунке 5 показана работа графического приложения *WPF* при нажатии кнопки *«Изменить».*

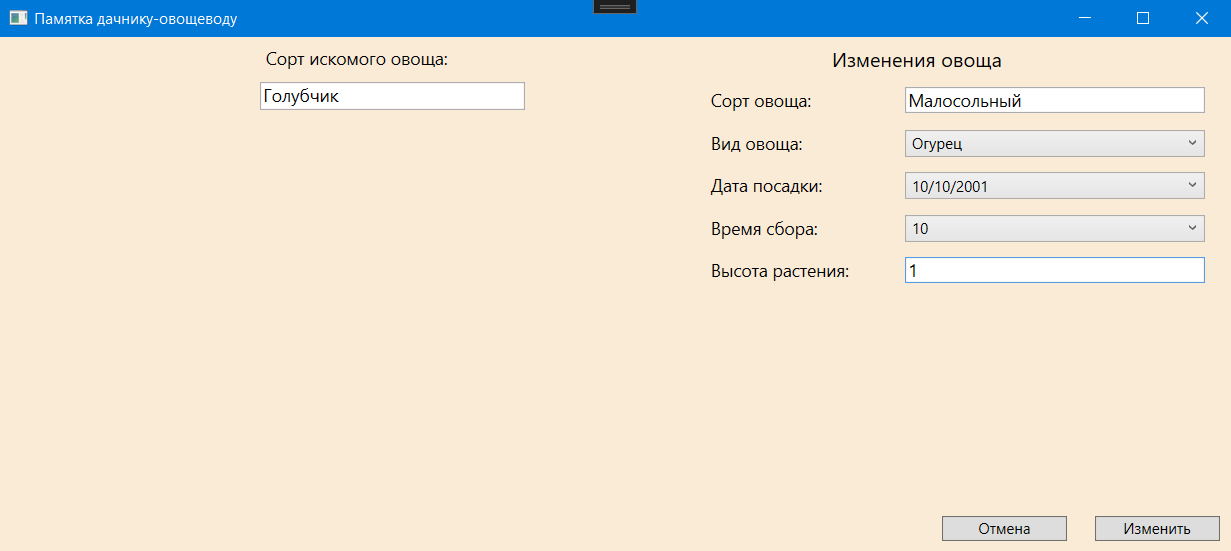


Рисунок 5 – Работа *WPF*-приложения при нажатии   
кнопки *«Изменить»*

На рисунке 6 описан журнал ветвей локального репозитория *Git*.

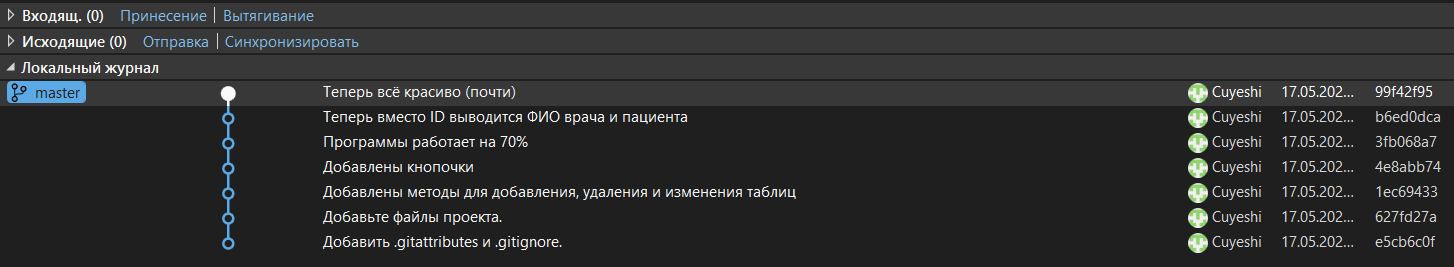


Рисунок 6 – Журнал ветвей репозитория *Git*

**Вывод:** в результате выполнения лабораторной работы были повторены знания об обобщениях, итераторах, коллекциях и изучены технологии доступа к данным *ADO.NET*.

**ПРИЛОЖЕНИЕ А**

**MainWindow.xaml:**

<Window x:Class="Memo.MainWindow"

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

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

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

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

xmlns:local="clr-namespace:Memo"

mc:Ignorable="d"

Title="Памятка дачнику-овощеводу" Height="450" Width="1000">

<Grid>

<Button Content="Загрузить" HorizontalAlignment="Left" VerticalAlignment="Top" Width="60" BorderBrush="{x:Null}" Background="{x:Null}" Click="LoadButton\_Click"/>

<Button Content="Добавить" HorizontalAlignment="Left" VerticalAlignment="Top" Width="60" Margin="60,0,0,0" Background="{x:Null}" BorderBrush="{x:Null}" Click="AddButton\_Click"/>

<Button Content="Удалить" HorizontalAlignment="Left" VerticalAlignment="Top" Width="60" Margin="120,0,0,0" BorderBrush="{x:Null}" Background="{x:Null}" Click="DeleteButton\_Click"/>

<Button Content="Изменить" HorizontalAlignment="Left" VerticalAlignment="Top" Width="60" Margin="180,0,0,0" Background="{x:Null}" BorderBrush="{x:Null}" Click="UpdateButton\_Click"/>

<Grid>

<DataGrid x:Name="dataGrid" Margin="0,20,0,0"

AutoGeneratingColumn="OnAutoGeneratingColumn"

IsReadOnly="True" ColumnWidth="250\*"

CanUserResizeColumns="False"

CanUserResizeRows="False"/>

<Frame x:Name="PageFrame" NavigationUIVisibility="Hidden"></Frame>

</Grid>

</Grid>

</Window>

**AddPage.xaml:**

<Page x:Class="Memo.AddPage"

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:Memo"

mc:Ignorable="d"

d:DesignHeight="450" d:DesignWidth="800"

Title="AddPage">

<Grid Background="AntiqueWhite">

<Button x:Name="submitButton" Content="Добавить" Click="SubmitButton\_Click" VerticalAlignment="Bottom" HorizontalAlignment="Right" MinWidth="100" Margin="0,0,10,10"/>

<Label Content="Cорт овоща:" HorizontalAlignment="Left" Margin="10,10,0,0" VerticalAlignment="Top" Width="150" FontSize="14"/>

<TextBox x:Name="Name" HorizontalAlignment="Left" TextWrapping="Wrap" Text="" Margin="180,14,0,0" VerticalAlignment="Top" Width="240" Height="21" FontSize="14"/>

<Label Content="Вид овоща:" HorizontalAlignment="Left" Margin="10,44,0,0" VerticalAlignment="Top" Width="159" FontSize="14"/>

<ComboBox x:Name="comboBoxType" HorizontalAlignment="Left" Margin="180,48,0,0" VerticalAlignment="Top" Width="240"/>

<Label Content="Дата посадки:" HorizontalAlignment="Left" Margin="10,78,0,0" VerticalAlignment="Top" Width="159" FontSize="14"/>

<ComboBox x:Name="comboBoxPlanting" HorizontalAlignment="Left" Margin="180,82,0,0" VerticalAlignment="Top" Width="240"/>

<Label Content="Время сбора:" HorizontalAlignment="Left" Margin="10,112,0,0" VerticalAlignment="Top" Width="159" FontSize="14"/>

<ComboBox x:Name="comboBoxHarvest" HorizontalAlignment="Left" Margin="180,116,0,0" VerticalAlignment="Top" Width="240"/>

<Label Content="Высота растения:" HorizontalAlignment="Left" Margin="10,146,0,0" VerticalAlignment="Top" Width="150" FontSize="14"/>

<TextBox x:Name="vegetableHeight" HorizontalAlignment="Left" Margin="180,150,0,0" TextWrapping="Wrap" Text="" VerticalAlignment="Top" Width="240" Height="21" FontSize="14"/>

<Button x:Name="cancelButton" Content="Отмена" Click="CancelButton\_Click" VerticalAlignment="Bottom" HorizontalAlignment="Right" MinWidth="100" Margin="0,0,132,10"/>

</Grid>

</Page>

**DeletePage.xaml:**  
<Page x:Class="Memo.DeletePage"

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:Memo"

mc:Ignorable="d"

d:DesignHeight="450" d:DesignWidth="800"

Title="DeletePage">

<Grid Background="AntiqueWhite">

<Button x:Name="submitButton" Content="Удалить" Click="SubmitButton\_Click" VerticalAlignment="Bottom" HorizontalAlignment="Right" MinWidth="100" Margin="0,0,10,10"/>

<Label Content="Сорт овоща:" HorizontalAlignment="Left" Margin="10,10,0,0" VerticalAlignment="Top" Width="150" FontSize="14"/>

<TextBox x:Name="Name" HorizontalAlignment="Left" Margin="180,14,0,0" TextWrapping="Wrap" Text="" VerticalAlignment="Top" Width="240" Height="21" FontSize="14"/>

<Button x:Name="cancelButton" Content="Отмена" Click="CancelButton\_Click" VerticalAlignment="Bottom" HorizontalAlignment="Right" MinWidth="100" Margin="0,0,132,10"/>

</Grid>

</Page>

**UpdatePage.xaml:**  
<Page x:Class="Memo.UpdatePage"

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:Memo"

mc:Ignorable="d"

d:DesignHeight="450" d:DesignWidth="800"

Title="UpdatePage">

<Grid Background="AntiqueWhite">

<Button x:Name="submitButton" Content="Изменить" Click="SubmitButton\_Click" VerticalAlignment="Bottom" HorizontalAlignment="Right" MinWidth="100" Margin="0,0,10,10"/>

<Label Content="Cорт овоща:" HorizontalAlignment="Right" Margin="0,36,272,0" VerticalAlignment="Top" Width="150" FontSize="14"/>

<TextBox x:Name="Name" HorizontalAlignment="Right" Margin="0,40,22,0" TextWrapping="Wrap" Text="" VerticalAlignment="Top" Width="240" Height="21" FontSize="14"/>

<Label Content="Вид овоща:" HorizontalAlignment="Right" Margin="0,70,263,0" VerticalAlignment="Top" Width="159" FontSize="14"/>

<ComboBox x:Name="comboBoxType" HorizontalAlignment="Right" Margin="0,74,22,0" VerticalAlignment="Top" Width="240"/>

<Label Content="Дата посадки:" HorizontalAlignment="Right" Margin="0,104,263,0" VerticalAlignment="Top" Width="159" FontSize="14"/>

<ComboBox x:Name="comboBoxPlanting" HorizontalAlignment="Right" Margin="0,108,22,0" VerticalAlignment="Top" Width="240"/>

<Label Content="Время сбора:" HorizontalAlignment="Right" Margin="0,138,263,0" VerticalAlignment="Top" Width="159" FontSize="14"/>

<ComboBox x:Name="comboBoxHarvest" HorizontalAlignment="Right" Margin="0,142,22,0" VerticalAlignment="Top" Width="240"/>

<Label Content="Высота растения:" HorizontalAlignment="Right" Margin="0,172,272,0" VerticalAlignment="Top" Width="150" FontSize="14"/>

<TextBox x:Name="vegetableHeight" HorizontalAlignment="Right" Margin="0,176,22,0" TextWrapping="Wrap" Text="" VerticalAlignment="Top" Width="240" Height="21" FontSize="14"/>

<Button x:Name="cancelButton" Content="Отмена" Click="CancelButton\_Click" VerticalAlignment="Bottom" HorizontalAlignment="Right" MinWidth="100" Margin="0,0,132,10"/>

<TextBox x:Name="searchName" HorizontalAlignment="Right" Margin="0,36,566,0" TextWrapping="Wrap" Text="" VerticalAlignment="Top" Width="212" Height="22" FontSize="14"/>

<Label Content="Сорт искомого овоща:" HorizontalAlignment="Right" Margin="0,2,526,0" VerticalAlignment="Top" Width="252" FontSize="14" Height="29"/>

<Label Content="Изменения овоща" HorizontalAlignment="Right" Margin="0,2,73,0" VerticalAlignment="Top" Width="252" FontSize="16" Height="29"/>

</Grid>

</Page>

**MainWindow.xaml.cs:**

using System.Windows;

using System.Windows.Controls;

using System.ComponentModel;

using Memo.DAL.ADO.Net;

using Memo.DAL.Interfaces;

using Memo.DAL.Repositories;

using Memo.Domain.Models;

using Memo.Domain;

using Memo.Service.Implementations;

using Memo.Service.Interfaces;

namespace Memo

{

public partial class MainWindow : Window

{

private readonly IVegetableService \_vegetableService;

private readonly IPlantingService \_plantingService;

private readonly ITypeService \_typeService;

private readonly IHarvestService \_harvestService;

public MainWindow()

{

InitializeComponent();

IDbContext dbContext = new SqlContext();

IBaseRepository<Vegetable> vegetableRepository = new VegetableRepository(dbContext);

\_vegetableService = new VegetableService(vegetableRepository);

IBaseRepository<Planting> plantingRepository = new PlantingRepository(dbContext);

\_plantingService = new PlantingService(plantingRepository);

IBaseRepository<Memo.Domain.Type> typeRepository = new TypeRepository(dbContext);

\_typeService = new TypeService(typeRepository);

IBaseRepository<Harvest> harvestRepository = new HarvestRepository(dbContext);

\_harvestService = new HarvestService(harvestRepository);

}

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

{

dataGrid.ItemsSource = \_vegetableService.GetAll();

}

private void OnAutoGeneratingColumn(object sender, DataGridAutoGeneratingColumnEventArgs e)

{

if (e.PropertyDescriptor is PropertyDescriptor descriptor)

{

e.Column.Header = descriptor.DisplayName;

}

}

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

{

PageFrame.Content = new AddPage(\_plantingService, \_harvestService, \_typeService, \_vegetableService);

}

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

{

PageFrame.Content = new DeletePage(\_vegetableService);

}

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

{

PageFrame.Content = new UpdatePage(\_plantingService, \_harvestService, \_typeService, \_vegetableService);

}

}

}

**AddPage.xaml.cs:**

using System.Windows;

using System.Windows.Controls;

using Memo.Domain.ViewModels;

using Memo.Service.Interfaces;

namespace Memo

{

public partial class AddPage : Page

{

private readonly List<PlantingViewModel> \_plantingViewModels;

private readonly List<TypeViewModel> \_typeViewModels;

private readonly List<HarvestViewModel> \_harvestViewModels;

private readonly IVegetableService \_vegetableService;

public AddPage(IPlantingService plantingService, IHarvestService harvestService, ITypeService typeService, IVegetableService vegetableService)

{

InitializeComponent();

\_vegetableService = vegetableService;

\_plantingViewModels = plantingService.GetAll();

\_typeViewModels = typeService.GetAll();

\_harvestViewModels = harvestService.GetAll();

foreach (TypeViewModel x in \_typeViewModels.Distinct())

{

comboBoxType.Items.Add(x.TypeV);

}

foreach (PlantingViewModel x in \_plantingViewModels.Distinct())

{

comboBoxPlanting.Items.Add(x.Planting);

}

foreach (HarvestViewModel x in \_harvestViewModels.Distinct())

{

comboBoxHarvest.Items.Add(x.HarvestTime);

}

}

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

{

try

{

if (Name.Text != string.Empty)

{

if (comboBoxType.SelectedItem != null &&

comboBoxHarvest.SelectedItem != null &&

comboBoxPlanting.SelectedItem != null)

{

if (double.TryParse(vegetableHeight.Text, out double height))

{

\_vegetableService.Create(new VegetableViewModel

{

Name = Name.Text,

HeightSm = height,

TypeName = comboBoxType.SelectedItem.ToString()!,

PlantingTime = Convert.ToDateTime(comboBoxPlanting.SelectedItem!),

HarvestTime = Convert.ToInt32(comboBoxHarvest.SelectedItem!),

});

}

}

}

}

catch (Exception ex)

{

MessageBox.Show(ex.Message);

}

Content = null;

}

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

{

Content = null;

}

}

}

**DeletePage.xaml.cs:**  
using System.Windows;

using System.Windows.Controls;

using Memo.Service.Interfaces;

namespace Memo

{

public partial class DeletePage : Page

{

private readonly IVegetableService \_vegetableService;

public DeletePage(IVegetableService vegetableService)

{

InitializeComponent();

\_vegetableService = vegetableService;

}

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

{

try

{

if (Name.Text.Trim() != string.Empty)

{

\_vegetableService.Delete(Name.Text);

}

}

catch (Exception ex)

{

MessageBox.Show(ex.Message);

}

Content = null;

}

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

{

Content = null;

}

}

}

**UpdatePage.xaml.cs:**  
using System.Windows;

using System.Windows.Controls;

using Memo.Domain.ViewModels;

using Memo.Service.Interfaces;

namespace Memo

{

public partial class UpdatePage : Page

{

private readonly List<PlantingViewModel> \_plantingViewModels;

private readonly List<TypeViewModel> \_typeViewModels;

private readonly List<HarvestViewModel> \_harvestViewModels;

private readonly IVegetableService \_vegetableService;

public UpdatePage(IPlantingService plantingService, IHarvestService harvestService, ITypeService typeService, IVegetableService vegetableService)

{

InitializeComponent();

\_vegetableService = vegetableService;

\_plantingViewModels = plantingService.GetAll();

\_typeViewModels = typeService.GetAll();

\_harvestViewModels = harvestService.GetAll();

foreach (TypeViewModel x in \_typeViewModels.Distinct())

{

comboBoxType.Items.Add(x.TypeV);

}

foreach (PlantingViewModel x in \_plantingViewModels.Distinct())

{

comboBoxPlanting.Items.Add(x.Planting);

}

foreach (HarvestViewModel x in \_harvestViewModels.Distinct())

{

comboBoxHarvest.Items.Add(x.HarvestTime);

}

}

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

{

try

{

VegetableViewModel vegetableViewModel = new();

if (Name.Text.Trim() != string.Empty)

{

if (comboBoxType.SelectedItem != null &&

comboBoxHarvest.SelectedItem != null &&

comboBoxPlanting.SelectedItem != null)

{

if (double.TryParse(vegetableHeight.Text, out double height))

{

\_vegetableService.Create(new VegetableViewModel

{

Name = Name.Text,

HeightSm = height,

TypeName = comboBoxType.SelectedItem.ToString()!,

PlantingTime = Convert.ToDateTime(comboBoxPlanting.SelectedItem!),

HarvestTime = Convert.ToInt32(comboBoxHarvest.SelectedItem!),

});

}

}

}

if (searchName.Text.Trim() != string.Empty)

{

\_vegetableService.Delete(searchName.Text);

\_vegetableService.Edit(searchName.Text.Trim(), vegetableViewModel);

}

}

catch (Exception ex)

{

//MessageBox.Show(ex.Message);

}

Content = null;

}

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

{

Content = null;

}

}

}

**App.config:**  
<?xml version="1.0" encoding="utf-8" ?>

<configuration>

<connectionStrings>

<add name="Db"

providerName="Microsoft.Data.SqlClient"

connectionString="Server=WIN-J9S0JJK5DN4\SQLEXPRESS;Database=MemoVegetable;Trusted\_Connection=True;TrustServerCertificate=true;" />

</connectionStrings>

</configuration>

**SqlContext.cs:**  
using System.Reflection;

using System.Data.SqlClient;

using Memo.DAL.Interfaces;

using Memo.Domain.Models;

using Memo.Domain;

using System.Configuration;

namespace Memo.DAL.ADO.Net;

public class SqlContext : IDbContext

{

public List<Vegetable> Vegetable { get; set; } = [];

public List<Domain.Type> Type { get; set; } = [];

public List<Planting> Planting { get; set; } = [];

public List<Harvest> Harvest { get; set; } = [];

private readonly string \_dbSettings;

private readonly SqlConnection \_connection;

public SqlContext()

{

\_dbSettings = ConfigurationManager.ConnectionStrings["DB"].ConnectionString;

\_connection = new SqlConnection(\_dbSettings);

SelectAll();

}

private void SelectAll()

{

\_connection.Open();

SqlCommand selectType = new("SELECT \* FROM Type", \_connection);

Type = SelectT(selectType);

SqlCommand selectPlanting = new("SELECT \* FROM Plant", \_connection);

Planting = SelectP(selectPlanting);

SqlCommand selectHarvest = new("SELECT \* FROM Harvest", \_connection);

Harvest = SelectH(selectHarvest);

SqlCommand selectVegetable = new("SELECT \* FROM Vegetable", \_connection);

Vegetable = SelectV(selectVegetable);

\_connection.Close();

}

private List<Vegetable> SelectV(SqlCommand command)

{

List<Vegetable> entities = new List<Vegetable>();

using (SqlDataReader reader = command.ExecuteReader())

{

while (reader.Read())

{

Vegetable entity = new();

PropertyInfo[] properties = typeof(Vegetable).GetProperties();

for (int i = 0; i < properties.Length; i++)

{

if (reader.GetName(i).Contains("ID") && i != 0)

{

int id = Convert.ToInt32(reader.GetValue(i));

if (entity is Vegetable)

{

if (reader.GetName(i).Contains("Type"))

{

List<Domain.Type> types = Type;

foreach (Domain.Type type in types)

{

if (type.Id == id)

properties[i].SetValue(entity, type);

}

}

if (reader.GetName(i).Contains("Plant"))

{

List<Planting> plantings = Planting;

foreach (Planting planting in plantings)

{

if (planting.Id == id)

properties[i].SetValue(entity, planting);

}

}

if (reader.GetName(i).Contains("Harvest"))

{

List<Harvest> harvests = Harvest;

foreach (Harvest harvest in harvests)

{

if (harvest.Id == id)

properties[i].SetValue(entity, harvest);

}

}

}

else

{

properties[i].SetValue(entity, null);

}

}

else

{

properties[i].SetValue(entity, Convert.ChangeType(reader.GetValue(i),

properties[i].PropertyType));

}

}

entities.Add(entity);

}

}

return entities;

}

private List<Planting> SelectP(SqlCommand command)

{

List<Planting> entities = new List<Planting>();

using (SqlDataReader reader = command.ExecuteReader())

{

while (reader.Read())

{

Planting entity = new();

PropertyInfo[] properties = typeof(Planting).GetProperties();

for (int i = 0; i < properties.Length; i++)

{

if (reader.GetName(i).Contains("ID") && i != 0)

{

int id = Convert.ToInt32(reader.GetValue(i));

}

else

{

properties[i].SetValue(entity, Convert.ChangeType(reader.GetValue(i),

properties[i].PropertyType));

}

}

entities.Add(entity);

}

}

return entities;

}

private List<Memo.Domain.Type> SelectT(SqlCommand command)

{

List<Memo.Domain.Type> entities = new List<Memo.Domain.Type>();

using (SqlDataReader reader = command.ExecuteReader())

{

while (reader.Read())

{

Memo.Domain.Type entity = new();

PropertyInfo[] properties = typeof(Memo.Domain.Type).GetProperties();

for (int i = 0; i < properties.Length; i++)

{

if (reader.GetName(i).Contains("ID") && i != 0)

{

int id = Convert.ToInt32(reader.GetValue(i));

}

else

{

properties[i].SetValue(entity, Convert.ChangeType(reader.GetValue(i),

properties[i].PropertyType));

}

}

entities.Add(entity);

}

}

return entities;

}

private List<Harvest> SelectH(SqlCommand command)

{

List<Harvest> entities = new List<Harvest>();

using (SqlDataReader reader = command.ExecuteReader())

{

while (reader.Read())

{

Harvest entity = new();

PropertyInfo[] properties = typeof(Harvest).GetProperties();

for (int i = 0; i < properties.Length; i++)

{

if (reader.GetName(i).Contains("ID") && i != 0)

{

int id = Convert.ToInt32(reader.GetValue(i));

}

else

{

properties[i].SetValue(entity, Convert.ChangeType(reader.GetValue(i),

properties[i].PropertyType));

}

}

entities.Add(entity);

}

}

return entities;

}

private void DeleteAll()

{

SqlCommand deleteVegetable = new("DELETE FROM Vegetable", \_connection);

deleteVegetable.ExecuteNonQuery();

}

private void InsertIntoVegetable()

{

string vegetableStr = "";

foreach (Vegetable vegetable in Vegetable)

{

List<Domain.Type> types = Type;

List<Planting> plants = Planting;

List<Harvest> harvests = Harvest;

foreach (Domain.Type type in types)

{

if (type.TypeV == vegetable.Type?.TypeV)

vegetable.Type = type;

}

foreach (Planting planting in plants)

{

if (planting.PlantingTime == vegetable.Planting?.PlantingTime)

vegetable.Planting = planting;

}

foreach (Harvest harvest in harvests)

{

if (harvest.HarvestTime == vegetable.Harvest?.HarvestTime)

vegetable.Harvest = harvest;

}

vegetableStr += vegetable + ", ";

}

vegetableStr = vegetableStr.Remove(vegetableStr.Length - 2);

SqlCommand insertVegetable = new("INSERT INTO Vegetable ([Name], TypeID, Height, PlantingID, HarvestID) " +

"VALUES " + vegetableStr, \_connection);

insertVegetable.ExecuteNonQuery();

}

private void UpdateAll()

{

DeleteAll();

InsertIntoVegetable();

}

public void SaveChanges()

{

\_connection.Open();

UpdateAll();

\_connection.Close();

}

}

**IBaseRepository.cs:**  
namespace Memo.DAL.Interfaces;

public interface IBaseRepository<T>

{

List<T> ReadAll();

bool Create(T entity);

T Read(int id);

bool Update(T oldEntity, T newEntity);

bool Delete(T entity);

}

**IDbContext.cs:**  
using Memo.Domain.Models;

using Memo.Domain;

namespace Memo.DAL.Interfaces

{

public interface IDbContext

{

List<Vegetable> Vegetable { get; }

List<Domain.Type> Type { get; }

List<Planting> Planting { get; }

List<Harvest> Harvest { get; }

void SaveChanges();

}

}

**HarvestRepository.cs:**  
using Memo.DAL.Interfaces;

using Memo.Domain.Models;

namespace Memo.DAL.Repositories;

public class HarvestRepository(IDbContext dbContext) : IBaseRepository<Harvest>

{

private readonly IDbContext \_dbContext = dbContext;

public List<Harvest> ReadAll()

{

return \_dbContext.Harvest;

}

public bool Create(Harvest harvest)

{

\_dbContext.Harvest.Add(harvest);

\_dbContext.SaveChanges();

return true;

}

public Harvest Read(int id)

{

if (id > 0 && id < \_dbContext.Harvest.Count)

{

Harvest harvestToRead = \_dbContext.Harvest.ElementAt(id);

return harvestToRead;

}

else

{

throw new IndexOutOfRangeException("Выход за пределы при попытке чтения из базы данных");

}

}

public bool Update(Harvest oldHarvest, Harvest newHarvest)

{

List<Harvest> harvestsToUpdate = \_dbContext.Harvest!;

Harvest? harvestToUpdate = null;

foreach (Harvest harvest in harvestsToUpdate)

{

if (harvest == oldHarvest)

{

harvestToUpdate = harvest;

break;

}

}

if (harvestToUpdate != null)

{

harvestToUpdate.HarvestTime = newHarvest.HarvestTime;

\_dbContext.SaveChanges();

return true;

}

else

{

throw new ArgumentNullException($"Овощь не найден в базе данных для обновления:{harvestToUpdate}");

}

}

public bool Delete(Harvest harvest)

{

bool isRemove = \_dbContext.Harvest.Remove(harvest);

\_dbContext.SaveChanges();

return isRemove;

}

}

**PlantingRepository.cs:**  
using Memo.DAL.Interfaces;

using Memo.Domain;

namespace Memo.DAL.Repositories;

public class PlantingRepository(IDbContext dbContext) : IBaseRepository<Planting>

{

private readonly IDbContext \_dbContext = dbContext;

public List<Planting> ReadAll()

{

return \_dbContext.Planting;

}

public bool Create(Planting planting)

{

\_dbContext.Planting.Add(planting);

\_dbContext.SaveChanges();

return true;

}

public Planting Read(int id)

{

if (id > 0 && id < \_dbContext.Planting.Count)

{

Planting plantingToRead = \_dbContext.Planting.ElementAt(id);

return plantingToRead;

}

else

{

throw new IndexOutOfRangeException("Выход за пределы при попытке чтения из базы данных");

}

}

public bool Update(Planting oldPlanting, Planting newPlanting)

{

List<Planting> plantingsToUpdate = \_dbContext.Planting!;

Planting? plantingToUpdate = null;

foreach (Planting planting in plantingsToUpdate)

{

if (planting == oldPlanting)

{

plantingToUpdate = planting;

break;

}

}

if (plantingToUpdate != null)

{

plantingToUpdate.PlantingTime = newPlanting.PlantingTime;

\_dbContext.SaveChanges();

return true;

}

else

{

throw new ArgumentNullException($"Овощь не найден в базе данных для обновления:{plantingToUpdate}");

}

}

public bool Delete(Planting planting)

{

bool isRemove = \_dbContext.Planting.Remove(planting);

\_dbContext.SaveChanges();

return isRemove;

}

}

**TypeRepository.cs:**  
using Memo.DAL.Interfaces;

using Memo.Domain;

namespace Memo.DAL.Repositories;

public class TypeRepository(IDbContext dbContext) : IBaseRepository<Memo.Domain.Type>

{

private readonly IDbContext \_dbContext = dbContext;

public List<Memo.Domain.Type> ReadAll()

{

return \_dbContext.Type;

}

public bool Create(Memo.Domain.Type type)

{

\_dbContext.Type.Add(type);

\_dbContext.SaveChanges();

return true;

}

public Memo.Domain.Type Read(int id)

{

if (id > 0 && id < \_dbContext.Type.Count)

{

Memo.Domain.Type typeToRead = \_dbContext.Type.ElementAt(id);

return typeToRead;

}

else

{

throw new IndexOutOfRangeException("Выход за пределы при попытке чтения из базы данных");

}

}

public bool Update(Memo.Domain.Type oldType, Memo.Domain.Type newType)

{

List<Memo.Domain.Type> typesToUpdate = \_dbContext.Type!;

Memo.Domain.Type? typeToUpdate = null;

foreach (Memo.Domain.Type type in typesToUpdate)

{

if (type == oldType)

{

typeToUpdate = type;

break;

}

}

if (typeToUpdate != null)

{

typeToUpdate.TypeV = newType.TypeV;

\_dbContext.SaveChanges();

return true;

}

else

{

throw new ArgumentNullException($"Овощь не найден в базе данных для обновления:{typeToUpdate}");

}

}

public bool Delete(Memo.Domain.Type type)

{

bool isRemove = \_dbContext.Type.Remove(type);

\_dbContext.SaveChanges();

return isRemove;

}

}

**VegetableRepository.cs:**  
using Memo.DAL.Interfaces;

using Memo.Domain;

namespace Memo.DAL.Repositories;

public class VegetableRepository(IDbContext dbContext) : IBaseRepository<Vegetable>

{

private readonly IDbContext \_dbContext = dbContext;

public List<Vegetable> ReadAll()

{

return \_dbContext.Vegetable;

}

public bool Create(Vegetable vegetable)

{

\_dbContext.Vegetable.Add(vegetable);

\_dbContext.SaveChanges();

return true;

}

public Vegetable Read(int id)

{

if (id > 0 && id < \_dbContext.Vegetable.Count)

{

Vegetable vegetableToRead = \_dbContext.Vegetable.ElementAt(id);

return vegetableToRead;

}

else

{

throw new IndexOutOfRangeException("Выход за пределы при попытке чтения из базы данных");

}

}

public bool Update(Vegetable oldVegetable, Vegetable newVegetable)

{

List<Vegetable?> vegetablesToUpdate = \_dbContext.Vegetable!;

Vegetable vegetableToUpdate = null;

foreach (Vegetable vegetable in vegetablesToUpdate)

{

if (vegetable == oldVegetable)

{

vegetableToUpdate = vegetable;

break;

}

}

if (vegetableToUpdate != null)

{

vegetableToUpdate.Name = newVegetable.Name;

vegetableToUpdate.Id = newVegetable.Id;

vegetableToUpdate.HeightSm = newVegetable.HeightSm;

vegetableToUpdate.Planting = newVegetable.Planting;

vegetableToUpdate.Type = newVegetable.Type;

vegetableToUpdate.Harvest = newVegetable.Harvest;

\_dbContext.SaveChanges();

return true;

}

else

{

throw new ArgumentNullException($"Овощь не найден в базе данных для обновления:{vegetableToUpdate}");

}

}

public bool Delete(Vegetable vegetable)

{

bool isRemove = \_dbContext.Vegetable.Remove(vegetable);

\_dbContext.SaveChanges();

return isRemove;

}

}

**Harvest.cs:**  
using Memo.Domain.ViewModels;

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace Memo.Domain.Models

{

public class Harvest

{

private static int \_id = 1;

public int Id { get; set; }

public int HarvestTime { get; set; }

public Harvest(int harvest)

{

Id = \_id;

HarvestTime = harvest;

\_id++;

}

public Harvest() : this(0)

{ }

public override string ToString()

{

return $"({Id}, {HarvestTime})";

}

public static implicit operator Harvest(HarvestViewModel planting)

{

return new Harvest

{

HarvestTime = planting.HarvestTime,

};

}

public override bool Equals(object? obj)

{

if (ReferenceEquals(this, obj)) return true;

if (obj is Harvest other)

{

if (HarvestTime == other.HarvestTime)

return true;

}

if (obj is null) return false;

return false;

}

public override int GetHashCode()

{

throw new NotImplementedException();

}

}

}

**Planting.cs:**  
using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using Memo.Domain.ViewModels;

namespace Memo.Domain;

public class Planting

{

private static int \_id = 1;

public int Id { get; set; }

public DateTime PlantingTime { get; set; }

public Planting(DateTime planting)

{

Id = \_id;

PlantingTime = planting;

\_id++;

}

public Planting() : this(new())

{ }

public override string ToString()

{

return $"({Id}, {PlantingTime})";

}

public static implicit operator Planting(PlantingViewModel planting)

{

return new Planting

{

PlantingTime = planting.Planting,

};

}

public override bool Equals(object? obj)

{

if (ReferenceEquals(this, obj)) return true;

if (obj is Planting other)

{

if (PlantingTime == other.PlantingTime)

return true;

}

if (obj is null) return false;

return false;

}

public override int GetHashCode()

{

throw new NotImplementedException();

}

}

**Type.cs:**  
using Memo.Domain.ViewModels;

namespace Memo.Domain;

public class Type

{

private static int \_id = 1;

public int Id { get; set; }

public string TypeV { get; set; }

public Type(string type)

{

Id = \_id;

TypeV = type;

\_id++;

}

public Type()

{

Id = \_id;

TypeV = string.Empty;

\_id++;

}

public override string ToString()

{

return $"({Id}, '{TypeV}')";

}

public static implicit operator Type(TypeViewModel variety)

{

return new Type

{

TypeV = variety.TypeV,

};

}

public override bool Equals(object? obj)

{

if (ReferenceEquals(this, obj)) return true;

if (obj is Type variety)

{

if (TypeV == variety.TypeV)

return true;

return false;

}

if (obj is null) return false;

return false;

}

public override int GetHashCode()

{

throw new NotImplementedException();

}

}

**Vegetable.cs:**  
using System.Globalization;

using Memo.Domain.Models;

using Memo.Domain.ViewModels;

namespace Memo.Domain;

public class Vegetable

{

private static int \_id = 1;

public int Id { get; set; } = \_id;

public string Name { get; set; }

public Type? Type { get; set; }

public double HeightSm { get; set; }

public Planting? Planting { get; set; }

public Harvest? Harvest { get; set; }

public Vegetable(string name, double heightSm, Planting planting, Type type, Harvest harvest)

{

Id = \_id;

Name = name;

HeightSm = heightSm;

Planting = planting;

Type = type;

Harvest = harvest;

\_id++;

}

public Vegetable()

{

Name = string.Empty;

HeightSm = 0.0;

Planting = null;

Type = null;

Harvest = null;

\_id++;

}

public override string ToString()

{

return $"('{Name}', {Type?.Id}, {HeightSm.ToString().Replace(',','.')}, {Planting?.Id}, {Harvest?.Id})";

}

public static implicit operator Vegetable(VegetableViewModel vegetableViewModel)

{

return new Vegetable

{

Name = vegetableViewModel.Name,

HeightSm = vegetableViewModel.HeightSm,

Planting = new Planting(vegetableViewModel.PlantingTime),

Type = new Type(vegetableViewModel.TypeName),

Harvest = new Harvest(vegetableViewModel.HarvestTime)

};

}

public override bool Equals(object? obj)

{

if (ReferenceEquals(this, obj)) return true;

if (obj is Vegetable b)

{

if (Type == b.Type &&

Planting == b.Planting &&

HeightSm == b.HeightSm) return true;

return false;

}

if (obj is null) return false;

return false;

}

public override int GetHashCode()

{

throw new NotImplementedException();

}

}

**HarvestViewModel.cs:**  
using Memo.Domain.Models;

namespace Memo.Domain.ViewModels

{

public class HarvestViewModel(int harvest)

{

public int HarvestTime { get; set; } = harvest;

public HarvestViewModel() : this(0) { }

public static implicit operator HarvestViewModel(Harvest harvest)

{

return new HarvestViewModel

{

HarvestTime = harvest.HarvestTime,

};

}

}

}

**PlantingViewModel.cs:**  
namespace Memo.Domain.ViewModels

{

public class PlantingViewModel(DateTime planting)

{

public DateTime Planting { get; set; } = planting;

public PlantingViewModel() : this(new()) { }

public static implicit operator PlantingViewModel(Planting planting)

{

return new PlantingViewModel

{

Planting = planting.PlantingTime,

};

}

}

}

**TypeViewModel.cs:**

namespace Memo.Domain.ViewModels

{

public class TypeViewModel(string type)

{

public string TypeV { get; set; } = type;

public TypeViewModel() : this("") { }

public static implicit operator TypeViewModel(Type variety)

{

return new TypeViewModel

{

TypeV = variety.TypeV,

};

}

}

}

**VegetableViewModel.cs:**

using Memo.Domain.ViewModels;

using Memo.Domain;

using System.ComponentModel;

namespace Memo.Domain.ViewModels

{

public class VegetableViewModel(string name, double heightSm, PlantingViewModel plantings, TypeViewModel types, HarvestViewModel harvest)

{

[DisplayName("Сорт овоща")]

public string Name { get; set; } = name;

[DisplayName("Высота овоща, см")]

public double HeightSm { get; set; } = heightSm;

[DisplayName("Рекомендуемая дата посадки")]

public DateTime PlantingTime { get; set; } = plantings.Planting;

[DisplayName("Вид овоща")]

public string TypeName { get; set; } = types.TypeV;

[DisplayName("Время сбора урожая")]

public int HarvestTime { get; set; } = harvest.HarvestTime;

public VegetableViewModel() : this("", 0.0, new(), new(), new()) { }

public static implicit operator VegetableViewModel(Vegetable vegetable)

{

ArgumentNullException.ThrowIfNull(vegetable.Type);

ArgumentNullException.ThrowIfNull(vegetable.Planting);

ArgumentNullException.ThrowIfNull(vegetable.Harvest);

return new VegetableViewModel

{

Name = vegetable.Name,

HeightSm = vegetable.HeightSm,

PlantingTime = vegetable.Planting.PlantingTime,

TypeName = vegetable.Type.TypeV,

HarvestTime = vegetable.Harvest.HarvestTime,

};

}

}

}

**HarvestService.cs:**

using Memo.Domain.Models;

using Memo.Domain.ViewModels;

using Memo.Service.Interfaces;

using Memo.DAL.Interfaces;

namespace Memo.Service.Implementations

{

public class HarvestService(IBaseRepository<Harvest> harvestRepository) : IHarvestService

{

private readonly IBaseRepository<Harvest> \_harvestRepository = harvestRepository;

public List<HarvestViewModel> GetAll()

{

try

{

List<Harvest> harvest = \_harvestRepository.ReadAll();

List<HarvestViewModel> harvestViewModels = [];

for (int i = 0; i < harvest.Count; i++)

{

ArgumentNullException.ThrowIfNull(harvest[i]);

harvestViewModels.Add(harvest[i]);

}

return harvestViewModels;

}

catch (ArgumentNullException)

{

throw new ArgumentNullException($"[GetAll]:Объект Harvest не найден");

}

catch (Exception ex)

{

throw new Exception($"[GetAll]:{ex.Message}");

}

}

}

}

**PlantingService.cs:**

using Memo.DAL.Interfaces;

using Memo.Domain.ViewModels;

using Memo.Domain;

using Memo.Service.Interfaces;

namespace Memo.Service.Implementations

{

public class PlantingService(IBaseRepository<Planting> plantingRepository) : IPlantingService

{

private readonly IBaseRepository<Planting> \_plantingRepository = plantingRepository;

public List<PlantingViewModel> GetAll()

{

try

{

List<Planting> planting = \_plantingRepository.ReadAll();

List<PlantingViewModel> plantingViewModels = [];

for (int i = 0; i < planting.Count; i++)

{

ArgumentNullException.ThrowIfNull(planting[i]);

plantingViewModels.Add(planting[i]);

}

return plantingViewModels;

}

catch (ArgumentNullException)

{

throw new ArgumentNullException($"[GetAll]:Объект Planting не найден");

}

catch (Exception ex)

{

throw new Exception($"[GetAll]:{ex.Message}");

}

}

}

}

**TypeService.cs:**

using Memo.DAL.Interfaces;

using Memo.Domain.ViewModels;

using Memo.Service.Interfaces;

namespace Memo.Service.Implementations

{

public class TypeService(IBaseRepository<Domain.Type> typeRepository) : ITypeService

{

private readonly IBaseRepository<Domain.Type> \_typeRepository = typeRepository;

public List<TypeViewModel> GetAll()

{

try

{

List<Domain.Type> type = \_typeRepository.ReadAll();

List<TypeViewModel> typeViewModels = [];

for (int i = 0; i < type.Count; i++)

{

ArgumentNullException.ThrowIfNull(type[i]);

typeViewModels.Add(type[i]);

}

return typeViewModels;

}

catch (ArgumentNullException)

{

throw new ArgumentNullException($"[GetAll]:Объект Type не найден");

}

catch (Exception ex)

{

throw new Exception($"[GetAll]:{ex.Message}");

}

}

}

}

**VegetableService.cs:**

using Memo.DAL.Interfaces;

using Memo.Domain.ViewModels;

using Memo.Domain;

using Memo.Service.Interfaces;

using System.Xml.Linq;

namespace Memo.Service.Implementations

{

public class VegetableService(IBaseRepository<Vegetable> vegetableRepository) : IVegetableService

{

private readonly IBaseRepository<Vegetable> \_vegetableRepository = vegetableRepository;

public bool Create(VegetableViewModel vegetableViewModel)

{

try

{

Vegetable vegetable = vegetableViewModel;

ArgumentNullException.ThrowIfNull(vegetable);

return \_vegetableRepository.Create(vegetable);

}

catch (ArgumentNullException)

{

throw new ArgumentNullException($"[Create]:Объект Vegetable не найден:{vegetableViewModel}");

}

catch (Exception ex)

{

throw new Exception($"[Create]:{ex.Message}");

}

}

public bool Delete(string name)

{

try

{

List<Vegetable>? vegetables = \_vegetableRepository.ReadAll();

for (int i = 0; i < vegetables.Count; i++)

{

if (vegetables[i].Name == name)

{

ArgumentNullException.ThrowIfNull(vegetables[i]);

return \_vegetableRepository.Delete(vegetables[i]);

}

}

return false;

}

catch (ArgumentNullException)

{

throw new ArgumentNullException($"[Delete]:Объект Vegetable не найден по названию: {name}");

}

catch (Exception ex)

{

throw new Exception($"[Delete]:{ex.Message}");

}

}

public bool Delete(int id)

{

try

{

List<Vegetable>? vegetables = \_vegetableRepository.ReadAll();

for (int i = 0; i < vegetables.Count; i++)

{

if (vegetables[i].Id == id)

{

ArgumentNullException.ThrowIfNull(vegetables[i]);

return \_vegetableRepository.Delete(vegetables[i]);

}

}

return false;

}

catch (ArgumentNullException)

{

throw new ArgumentNullException($"[Delete]:Объект Vegetable не найден по id: {id}");

}

catch (Exception ex)

{

throw new Exception($"[Delete]:{ex.Message}");

}

}

public bool Edit(int id, VegetableViewModel vegetableViewModel)

{

try

{

List<Vegetable>? vegetables = \_vegetableRepository.ReadAll();

Vegetable newVegetable = vegetableViewModel;

for (int i = 0; i < vegetables.Count; i++)

{

if (vegetables[i].Id == id)

{

ArgumentNullException.ThrowIfNull(vegetables[i]);

ArgumentNullException.ThrowIfNull(newVegetable);

return \_vegetableRepository.Update(vegetables[i], newVegetable);

}

}

return false;

}

catch (ArgumentNullException ex)

{

throw new ArgumentNullException($"[Edit]:Объект Vegetable не найден по:{ex.ParamName}");

}

catch (Exception ex)

{

throw new Exception($"[Edit]:{ex.Message}");

}

}

public bool Edit(string name, VegetableViewModel vegetableViewModel)

{

try

{

List<Vegetable>? vegetables = \_vegetableRepository.ReadAll();

Vegetable newVegetable = vegetableViewModel;

for (int i = 0; i < vegetables.Count; i++)

{

if (vegetables[i].Name == name)

{

ArgumentNullException.ThrowIfNull(vegetables[i]);

ArgumentNullException.ThrowIfNull(newVegetable);

return \_vegetableRepository.Update(vegetables[i], newVegetable);

}

}

return false;

}

catch (ArgumentNullException ex)

{

throw new ArgumentNullException($"[Edit]:Объект Vegetable не найден по:{ex.ParamName}");

}

catch (Exception ex)

{

throw new Exception($"[Edit]:{ex.Message}");

}

}

public VegetableViewModel Get(int id)

{

try

{

List<Vegetable>? vegetables = \_vegetableRepository.ReadAll();

for (int i = 0; i < vegetables.Count; i++)

{

if (vegetables[i].Id == id)

{

ArgumentNullException.ThrowIfNull(vegetables[i]);

VegetableViewModel vegetableViewModel = vegetables[i];

return vegetableViewModel;

}

}

VegetableViewModel vegetableView = new VegetableViewModel();

return vegetableView;

}

catch (ArgumentNullException)

{

throw new ArgumentNullException($"[Get]:Объект Vegetable не найден по id:{id}");

}

catch (Exception ex)

{

throw new Exception($"[Get]:{ex.Message}");

}

}

public List<VegetableViewModel> GetAll()

{

try

{

List<Vegetable> vegetable = \_vegetableRepository.ReadAll();

List<VegetableViewModel> vegetableViewModels = [];

for (int i = 0; i < vegetable.Count; i++)

{

ArgumentNullException.ThrowIfNull(vegetable[i]);

vegetableViewModels.Add(vegetable[i]);

}

return vegetableViewModels;

}

catch (ArgumentNullException)

{

throw new ArgumentNullException($"[GetAll]:Объект Vegetable не найден");

}

catch (Exception ex)

{

throw new Exception($"[GetAll]:{ex.Message}");

}

}

public VegetableViewModel GetByName(string name)

{

try

{

List<Vegetable>? vegetables = \_vegetableRepository.ReadAll();

for (int i = 0; i < vegetables.Count; i++)

{

if (vegetables[i].Name == name)

{

ArgumentNullException.ThrowIfNull(vegetables[i]);

VegetableViewModel vegetableViewModel = vegetables[i];

return vegetableViewModel;

}

}

VegetableViewModel vegetableView = new VegetableViewModel();

return vegetableView;

}

catch (ArgumentNullException)

{

throw new ArgumentNullException($"[GetByName]:Объект Vegetable не найден по имени: {name}");

}

catch (Exception ex)

{

throw new Exception($"[GetByName]:{ex.Message}");

}

}

}

}

**IHarvestService.cs:**

using Memo.Domain.ViewModels;

namespace Memo.Service.Interfaces

{

public interface IHarvestService

{

List<HarvestViewModel> GetAll();

}

}

**IPlantingService.cs:**

using Memo.Domain.ViewModels;

namespace Memo.Service.Interfaces

{

public interface IPlantingService

{

List<PlantingViewModel> GetAll();

}

}

**ITypeService.cs:**

using Memo.Domain.ViewModels;

namespace Memo.Service.Interfaces

{

public interface ITypeService

{

List<TypeViewModel> GetAll();

}

}

**IVegetableService.cs:**

using Memo.Domain.ViewModels;

namespace Memo.Service.Interfaces

{

public interface IVegetableService

{

List<VegetableViewModel> GetAll();

VegetableViewModel GetByName(string name);

VegetableViewModel Get(int id);

bool Create(VegetableViewModel viewModel);

bool Delete(int id);

bool Delete(string name);

bool Edit(int id, VegetableViewModel viewModel);

bool Edit(string name, VegetableViewModel viewModel);

}

}