**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 №8

по дисциплине: «Объектно-ориентированное проектирование и   
программирование»

на тему: **«**Технологии доступа к данным *LINQ***»**

Выполнил: студент гр. ИТИ-21

Бахонько А. М.

Принял: преподаватель

Башаримов Ю. С.

Гомель 2024

**Цель работы**: Изучить технологии *LINQ*.

**Задание:**

* Реализовать *CRUD* операции, используя БД из 7-ой работы и технологию *LINQ to SQL* или *LINQ to DataSets*.
* Разработать 2-3 запроса на фильтрацию данных по разным полям на свое усмотрение (Например, запросы на ограничения, функции агрегации и группировки) и реализовать их, используя технологию *LINQ to Object*.
* 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** |
| 2 | Памятка дачнику-овощеводу: Вид овоща (картофель, помидор, огурец, перец, редис, салат), сорт, высота растения, рекомендуемая дата посадки, время сбора урожая |

**Ход работы**

На основании предметной области разработаем базу данных MemoVegeta-ble в СУБД MSSQL, состоящую из 4 таблиц (Vegetable, Type, Plant, Harvest). Диаграмма базы данных представлена на рисунке 1.

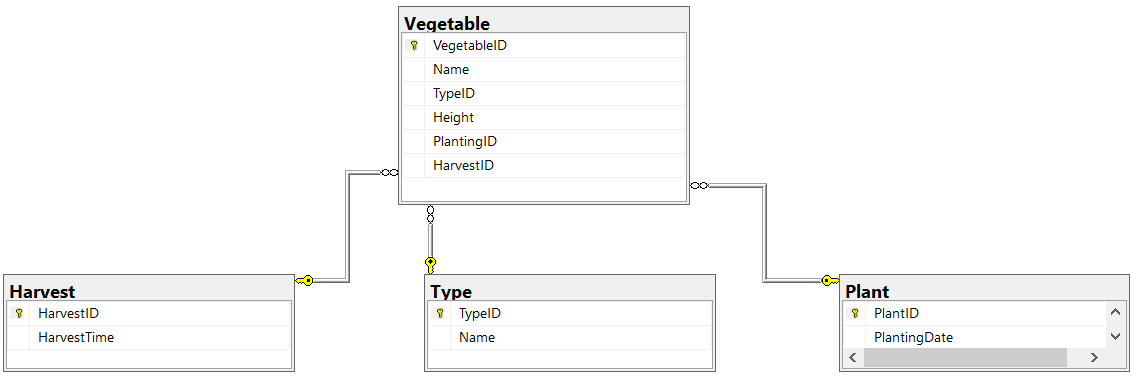


Рисунок 1 – Диаграмма базы данных

Далее разработаем *WPF*-приложение для работы с созданной базой данных *MemoVegetable,* реализуя *CRUD* операции используя технологию *LINQ to DataSets.*

На рисунке 2 показана работа графического приложения *WPF* при запуске*.*

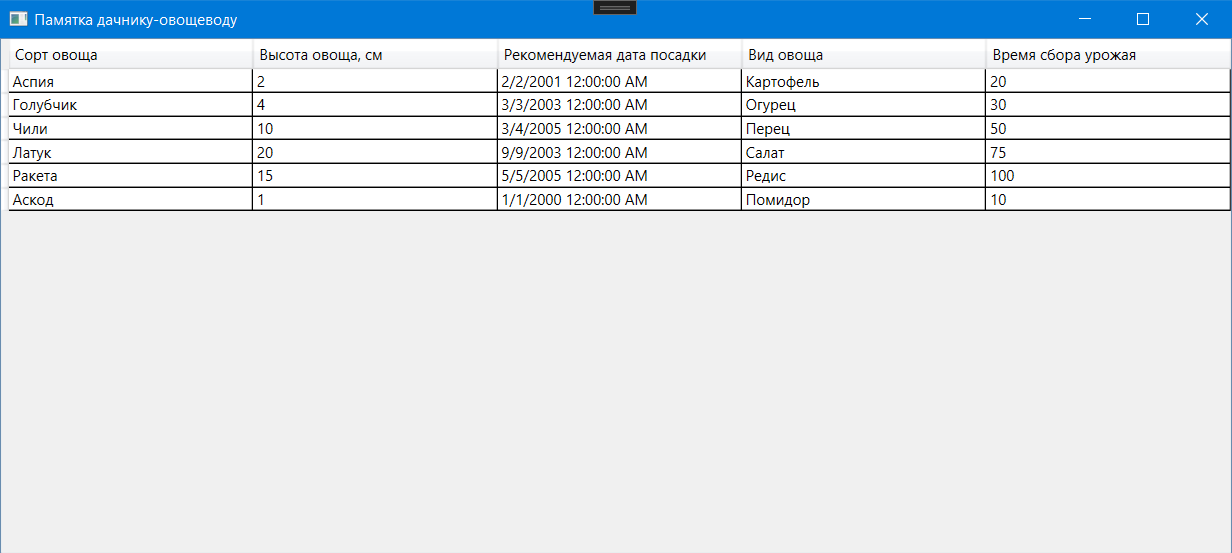


Рисунок 2 – *WPF*-приложения при запуске

На рисунке 3 показана работа графического приложения *WPF* при нажатии кнопки *«F1».*

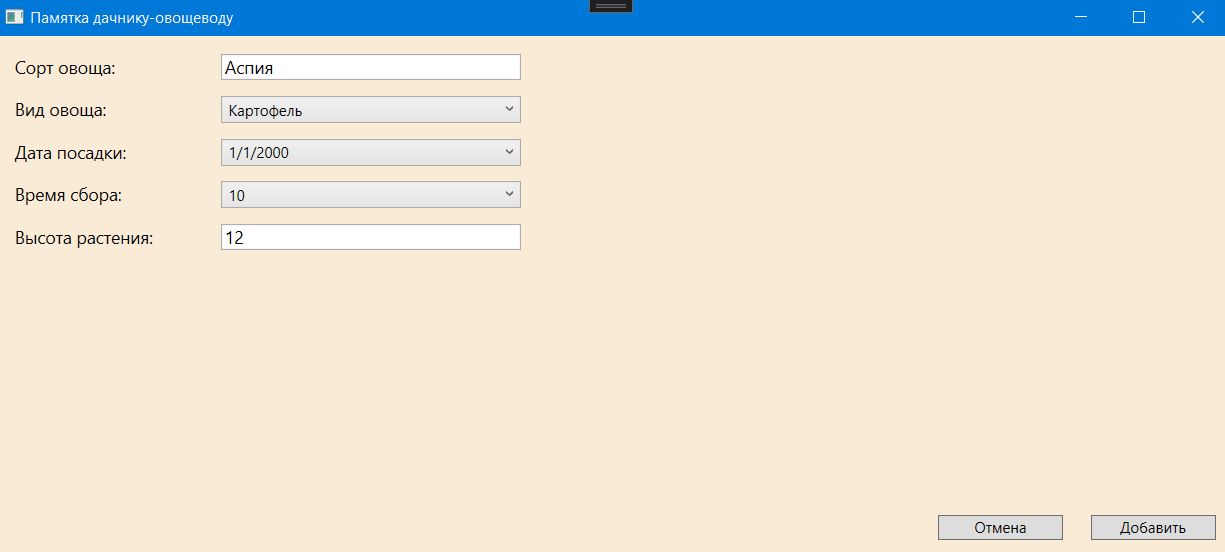


Рисунок 3 – Скриншот работы *WPF*-приложения при нажатии   
кнопки *«F1»*

На рисунке 4 показана работа графического приложения *WPF* при нажатии кнопки *«F2».*

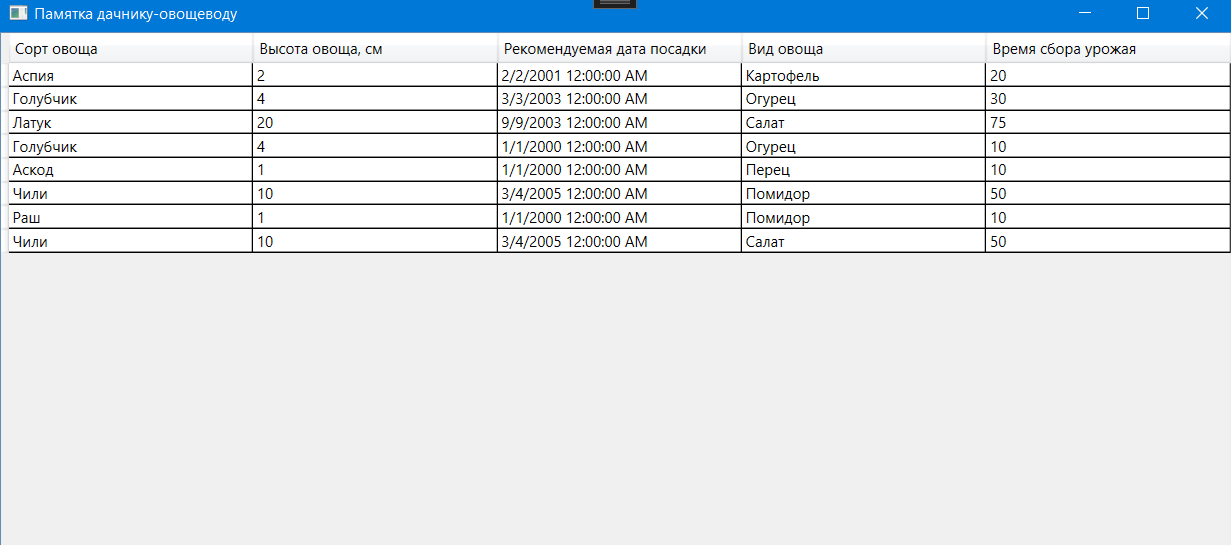


Рисунок 4 – Скриншот работы *WPF*-приложения при нажатии   
кнопки *«F2»*

На рисунке 5 показана работа графического приложения *WPF* при нажатии кнопки *«F3».*

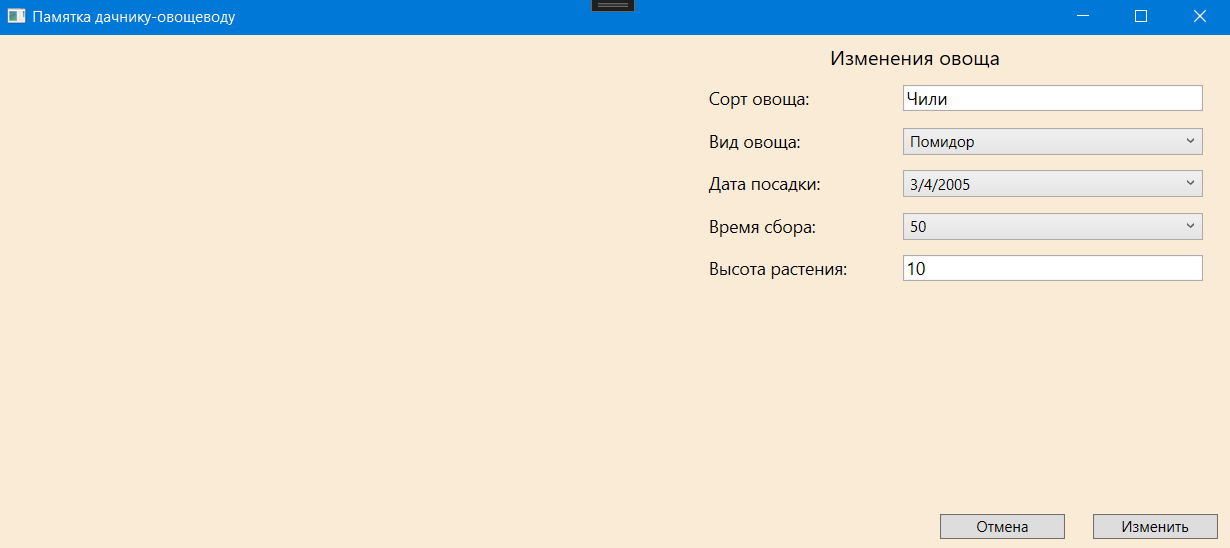


Рисунок 5 – Скриншот работы *WPF*-приложения при нажатии   
кнопки *«F3»*

На рисунке 6 показана работа графического приложения *WPF* при нажатии кнопки *«F4».*



Рисунок 6 – Скриншот работы *WPF*-приложения при нажатии   
кнопки *«F4»*

На рисунке 7 показана работа графического приложения *WPF* при нажатии кнопки *«F5».*



Рисунок 7 – Скриншот работы *WPF*-приложения при нажатии   
кнопки *«F5»*

На рисунке 8 показана работа графического приложения *WPF* при нажатии кнопки *«F6».*

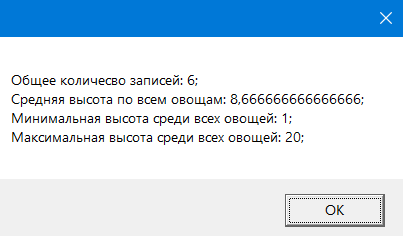


Рисунок 8 – Скриншот работы *WPF*-приложения при нажатии   
кнопки *«F6»*

На рисунке 9 описан журнал ветвей локального репозитория *Git*.

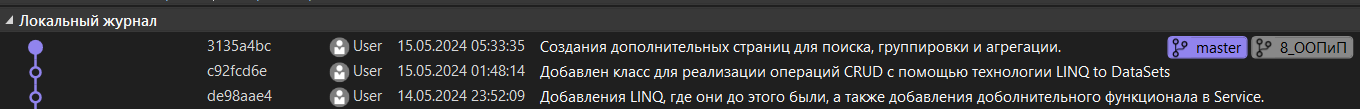


Рисунок 9 – Скриншот журнала ветвей репозитория *Git*

В приложении А представлен код программы библиотеки приложения.

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

**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>

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

AutoGeneratingColumn="OnAutoGeneratingColumn"

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

CanUserResizeColumns="False"

CanUserResizeRows="False"/>

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

</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.ViewModels;

using Memo.Domain;

using Memo.Service.Implementations;

using Memo.Service.Interfaces;

using System.Windows.Input;

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();

this.KeyDown += new KeyEventHandler(MainWindow\_KeyDown);

IDbContext dbContext = new SqlContextLinq();

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);

dataGrid.ItemsSource = \_vegetableService.GetAll();

}

private void OnAutoGeneratingColumn(object sender, DataGridAutoGeneratingColumnEventArgs e)

{

if (e.PropertyDescriptor is PropertyDescriptor descriptor)

{

e.Column.Header = descriptor.DisplayName;

}

}

private void MainWindow\_KeyDown(object sender, KeyEventArgs e)

{

switch (e.Key)

{

case (Key.F1):

{

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

dataGrid.ItemsSource = \_vegetableService.GetAll();

}

break;

case (Key.F2):

{

VegetableViewModel vegetableViewModel = (dataGrid.SelectedItem as VegetableViewModel)!;

if (vegetableViewModel != null)

{

\_vegetableService.Delete(vegetableViewModel.Name);

dataGrid.ItemsSource = \_vegetableService.GetAll();

}

}

break;

case (Key.F3):

{

VegetableViewModel vegetableViewModel = (dataGrid.SelectedItem as VegetableViewModel)!;

if (vegetableViewModel != null)

{

\_vegetableService.Delete(vegetableViewModel.Name);

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

}

}

break;

case (Key.F4):

{

SearchPage searchPage = new(\_vegetableService)

{

DataContext = this

};

PageFrame.Content = searchPage;

}

break;

case (Key.F5):

{

GroupPage groupPage = new(\_vegetableService, \_typeService, \_harvestService)

{

DataContext = this

};

PageFrame.Content = groupPage;

}

break;

case (Key.F6):

{

MessageBox.Show($"Общее количесво записей: {\_vegetableService.Count()};\n" +

$"Средняя высота по всем овощам: {\_vegetableService.HeightAvg()};\n" +

$"Минимальная высота среди всех овощей: {\_vegetableService.HeightMin()};\n" +

$"Максимальная высота среди всех овощей: {\_vegetableService.HeightMax()};\n");

}

break;

case (Key.Tab):

{

dataGrid.ItemsSource = \_vegetableService.GetAll();

}

break;

}

}

}

}

**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>

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

using Memo.Domain;

using System.Configuration;

using System.Data.SqlClient;

using System.Data;

using Memo.Domain.Models;

namespace Memo.DAL.ADO.Net;

public class SqlContextLinq : IDbContext

{

private readonly SqlConnection \_connection;

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 DataTable \_dtVegetable;

private readonly DataTable \_dtType;

private readonly DataTable \_dtPlanting;

private readonly DataTable \_dtHarvest;

public SqlContextLinq()

{

string dbSettings = ConfigurationManager.ConnectionStrings["DB"].ConnectionString;

\_connection = new SqlConnection(dbSettings);

SqlDataAdapter \_adapterVegetable = new("SELECT \* FROM Vegetable", \_connection);

SqlDataAdapter \_adapterType = new("SELECT \* FROM Type", \_connection);

SqlDataAdapter \_adapterPlanting = new("SELECT \* FROM Plant", \_connection);

SqlDataAdapter \_adapterHarvest = new("SELECT \* FROM Harvest", \_connection);

\_dtVegetable = new();

\_dtType = new();

\_dtPlanting = new();

\_dtHarvest = new();

\_connection.Open();

\_adapterVegetable.Fill(\_dtVegetable);

\_adapterType.Fill(\_dtType);

\_adapterPlanting.Fill(\_dtPlanting);

\_adapterHarvest.Fill(\_dtHarvest);

\_connection.Close();

SelectVegetable();

}

private void SelectType()

{

var queryTypes = \_dtType.AsEnumerable().Select(types => new

{

Id = types.Field<int>("TypeID"),

TypeV = types.Field<string>("Name"),

});

Type = [];

foreach (var query in queryTypes)

{

if (query.TypeV != null)

{

Type.Add(new Domain.Type

{

Id = query.Id,

TypeV = query.TypeV,

});

}

}

}

private void SelectPlanting()

{

var queryPlantings = \_dtPlanting.AsEnumerable().Select(plantings => new

{

Id = plantings.Field<int>("PlantID"),

PlantingTime = plantings.Field<DateTime>("PlantingDate"),

});

Planting = [];

foreach (var query in queryPlantings)

{

if (query.PlantingTime != null)

{

Planting.Add(new Planting

{

Id = query.Id,

PlantingTime = Convert.ToDateTime(query.PlantingTime),

});

}

}

}

private void SelectHarvest()

{

var queryHarvests = \_dtHarvest.AsEnumerable().Select(harvests => new

{

Id = harvests.Field<int>("HarvestID"),

HarvestTime = harvests.Field<int>("HarvestTime"),

});

Harvest = [];

foreach (var query in queryHarvests)

{

if (query.HarvestTime != 0)

{

Harvest.Add(new Harvest

{

Id = query.Id,

HarvestTime = query.HarvestTime,

});

}

}

}

private void SelectVegetable()

{

SelectType();

SelectPlanting();

SelectHarvest();

var queryVegetable = \_dtVegetable.AsEnumerable().Select(vegetable => new

{

Id = vegetable.Field<int>("VegetableID"),

Name = vegetable.Field<string>("Name"),

Type = vegetable.Field<int>("TypeID"),

HeightSm = vegetable.Field<decimal>("Height"),

Planting = vegetable.Field<int>("PlantingID"),

Harvest = vegetable.Field<int>("HarvestID")

});

Vegetable = [];

foreach (var query in queryVegetable)

{

if (query.Name != null)

{

Domain.Type qType = Type.Find(x => x.Id == query.Type)!;

Planting qPlanting = Planting.Find(x => x.Id == query.Planting)!;

Harvest qHarvest = Harvest.Find(x => x.Id == query.Harvest)!;

Vegetable.Add(new Vegetable

{

Id = query.Id,

Name = query.Name,

Type = qType,

HeightSm = Convert.ToDouble(query.HeightSm),

Planting = qPlanting,

Harvest = qHarvest,

});

}

}

}

private void DeleteAll()

{

SqlDataAdapter adapter = new()

{

DeleteCommand = new("DELETE FROM Vegetable", \_connection)

};

adapter.DeleteCommand.ExecuteNonQuery();

\_dtVegetable.Clear();

adapter.Update(\_dtVegetable);

\_dtVegetable.AcceptChanges();

}

private void InsertIntoType()

{

foreach (Domain.Type type in Type)

{

DataRow row = \_dtType.NewRow();

row["[Name]"] = type.TypeV;

\_dtType.Rows.Add(row);

}

SqlDataAdapter adapter = new()

{

InsertCommand = new("INSERT INTO Type ([Name]) " +

"VALUES (@Name)", \_connection)

};

adapter.InsertCommand.Parameters.Add("@Name", SqlDbType.VarChar, 50, "[Name]");

adapter.Update(\_dtType);

\_dtType.AcceptChanges();

}

private void InsertIntoPlanting()

{

foreach (Planting planting in Planting)

{

DataRow row = \_dtPlanting.NewRow();

row["PlantingDate"] = planting.PlantingTime;

\_dtPlanting.Rows.Add(row);

}

SqlDataAdapter adapter = new()

{

InsertCommand = new("INSERT INTO Plant (PlantingDate) " +

"VALUES (@Date)", \_connection),

};

adapter.InsertCommand.Parameters.Add("@Date", SqlDbType.Date, 100, "PlantingDate");

adapter.Update(\_dtPlanting);

\_dtPlanting.AcceptChanges();

}

private void InsertIntoHarvest()

{

foreach (Harvest harvest in Harvest)

{

DataRow row = \_dtHarvest.NewRow();

row["HarvestTime"] = harvest.HarvestTime;

\_dtHarvest.Rows.Add(row);

}

SqlDataAdapter adapter = new()

{

InsertCommand = new("INSERT INTO Harvest (HarvestTime) " +

"VALUES (@Time)", \_connection),

};

adapter.InsertCommand.Parameters.Add("@Time", SqlDbType.Int, 10, "HarvestTime");

adapter.Update(\_dtHarvest);

\_dtHarvest.AcceptChanges();

}

private void InsertIntoVegetable()

{

foreach (Vegetable vegetable in Vegetable)

{

vegetable.Type = Type.Find(s => s.TypeV == vegetable.Type?.TypeV);

vegetable.Planting = Planting.Find(s => s.PlantingTime == vegetable.Planting?.PlantingTime);

vegetable.Harvest = Harvest.Find(s => s.HarvestTime == vegetable.Harvest?.HarvestTime);

DataRow row = \_dtVegetable.NewRow();

row["Name"] = vegetable.Name;

row["TypeID"] = vegetable.Type?.Id;

row["Height"] = vegetable.HeightSm;

row["PlantingID"] = vegetable.Planting?.Id;

row["HarvestID"] = vegetable.Harvest?.Id;

\_dtVegetable.Rows.Add(row);

}

SqlDataAdapter adapter = new()

{

InsertCommand = new("INSERT INTO Vegetable (Name, TypeID, Height, PlantingID, HarvestID) " +

"VALUES (@Name, @TypeID, @HeightSm, @PlantingID, @HarvestID)", \_connection)

};

adapter.InsertCommand.Parameters.Add("@Name", SqlDbType.VarChar, 50, "Name");

adapter.InsertCommand.Parameters.Add("@TypeID", SqlDbType.Int, 4, "TypeID");

adapter.InsertCommand.Parameters.Add("@HeightSm", SqlDbType.Decimal, 18, "Height");

adapter.InsertCommand.Parameters.Add("@PlantingID", SqlDbType.Int, 4, "PlantingID");

adapter.InsertCommand.Parameters.Add("@HarvestID", SqlDbType.Int, 4, "HarvestID");

adapter.Update(\_dtVegetable);

\_dtVegetable.AcceptChanges();

}

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)

{

Harvest? harvestToUpdate = \_dbContext.Harvest.Find(s => s == oldHarvest);

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)

{

Planting? plantingToUpdate = \_dbContext.Planting.Find(s => s == oldPlanting);

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)

{

Domain.Type? varietyToUpdate = \_dbContext.Type.Find(s => s == oldType);

if (varietyToUpdate != null)

{

varietyToUpdate.TypeV = newType.TypeV;

\_dbContext.SaveChanges();

return true;

}

else

{

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

}

}

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)

{

Vegetable? vegetableToUpdate = \_dbContext.Vegetable.Find(vegetable => vegetable == oldVegetable);

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

{

Vegetable? vegetable = \_vegetableRepository.ReadAll().FirstOrDefault(vegetable => vegetable.Name == name);

ArgumentNullException.ThrowIfNull(vegetable);

return \_vegetableRepository.Delete(vegetable);

}

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

{

Vegetable? vegetable = \_vegetableRepository.ReadAll().FirstOrDefault(vegetable => vegetable.Id == id);

ArgumentNullException.ThrowIfNull(vegetable);

return \_vegetableRepository.Delete(vegetable);

}

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

{

Vegetable? oldVegetable = \_vegetableRepository.ReadAll().FirstOrDefault(vegetable => vegetable.Id == id);

Vegetable newVegetable = vegetableViewModel;

ArgumentNullException.ThrowIfNull(oldVegetable);

ArgumentNullException.ThrowIfNull(newVegetable);

return \_vegetableRepository.Update(oldVegetable, newVegetable);

}

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

{

Vegetable? oldVegetable = \_vegetableRepository.ReadAll().FirstOrDefault(vegetable => vegetable.Name == name);

ArgumentNullException.ThrowIfNull(oldVegetable);

Vegetable newVegetable = vegetableViewModel;

ArgumentNullException.ThrowIfNull(newVegetable);

newVegetable.Id = oldVegetable.Id;

return \_vegetableRepository.Update(oldVegetable, newVegetable);

}

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

{

Vegetable? vegetable = \_vegetableRepository.ReadAll().FirstOrDefault(vegetable => vegetable.Id == id);

ArgumentNullException.ThrowIfNull(vegetable);

VegetableViewModel vegetableViewModel = vegetable;

return vegetableViewModel;

}

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

{

Vegetable? vegetable = \_vegetableRepository.ReadAll().FirstOrDefault(vegetable => vegetable.Name == name);

ArgumentNullException.ThrowIfNull(vegetable);

VegetableViewModel vegetableViewModel = vegetable;

return vegetableViewModel;

}

catch (ArgumentNullException)

{

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

}

catch (Exception ex)

{

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

}

}

public List<VegetableViewModel> FindByName(string name)

{

if (name == null)

{

return new List<VegetableViewModel>();

}

return GetAll().FindAll(vegetable => vegetable.Name == name);

}

public List<VegetableViewModel> GroupByType(string name)

{

var Groups = from vegetable in GetAll()

group vegetable by vegetable.TypeName

into vegetableGroups

select new

{

Name = vegetableGroups.Key,

Count = vegetableGroups.Count(),

Vegetable = from vegetableGroup in vegetableGroups

select vegetableGroup

};

return Groups.Where(x => x.Name == name).Select(x => x.Vegetable).First().ToList();

}

public List<VegetableViewModel> GroupByHarvest(string name)

{

var Groups = from vegetable in GetAll()

group vegetable by vegetable.HarvestTime

into vegetableGroups

select new

{

Name = vegetableGroups.Key,

Count = vegetableGroups.Count(),

Vegetable = from vegetableGroup in vegetableGroups

select vegetableGroup

};

return Groups.Where(x => x.Name.ToString() == name).Select(x => x.Vegetable).First().ToList();

}

public int Count()

{

return GetAll().Count();

}

public double HeightAvg()

=> GetAll().Average(x => x.HeightSm);

public double HeightMin()

=> GetAll().Min(x => x.HeightSm);

public double HeightMax()

=> GetAll().Max(x => x.HeightSm);

}

}

**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);

public List<VegetableViewModel> FindByName(string name);

public List<VegetableViewModel> GroupByType(string name);

public List<VegetableViewModel> GroupByHarvest(string name);

public int Count();

double HeightAvg();

double HeightMin();

double HeightMax();

}

}