МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РФ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ ЛИПЕЦКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ

Институт компьютерных наук
Кафедра автоматизированных систем управления

ОТЧЕТ

о производственной практике

"Проектно-технологическая практика"

в ООО "ОНСОФТ"

Студент АС-21-1 Станиславчук С.М.

Руководитель от кафедры Болдырихин О.В.

Руководитель ВКР
канд. техн. наук Гаев Л.В.

Руководитель от предприятия Кретов А.Н.

Липецк 2024 г.

Липецкий государственный технический университет

Институт компьютерных наук

Кафедра автоматизированных систем управления

ЗАДАНИЕ НА ПРОИЗВОДСТВЕННУЮ ПРАКТИКУ

Студенту __Станиславчуку Сергею Михайловичу_____группы ____АС-21-1_____

Направление (специальность) _09.03.01 "Информатика и вычислительная техника"

Изучить информацию по общей программе практики:

1. Структуру, деятельность, продукты для анализа бизнес-процессов, задачи отдела разработки.

2. Используемые на предприятии и в подразделении средства автоматизации.

3. Действующий порядок разработки и использования средств автоматизации: техническую политику отдела разработки в области систем автоматизации, инструменты реализации технической политики, требования к компонентам систем автоматизации и т.п.

Выполнить индивидуальное задание по разработке прототипа клиент-приложения информационной системы учета медицинских инструментов на C#:

1. Ознакомиться с паттерном проектирования MVVM.

2. Разработать редактируемый список мед. инструментов.

3. Разработать редактируемые списки сотрудников и мест хранения инструментов.

Руководитель практики от ЛГТУ Старший преподаватель Болдырихин О.В._____

Задание принял к исполнению студент ____Станиславчук С.М._____

Аннотация

С. 27. Ил. 7. Литература 4 назв. Прил. 2.

Документ включает в себя описание характеристик предприятия и выполнение задания, выданного руководителем практики. Содержание указанных разделов соответствует стандартам ЕСПД (Единая система программной документации) соответствующих наименований. В работе рассмотрены основные этапы выполнения задания, включая разработку программного обеспечения на языке C# с использованием технологий Avalonia, dotNET, Git.

Оглавление

Введение

Для организации разработки было принято решение использовать разделение на задачи. Таким образом были выделены следующие блоки для реализации:

1. Понять принципы работы паттерна MVVM

2. Изучить фреймворк Avalonia.

3. Написать модели, бизнес-логику и представление редактируемых списков мед. инструментов, мест хранения и сотрудников.

Реализация третьего пункта подразумевает использование следующих инструментов: для ведения контроля версий - Git, для упрощения написания кода использовался редактор текста Nvim.

1. Краткое описание подразделения – места практики

1.1. Характеристика деятельности организации

Общество с ограниченной ответственность «ОНСОФТ» - "охватывают все этапы разработки программного обеспечения, включая анализ требований, проектирование архитектуры, разработку, тестирование, внедрение и поддержку".

Разработка программного обеспечения для retail оборудования: весы, принтеры, сканеры, QR-дисплеи:

- Кроссплатформенные драйвера
- Android приложения и сервисы
- Веб-системы мониторинга и управления парком оборудования
- Системы разворачивания и доставки обновлений
- Драйвера библиотек подключаемого оборудования 1С, с возможностью последующей сертификации

Разработка и поддержка высоконагруженных web приложений

Технологический стек:

- Python, TypeScript, JavaScript, Kotlin, C++
- React, Vue, Vite
- PostgreSQL, Redis
- Docker, Nginx

Разработка программного обеспечения для ККТ

Основные направления разработки:

- Фискальное ядро ККТ
- Кроссплатформенные драйвера ККТ
- Плагины и обертки над существующими драйверами для интеграции во фронт-офисные решения
- Приложения для ОС "Эвотор", включая публикацию и поддержку в магазине приложений

Отдел прохождения практики — отдел разработки ритейл решений.

2. Описание выполнения общей программы

2.1. Описание паттерна проектирования MVVM.

Паттерн MVVM (Model-View-ViewModel) позволяет отделить логику приложения от визуальной части (представления). Данный паттерн является архитектурным, то есть он задает общую архитектуру приложения.

MVVM состоит из трех компонентов: модели (Model), модели представления (ViewModel) и представления (View).

Модель описывает используемые в приложении данные. Модели могут содержать логику, непосредственно связанную этими данными, например, логику валидации свойств модели. В то же время модель не должна содержать никакой логики, связанной с отображением данных и взаимодействием с визуальными элементами управления. Прямое взаимодействие между моделью и представлением отсутствует.

View или представление определяет визуальный интерфейс, через который пользователь взаимодействует с приложением. Применительно к WPF представление - это код в xaml, который определяет интерфейс в виде кнопок, текстовых полей и прочих визуальных элементов. Хотя окно (класс Window) в WPF может содержать как интерфейс в xaml, так и привязанный к нему код C#, однако в идеале код C# не должен содержать какой-то логики, кроме разве что конструктора, который вызывает метод InitializeComponent и выполняет начальную инициализацию окна. Вся же основная логика приложения выносится в компонент ViewModel.

ViewModel или модель представления связывает модель и представление через механизм привязки данных. Если в модели изменяются значения свойств, при реализации моделью интерфейса INotifyPropertyChanged автоматически идет изменение отображаемых данных в представлении, хотя напрямую модель и представление не связаны. ViewModel также содержит логику по получению данных из

модели, которые потом передаются в представление. И также VewModel определяет логику по обновлению данных в модели. Поскольку элементы представления, то есть визуальные компоненты типа кнопок, не используют события, то представление взаимодействует с ViewModel посредством команд.

Итогом применения паттерна MVVM является функциональное разделение приложения на три компонента, которые проще разрабатывать и тестировать, а также в дальнейшем модифицировать и поддерживать.

2.2. Обзор Avalonia

Avalonia - это фреймворк пользовательского интерфейса для создания кроссплатформенных приложений на .NET. Он использует собственную технологию отрисовки элементов, что обеспечивает единый внешний вид и обработку событий на разных платформах, таких как Windows, macOS, Linux, Android, iOS и WebAssembly.

Приложения на Avalonia написаны на C#, что позволяют быстро создавать прототипы приложений, которые со временем могут стать полноценными приложениями. В отличии от других инструментов, которые могут быть ограничены своими API или производительностью, Avalonia имеет доступ ко всем возможностям выбранной платформы, а также обеспечивает высокую производительность.

Поскольку Avalonia используется для создания кроссплатформенных приложений, то распространен подход по создание Shared-проекта для создания общей кодовой базы, на которую потом ссылаются проекты под конкретную платформу.

Avalonia включает два варианта описания пользовательского интерфейса. Первый - описание элементов в коде посредствам обращения к Avalonia API. Второй - использовать XAML (Extensible Application Markup Language - расширяемый язык разметки приложений).

3. Описание выполнения индивидуального задания

Прежде чем начать работу с языком C#, стоит установить набор для разработки — dotNET SDK. SDK представляет собой пакет инструментов для разработки программного обеспечения.

Первым делом скачать SDK версии 8. Так как это версия самая стабильная на данный момент. В Arch Linux это можно сделать при помощи команды «pacman -Sy dotnet-sdk dotnet-runtime».

Для разработки понадобится Git. Для того, чтобы установить его, следует открыть командную строку Linux и ввести соответствующую команду (Arch Linux): «pacman -Sy git» (рисунок 1).

Рисунок 1 – Установка Git при помощи менеджера пакетов pacman

После завершения установки можно перейти к написанию скриптов проекта. Для этого создадим проект avalonia командами «dotnet new install Avalonia.Templates» и «dotnet new avalonia.mvvm -o MedTools» [2]. Чтобы запустить проект, нужно выполнить команду «dotnet run».

Проект разделен на директории модулей: Model, ViewModel, View, а также дополнительные — Static и Data. Структура проекта показана на рисунке 1.

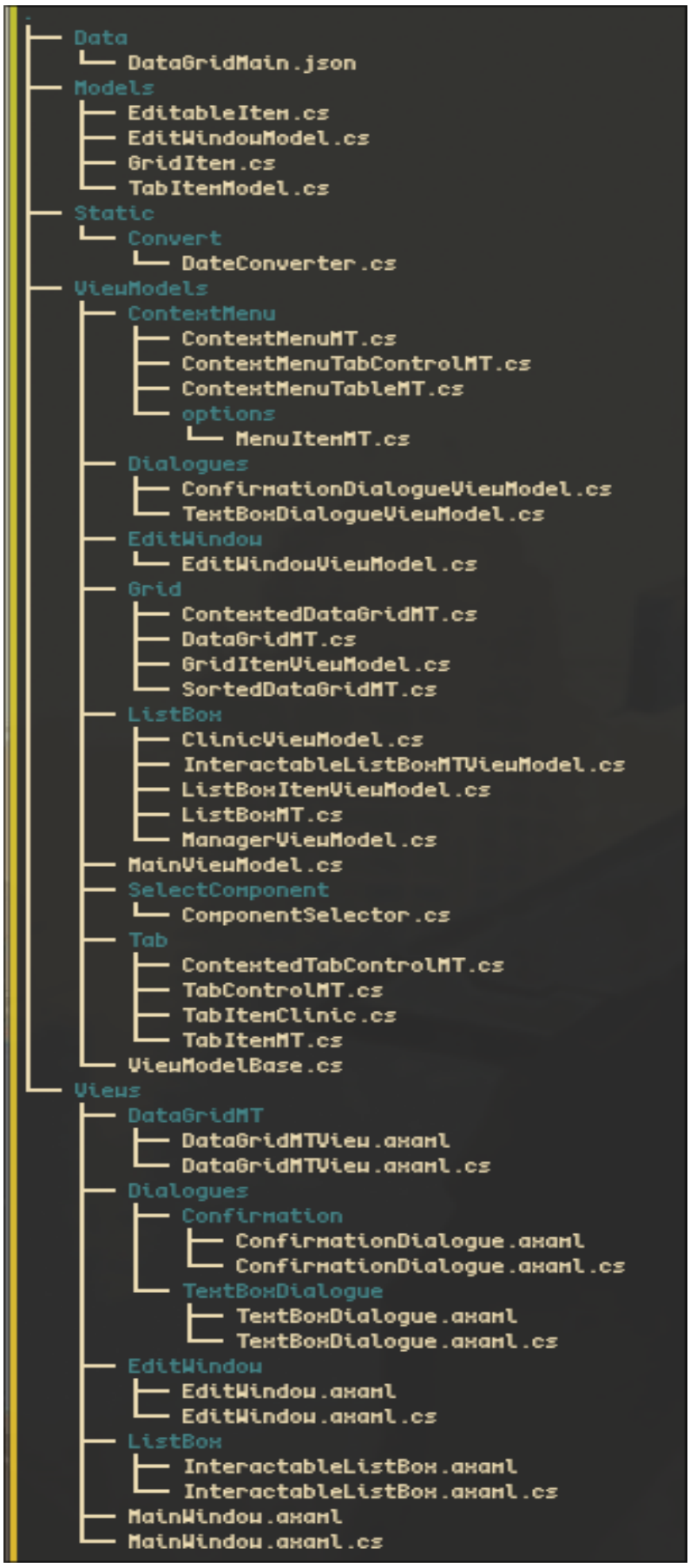

Рисунок 1 – Структура проекта

На данный момент, для выполнения задания буду работать с

кластером файлов DataGridMT. Помимо отображения списков инструментов, необходимо предусмотреть возможность сортировки отображаемых наборов по столбцам, изменение их ширины, добавление, редактирование, очистка, удаление записей. Команды по типу добавления, очистки и удаления вызываются из контекстного меню. Редактирование инструмента выполняется в отдельном окне, с последующим сохранением сделанных изменений.

Экземпляр окна с отображаемым списком инструментов (DataGridMT) представлен на рисунке 2.

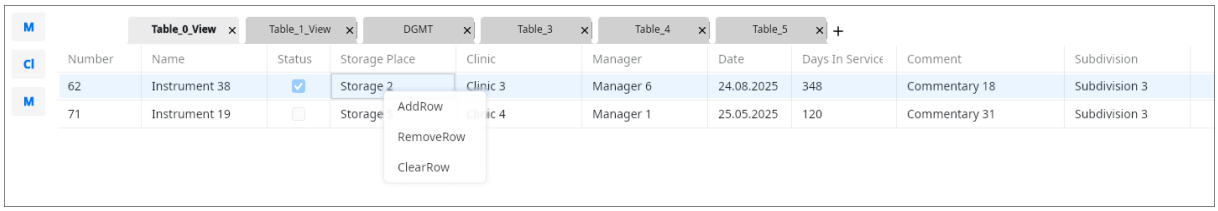

Рисунок 2 – Окно со списком инструментов (активно контекстное меню)

Модальное окно для редактирования характеристик медицинского набора инструментов представлен на рисунке 3.

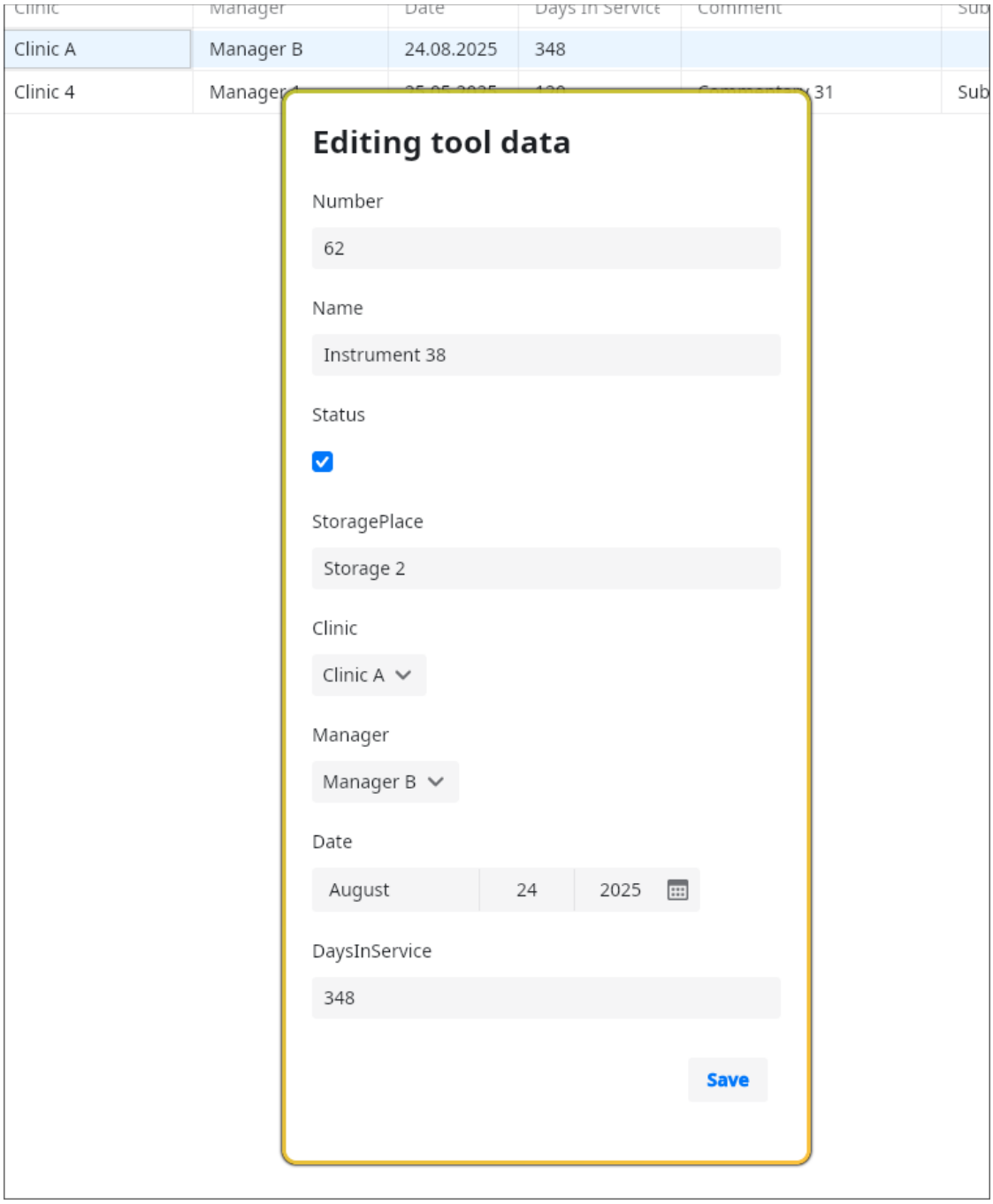

Рисунок 3 – Модальное окно редактирования инструмента

Для страницы сотрудников необходимо отображать список всех сотрудников. Результат представлен на рисунке 4.

Рисунок 4 – Редактируемый список сотрудников

Помимо отображения информации, необходимо предусмотреть возможность добавления, удаления отображаемых сотрудников с подтверждением в модальном диалоговом окне. Вызов окон происходит по нажатии соответствующих кнопок «+» и «-». Вызов окна с добавлением нового менеджера представлен на рисунке 5.

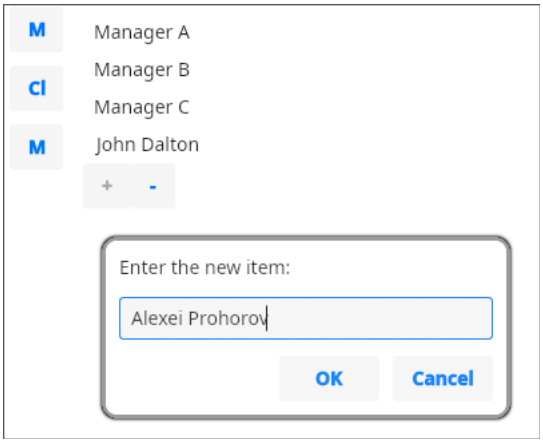

Рисунок 5 – Окно добавления нового элемента в список

Вызов окна удаления менеджера представлен на рисунке 6.

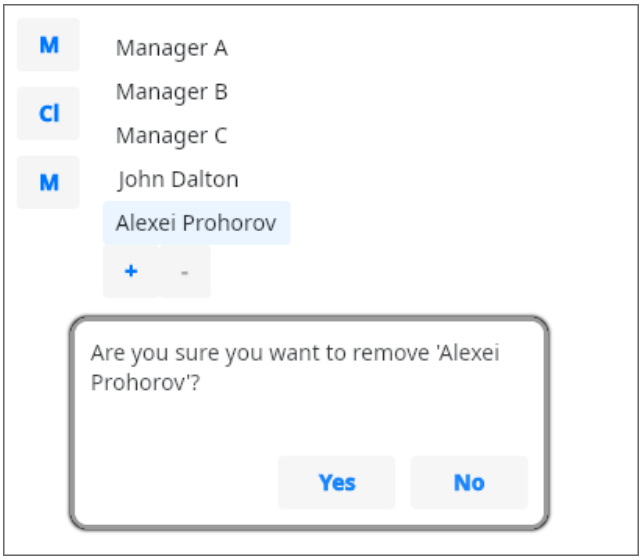

Рисунок 6 – Окно удаления элемента из списка

4. ВКР

4.1 Постановка задачи

Медицинские инструменты являются неотъемлемой частью работы любой клиники или больницы. Однако, учитывая их высокую стоимость, клиники нередко предпочитают арендовать инструменты, а не приобретать их в собственность. Это связано с тем, что многие инструменты являются многоразовыми, и их состояние поддерживается специальным персоналом. Однако процесс учета, транспортировки и контроля состояния инструментов часто занимает значительное время, что может негативно сказаться на оперативности оказания медицинской помощи.

В условиях, когда каждая минута на счету, особенно в экстренных ситуациях, сокращение времени ожидания поставки медицинских инструментов может стать решающим фактором для спасения жизни пациента. Поэтому разработка программы, которая автоматизирует процессы учета, транспортировки и контроля состояния медицинских инструментов, является актуальной и крайне важной задачей.

Изучая существующие приложения, были найдены только веб-сервисы, требующие постоянного подключения к сети интернет. Мое приложение не будет требовать постоянного подключения к сети: только в момент идентификации и аутентификации. Такой функционал как редактирование списков, добавление и удаление новых элементов будет всё еще сохранять свое состояние, используя сохранение на локальном устройстве пользователя, а при восстановлении соединения данные будут синхронизованы с удаленной базой данных.

4.2 Решаемые задачи

1. Учет хранимых медицинских инструментов

Ведение актуального учета всех медицинских инструментов, находящихся на хранении в клинике или на складе. Это включает информацию о каждом инструменте: тип, серийный номер, состояние, срок

аренды, дата поступления, история использования и другие параметры.

Система регистрирует каждый инструмент в базе данных, обновляет информацию о его состоянии и предоставляет доступ к данным в режиме реального времени.

2. Запросы медицинских инструментов

Система предоставляет интерфейс для создания запросов на поставку инструментов. Запросы могут быть обработаны пользователем с учетом наличия инструментов на складе и их местоположения.

Система регистрирует каждый инструмент в базе данных, обновляет информацию о его состоянии и предоставляет доступ к данным в режиме реального времени.

3. Журналирование перемещений инструментов

Ведение подробного журнала всех перемещений медицинских инструментов между клиниками, складами и другими объектами. Это включает информацию о дате, времени, месте отправки и получения, а также о лицах, ответственных за перемещение.

Система автоматически фиксирует каждое перемещение инструментов, сохраняя данные в журнале. Это позволяет отслеживать историю движения инструментов и контролировать их местоположение.

4.3 Диаграмма использования

Диаграмма использования АИС «Учет медицинских инструментов» представлена на рисунке 6.

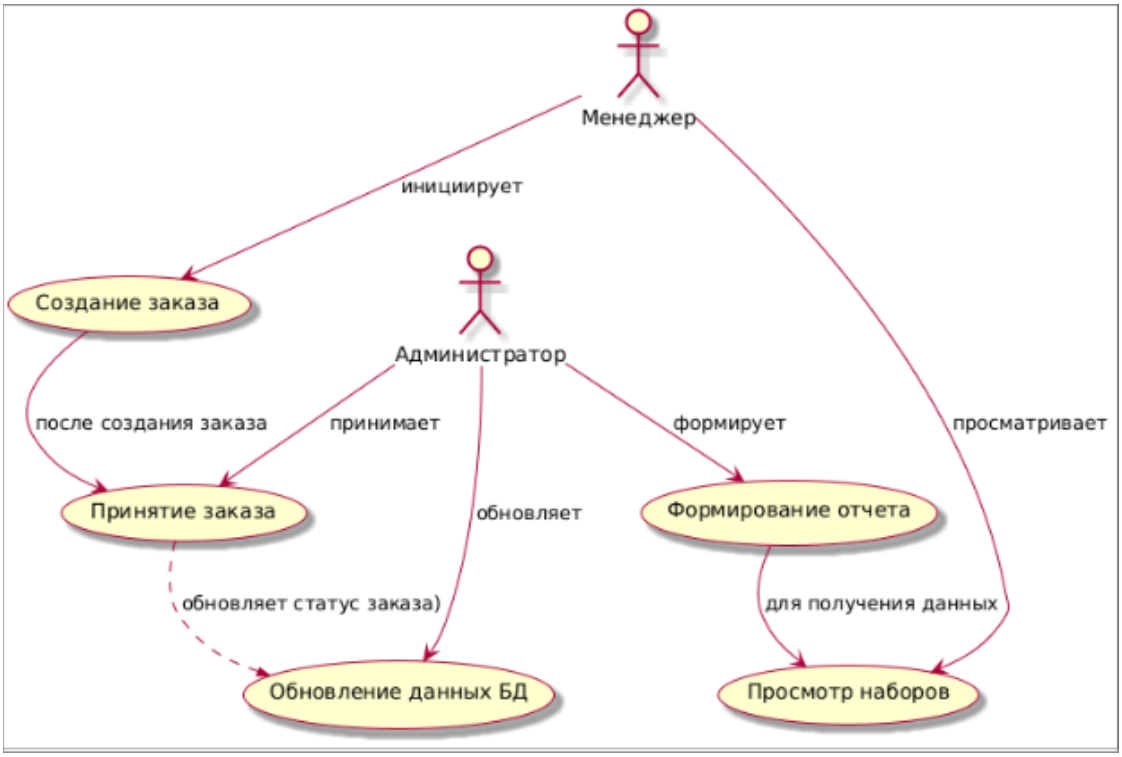

Рисунок 7 – Диаграмма использования

5. Описание информации, полученной на практике для выполнения ВКР

В ходе прохождения практики в отделе разработки мной были получены практические навыки разработки приложений с использованием фреймворка Avalonia и языка программирования C#. Данное техническое решение обладает встроенной поддержкой кроссплатформенности, что позволит при разработки информационной системы писать единую кодовую базу для разных платформ с дальнейшей возможностью сборки приложения под все существующие платформы.

Также в ходе практики был получен опыт работы с приложением, использующим архитектуру MVVM. Используя такой подход, каждый модуль системы является независимым, что упрощает разработку, так как модели, бизнес-логика и отображение слабо связаны между собой и изменение кода одного компонента редко требует изменения работы другого.

Заключение

По итогам прохождения производственной практики в отделе разработки были получены навыки разработки C# и Avalonia для приложений под Windows. Была поставлена задача ВКР.

Библиографический список

1. Методические указания к производственной практике и выпускной квалификационной работе бакалавра [Текст] – Липецк: Издательство Липецкого государственного технического университета, 2024. – 32 с

2. Avalonia Documentation [Электронный ресурс] / Режим доступа к данным: https://docs.avaloniaui.net/docs/, свободный.

3. MVVM Pattern [Электронный ресурс] / Режим доступа к данным: https://docs.avaloniaui.net/docs/concepts/the-mvvm-pattern/, свободный.

4. Git Documentation [Электронный ресурс] / Режим доступа к данным: https://git-scm.com/doc, свободный.

Приложение А

Код модуля бизнес-логики DataGridMT

DataGridMT.cs

```
using Avalonia.Controls;
using System;
using System.Collections.Generic;
using System.IO;
using System.Text.Json;
using MedTools.Models;
using MedTools.Services;
using System.Collections.ObjectModel;
using System.Diagnostics;
using System.Collections.Specialized;

namespace MedTools.ViewModels;

public partial class DataGridMT : DataGrid
{
    // To render this element in MainWindow.axaml
    protected override Type StyleKeyOverride => typeof(DataGrid);
    public ObservableCollection<GridItemViewModel> ItemsCollection
    {
        get
        {
            if (this.ItemsSource is ObservableCollection<GridItemViewModel> items)
            {
                return items;
            }
            else
            {
                return new ObservableCollection<GridItemViewModel>();
            }
        }
    }
    protected new string Name;


    private const bool _isReadOnly = true;
    private const bool _autogenColumns = true;
    private const bool _canUserResizeColumns = true;
    private readonly bool _enableLog;
    private static string _dataFilePath = GetDataFilePath();
    protected GridItemViewModel _selectedItem
    {
        get
        {
            if (this.SelectedItem is GridItemViewModel item)
            {
                return item;
            }
            return new GridItemViewModel();
        }
    }


    public DataGridMT()
    {
        InitDataGrid();

        Name = "Table_";
    }

    public DataGridMT(string name)
    {
        this.Name = name;
        InitDataGrid();
    }
```

```
public void ReplaceGridItem(GridItemViewModel oldItem, GridItemViewModel newItem)
{
    if (oldItem == null || newItem == null)
    {
        return;
    }
    if (ItemsCollection.Contains(oldItem))
    {
        int id = ItemsCollection.IndexOf(oldItem);
        ItemsCollection[id] = newItem;
    }
}

public void ClearAll()
{
    Logger.Info($"Clearing all rows...");
    var _itemsCollection = new ObservableCollection<GridItemViewModel>(ItemsCollection);
    foreach (var item in _itemsCollection)
    {
        ClearRow(ItemsCollection, item);
    }
}

protected void AddRow()
{
    ItemsCollection.Add(GenerateRandomItem());
    Logger.Info("Row added");
}

protected void AddRow(ObservableCollection<GridItemViewModel> items, GridItemViewModel element)
{
    items.Add(element);
    Logger.Info("Row added");
}

protected void RemoveRow(object? element)
{
    if (element == null)
    {
        Logger.Error($"Element '{element}' is null");
        return;
    }
    if (!(element is GridItemViewModel givm) || !ItemsCollection.Contains(givm))
    {
        Logger.Error($"{element} not type of GridItemViewModel or row not found");
        return;
    }

    Logger.Info(ItemsCollection.Remove(givm) ?
            $"Row '{element}' removed" :
            $"Can't remove row '{element}'");
}

protected bool RemoveRow(ObservableCollection<GridItemViewModel> items, GridItemViewModel element)
{
    if (element == null)
    {
        Logger.Error($"Element '{element}' is null");
        return false;
    }
    if (!items.Contains(element))
    {
        Logger.Error($"Row '{element}' not found");
        return false;
    }

    bool rowRemoved = ItemsCollection.Remove(element);
    Logger.Info(rowRemoved ?
            $"Row '{element}' removed" :
            $"Can't remove row '{element}'");
    return rowRemoved;
```

```
}

protected void ClearRow(object? item)
{
    if (item == null)
    {
        return;
    }
    if (item is GridItemViewModel itemGIVM)
    {
        ClearRow(ItemsCollection, itemGIVM);
    }
}

protected bool ClearRow(ObservableCollection<GridItemViewModel> items, GridItemViewModel element)
{
    if (element == null)
    {
        Logger.Error("Element is null");
        return false;
    }
    if (!items.Contains(element))
    {
        Logger.Error($"Row '{element}' not found");
        return false;
    }

    var index = items.IndexOf(element);
    items[index] = new GridItemViewModel();

    Logger.Info($"Row id={index} cleared");
    return items[index] != null;
}

protected void InitEmptyDataGrid()
{
    Logger.Info("Selected grid items is null! Initializing...");
    var newTableItems = new ObservableCollection<GridItemViewModel>();
}

/// <summary> Update displayable collection content on its change
private void OnCollectionUpdate(object? sender, NotifyCollectionChangedEventArgs e)
{
    var newItems = this.ItemsCollection;
    this.ItemsSource = null;
    this.ItemsSource = newItems;
    Logger.Info($"Total items: {ItemsCollection.Count}");
}

private void InitDataGrid()
{
    this.IsReadOnly = _isReadOnly;
    this.AutoGenerateColumns = _autogenColumns;
    this.CanUserResizeColumns = _canUserResizeColumns;

    var rawData = LoadDataFromJson(_dataFilePath);
    ItemsSource = rawData;
    DataContext = ItemsSource;

    // WARNING: DEBUG THING
    AddRow(ItemsCollection, GenerateRandomItem());
    ItemsCollection.CollectionChanged += OnCollectionUpdate;
    Logger.Info($"{this.Name} initialized");
}

private static string GetDataFilePath()
{
    var enviroment = System.Environment.CurrentDirectory;
    _dataFilePath = Path.Combine(enviroment, "Data", "DataGridMain.json");

    return _dataFilePath;
}
```

```
    private ObservableCollection<GridItemViewModel> LoadDataFromJson(string filePath)
    {
        Logger.Info("Loading data from json...");

        try
        {
            string json = File.ReadAllText(filePath);
            var items = JsonSerializer.Deserialize<ObservableCollection<GridItemViewModel>>(json);
            Logger.Info("Total items = " + items?.Count);

            return items ?? new ObservableCollection<GridItemViewModel>();
        }
        catch (System.Exception ex)
        {
            var items = new List<GridItemViewModel>();
            items.Add(GenerateRandomItem());
            Trace.TraceError($"[{this.Name}, LoadDataFromJson()] Error while reading file: {ex.Message}");

            return new ObservableCollection<GridItemViewModel>(items);
        }
    }

    private GridItemViewModel GenerateRandomItem()
    {
        DateTime dt = DateTime.Now.AddDays(new Random().Next(1, 365));
        string s = DateConverter.ToString(dt);

        var gridItemFields = new GridItem()
        {
            Number = new Random().Next(1, 100).ToString(),
            Name = "Instrument " + new Random().Next(1, 100),
            Status = new Random().Next(0, 2) == 1,
            StoragePlace = "Storage " + new Random().Next(1, 10),
            Clinic = "Clinic " + new Random().Next(1, 5),
            Manager = "Manager " + new Random().Next(1, 10),
            Date = s,
            DaysInService = new Random().Next(1, 500),
            Comment = "Commentary " + new Random().Next(1, 100),
            Subdivision = "Subdivision " + new Random().Next(1, 5)
        };

        return new GridItemViewModel(gridItemFields);
    }

}
```

ContextedDataGridMT.java

```
using System;
using System.Windows.Input;
using Avalonia.Controls;
using Avalonia.Interactivity;
using CommunityToolkit.Mvvm.Input;
using MedTools.Services;
using MedTools.Views;

namespace MedTools.ViewModels;

/// <summary> DataGrid with context menu and ShowEditWindow dialog <summary>
public class ContextedDataGridMT : DataGridMT
{
    public ContextMenuTableMT Cmt { get; }

    public ContextedDataGridMT() : base()
    {
        Cmt = new ContextMenuTableMT(this);
    }

    public ContextedDataGridMT(string name) : base(name)
```

```
    {
        Cmt = new ContextMenuTableMT(this);
        Name = name;
    }

    public ICommand AddRowCommand => new RelayCommand(AddRow);
    public ICommand ClearRowCommand => new RelayCommand<object>(ClearRow);
    public ICommand RemoveRowCommand => new RelayCommand<object>(RemoveRow);

    protected Action? OnEditWindowSaveAction;

    private GridItemViewModel? _editedItem;

    public void ShowEditWindow(object? sender, RoutedEventArgs e)
    {
        var viewDG = sender as DataGridMTView;
        if (viewDG == null)
        {
            Logger.Error("View dg is null. Cancel method.");
            return;
        }
        var contentDG = viewDG.Content as DataGrid;
        if (contentDG == null)
        {
            Logger.Error("Content gd is null. Cancel method.");
            return;
        }

        if (IsValidSource(e.Source) &&
                sender is DataGridMTView dg &&
                contentDG.SelectedItem is GridItemViewModel item)
        {
            EditWindow editWindow = new EditWindow();
            editWindow.ShowDialogue(item);
            _editedItem = item;

            if (editWindow.DataContext is EditWindowViewModel ewvm)
            {
                ewvm.OnSaveAction += GetEditedItem;
            };
        }
    }

    private void GetEditedItem(GridItemViewModel item)
    {
        if (_editedItem == null)
            return;

        this.ReplaceGridItem(_editedItem, item);
    }

    private bool IsValidSource(object? source)
    {
        return source is TextBlock tb && tb.Name == "CellTextBlock" ||
               source is Border border && border.Name == "CellBorder";
    }
}
```

Приложение Б

Код модуля бизнес-логики InteractableListBoxViewModel

InteractableListBoxViewModel.cs

```
using System.Collections.ObjectModel;
using System.Threading.Tasks;
using System.Windows.Input;
using Avalonia.Controls;
using CommunityToolkit.Mvvm.Input;
using MedTools.Services;

namespace MedTools.ViewModels;

public class InteractableListBoxMTViewModel
{
    public ListBoxMT<ListBoxItemViewModel> ItemsContainer { get; set; }

    public ICommand AddItemCommand { get; set; }
    public ICommand RemoveItemCommand { get; set; }

    private TextBoxDialogueViewModel? tbDialog;
    private ConfirmationDialogueViewModel? confDialog;


    public InteractableListBoxMTViewModel(ObservableCollection<ListBoxItemViewModel> itemsSource)
    {
        ItemsContainer = new ListBoxMT<ListBoxItemViewModel>(itemsSource);

        AddItemCommand = new AsyncRelayCommand(AddItem);
        RemoveItemCommand = new AsyncRelayCommand(RemoveItem);
    }


    private async Task AddItem()
    {
        if (tbDialog != null && tbDialog.IsShown)
            return;

        tbDialog = new TextBoxDialogueViewModel("Add Item", "Enter the new item:");
        ListBoxItemViewModel result = await tbDialog.ShowDialogAsync();

        if (result != null && result.Name != string.Empty)
        {
            if (!CollectionContains(ItemsContainer.Items, result))
            {
                ItemsContainer.AddItem(result);
            }
        }
    }

    private async Task RemoveItem()
    {
        if (confDialog != null && confDialog.IsShown)
            return;

        var desktop = DesktopAccesser.DesktopInstance;
        if (desktop == null || desktop.MainWindow == null)
            return;

        var selectedItem = ItemsContainer.SelectedItem;
        if (selectedItem != null && selectedItem is ListBoxItemViewModel selItemListBox)
        {
            confDialog = new ConfirmationDialogueViewModel("Remove Item", $"Are you sure you want to remove
'{selItemListBox.ToString()}'?");

            var result = await confDialog.ShowDialogAsync();

            if (result && CollectionContains(ItemsContainer.Items, selItemListBox))
```

```
            {
                ItemsContainer.RemoveItem(selItemListBox);
            }
        }
    }

    private bool CollectionContains(ItemCollection items, ListBoxItemViewModel checkableItem)
    {
        bool contains = false;

        if (checkableItem is ListBoxItemViewModel checkItem)
        {
            foreach (ListBoxItemViewModel? item in items)
            {
                if (item != null)
                    contains = checkItem.Equals(item);
                if (contains)
                    return contains;
            }
        }

        return contains;
    }
}
```