МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РФ
федеральное государственное автономное образовательное учреждение высшего образования
«Санкт–Петербургский государственный университет
аэрокосмического приборостроения»

ФАКУЛЬТЕТ СРЕДНЕГО ПРОФЕССИОНАЛЬНОГО ОБРАЗОВАНИЯ

ОТЧЕТ
ЗАЩИЩЕН С ОЦЕНКОЙ
РУКОВОДИТЕЛЬ _____

преподаватель		
должность, уч. степень, звание	подпись, дата	инициалы, фамилия

ОТЧЕТ ПО ЛАБОРАТОРНЫМ РАБОТАМ

по дисциплине МДК 04.01

РАБОТУ ВЫПОЛНИЛ
СТУДЕНТ ГР. № С326 _____ Э.С. Тигранян
подпись, дата — инициалы, фамилия

Санкт-Петербург 2024

Оглавление

Лабораторная работа 1

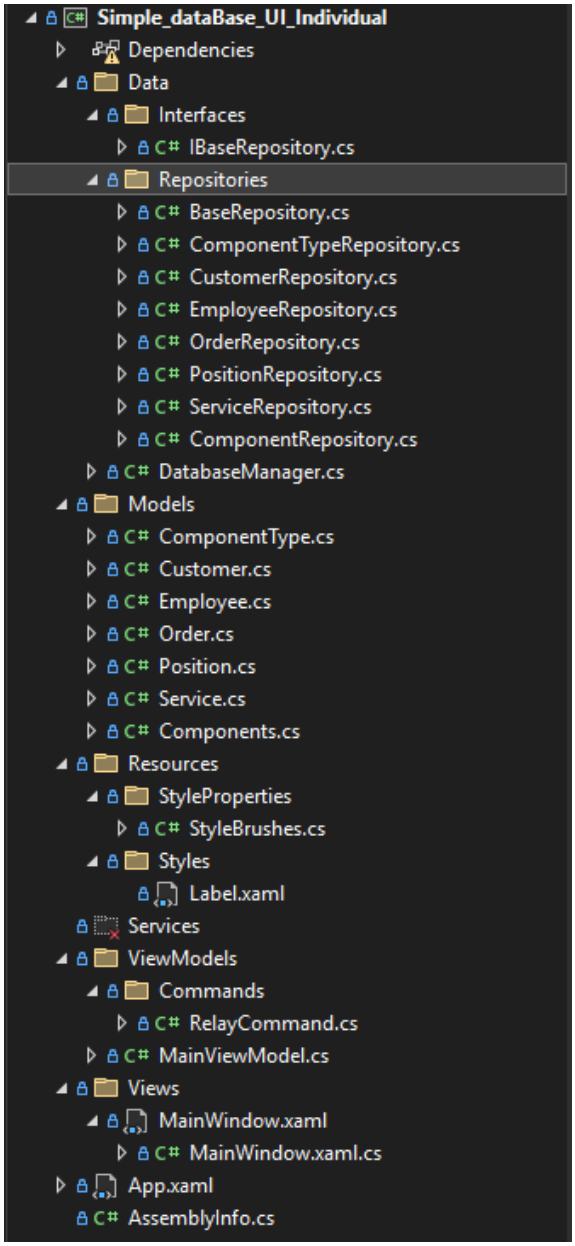
Задание

Клиентское приложение для взаимодействия с SQLite в соответствии с выданным вариантом.

1. Создать и связать любые три таблицы из индивидуального задания.
2. Показать, как работают следующие ограничения:

Вариант №19**:** БД Компьютерной фирмы.
Таблицы:
1) Сотрудники (Код сотрудника, ФИО, Возраст, Пол, Адрес, Телефон, Паспортные данные, Код должности).
2) Должности (Код должности, Наименование должности, Оклад, Обязанности,
Требования)
3) Виды комплектующих (Код вида, Наименование, Описание)
4) Комплектующие (Код комплектующего, Код вида, Марка, Фирма производитель, Страна производитель, Дата выпуска, Характеристики, Срок гарантия, Описание, Цена)
5) Заказчики (Код заказчика, ФИО, Адрес, Телефон).
6) Услуги (Код услуги, Наименование, Описание, Стоимость)
7) Заказы (Дата заказа, Дата исполнения, Код заказчика, Код комплектующего 1,
Код комплектующего 2, Код комплектующего 3, Доля предоплаты, Отметка об оплате, Отметка об исполнении, Общая стоимость, Срок общей гарантии, Код услуги 1, Код услуги 2, Код услуги 3, Код сотрудника).
Запросы:
Отобразить заказы отдельных заказчиков;
Отобразить комплектующие определенного производителя.

Структура проекта:



Структура проекта

Код MainWindow.xaml

Файл XAML-разметки главного окна приложения.

Определяет элементы интерфейса:

- ListBox — список таблиц базы данных;
- DataGrid — отображение и редактирование данных выбранной таблицы;
- привязки (Binding) к свойствам MainViewModel.

Все визуальные элементы связаны с данными через механизм привязки WPF.

```
<Window x:Class="Simple_dataBase_UI_Individual.Views.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:Simple_dataBase_UI_Individual.Views"
        xmlns:vm="clr-namespace:Simple_dataBase_UI_Individual.ViewModels"
        xmlns:i="http://schemas.microsoft.com/xaml/behaviors"
        mc:Ignorable="d"
        Title="MainWindow" Height="450" Width="800">
    <Window.DataContext>
        <vm:MainViewModel/>
    </Window.DataContext>
    <Border
        Background="DarkSlateGray"
        >

    <Grid Grid.RowSpan="1" Grid.Row="1" Grid.ColumnSpan="2">
        <Grid.RowDefinitions>
            <RowDefinition Height="auto"/>
            <RowDefinition Height="auto"/>
            <RowDefinition/>
        </Grid.RowDefinitions>
        <Grid.ColumnDefinitions>
            <ColumnDefinition Width="auto"/>
            <ColumnDefinition/>
        </Grid.ColumnDefinitions>



        <Label  Content="Tables"
                Style="{StaticResource S_Header}"
                ></Label>
        <Label  Content="Data"
                Grid.Column="1"
                Style="{StaticResource S_Header}"
                ></Label>

        <ListBox Grid.Row="1"
                 ItemsSource="{Binding Tables}"
                 SelectedItem="{Binding SelectedTable}"
                 Background="SlateGray"
                 Foreground="White"
                 FontWeight="DemiBold"
                 >
            <i:Interaction.Triggers>
                <i:EventTrigger EventName="SelectionChanged">
                    <i:InvokeCommandAction Command="{Binding
SelectionTableCommand}"></i:InvokeCommandAction>
                </i:EventTrigger>
            </i:Interaction.Triggers>
        </ListBox>



        <DataGrid Grid.Column="1" Grid.RowSpan="2" Grid.Row="1"
                  ItemsSource="{Binding DataTableC}"
                  Background="SlateGray"
                  Foreground="Black"
                  >
            <i:Interaction.Triggers>
                <i:EventTrigger EventName="CellEditEnding">
                    <i:InvokeCommandAction Command="{Binding CellEditEndingCommand}"
                                PassEventArgsToCommand="True"/>
                </i:EventTrigger>
```

```
                <i:EventTrigger EventName="RowEditEnding">
                    <i:InvokeCommandAction Command="{Binding RowEditEndingCommand}"
                                           PassEventArgsToCommand="True"/>
                </i:EventTrigger>
            </i:Interaction.Triggers>
        </DataGrid>
        </Grid>
    </Border>
</Window>
```

Код IBaseRepository.cs

Интерфейс репозитория, определяющий базовый контракт для работы с данными.

Содержит объявления стандартных CRUD-операций:

- получение всех записей;
- добавление;
- обновление;
- удаление;
- сохранение изменений.

Интерфейс обеспечивает единообразие работы с различными сущностями базы данных.

```
using System;
using System.Collections.Generic;
using System.Data;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace Simple_dataBase_UI_Individual.Data.Interfaces
{
    public interface IBaseRepository<T> where T : class
    {
        DataTable GetAll();
        void Add(T entity);
        void Update(T entity);
        void Delete(int id);
        void Save();
    }
}
```

Код BaseRepository

Базовый абстрактный класс репозитория, реализующий общую логику взаимодействия с SQLite:

- хранит подключение к базе данных;

- выполняет SQL-запросы;
- реализует метод GetAll() для получения данных в виде DataTable.

Данный класс не зависит от конкретной модели и используется как основа для специализированных репозиториев.

```
using Simple_dataBase_UI_Individual.Data;
using Simple_dataBase_UI_Individual.Data.Interfaces;
using System;
using System.Collections.Generic;
using System.Data;
using System.Data.SQLite;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows;
namespace Simple_dataBase_UI_Individual.Data.Repositories
{
    public abstract class BaseRepository<T> : IBaseRepository<T> where T : class
    {
        protected readonly string tableName;
        public BaseRepository()
        {
            this.tableName = typeof(T).Name;
        }
        public T CreateInstance()
        {
            return Activator.CreateInstance<T>();
        }
        public virtual DataTable GetAll()
        {
            DataTable dataTable = new DataTable();
            try
            {
                string sqlQuery = $"SELECT * FROM \"{tableName}\"";
                using (SQLiteDataAdapter adapter = new SQLiteDataAdapter(sqlQuery,
DatabaseManager.m_dbConn))
                {
                    adapter.Fill(dataTable);
                }
                Console.WriteLine($"Retrieved {dataTable.Rows.Count} rows from
{tableName}");
            }
            catch (Exception ex)
            {
                Console.WriteLine($"Error getting all from {tableName}:
{ex.Message}");
                MessageBox.Show($"Error loading data from {tableName}:
{ex.Message}");
            }
            return dataTable;
        }
        public virtual T CreateInstanceFromDataRow(DataRow row) { throw new
NotImplementedException(); }
        public virtual void Add(T entity){throw new NotImplementedException();}
        public virtual void Delete(int id) { throw new NotImplementedException(); }
        public virtual void Save() { throw new NotImplementedException(); }
        public virtual void Update(T entity) { throw new NotImplementedException();
}
    }
}
```

Код ComponentTypeRepository

Конкретная реализация репозитория для сущности ComponentType.

Наследуется от BaseRepository<ComponentType> и реализует:

- преобразование строк таблицы (DataRow) в объекты ComponentType;
- SQL-команды INSERT, UPDATE, DELETE;
- проверку существования записи по идентификатору.

Таким образом, данный репозиторий полностью инкапсулирует логику работы с таблицей «ComponentType» в базе данных SQLite.

Остальные репозитории организованы по подобному принципу

```
using Simple_dataBase_UI_Individual.Data;
using Simple_dataBase_UI_Individual.Data.Interfaces;
using Simple_dataBase_UI_Individual.Models;
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Data.SQLite;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows;



//  Виды комплектующих(
//          Код вида,
//          Наименование,
//          Описание)



namespace Simple_dataBase_UI_Individual.Data.Repositories
{
    public class ComponentTypeRepository : BaseRepository<ComponentType>
    {
        public ComponentTypeRepository(string dbFilePath) { }



        public override ComponentType CreateInstanceFromDataRow(DataRow row)
        {
            try
            {
                var componentType = new ComponentType();



                if (!row.IsNull("id") && row["id"] != DBNull.Value) {
componentType.Id = Convert.ToInt32(row["id"]); }
```

```
                else { componentType.Id = 0; }



                componentType.Name = row.IsNull("name") ? "" :
row["name"].ToString();
                componentType.Description = row.IsNull("de  scription") ? "" :
row["description"].ToString();



                return componentType;
            }
            catch (Exception ex)
            {
                Console.WriteLine($"Error creating ComponentType from DataRow:
{ex.Message}");
                throw;
            }
        }



        public bool IsIdExists(int id)
        {
            try
            {
                using (var command = new SQLiteCommand(DatabaseManager.m_dbConn))
                {
                    command.CommandText = "SELECT COUNT(*) FROM ComponentType WHERE
id = @id";
                    command.Parameters.AddWithValue("@id", id);



                    int count = Convert.ToInt32(command.ExecuteScalar());
                    return count > 0;
                }
            }
            catch (Exception ex)
            {
                Console.WriteLine($"Error checking ID existence: {ex.Message}");
                return false;
            }
        }



        public override void Add(ComponentType entity)
        {
            try
            {
                // Проверяем, не существует ли уже запись с таким ID
                if (entity.Id != 0 && IsIdExists(entity.Id))
                {
                    MessageBox.Show($"ComponentType with ID {entity.Id} already
exists!");
                    return;
                }
```

```
                using (var command = new SQLiteCommand(DatabaseManager.m_dbConn))
                {
                    if (entity.Id != 0)
                    {
                        command.CommandText = @"
                            INSERT INTO ComponentType (id, name, description)
                            VALUES (@id, @name, @description)";
                        command.Parameters.AddWithValue("@id", entity.Id);
                    }
                    else
                    {
                        command.CommandText = @"
                            INSERT INTO ComponentType (name, description)
                            VALUES (@name, @description)";
                    }



                    command.Parameters.AddWithValue("@name", entity.Name ?? "");
                    command.Parameters.AddWithValue("@description",
entity.Description ?? "");



                    int rowsAffected = command.ExecuteNonQuery();
                    Console.WriteLine($"Rows affected: {rowsAffected}");



                    // Если ID не был указан, получаем сгенерированный ID
                    if (entity.Id == 0)
                    {
                        command.CommandText = "SELECT last_insert_rowid()";
                        entity.Id = Convert.ToInt32(command.ExecuteScalar());
                    }
                }
            }
            catch (SQLiteException ex)
            {
                MessageBox.Show("Error adding component type: " + ex.Message);
            }
        }



        public override void Update(ComponentType entity)
        {
            try
            {
                using (var command = new SQLiteCommand(DatabaseManager.m_dbConn))
                {
                    command.CommandText = @"
                        UPDATE ComponentType
                        SET name = @name,
                            description = @description
                        WHERE id = @id";



                    command.Parameters.AddWithValue("@name", entity.Name ?? "");
                    command.Parameters.AddWithValue("@description",
entity.Description ?? "");
```

```
                    command.Parameters.AddWithValue("@id", entity.Id);



                    command.ExecuteNonQuery();
                }
            }
            catch (SQLiteException ex)
            {
                MessageBox.Show("Error updating component type: " + ex.Message);
            }
        }
    }
}
```

Код ComponentType.cs

Содержит модель сущности базы данных «Тип комплектующего».

Класс ComponentType представляет одну запись таблицы и включает свойства:

- Id — уникальный идентификатор записи;
- Name — наименование типа комплектующего;
- Description — описание.

Модель не содержит логики работы с базой данных и используется для переноса данных между слоями приложения.

Остальные модели организованы подобным образов

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;



namespace Simple_dataBase_UI_Individual.Models
{
    //  Виды комплектующих(
    //          Код вида,
    //          Наименование,
    //          Описание)
    public class ComponentType
    {
        public ComponentType() { }
        public ComponentType(int Id, string Name, string Description) {
            this.Id = Id;
            this.Name = Name;
            this.Description = Description;
        }
        public ComponentType(List<object> array)
        {
            if (array == null || array.Count < 3)
                throw new ArgumentException("Array must contain at least 3
elements");
```

```
            this.Id = Convert.ToInt32(array[0]);
            this.Name = array[1]?.ToString() ?? string.Empty;
            this.Description = array[2]?.ToString() ?? string.Empty;
        }
        public int Id { get; set; }
        public string Name { get; set; }
        public string Description { get; set; }
    }
}
```

Код RelayCommand

Реализует интерфейс ICommand для использовани команд в интерфейсе

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Input;
//using Simple_dataBase_UI_Individual.Models;



namespace Simple_dataBase_UI_Individual.ViewModels.Commands
{
    public class RelayCommand : ICommand
    {
        private readonly Action<object> _execute;
        private readonly Func<object, bool> _canExecute;



        public RelayCommand(Action<object> execute, Func<object, bool> canExecute =
null)
        {
            _execute = execute ?? throw new ArgumentNullException(nameof(execute));
            _canExecute = canExecute;
        }
        public bool CanExecute(object parameter)
        {
            return _canExecute == null || _canExecute(parameter);
        }



        public void Execute(object parameter)
        {
            _execute(parameter);
        }



        public event EventHandler CanExecuteChanged
        {
            add { CommandManager.RequerySuggested += value; }
            remove { CommandManager.RequerySuggested -= value; }
```

```
        }
    }
}
```

Код MainViewModel.cs

Главная ViewModel приложения. Выполняет следующие функции:

- хранит список доступных таблиц базы данных;
- отслеживает выбранную пользователем таблицу;
- инициализирует соответствующий репозиторий;
- загружает данные из базы данных и предоставляет их представлению;
- обрабатывает события редактирования данных в таблице.

ViewModel не содержит SQL-запросов напрямую, а работает исключительно через слой репозиториев.

```
using Simple_dataBase_UI_Individual.Data;
using Simple_dataBase_UI_Individual.Data.Interfaces;
using Simple_dataBase_UI_Individual.Data.Repositories;
using Simple_dataBase_UI_Individual.ViewModels.Commands;
using System;
using System.Collections.Generic;
using System.Collections.ObjectModel;
using System.ComponentModel;
using System.Data;
using System.Linq;
using System.Runtime.CompilerServices;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Controls;
using System.Windows.Input;
using System.Windows.Threading;



namespace Simple_dataBase_UI_Individual.ViewModels
{
    class MainViewModel : INotifyPropertyChanged
    {
        private object _selectedTable;
        public object SelectedTable
        {
            get { return _selectedTable; }
            set {
                _selectedTable = value;
                OnPropertyChanged();
            }
        }



        private DataTable _dataTable;
        public DataTable DataTableC
        {
```

```
    get { return _dataTable; }
    set { _dataTable = value;
        OnPropertyChanged();
    }
}
public ObservableCollection<string> Tables { get; set; }



private ObservableCollection<object> _items;
public ObservableCollection<object> Items
{
    get { return _items; }
    set { _items = value;
        OnPropertyChanged();
    }
}



private readonly Dictionary<string, object> _repositories;



public MainViewModel()
{
    DatabaseManager.getInstance("testdb.sqlite");



    _repositories = new Dictionary<string, object> {
        { "ComponentType", new ComponentTypeRepository("testdb")},
        { "Employee", new EmployeeRepository("testdb")},
        { "Order", new OrderRepository("testdb")},
        { "Position", new PositionRepository("testdb")},
        { "Service", new ServiceRepository("testdb")},
        { "Component", new ComponentRepository("testdb")},
        { "Customer", new CustomerRepository("testdb")}
        };



    SelectionTableCommand = new RelayCommand(SelectionTable);
    SelectionChangedCommand = new RelayCommand(SelectionChanged);
    CellEditEndingCommand = new RelayCommand(CellEditEnding);
    RowEditEndingCommand = new RelayCommand(RowEditEnding);



    Tables = new ObservableCollection<string>(_repositories.Keys);
}
public ICommand SelectionTableCommand { get; }
public ICommand SelectionChangedCommand { get; }
public ICommand CellEditEndingCommand { get; }
public ICommand RowEditEndingCommand { get; }
public void CellEditEnding(object parameter)
{
```

```
        }
        public void SelectionChanged(object parameter)
        {



        }
        public void RowEditEnding(object parameter)
        {
            if (parameter is DataGridRowEditEndingEventArgs e)
            {
                if (e.EditAction == DataGridEditAction.Commit)
                {
                    if (e.Row.DataContext is DataRowView dataRowView)
                    {
                        Dispatcher dispatcher =
System.Windows.Application.Current.Dispatcher;
                        Action myAction = delegate ()
                        {
                            ProcessRowEdit(dataRowView.Row);
                        };
                        dispatcher.BeginInvoke(myAction,
DispatcherPriority.Background);
                    }
                }
            }
        }
        private void ProcessRowEdit(DataRow dataRow)
        {
            try
            {
                bool isNewRow = dataRow.RowState == DataRowState.Added ||
                                dataRow.RowState == DataRowState.Detached ||
                                dataRow.IsNull("id") ||
                                dataRow["id"] == DBNull.Value ||
                                Convert.ToInt32(dataRow["id"]) == 0;



                dynamic repository = _repositories[SelectedTable.ToString()];
                var entity = repository.CreateInstanceFromDataRow(dataRow);



                if (isNewRow)
                {
                    repository.Add(entity);
                    dataRow["id"] = entity.Id;
                }
                else
                {
                    repository.Update(entity);
                }
                Dispatcher dispatcher =
System.Windows.Application.Current.Dispatcher;
                Action myAction = delegate ()
                {
                    RefreshDataTable(repository);
                };
                dispatcher.BeginInvoke(myAction,
DispatcherPriority.ApplicationIdle);
            }
            catch (Exception ex) { }
```

```
        }



        private void RefreshDataTable(dynamic repository)
        {
            var newDataTable = repository.GetAll();
            DataTableC = newDataTable;
        }
        private void SelectionTable(object parameter)
        {
            dynamic repository = _repositories[SelectedTable.ToString()];
            RefreshDataTable(repository);
        }



        public event PropertyChangedEventHandler? PropertyChanged;
        protected virtual void OnPropertyChanged([CallerMemberName] string
propertyName = null)
        {
            PropertyChanged?.Invoke(this, new
PropertyChangedEventArgs(propertyName));
        }
    }
}
```

Описание кода

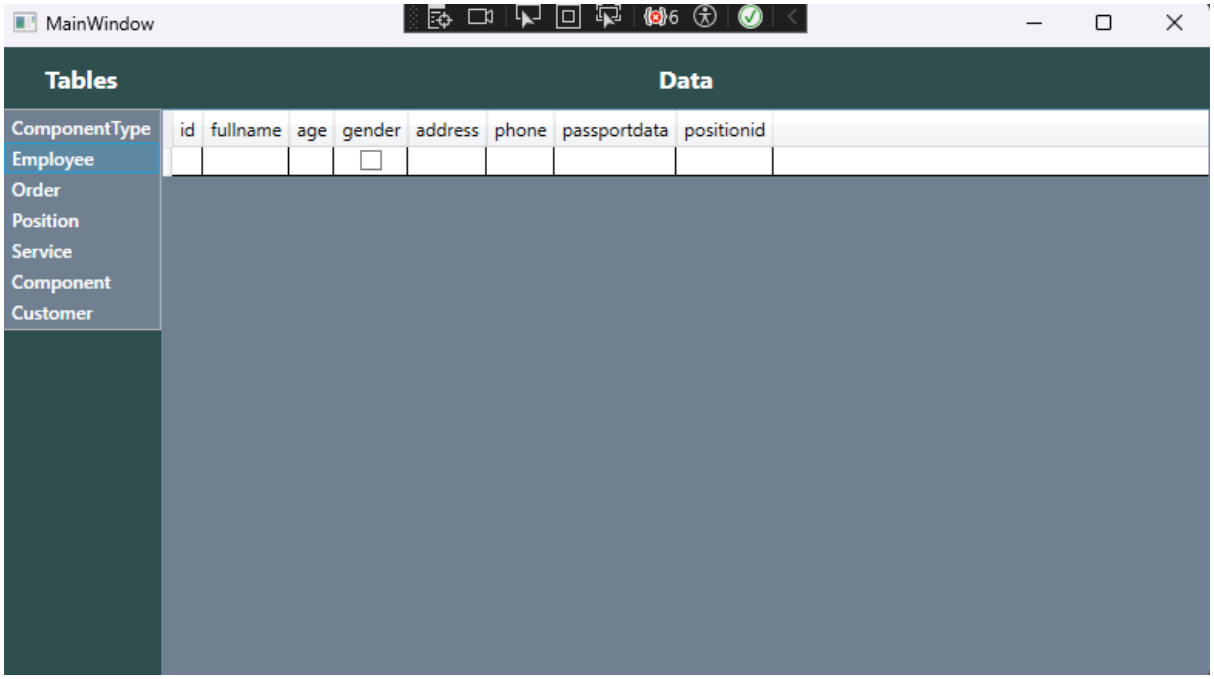
Взаимодействие файлов и слоёв приложения

1. Представление (View) (MainWindow.xaml) устанавливает объект MainViewModel в качестве DataContext, тем самым связывая элементы пользовательского интерфейса со свойствами и командами ViewModel.
2. Пользователь выбирает таблицу базы данных в элементе управления интерфейса (ListBox).
3. MainViewModel анализирует выбранное значение и определяет, какой конкретный репозиторий соответствует данной таблице (например, ComponentTypeRepository).
4. ViewModel обращается к выбранному репозиторию и вызывает метод GetAll() для получения данных.
5. Репозиторий выполняет SQL-запрос к базе данных SQLite, используя активное подключение к БД.
6. Результат SQL-запроса извлекается в виде набора строк, которые:
    - либо преобразуются в объекты модели (например, ComponentType),

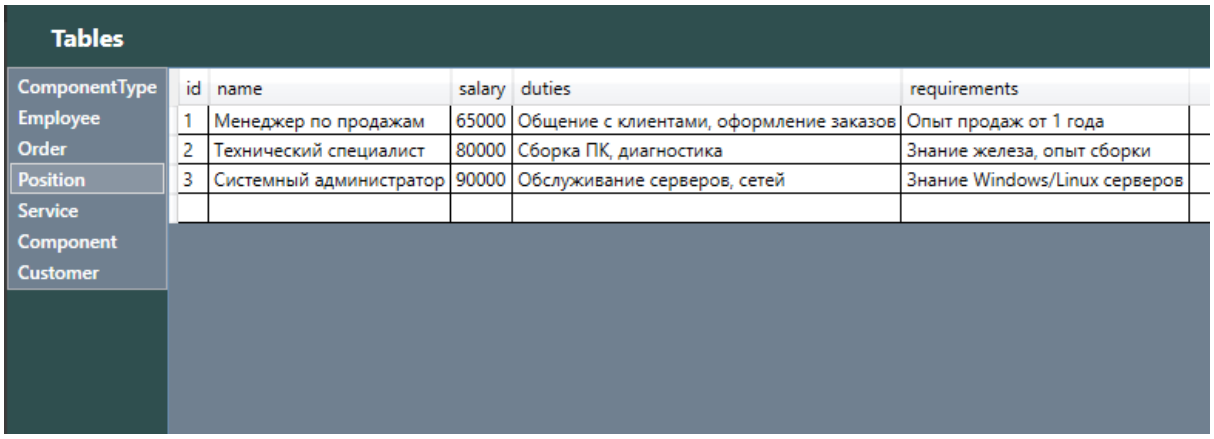
- либо формируются в структуру DataTable, связанную с моделью по составу полей.

7. Модель (ComponentType) на данном этапе выступает как представление сущности предметной области:
   каждая строка таблицы базы данных логически соответствует экземпляру модели, содержащему строго типизированные свойства (Id, Name, Description).
8. Репозиторий возвращает данные (модели или DataTable, сформированный на их основе) обратно во ViewModel.
9. ViewModel сохраняет полученные данные в своих свойствах и предоставляет их представлению через механизм привязки данных (Binding).
10. Представление (View) отображает данные в элементе DataGrid и предоставляет пользователю возможность их редактирования.
11. При добавлении или изменении строки пользовательским интерфейсом ViewModel формирует или обновляет соответствующий объект модели (ComponentType) на основе введённых данных.
12. ViewModel передаёт объект модели в репозиторий, вызывая методы Add() или Update().
13. Репозиторий, используя данные модели, формирует параметризованные SQL-запросы и сохраняет изменения в базе данных SQLite.

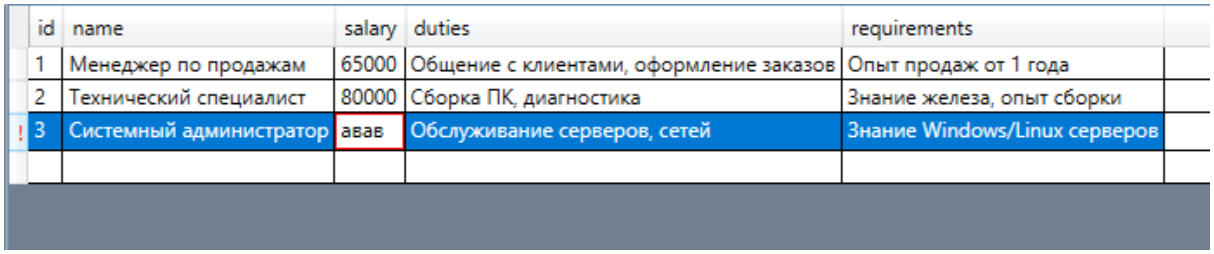
Результат



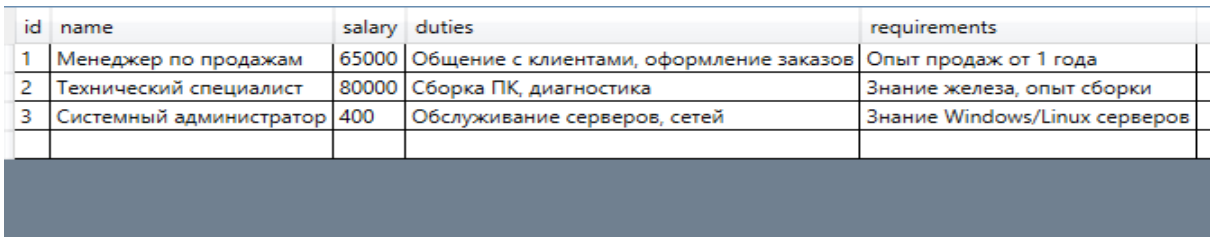
Интерфейс приложения



Пример заполнения таблицы

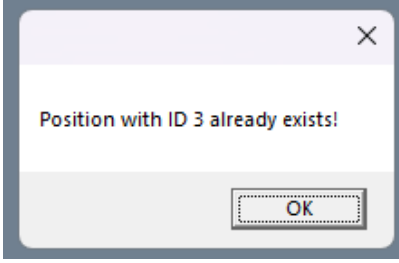


Пример попытки внесения некорректных данных



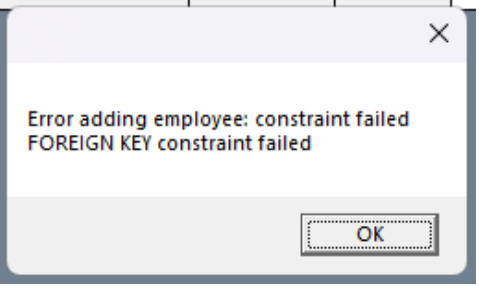
Пример изменения таблицы

id	name	salary	duties	requirements
1	Менеджер по продажам	65000	Общение с клиентами, оформление заказов	Опыт продаж от 1 года
2	Технический специалист	80000	Сборка ПК, диагностика	Знание железа, опыт сборки
3	Системный администратор	90000	Обслуживание серверов, сетей	Знание Windows/Linux серверов
3				



Попытка создание записи с не уникальным ключом

id	fullname	age	gender	address	phone	passportdata	positionid
101	Иванов Алексей Петрович	28	☑	Москва, ул. Ленина, 15	+7(495)111-22-33	4510 123456	76
			☐				



Попытка создания записи в таблицы с несуществующим Foreign Key

Tables

ComponentType
Employee
Order
Position
Service
Component
Customer

id	fullname	age	gender	address	phone	passportdata	positionid
101	Иванов Алексей Петрович	28	☑	Москва, ул. Ленина, 15	+7(495)111-22-33	4510 123456	1
102	Петрова Мария Сергеевна	32	☐	Москва, пр. Мира, 28	+7(495)222-33-44	4510 654321	2
103	Сидоров Дмитрий Иванович	35	☑	Москва, ул. Пушкина, 10	+7(495)333-44-55	4510 789012	3
			☐				

Пример заполнения таблицы с Foreign Key

Tables

ComponentType
Employee
Order
Position
Service
Component
Customer

id	name	description	price
1	Диагностика компьютера	Полная проверка оборудования и программного обеспечения	1500
2	Сборка ПК на заказ	Комплексная сборка компьютера из выбранных комплектующих	3000
3	Установка ОС	Установка и настройка операционной системы Windows/Linux	2000

Пример заполнения таблицы Service

Лабораторная работа 2

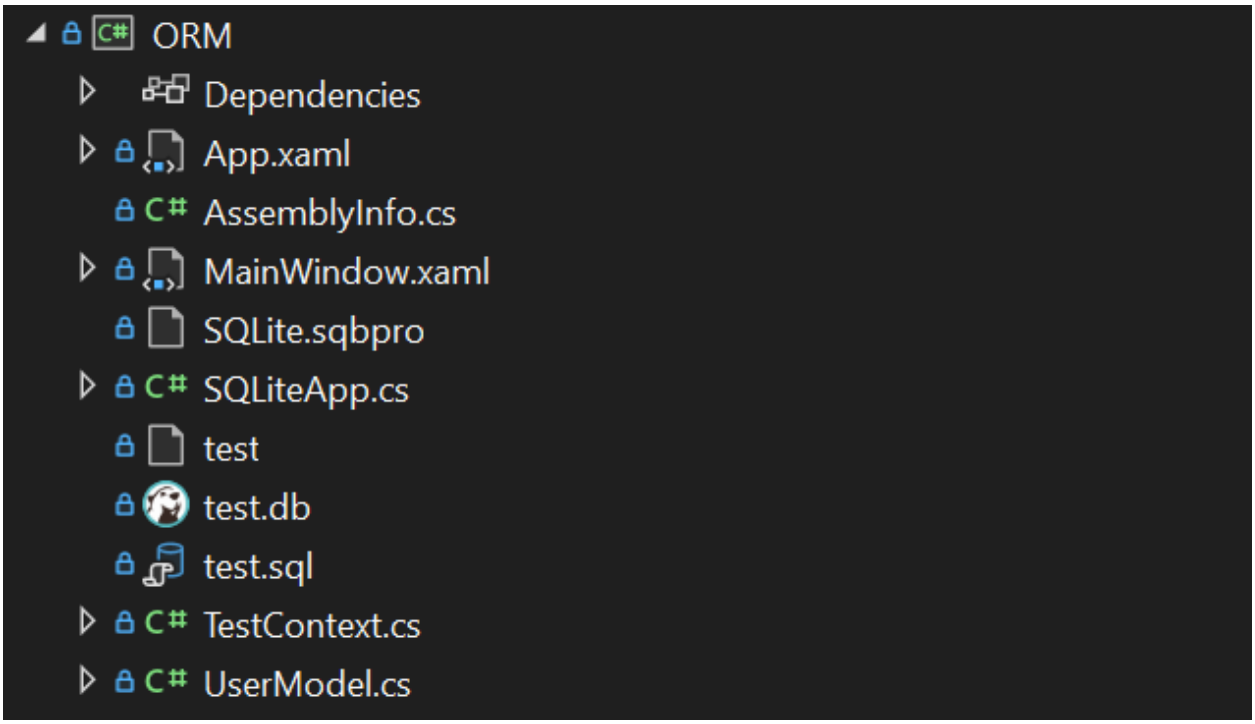
Тема: Взаимодействие с встраиваемой СУБД SQLite. Подключение к базе данных и создание таблиц с помощью ORM-фреймворка Entity Framework.

Цель работы : получение практических навыков при использовании Microsoft.EntityFrameworkCore.Sqlite.

Задание 1

Изучить подход code first.

Определим в проекте класс **User**, объекты которого будут храниться в базе данных. Установим подключение к БД и добавим данные в БД через приложение(на момент запуска приложения БД не существует).



Структура проекта

SQLiteApp.cs

Класс контекста базы данных Entity Framework. Отвечает за подключение к SQLite и управление сущностями приложения. Содержит DbSet<User>, через который выполняется создание таблицы и работа с данными. Контекст реализован как Singleton, что обеспечивает использование одного подключения к базе данных во всём приложении.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using Microsoft.EntityFrameworkCore;
using Models;

namespace SQLiteApp
{
    public class ApplicationContext : DbContext
    {
        private static ApplicationContext _context;
        public DbSet<User> Users { get; set; } = null;
        public static ApplicationContext GetContext()
        {
            if( _context == null )
                _context = new ApplicationContext();
            return _context;
        }
        protected override void OnConfiguring(DbContextOptionsBuilder
optionsBuilder)
        {
            optionsBuilder.UseSqlite(@"Data Source=C:\Secret\Fork\ORM\ORM\test.db");
        }
    }
}
```

TestContext.cs

Файл контекста базы данных Entity Framework. Наследуется от DbContext и задаёт подключение к SQLite через OnConfiguring() (строка подключения вида Data Source=test.db). Также содержит метод OnModelCreating(), который используется для настройки схемы базы (если требуется), и оставлен частичным (partial) для расширения без изменения автосгенерированного кода.

```
using System;
using System.Collections.Generic;
using Microsoft.EntityFrameworkCore;

namespace ORM;

public partial class TestContext : DbContext
{
    public TestContext()
    {
    }

    public TestContext(DbContextOptions<TestContext> options)
        : base(options)
    {
    }

    protected override void OnConfiguring(DbContextOptionsBuilder optionsBuilder)
#warning To protect potentially sensitive information in your connection string, you
should move it out of source code. You can avoid scaffolding the connection string
by using the Name= syntax to read it from configuration - see
```

```
https://go.microsoft.com/fwlink/?linkid=2131148. For more guidance on storing
connection strings, see https://go.microsoft.com/fwlink/?LinkId=723263.
        => optionsBuilder.UseSqlite("Data Source=test.db");

    protected override void OnModelCreating(ModelBuilder modelBuilder)
    {
        OnModelCreatingPartial(modelBuilder);
    }

    partial void OnModelCreatingPartial(ModelBuilder modelBuilder);
}
```

UserModel.cs

Модель сущности пользователя. Описывает структуру таблицы базы данных (Id, Name, Age). Используется Entity Framework для автоматического создания таблицы при первом запуске приложения. Класс содержит только свойства и не включает логику работы с БД.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace Models
{
    public class User
    {
        public int Id { get; set; }
        public string? Name { get; set; }
        public int Age { get; set; }
    }
}
```

MainWindow.xaml

XAML-разметка главного окна приложения. Содержит элемент DataGrid, предназначенный для отображения данных из базы данных SQLite. Интерфейс служит для визуального представления данных, полученных через Entity Framework.

```
<Window x:Class="ORM.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:ORM"
        mc:Ignorable="d"
        Title="MainWindow" Height="450" Width="800">
    <Grid>
        <DataGrid x:Name="data"></DataGrid>
    </Grid>
</Window>
```

MainWindow.xaml.cs

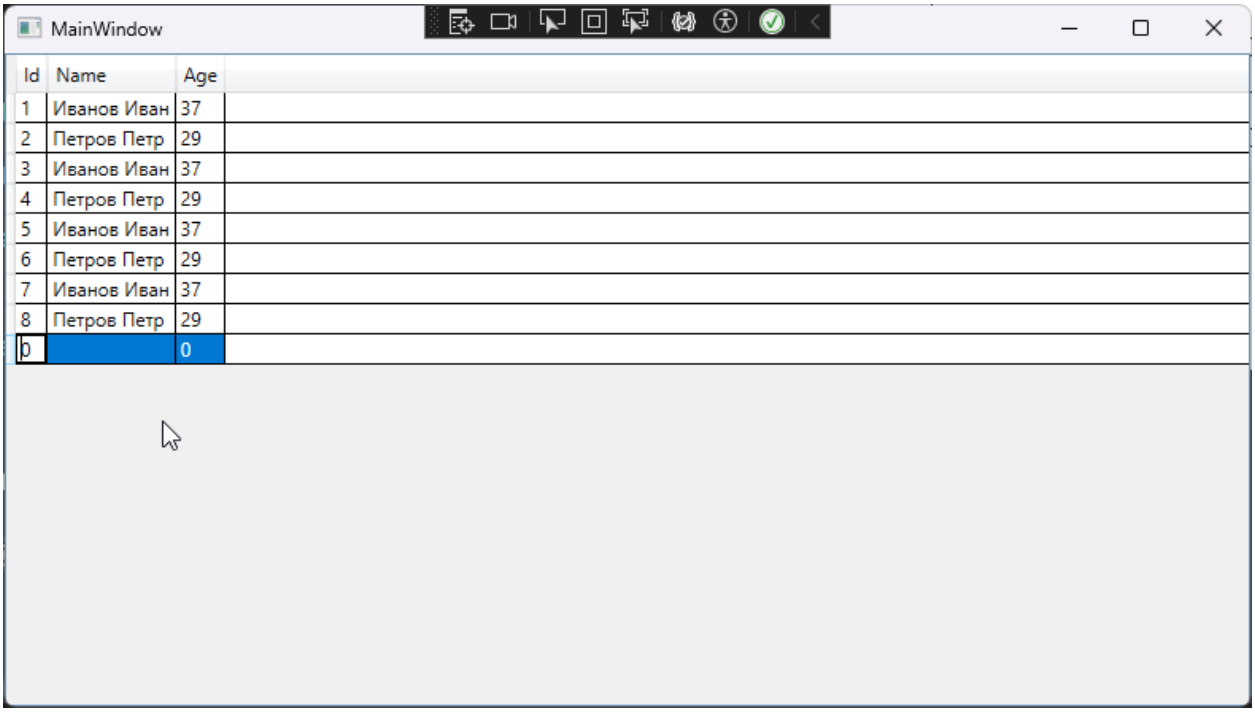
Код-бихайнд главного окна. Выполняет инициализацию базы данных, создание тестовых записей пользователей и загрузку данных из таблицы Users. Получает данные через контекст базы данных и передаёт их в DataGrid.

```
namespace ORM
{
    /// <summary>
    /// Interaction logic for MainWindow.xaml
    /// </summary>
    public partial class MainWindow : Window
    {
        public MainWindow()
        {
            InitializeComponent();
            Load();
        }
    public void Load()
        {
            User user1 = new User { Name = "Иванов Иван", Age = 37 };
            User user2 = new User { Name = "Петров Петр", Age = 29 };
            var db = ApplicationContext.GetContext();
            db.Database.EnsureCreated();
            db.Users.AddRange(user1, user2);
            db.SaveChanges();
            data.ItemsSource = db.Users.ToList();
        }
    }
}
```

Описание кода

При запуске приложения создаётся окно MainWindow. В коде MainWindow.xaml.cs инициализируется контекст ApplicationContext. Entity Framework на основе модели User автоматически создаёт базу данных и таблицу (подход Code First). Далее в таблицу добавляются записи пользователей, после чего данные извлекаются из базы и отображаются в элементе DataGrid.

Результат

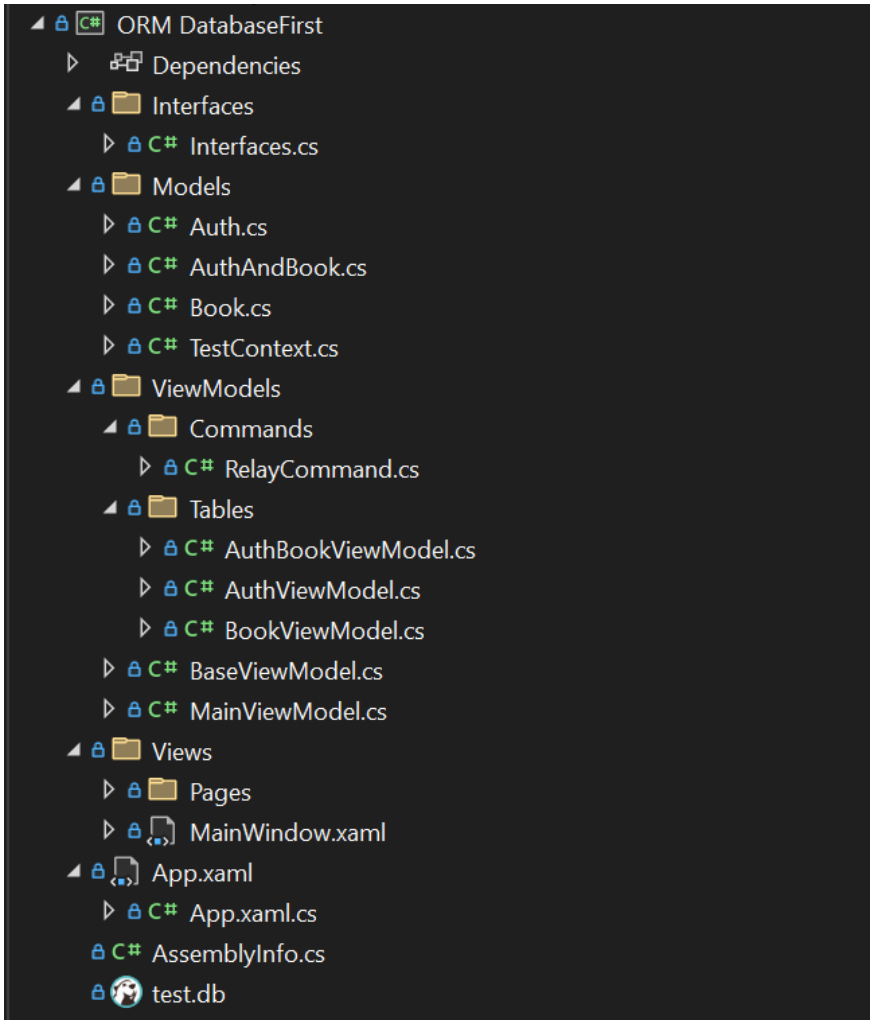


Главный экран приложения

Задание 2

Изучить подход databasefirst.

Создать БД из трех связанных таблиц в DB Browser for SQLite(Таблица Авторы, Книги, Книги_Авторы). Используя команду обратного проектирования, получить классы сущностей и контекстов на основе схемы существующей базы данных.



Структура проекта

Папка Interfaces

Interfaces.cs

Интерфейс, определяющий общий метод Load(). Используется для унификации загрузки данных в различных ViewModel и обеспечения единого подхода к работе с таблицами базы данных.

```
namespace Interfaces
{
    public interface Tables
    {
        public void Load();
    }
}
```

Папка Models

Auth.cs

Модель сущности «Автор», автоматически сгенерированная Entity Framework. Соответствует таблице auth и содержит навигационное свойство для связи с книгами.

```
namespace Models;
public partial class Auth
{
    public int Id { get; set; }
    public string Name { get; set; } = null!;
    public int? Age { get; set; }
    public virtual ICollection<Book> Books { get; set; } = new List<Book>();
}
```

AuthAndBook.cs

Вспомогательная модель для описания промежуточной таблицы связи между авторами и книгами.

```
namespace ORM_DatabaseFirest.Models
{
    public class AuthAndBook
    {
        public int AuthId { get; set; }
        public int BookId { get; set; }
    }
}
```

Book.cs

Модель сущности «Книга». Соответствует таблице books и содержит коллекцию авторов, что позволяет реализовать связь «многие-ко-многим».

```
namespace Models;
public partial class Book
{
    public int Id { get; set; }
    public string Title { get; set; } = null!;
    public int CountPage { get; set; }
    public double? Price { get; set; }
    public virtual ICollection<Auth> Auths { get; set; } = new List<Auth>();
}
```

TestContext.cs

Контекст базы данных, сгенерированный при обратном проектировании. Описывает таблицы базы данных и связи между ними. Используется для получения и сохранения данных при подходе Database First.

```
namespace Models;

public partial class TestContext : DbContext
{
    static TestContext context;
    private TestContext(){}
    public static TestContext GetContext()
    {
```

```
        if (context == null) context = new TestContext();
        return context;
    }
    private TestContext(DbContextOptions<TestContext> options)
        : base(options)
    {
    }
    public virtual DbSet<Auth> Auths { get; set; }
    public virtual DbSet<Book> Books { get; set; }
    protected override void OnConfiguring(DbContextOptionsBuilder optionsBuilder)
#warning To protect potentially sensitive information in your connection string, you should move it out of source code. You can avoid scaffolding the connection string by using the Name= syntax to read it from configuration - see https://go.microsoft.com/fwlink/?linkid=2131148. For more guidance on storing connection strings, see https://go.microsoft.com/fwlink/?LinkId=723263.
        => optionsBuilder.UseSqlite("Data Source=C:\\Secret\\Laboratory-work\\3 Курс\\МДК 04.01\\Лабораторная работа 2\\ORM DatabaseFirst\\Лабораторная 2\\test.db");
    protected override void OnModelCreating(ModelBuilder modelBuilder)
    {
        modelBuilder.Entity<Auth>(entity =>
        {
            entity.ToTable("auth");

            entity.Property(e => e.Id)
                .ValueGeneratedNever()
                .HasColumnName("id");
            entity.Property(e => e.Age).HasColumnName("age");
            entity.Property(e => e.Name).HasColumnName("name");

            entity.HasMany(d => d.Books).WithMany(p => p.Auths)
                .UsingEntity<Dictionary<string, object>>(
                    "AuthBook",
                    r => r.HasOne<Book>().WithMany()
                        .HasForeignKey("BooksId")
                        .OnDelete(DeleteBehavior.ClientSetNull),
                    l => l.HasOne<Auth>().WithMany()
                        .HasForeignKey("AuthId")
                        .OnDelete(DeleteBehavior.ClientSetNull),
                    j =>
                    {
                        j.HasKey("AuthId", "BooksId");
                        j.ToTable("auth_book");
                        j.IndexerProperty<int>("AuthId").HasColumnName("auth_id");
                        j.IndexerProperty<int>("BooksId").HasColumnName("books_id");
                    });
        });
        modelBuilder.Entity<Book>(entity =>
        {
            entity.ToTable("books");

            entity.Property(e => e.Id).ValueGeneratedNever();
        });
        OnModelCreatingPartial(modelBuilder);
    }
    partial void OnModelCreatingPartial(ModelBuilder modelBuilder);
}
```

Папка ViewModels

RelayCommand.cs

Класс команды, реализующий интерфейс ICommand. Используется для обработки действий пользователя в рамках паттерна MVVM.

```
namespace ORM_DatabaseFirest.ViewModels.Commands
{
    internal class RelayCommand : ICommand
    {
        private readonly Action<object> _execute;
        public RelayCommand(Action<object> execute) {
            _execute = execute;
        }
        public event EventHandler? CanExecuteChanged;
        public bool CanExecute(object parameter)
        {
            return true;
        }
        public void Execute(object parameter)
        {
            _execute(parameter);
        }
    }
}
```

AuthBookViewModel.cs

ViewModel, отвечающая за загрузку и представление связанных данных (авторы и книги). Получает данные через TestContext и подготавливает их для отображения в интерфейсе.

```
namespace ORM_DatabaseFirest.ViewModels
{
    public class AuthBookViewModel : MainViewModel, Tables
    {
        public AuthBookViewModel() : base(){
            Load();
        }
        public void Load()
        {
            var db = TestContext.GetContext();
            var authBooks = db.Auths.Include(a => a.Books).ToList();

            var dataForGrid = authBooks
                .SelectMany(a => a.Books.Select(book => new AuthAndBook
                {
                    AuthId = a.Id,
                    BookId = book.Id,
                }))
                .ToList();

            if (AuthAndBookData == null)
            {
                AuthAndBookData = new
ObservableCollection<AuthAndBook>(dataForGrid);
            }
            else
            {
                AuthAndBookData.Clear();
                foreach (var item in dataForGrid)
                {
```

```
                AuthAndBookData.Add(item);
            }
        }
    }

    #region AddProperties
    private int _authId;
    public int AuthId
    {
        get { return _authId; }
        set { _authId = value;
            OnPropertyChanged();
        }
    }
    private int _booksId;
    public int BooksId
    {
        get { return _booksId; }
        set { _booksId = value;
            OnPropertyChanged();
        }
    }
    #endregion

    public override void AddData(object parameter)
    {
        TestContext db = TestContext.GetContext();
        try
        {
            var book = db.Books.Find(Convert.ToInt32(BooksId));
            var auth = db.Auths.Find(Convert.ToInt32(AuthId));

            if (auth == null)
            {
                MessageBox.Show("Введён несуществующий автор");
                return;
            }
            else if (book == null)
            {
                MessageBox.Show("Введена несуществующая книга");
                return;
            }
            else if (auth.Books.Any(b => b.Id == book.Id))
            {
                MessageBox.Show("Эта книга уже связана с данным автором!");
                return;
            }
            else
            {
                auth.Books.Add(book);
            }

            db.SaveChanges();
            Load(); // Вызов после сохранения
        }
        catch
        {
            MessageBox.Show("Введено неверное значение", "Ошибка");
        }
    }
    private ObservableCollection<AuthAndBook> _authAndBookData;
    public ObservableCollection<AuthAndBook> AuthAndBookData
    {
        get { return _authAndBookData; }
        set { _authAndBookData = value;
```

```
                OnPropertyChanged();
            }
        }
    }
}
```

AuthViewModel.cs

ViewModel для работы с таблицей авторов. Через TestContext загружает данные из таблицы Auths и предоставляет их представлению. Использует модели Auth и обновляет интерфейс при изменении данных.

```
using Models;
using System;
using System.Collections.Generic;
using System.Collections.ObjectModel;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using ViewModels;
using Interfaces;

namespace ORM_DatabaseFirest.ViewModels
{
    public class AuthViewModel : MainViewModel, Tables
    {
        public AuthViewModel() {
            Load();
        }
        public void Load()
        {
            var db = TestContext.GetContext();
            AuthsData = new ObservableCollection<Auth>(db.Auths.ToList());
        }

        #region AddProperties
        private string _title;
        public string Title
        {
            get { return _title; }
            set
            {
                _title = value;
                OnPropertyChanged();
            }
        }

        private int _countPage;

        public int CountPage
        {
            get { return _countPage; }
            set
            {
                _countPage = value;
                OnPropertyChanged();
            }
        }

        private double _price;
        public double Price
```

```
        {
            get { return _price; }
            set
            {
                _price = value;
                OnPropertyChanged();
            }
        }

        private int _book_id;

        public int Book_Id
        {
            get { return _book_id; }
            set
            {
                _book_id = value;
                OnPropertyChanged();
            }
        }
        #endregion

        private ObservableCollection<Auth> _authsData;
        public ObservableCollection<Auth> AuthsData
        {
            get { return _authsData; }
            set
            {
                _authsData = value;
                OnPropertyChanged();
            }
        }
    }
}
```

BookViewModel.cs

ViewModel для работы с таблицей книг. Получает список книг из базы данных и передаёт его в представление.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using ViewModels;
using Models;
using Microsoft.EntityFrameworkCore;
using System.Windows.Documents;
using System.Collections.ObjectModel;
using Interfaces;

namespace ORM_DatabaseFirest.ViewModels
{
    public class BookViewModel : MainViewModel, Tables
    {
        public BookViewModel()
        {
            Load();
        }
        public void Load()
        {
            var db = TestContext.GetContext();
            BooksData = new ObservableCollection<Book>(db.Books.ToList());
```

```
        }

        #region AddProperties
        private string _name;
        public string Name
        {
            get { return _name; }
            set
            {
                _name = value;
                OnPropertyChanged();
            }
        }

        private int _age;

        public int Age
        {
            get { return _age; }
            set
            {
                _age = value;
                OnPropertyChanged();
            }
        }

        private int _auth_id;

        public int Auth_Id
        {
            get { return _auth_id; }
            set
            {
                _auth_id = value;
                OnPropertyChanged();
            }
        }
        #endregion

        private ObservableCollection<Book> _booksData;
        public ObservableCollection<Book> BooksData
        {
            get { return _booksData; }
            set
            {
                _booksData = value;
                OnPropertyChanged();
            }
        }
    }
}
```

BaseViewModel.cs

Базовая ViewModel, реализующая интерфейс INotifyPropertyChanged.

Используется для уведомления интерфейса об изменении данных.

```
using System.ComponentModel;
using System.Runtime.CompilerServices;

namespace ViewModels
{
    public class BaseViewModel : INotifyPropertyChanged
    {
```

```
        public event PropertyChangedEventHandler? PropertyChanged;
        protected void OnPropertyChanged([CallerMemberName] string propertyName =
null )
        {
            PropertyChanged?.Invoke(propertyName, new
PropertyChangedEventArgs(propertyName));
        }
    }
}
```

MainViewModel.cs

Основная ViewModel приложения. Содержит общую логику работы с данными и команды для взаимодействия с интерфейсом.

```
using Microsoft.EntityFrameworkCore;
using Models;
using ORM_DatabaseFirest.Models;
using ORM_DatabaseFirest.ViewModels.Commands;
using System.Windows.Input;

namespace ViewModels
{
    public class MainViewModel : BaseViewModel
    {
        public MainViewModel()
        {
            InitializeValues();
            AddDataCommand = new RelayCommand(AddData);
        }
        private void InitializeValues()
        {
            var db = TestContext.GetContext();
            db.Database.EnsureCreated();

            foreach (var auth in db.Auths.Include(a => a.Books))
            {
                auth.Books.Clear();
            }

            foreach (var book in db.Books.Include(b => b.Auths))
            {
                book.Auths.Clear();
            }

            db.SaveChanges();

            foreach (var i in db.Auths)
            {

                db.Auths.Remove(i);
            }
            Auth user1 = new Auth { Id = 1, Name = "Эдик", Age = 42 };
            Auth user2 = new Auth { Id = 2, Name = "Антон", Age = 19 };
            foreach (var i in db.Books)
            {
                db.Books.Remove(i);
            }
            Book book1 = new Book { Id = 1, Title = "Поле с плотностью 4 см",
CountPage = 1674 };
            Book book2 = new Book { Id = 2, Title = "Ну оно опять не работает",
CountPage = 54 };

            AuthAndBook authAndBook = new AuthAndBook { AuthId = 1, BookId = 1 };
```

```
            db.Auths.AddRange(user1, user2);
            db.Books.AddRange(book1, book2);

            var bookz = db.Books.Find(Convert.ToInt32(1));
            var authz = db.Auths.Find(Convert.ToInt32(1));
            authz.Books.Add(bookz);

            db.SaveChanges();
        }
        public ICommand AddDataCommand { get; set; }
        public virtual void AddData(object parameter) {; }
    }
}
```

```
<Page x:Class="ORM_DatabaseFirest.Views.Pages.AuthBookView"
        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:ORM_DatabaseFirest.Views.Pages"
        mc:Ignorable="d"
        Title="AuthBookView" Height="450" Width="800">
    <Grid>
        <Grid.RowDefinitions>
            <RowDefinition Height="auto"></RowDefinition>
            <RowDefinition Height="auto"></RowDefinition>
        </Grid.RowDefinitions>
        <DataGrid Grid.Row="0" ItemsSource="{Binding AuthAndBookData}"
Height="auto"></DataGrid>
        <StackPanel Grid.Row="1" Orientation="Horizontal">
            <TextBlock Text="Id автора"/>
            <TextBox Text="{Binding AuthId}" Width="75"/>
            <TextBlock Text="Id книги"/>
            <TextBox Text="{Binding BooksId}" Width="75"/>
            <Button Command="{Binding AddDataCommand}" Content="Добавить"/>
        </StackPanel>
    </Grid>
</Page>
```

```
<Page x:Class="ORM_DatabaseFirest.Views.Pages.AuthView"
      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:ORM_DatabaseFirest.Views.Pages"
      mc:Ignorable="d"
      d:DesignHeight="450" d:DesignWidth="800"
      Title="AuthView">

    <Grid>
        <Grid.RowDefinitions>
            <RowDefinition Height="auto"/>
            <RowDefinition Height="auto"/>
        </Grid.RowDefinitions>
        <DataGrid Grid.Row="0" ItemsSource="{Binding AuthsData}"></DataGrid>
    </Grid>
</Page>
```

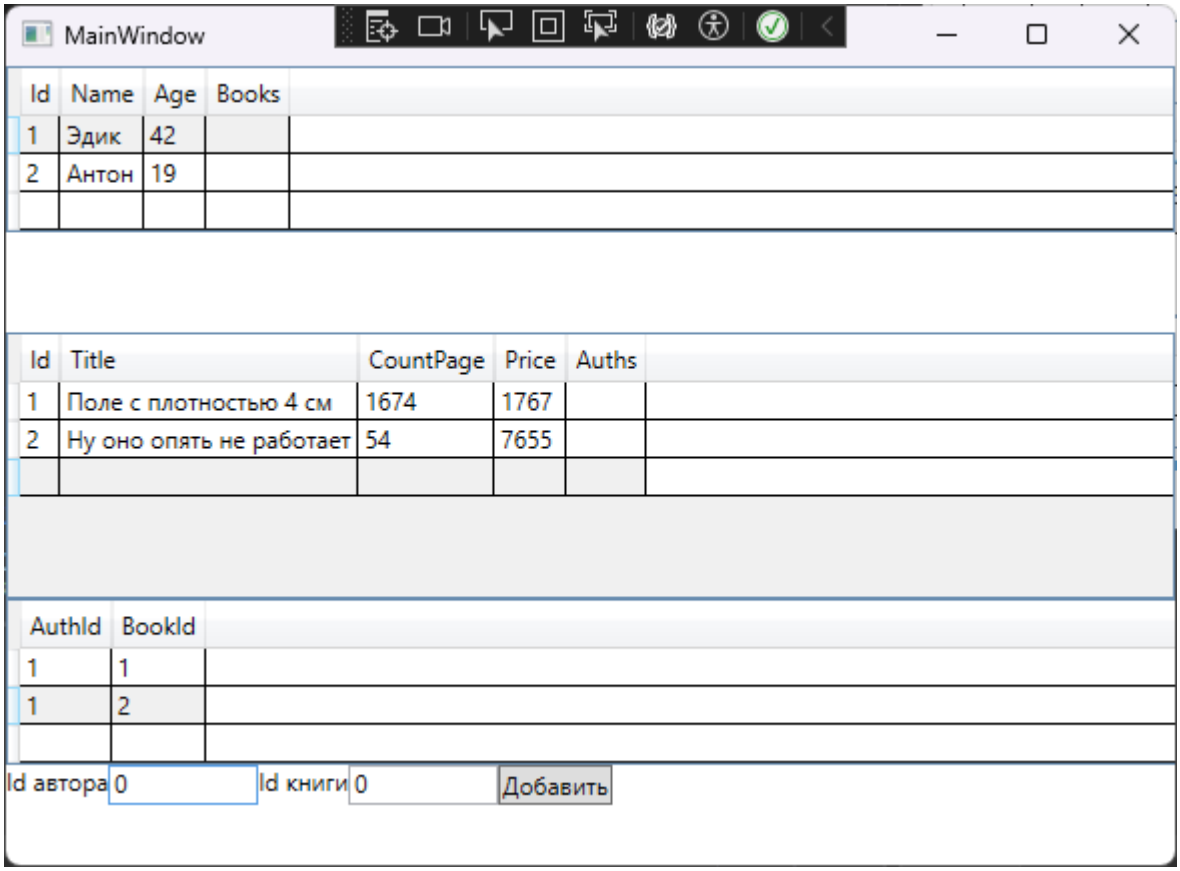
```
<Page x:Class="ORM_DatabaseFirest.Views.Pages.BookView"
      xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
      xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
```

```
      xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
      xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
      xmlns:local="clr-namespace:ORM_DatabaseFirest.Views.Pages"
      mc:Ignorable="d"
      d:DesignHeight="450" d:DesignWidth="800"
      Title="BookView">

    <Grid>
        <DataGrid ItemsSource="{Binding BooksData}"></DataGrid>
    </Grid>
</Page>


<Window x:Class="Views.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:Views"
        mc:Ignorable="d"
        Title="MainWindow" Height="450" Width="600">

    <Grid>
        <Grid.RowDefinitions>
            <RowDefinition/>
            <RowDefinition/>
            <RowDefinition/>
        </Grid.RowDefinitions>
        <Frame Grid.Row="0" Source="Pages/AuthView.xaml"></Frame>
        <Frame Grid.Row="1" Source="Pages/BookView.xaml"></Frame>
        <Frame Grid.Row="2" Source="Pages/AuthBookView.xaml"></Frame>
    </Grid>
</Window>
```

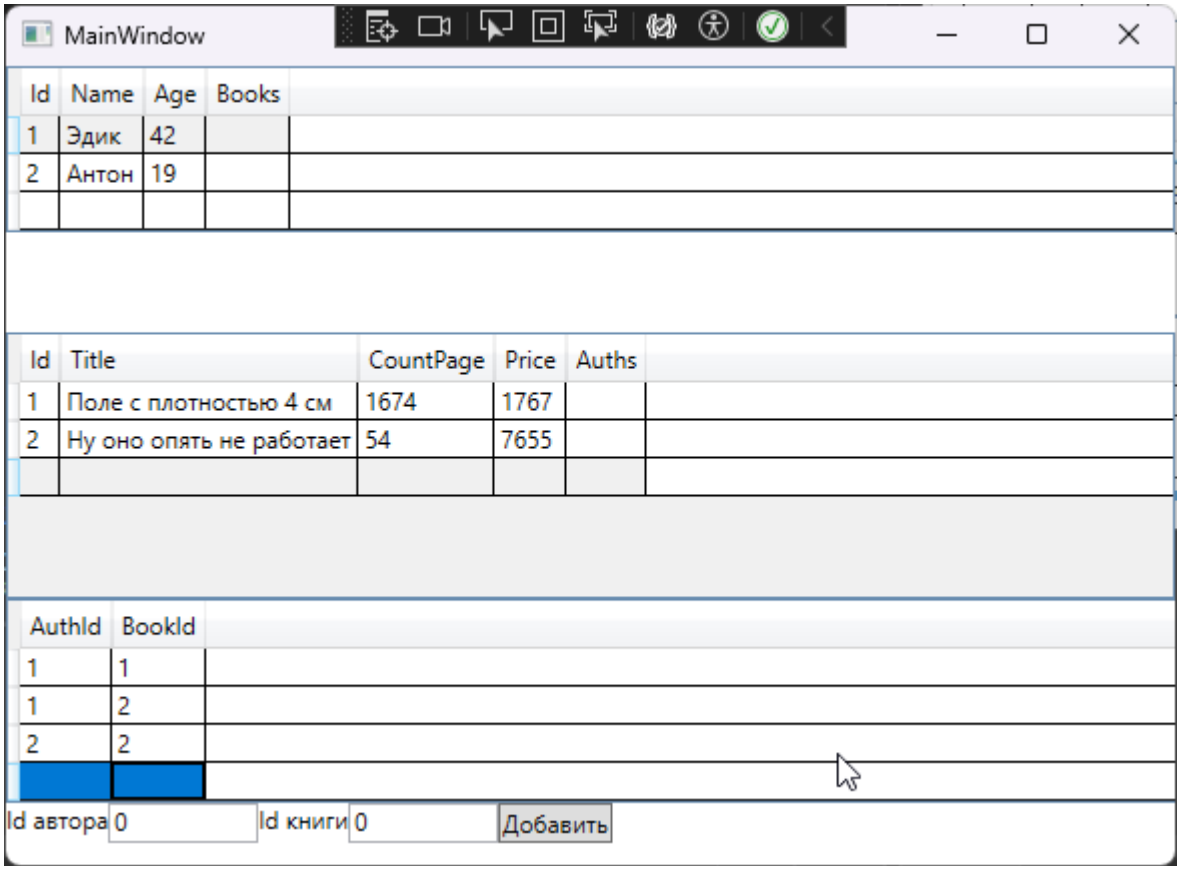
Описание кода

База данных заранее создаётся во внешнем инструменте и содержит связанные таблицы. Entity Framework выполняет обратное проектирование, формируя модели и контекст. ViewModel обращается к TestContext для загрузки данных, включая связанные сущности. Полученные данные передаются в представление и отображаются пользователю без прямого обращения View к базе данных.

Результат



Главный экран приложения

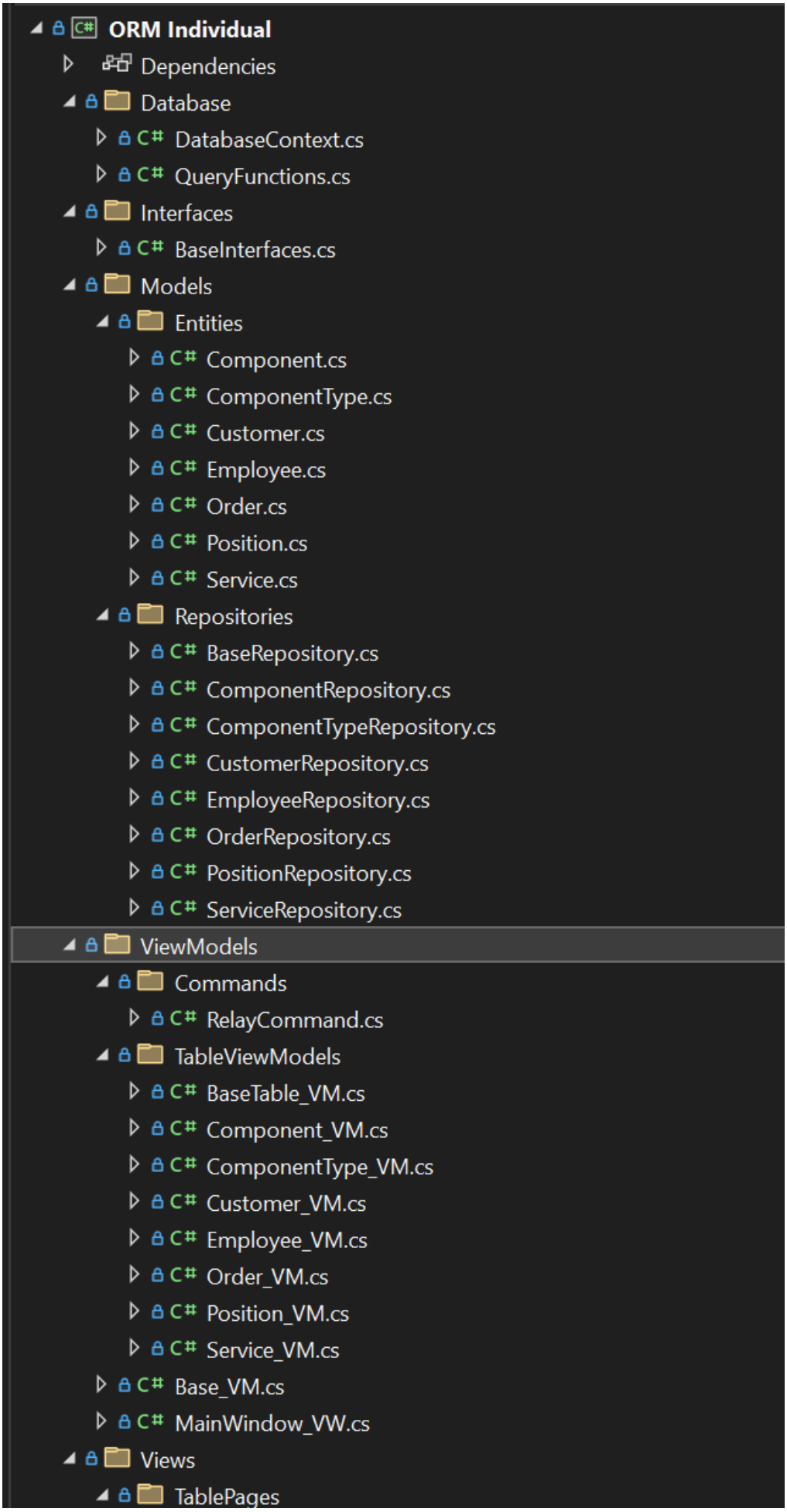


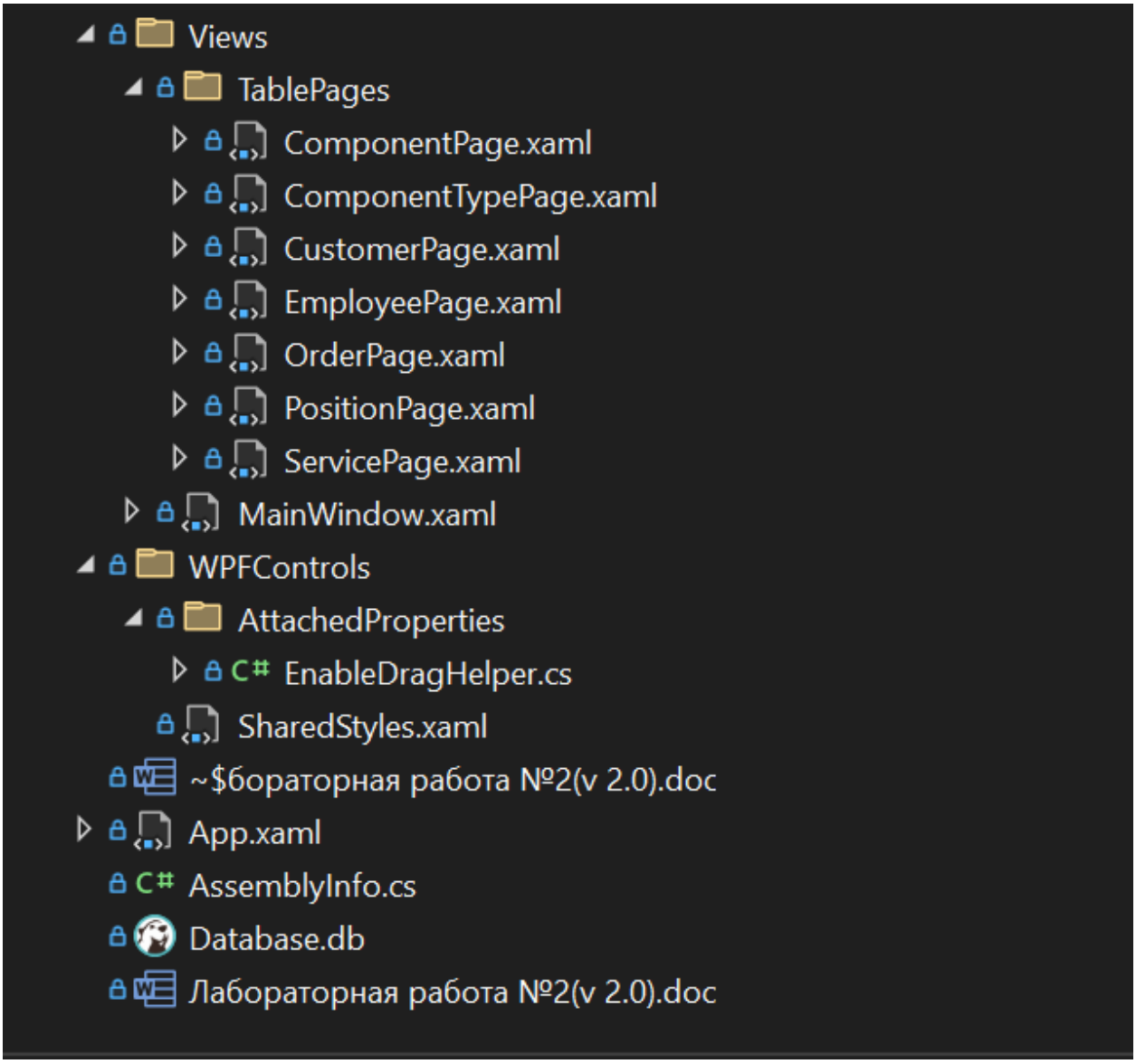
Добавление записи

Индивидуальное задание

Выполнять задание для варианта, выданного преподавателем в лабораторной работе №1.

1. Используя подход database first, создать настольное приложение, которое будет взаимодействовать с созданной БД(из трех связанных таблиц).
2. Реализовать **функции добавления, изменения и удаления данных** в созданные таблицы(через интерфейс приложения).
3. Сформулировать запросы для заданной предметной области:
   - на выборку(2 запроса с различными условиями),
   - на использование статистических функций(1 запрос),
   - на соединение таблиц.


ORM Individual
Dependencies
Database
DatabaseContext.cs
QueryFunctions.cs
Interfaces
BaseInterfaces.cs
Models
Entities
Component.cs
ComponentType.cs
Customer.cs
Employee.cs
Order.cs
Position.cs
Service.cs
Repositories
BaseRepository.cs
ComponentRepository.cs
ComponentTypeRepository.cs
CustomerRepository.cs
EmployeeRepository.cs
OrderRepository.cs
PositionRepository.cs
ServiceRepository.cs
ViewModels
Commands
RelayCommand.cs
TableViewModels
BaseTable_VM.cs
Component_VM.cs
ComponentType_VM.cs
Customer_VM.cs
Employee_VM.cs
Order_VM.cs
Position_VM.cs
Service_VM.cs
Base_VM.cs
MainWindow_VW.cs
Views
TablePages



Структура проекта

Папка Database

DatabaseContext.cs

Контекст базы данных Entity Framework для индивидуального задания.

Содержит DbSet всех таблиц базы данных и настройку связей между ними.

Используется для подключения к SQLite и выполнения операций с данными.

```
using System.IO;
using Microsoft.EntityFrameworkCore;
using ORM_Individual.Models.Entities;

namespace ORM_Individual.Models.Database;

public partial class DatabaseContext : DbContext
{
    public DatabaseContext(){}
    public DatabaseContext(DbContextOptions<DatabaseContext> options)
        : base(options)
```

```
    {
    }
    static DatabaseContext context;
    public static DatabaseContext GetContext()
    {
        if (context == null) context = new DatabaseContext();
        return context;
    }
    public virtual DbSet<Component> Components { get; set; }
    public virtual DbSet<ComponentType> ComponentTypes { get; set; }
    public virtual DbSet<Customer> Customers { get; set; }
    public virtual DbSet<Employee> Employees { get; set; }
    public virtual DbSet<Order> Orders { get; set; }
    public virtual DbSet<Position> Positions { get; set; }
    public virtual DbSet<Service> Services { get; set; }
    protected override void OnConfiguring(DbContextOptionsBuilder optionsBuilder)
    {
        if (optionsBuilder.IsConfigured)
        {
            return;
        }
        var dbPath = GetDatabasePath();
        optionsBuilder.UseSqlite($"Data Source={dbPath}");
    }
    private static string GetDatabasePath()
    {
        var baseDirectory = AppContext.BaseDirectory;
        var projectRoot = FindProjectRoot(baseDirectory);
        var dbDirectory = projectRoot ?? baseDirectory;
        var path = Path.Combine(dbDirectory, "Database.db");
        var directory = Path.GetDirectoryName(path);
        if (!string.IsNullOrEmpty(directory) && !Directory.Exists(directory))
        {
            Directory.CreateDirectory(directory);
        }
        return path;
    }
    private static string? FindProjectRoot(string startDirectory)
    {
        var directoryInfo = new DirectoryInfo(startDirectory);
        while (directoryInfo != null)
        {
            var projectFile = Path.Combine(directoryInfo.FullName, "ORM
Individual.csproj");
            if (File.Exists(projectFile))
            {
                return directoryInfo.FullName;
            }
            directoryInfo = directoryInfo.Parent;
        }
        return null;
    }
    protected override void OnModelCreating(ModelBuilder modelBuilder)
    {
        modelBuilder.Entity<Component>(entity =>
        {
            entity.ToTable("Component");
            entity.Property(e => e.Id)
                .ValueGeneratedOnAdd()
                .HasColumnName("id");
            entity.Property(e => e.Brand).HasColumnName("brand");
            entity.Property(e => e.Description).HasColumnName("description");
            entity.Property(e =>
e.ManufacturerCompany).HasColumnName("manufacturer_company");
```

```
            entity.Property(e =>
e.ManufacturerCountry).HasColumnName("manufacturer_country");
            entity.Property(e => e.Price)
                .HasColumnType("DECIMAL")
                .HasColumnName("price");
            entity.Property(e => e.ReleaseDate).HasColumnName("release_date");
            entity.Property(e => e.Specifications).HasColumnName("specifications");
            entity.Property(e => e.TypeId).HasColumnName("type_id");
            entity.Property(e => e.Warranty)
                .HasColumnType("INT")
                .HasColumnName("warranty");

            entity.HasOne(d => d.Type).WithMany(p => p.Components).HasForeignKey(d
=> d.TypeId);
        });
        modelBuilder.Entity<ComponentType>(entity =>
        {
            entity.ToTable("ComponentType");

            entity.Property(e => e.Id)
                .ValueGeneratedOnAdd()
                .HasColumnName("id");
            entity.Property(e => e.Description).HasColumnName("description");
            entity.Property(e => e.Name).HasColumnName("name");
        });
        modelBuilder.Entity<Customer>(entity =>
        {
            entity.ToTable("Customer");
            entity.Property(e => e.Id)
                .ValueGeneratedOnAdd()
                .HasColumnName("id");
            entity.Property(e => e.Address).HasColumnName("address");
            entity.Property(e => e.FullName).HasColumnName("full_name");
            entity.Property(e => e.Phone).HasColumnName("phone");
        });
        modelBuilder.Entity<Employee>(entity =>
        {
            entity.ToTable("Employee");
            entity.Property(e => e.Id)
                .ValueGeneratedOnAdd()
                .HasColumnName("id");
            entity.Property(e => e.Address).HasColumnName("address");
            entity.Property(e => e.Age)
                .HasColumnType("INT")
                .HasColumnName("age");
            entity.Property(e => e.FullName).HasColumnName("full_name");
            entity.Property(e => e.Gender)
                .HasColumnType("BOOL")
                .HasColumnName("gender");
            entity.Property(e => e.PassportData).HasColumnName("passport_data");
            entity.Property(e => e.Phone).HasColumnName("phone");
            entity.Property(e => e.PositionId).HasColumnName("position_id");
            entity.HasOne(d => d.Position).WithMany(p =>
p.Employees).HasForeignKey(d => d.PositionId);
        });
        modelBuilder.Entity<Order>(entity =>
        {
            entity.ToTable("Order");

            entity.Property(e => e.Id)
                .ValueGeneratedOnAdd()
                .HasColumnName("id");
            entity.Property(e => e.CompletionDate).HasColumnName("completion_date");
            entity.Property(e => e.Component1Id).HasColumnName("component1_id");
            entity.Property(e => e.Component2Id).HasColumnName("component2_id");
```

```
            entity.Property(e => e.Component3Id).HasColumnName("component3_id");
            entity.Property(e => e.CustomerId).HasColumnName("customer_id");
            entity.Property(e => e.EmployeeId).HasColumnName("employee_id");
            entity.Property(e => e.IsCompleted)
                .HasColumnType("BOOL")
                .HasColumnName("is_completed");
            entity.Property(e => e.IsPaid)
                .HasColumnType("BOOL")
                .HasColumnName("is_paid");
            entity.Property(e => e.OrderDate).HasColumnName("order_date");
            entity.Property(e => e.Prepayment)
                .HasColumnType("DECIMAL")
                .HasColumnName("prepayment");
            entity.Property(e => e.Service1Id).HasColumnName("service1_id");
            entity.Property(e => e.Service2Id).HasColumnName("service2_id");
            entity.Property(e => e.Service3Id).HasColumnName("service3_id");
            entity.Property(e => e.TotalAmount)
                .HasColumnType("DECIMAL")
                .HasColumnName("total_amount");
            entity.Property(e => e.TotalWarranty).HasColumnName("total_warranty");

            entity.HasOne(d => d.Component1).WithMany(p =>
p.OrderComponent1s).HasForeignKey(d => d.Component1Id);

            entity.HasOne(d => d.Component2).WithMany(p =>
p.OrderComponent2s).HasForeignKey(d => d.Component2Id);

            entity.HasOne(d => d.Component3).WithMany(p =>
p.OrderComponent3s).HasForeignKey(d => d.Component3Id);

            entity.HasOne(d => d.Customer).WithMany(p => p.Orders).HasForeignKey(d
=> d.CustomerId);

            entity.HasOne(d => d.Employee).WithMany(p => p.Orders).HasForeignKey(d
=> d.EmployeeId);

            entity.HasOne(d => d.Service1).WithMany(p =>
p.OrderService1s).HasForeignKey(d => d.Service1Id);

            entity.HasOne(d => d.Service2).WithMany(p =>
p.OrderService2s).HasForeignKey(d => d.Service2Id);

            entity.HasOne(d => d.Service3).WithMany(p =>
p.OrderService3s).HasForeignKey(d => d.Service3Id);
        });

        modelBuilder.Entity<Position>(entity =>
        {
            entity.ToTable("Position");

            entity.Property(e => e.Id)
                .ValueGeneratedOnAdd()
                .HasColumnName("id");
            entity.Property(e => e.Duties).HasColumnName("duties");
            entity.Property(e => e.Name).HasColumnName("name");
            entity.Property(e => e.Requirements).HasColumnName("requirements");
            entity.Property(e => e.Salary)
                .HasColumnType("DECIMAL")
                .HasColumnName("salary");
        });

        modelBuilder.Entity<Service>(entity =>
        {
            entity.ToTable("Service");
```

```
            entity.Property(e => e.Id)
                .ValueGeneratedOnAdd()
                .HasColumnName("id");
            entity.Property(e => e.Description).HasColumnName("description");
            entity.Property(e => e.Name).HasColumnName("name");
            entity.Property(e => e.Price)
                .HasColumnType("DECIMAL")
                .HasColumnName("price");
        });

        OnModelCreatingPartial(modelBuilder);
    }

    partial void OnModelCreatingPartial(ModelBuilder modelBuilder);
}
```

Папка Interfaces

BaseInterface.cs

Файл с интерфейсами проекта. Содержит интерфейс сущности с идентификатором, интерфейс репозитория с базовыми CRUD-методами и вспомогательный интерфейс для взаимодействия с элементами интерфейса.

```
using System.Collections.ObjectModel;
using System.Data;
using System.Windows.Controls;

namespace ORM_Individual.Interfaces
{
    public interface IRepository<T> where T : IEntity
    {
        ObservableCollection<T> GetAll();
        T Add(T entity);
        T Update(T entity);
        void Remove(int id);
        T? FindById(int id);
        T CreateInstanceFromDataRow(DataRow dataRow);
        ObservableCollection<T> IdQueries(ObservableCollection<T> entities, int
IdMoreThan, int IdLessThan);
    }
    public interface IEntity
    {
        public int Id { get; set; }
    }
    public interface ViewCodeBehind
    {
        public Control GetDataGrid();
    }
}
```

Папка Models

Component.cs

Модель сущности «Комплектующее». Описывает структуру таблицы базы данных и содержит внешний ключ для связи с типом комплектующих.

```
using ORM_Individual.Interfaces;
```

```
namespace ORM_Individual.Models.Entities;

public partial class Component : IEntity
{
    public int Id { get; set; }

    public int? TypeId { get; set; }

    public string? Brand { get; set; }

    public string? ManufacturerCompany { get; set; }

    public string? ManufacturerCountry { get; set; }

    public string? ReleaseDate { get; set; }

    public string? Specifications { get; set; }

    public int? Warranty { get; set; }

    public string? Description { get; set; }

    public decimal? Price { get; set; }

    public virtual ICollection<Order> OrderComponent1s { get; set; } = new
List<Order>();

    public virtual ICollection<Order> OrderComponent2s { get; set; } = new
List<Order>();

    public virtual ICollection<Order> OrderComponent3s { get; set; } = new
List<Order>();

    public virtual ComponentType? Type { get; set; }
}
```

BaseRepository.cs

Базовый репозиторий для работы с сущностями базы данных. Реализует общие операции добавления, обновления, удаления и получения данных.

```
using ORM_Individual.Interfaces;
using Microsoft.EntityFrameworkCore;
using System.Collections.ObjectModel;
using System.Data;
using ORM_Individual.Models.Database;


namespace ORM_Individual.Models.Repositories
{
    public abstract class BaseRepository<T> : IRepository<T> where T : class,
IEntity
    {
        protected DatabaseContext Context { get; }
        protected DbSet<T> Set { get; }
        protected BaseRepository()
        {
            Context = DatabaseContext.GetContext();
            Set = Context.Set<T>();
        }
        public virtual T Add(T entity)
```

```
        {
            if (entity == null)
            {
                throw new ArgumentNullException(nameof(entity));
            }

            Set.Add(entity);
            Context.SaveChanges();

            return entity;
        }
        public virtual T CreateInstance()
        {
            return Activator.CreateInstance<T>();
        }
        public virtual ObservableCollection<T> GetAll()
        {
            return new ObservableCollection<T>(Set.AsNoTracking().ToList());
        }
        public virtual void Remove(int id)
        {
            var entity = FindById(id);
            if (entity == null)
            {
                return;
            }
            Set.Remove(entity);
            Context.SaveChanges();
        }
        public virtual T Update(T entity)
        {
            if (entity == null)
            {
                throw new ArgumentNullException(nameof(entity));
            }

            Set.Update(entity);
            Context.SaveChanges();
            return entity;
        }
        public virtual T? FindById(int id)
        {
            return Set.Find(id);
        }
        public virtual T CreateInstanceFromDataRow(DataRow row) { throw new
NotImplementedException(); }

        #region Queries
        public  ObservableCollection<T> IdQueries(ObservableCollection<T> entities,
int IdMoreThan, int IdLessThan)
        {
            return new ObservableCollection<T>(
                (IEnumerable<T>)
                (
                from entity in entities
                where entity.Id >= IdMoreThan && entity.Id <= IdLessThan
                select entity)
                );
        }
        #endregion
    }
}
```

ComponentRepository.cs

Репозиторий для таблицы комплектующих. Используется для выполнения CRUD-операций и загрузки связанных данных.

```
using Microsoft.EntityFrameworkCore;
using ORM_Individual.Models.Entities;
using System.Collections.ObjectModel;

namespace ORM_Individual.Models.Repositories
{
    public class ComponentRepository : BaseRepository<Component>
    {
        public override ObservableCollection<Component> GetAll()
        {
            return new ObservableCollection<Component>(
                Set.Include(c => c.Type)
                   .AsNoTracking()
                   .ToList());
        }
    }
}
```

PositionRepository.cs

Репозиторий для таблицы должностей. Содержит методы для выполнения запросов со статистическими функциями, такими как поиск максимального и минимального оклада.

```
using ORM_Individual.Models.Entities;
using System.Collections.ObjectModel;

namespace ORM_Individual.Models.Repositories
{
    public class PositionRepository : BaseRepository<Position>
    {
        public ObservableCollection<Position>
SalaryQueryMax(ObservableCollection<Position> entities)
        {
            return new ObservableCollection<Position>(
                from entity in entities
                where entity.Salary == Context.Positions.Select(p =>
p.Salary).AsEnumerable().Max()
                select entity
                );
        }
        public ObservableCollection<Position>
SalaryQueryMin(ObservableCollection<Position> entities)
        {
            return new ObservableCollection<Position>(
                from entity in entities
                where entity.Salary == Context.Positions.Select(p =>
p.Salary).AsEnumerable().Min()
                select entity
                );
            }
    }
}
```

EmployeeRepository.cs

Репозиторий для работы с таблицей сотрудников. Содержит методы получения списка сотрудников с учётом связанных данных из таблицы должностей. Также включает запросы с объединением таблиц и фильтрацией по значениям оклада.

```
using Microsoft.EntityFrameworkCore;
using ORM_Individual.Models.Entities;
using System.Collections.ObjectModel;

namespace ORM_Individual.Models.Repositories
{
    public class EmployeeRepository : BaseRepository<Employee>
    {
        public override ObservableCollection<Employee> GetAll()
        {
            return new ObservableCollection<Employee>(
                Set.Include(e => e.Position)
                   .AsNoTracking()
                   .ToList());
        }

        public ObservableCollection<Employee>
FilterByPositionSalary(ObservableCollection<Employee> entities, decimal salaryFrom)
        {
            return new ObservableCollection<Employee>(
                from employee in entities
                join position in Context.Positions on employee.PositionId equals
position.Id
                select employee
            );
        }
    }
}
```

Папка ViewModels

RelayCommand.cs

Реализация интерфейса ICommand. Используется для обработки действий пользователя в интерфейсе приложения.

```
using System.Windows.Input;

namespace ORM_Individual.ViewModels.Commands
{
    public class RelayCommand : ICommand
    {
        public Action<object> _execute;
        public RelayCommand(Action<object> execute) {
            _execute = execute;
        }
        public event EventHandler? CanExecuteChanged;
        public bool CanExecute(object? parameter)
```

```
        {
            return true;
        }
        public void Execute(object? parameter)
        {
            _execute(parameter);
        }
    }
}
```

BaseTable_VM.cs

Базовая ViewModel, реализующая INotifyPropertyChanged. Обеспечивает обновление интерфейса при изменении данных.

```
using ORM_Individual.Interfaces;
using ORM_Individual.Models.Database;
using ORM_Individual.ViewModels.Commands;
using System.Collections.ObjectModel;
using System.Data;
using System.Windows.Controls;
using System.Windows.Input;
using Newtonsoft.Json;

namespace ORM_Individual.ViewModels.TableViewModels
{
    public abstract class BaseTable_VM<T> : Base_VM, IDisposable where T : IEntity
    {
        private static bool _databaseInitialized;
        public ObservableCollection<T> _source = new();
        protected IRepository<T> Repository { get; }
        public ICommand RowEditEndingCommand { get; }
        public ICommand SaveRowCommand { get; }
        public ICommand DeleteRowsCommand { get; }
        public virtual ObservableCollection<T> Source
        {
            get => _source;
            set
            {
                _source = value;
                OnPropertyChanged();
            }
        }

        private string _serialised;
        public string Serialised
        {
            get { return _serialised; }
            set { _serialised = value;
                OnPropertyChanged();
            }
        }

        private string temp {  get; set; }

        protected BaseTable_VM(IRepository<T> repository)
        {
            Repository = repository ?? throw new
ArgumentNullException(nameof(repository));
            EnsureDatabase();
            LoadSource();
            IsIdQuery = false;
```

```
            RowEditEndingCommand = new RelayCommand(RowEditEnding);
            DeleteRowsCommand = new RelayCommand(DeleteRows);
            UseIdQueryCommand = new RelayCommand(UserIdQuery);
            UpdateSerString();
        }
        public void RowEditEnding(object parameter)
        {
            if (parameter is DataGridRowEditEndingEventArgs e)
            {
                e.Row.BindingGroup?.CommitEdit();
                if ((e.EditAction == DataGridEditAction.Commit) &&
(e.Row.DataContext is IEntity entity))
                {
                    try
                    {
                        foreach (IEntity row in Repository.GetAll())
                        {
                            if (row.Id == entity.Id)
                            {
                                Repository.Update((T)entity);
                                return;
                            }
                        }
                        Repository.Add((T)entity);
                        UpdateSerString();
                    }
                    catch (Exception ex) {
                        LoadSource();
                    }
                }
            }
        }
        private void UpdateSerString()
        {
            Serialised = string.Empty;

            var settings = new JsonSerializerSettings
            {
                ReferenceLoopHandling = ReferenceLoopHandling.Ignore,
                PreserveReferencesHandling = PreserveReferencesHandling.Objects,
                Formatting = Formatting.Indented
            };
            temp = JsonConvert.SerializeObject(Source, settings);
            foreach (var str in temp.Split(','))
            {
                if (!(str[0] == '{' || str[1] == '{'))
                {
                    Serialised += ('\t' + str + "\n");
                }
                else
                {
                    Serialised += (str + "\n");
                }
            }
        }
        public void DeleteRows(object parameter)
        {
            if (parameter is KeyEventArgs e)
            {
                if (e.Key != Key.Delete)
                    return;
                if (e.OriginalSource is TextBox)
                    return;
                var grid = (DataGrid)e.Source;
```

```
        try
        {
            var toDelete = grid.SelectedItems.Cast<T>().ToList();
            foreach (T item in toDelete)
            {
                Source.Remove(item);
                if (item is IEntity entity)
                {
                    Repository.Remove(entity.Id);
                }
            }
        }
        catch (Exception ex) {
            LoadSource();
        }
    }
}
protected void LoadSource()
{
    Source = Repository.GetAll();
}
private static void EnsureDatabase()
{
    var context = DatabaseContext.GetContext();
    context.Database.EnsureCreated();
    _databaseInitialized = true;
}

#region Queries
public ICommand UseIdQueryCommand { get; }
public bool IsIdQuery;
private void UserIdQuery(object parameter)
{
    if(IsIdQuery == false)
    {
        IsIdQuery = true;
        IdQuerySelect();
    }
    else
    {
        IsIdQuery = false;
        LoadSource();
    }
}
private void IdQuerySelect()
{
    if (IsIdQuery == true) {
        Source = Repository.IdQueries(Source, IdFrom, IdTo);
    }
}

public void Dispose()
{
    throw new NotImplementedException();
}

private int _idFrom;
public int IdFrom
{
    get { return _idFrom; }
    set { _idFrom = value;
        OnPropertyChanged();
        IdQuerySelect();
    }
}
```

```
        private int _idTo;
        public int IdTo
        {
            get { return _idTo; }
            set
            {
                _idTo = value;
                OnPropertyChanged();
                IdQuerySelect();
            }
        }
        #endregion
    }
}
```

Employee_VM.cs

ViewModel для таблицы сотрудников. Использует репозиторий сотрудников и поддерживает выполнение специальных запросов, связанных с должностями и окладами.

```
using ORM_Individual.Models.Entities;
using ORM_Individual.Models.Repositories;
using ORM_Individual.ViewModels.Commands;
using System.Collections.ObjectModel;
using System.Windows.Controls;
using System.Windows.Input;

namespace ORM_Individual.ViewModels.TableViewModels
{
    public class Employee_VM : BaseTable_VM<Employee>
    {
        private readonly EmployeeRepository _employeeRepository;

        private bool _usePositionSalaryQuery;
        private decimal _positionSalaryFrom;

        public ICommand PositionSalaryQueryCommand { get; }

        public Employee_VM() : base(new EmployeeRepository())
        {
            _employeeRepository = (EmployeeRepository)Repository;
            PositionSalaryQueryCommand = new RelayCommand(PositionSalaryQuery);
        }

        public decimal PositionSalaryFrom
        {
            get => _positionSalaryFrom;
            set
            {
                _positionSalaryFrom = value;
                OnPropertyChanged();
                ApplyPositionSalaryQuery();
            }
        }

        public virtual ObservableCollection<Employee> Source
        {
            get => _source;
            set
            {
                _source = value;
```

```
                OnPropertyChanged();

            }
        }

        public DataGridColumn[] temp;

        private void PositionSalaryQuery(object parameter)
        {
            _usePositionSalaryQuery = !_usePositionSalaryQuery;
            var dataGrid = parameter as DataGrid;

            if (temp == null)
            {
                temp = new DataGridColumn[8];
                dataGrid.Columns.CopyTo(temp,0);
            }
            if (_usePositionSalaryQuery)
            {
                dataGrid.AutoGenerateColumns = true;
                dataGrid.CanUserAddRows = false;
                dataGrid.CanUserDeleteRows = false;

                dataGrid.Columns.Clear();
                ApplyPositionSalaryQuery();
            }
            else
            {
                LoadSource();
                dataGrid.AutoGenerateColumns = false;
                foreach (var column in temp)
                {
                    dataGrid.Columns.Add(column);
                }
            }
        }

        private void ApplyPositionSalaryQuery()
        {
            if (_usePositionSalaryQuery)
            {
                var temp = Source;
                Source = _employeeRepository.FilterByPositionSalary(Source,
PositionSalaryFrom);
            }
        }
    }
}
```

Position_VM.cs

ViewModel для таблицы должностей. Загружает данные из таблицы Position и выполняет запросы на получение минимального и максимального значения оклада с выводом результата в интерфейс.

```
using ORM_Individual.Models.Entities;
using ORM_Individual.Models.Repositories;
using ORM_Individual.ViewModels.Commands;
using System.Windows.Input;

namespace ORM_Individual.ViewModels.TableViewModels
```

```
{
    public class Position_VM : BaseTable_VM<Position>
    {
        public Position_VM() : base(new PositionRepository())
        {
            MaxSalaryQueryCommand = new RelayCommand(MaxSalaryQuery);
            MinSalaryQueryCommand = new RelayCommand(MinSalaryQuery);
            IsUnderFunction = false;
        }
        #region Queries
        public ICommand MaxSalaryQueryCommand { get; }
        public ICommand MinSalaryQueryCommand { get; }
        public bool IsUnderFunction { get; set; }
        private void MaxSalaryQuery(object parameter)
        {
            if (Repository is PositionRepository positionrep)
            {
                if (!IsUnderFunction)
                {

                    Source = positionrep.SalaryQueryMax(Source);
                    IsUnderFunction = true;
                }
                else
                {
                    LoadSource();
                }
            }
        }
        private void MinSalaryQuery(object parameter)
        {
            if (Repository is PositionRepository positionrep)
            {
                if (!IsUnderFunction)
                {
                    Source = positionrep.SalaryQueryMin(Source);
                    IsUnderFunction = true;
                }
                else
                {
                    LoadSource();
                    IsUnderFunction = false;
                }
            }
        }
        #endregion
    }
}
```

Base_VM.cs

Базовая ViewModel, реализующая INotifyPropertyChanged. Обеспечивает обновление интерфейса при изменении данных.

```
using System.ComponentModel;
using System.Runtime.CompilerServices;

namespace ORM_Individual.ViewModels
{
    public class Base_VM : INotifyPropertyChanged
    {
        public event PropertyChangedEventHandler? PropertyChanged;
```

```
        protected void OnPropertyChanged([CallerMemberName] string? propertyName =
null)
        {
            PropertyChanged?.Invoke(this, new
PropertyChangedEventArgs(propertyName));
        }
    }
}
```

MainWindow_VW.cs

ViewModel главного окна. Хранит список доступных таблиц (Tables) и коллекцию открытых “окон-таблиц” (TableFrames). Реализует команды добавления и удаления таблиц: при добавлении создаёт нужную Page (ComponentPage, EmployeePage и т.д.), помещает её во Frame внутри Border, делает этот блок перетаскиваемым (DragBehavior) и добавляет на Canvas.

```
using ORM_Individual.ViewModels.Commands;
using ORM_Individual.Views.TablePages;
using ORM_Individual.WPFControls.AttachedProperties;
using System.Collections.ObjectModel;
using System.Windows;
using System.Windows.Controls;
using System.Windows.Input;
using System.Windows.Media;

namespace ORM_Individual.ViewModels
{
    public class MainWindow_VW : Base_VM
    {
        public ICommand AddTableCommand { get; }
        public ICommand DeleteTableCommand { get; }
        public ObservableCollection<string> Tables { get; set; }
        public ObservableCollection<Border> TableFrames { get; set; }

        private readonly Dictionary<string, Func<Page>> _pages;
        public MainWindow_VW()
        {
            _pages = new Dictionary<string, Func<Page>> {
            { "Component",     () => new ComponentPage() },
            { "ComponentType", () => new ComponentTypePage() },
            { "Employee",      () => new EmployeePage() },
            { "Order",         () => new OrderPage() },
            { "Position",      () => new PositionPage() },
            { "Service",       () => new ServicePage() },
            { "Customer",      () => new CustomerPage() }
            };

            Tables = new ObservableCollection<string>(_pages.Keys);
            TableFrames = new ObservableCollection<Border>();
            AddTableCommand = new RelayCommand(AddTable);
            DeleteTableCommand = new RelayCommand(DeleteTable);
        }
        public void DeleteTable(object parameter)
        {
            if(parameter is Border border)
            {
                TableFrames.Remove(border);
```

```
                OnPropertyChanged(nameof(TableFrames));
            }
        }
        public void AddTable(object parameter)
        {
            if (parameter == null) return;

            string tableName = parameter.ToString();
            if (_pages.ContainsKey(tableName))
            {
                var page = _pages[tableName]();

                var frame = new Frame();
                frame.Navigate(page);

                var border = new Border
                {
                    Background = Brushes.Wheat,
                    BorderBrush = Brushes.Black,
                    BorderThickness = new Thickness(1),
                    CornerRadius = new CornerRadius(5)
                };
                var mi = new MenuItem
                {
                    Header = "Delete Table",
                    Command = DeleteTableCommand,
                    CommandParameter = border
                };
                var cm = new ContextMenu();
                cm.Items.Add(mi);
                border.ContextMenu = cm;

                DragBehavior.SetIsDraggable(border, true);

                border.Child = frame;

                Canvas.SetLeft(border, 50 + (TableFrames.Count * 20));
                Canvas.SetTop(border, 50 + (TableFrames.Count * 20));

                TableFrames.Add(border);

                OnPropertyChanged(nameof(TableFrames));
            }
        }
    }
}
```

Папка Views

ComponentPage.xaml

Страница WPF для таблицы комплектующих. Содержит DataGrid и элементы управления для выполнения команд ViewModel (добавление/удаление/сохранение) и отображения данных Component.

```
<Page x:Class="ORM_Individual.Views.TablePages.ComponentPage"
      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:viewModels="clr-namespace:ORM_Individual.ViewModels.TableViewModels"
      xmlns:behaviors="clr-namespace:ORM_Individual.WPFControls.AttachedProperties"
      xmlns:i="http://schemas.microsoft.com/xaml/behaviors"
      mc:Ignorable="d"
      d:DesignHeight="450" d:DesignWidth="800"
      Height="auto"
      Width="auto"
      Title="ComponentPage">

    <Page.DataContext>
        <viewModels:Component_VM />
    </Page.DataContext>

    <Grid>
        <Grid.RowDefinitions>
            <RowDefinition Height="23"/>
            <RowDefinition Height="Auto"/>
        </Grid.RowDefinitions>
        <TextBlock Text="Component Table"
                   Grid.ColumnSpan="2"
                   Style="{StaticResource PageHeaderTextBlockStyle}"
                   Background="DarkSlateGray"
                   />

        <Border Grid.Row="1"
                Style="{StaticResource TableBorderStyle}">
            <Grid>
                <Grid.ColumnDefinitions>
                    <ColumnDefinition Width="auto" />
                    <ColumnDefinition Width="auto" />
                </Grid.ColumnDefinitions>
                <DataGrid
                    Grid.Row="1"
                    Grid.Column="0"
                    ItemsSource="{Binding Source, Mode=TwoWay}"
                    Style="{StaticResource TableDataGridStyle}">
                    <DataGrid.Columns>
                        <DataGridTextColumn Header="Id" Binding="{Binding Id}"/>
                        <DataGridTextColumn Header="Type" Binding="{Binding
TypeId}"/>
                        <DataGridTextColumn Header="Brand" Binding="{Binding
Brand}"/>
                        <DataGridTextColumn Header="Manufacturer" Binding="{Binding
ManufacturerCompany}"/>
                        <DataGridTextColumn Header="Country" Binding="{Binding
ManufacturerCountry}"/>
                        <DataGridTextColumn Header="Release Date" Binding="{Binding
ReleaseDate}"/>
                        <DataGridTextColumn Header="Specifications"
Binding="{Binding Specifications}"/>
                        <DataGridTextColumn Header="Warranty" Binding="{Binding
Warranty}"/>
                        <DataGridTextColumn Header="Description" Binding="{Binding
Description}"/>
                        <DataGridTextColumn Header="Price" Binding="{Binding
Price}"/>
                    </DataGrid.Columns>
                    <i:Interaction.Triggers>
                        <i:EventTrigger EventName="RowEditEnding">
                            <i:InvokeCommandAction Command="{Binding
RowEditEndingCommand}"
                                                   PassEventArgsToCommand="True"/>
                        </i:EventTrigger>
                        <i:EventTrigger EventName="PreviewKeyDown">
                            <i:InvokeCommandAction
```

```
                                        Command="{Binding DeleteRowsCommand}"
                                        PassEventArgsToCommand="True" />
                            </i:EventTrigger>
                        </i:Interaction.Triggers>
                        <DataGrid.ContextMenu>
                            <ContextMenu>
                                <MenuItem Header="Delete Table"></MenuItem>
                            </ContextMenu>
                        </DataGrid.ContextMenu>
                    </DataGrid>
                    <Expander Grid.Column="1"
                              Grid.Row="1"
                              Style="{StaticResource QueryExpanderStyle}">
                        <Border
                                Style="{StaticResource QueryContainerBorderStyle}">
                            <Grid>
                                <Grid.RowDefinitions>
                                    <RowDefinition Height="auto"/>
                                    <RowDefinition/>
                                    <RowDefinition/>
                                    <RowDefinition/>
                                    <RowDefinition/>
                                </Grid.RowDefinitions>
                                <TextBlock
                                    Grid.Row="0"
                                    Grid.ColumnSpan="2"
                                    Text="Queries"
                                    Style="{StaticResource QueryTitleTextBlockStyle}" />
                                <Border
                                       Grid.Row="1"
                                       Grid.Column="0"
                                       Style="{StaticResource QueryGroupBorderStyle}">
                                    <Grid>
                                        <Grid.ColumnDefinitions>
                                            <ColumnDefinition/>
                                            <ColumnDefinition/>
                                        </Grid.ColumnDefinitions>
                                        <Grid.RowDefinitions>
                                            <RowDefinition/>
                                            <RowDefinition/>
                                            <RowDefinition/>
                                            <RowDefinition/>
                                            <RowDefinition/>
                                        </Grid.RowDefinitions>
                                        <TextBlock
                                            Grid.Row="0"
                                            Text="Id"
                                            Style="{StaticResource
QueryLabelTextBlockStyle}" />
                                        <Button
                                            Grid.Column="1"
                                            Content="+"
                                            Style="{StaticResource QueryAddButtonStyle}"
                                            Command="{Binding UseIdQueryCommand}" />
                                        <TextBlock
                                            TextAlignment="Right"
                                            Grid.Row="1"
                                            Text="from"
                                            Style="{StaticResource
QueryLabelTextBlockStyle}" />
                                        <TextBox
                                            Grid.Row="1"
                                            Grid.Column="1"
                                            Text="{Binding IdFrom, Mode=TwoWay}" />
                                        <TextBlock
```

```
                                        TextAlignment="Right"
                                        Grid.Row="2"
                                        Text="to"
                                        Style="{StaticResource
QueryLabelTextBlockStyle}" />
                                <TextBox
                                        Grid.Row="2"
                                        Grid.Column="1"
                                        Text="{Binding IdTo, Mode=TwoWay}" />
                            </Grid>
                        </Border>
                    </Grid>

                </Border>
            </Expander>
        </Grid>
    </Border>
</Grid>
</Page>
```

EmployeePage.xaml

Страница WPF для таблицы сотрудников. Отображает список сотрудников в DataGrid, поддерживает редактирование строк и вызывает команды Employee_VM для управления записями.

```
<Page x:Class="ORM_Individual.Views.TablePages.EmployeePage"
      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:viewModels="clr-namespace:ORM_Individual.ViewModels.TableViewModels"
      xmlns:behaviors="clr-namespace:ORM_Individual.WPFControls.AttachedProperties"
      xmlns:i="http://schemas.microsoft.com/xaml/behaviors"
      mc:Ignorable="d"
      d:DesignHeight="450" d:DesignWidth="800"
      Height="auto"
      Width="auto"
      Title="EmployeePage">

    <Page.DataContext>
        <viewModels:Employee_VM />
    </Page.DataContext>

    <Grid>
        <Grid.RowDefinitions>
            <RowDefinition Height="23"/>
            <RowDefinition Height="Auto"/>
        </Grid.RowDefinitions>
        <TextBlock Text="Employee Table"
                   Grid.ColumnSpan="2"
                   Style="{StaticResource PageHeaderTextBlockStyle}"
                   Background="MediumPurple"
                   />

        <Border Grid.Row="1"
                Style="{StaticResource TableBorderStyle}">
            <Grid>
                <Grid.ColumnDefinitions>
                    <ColumnDefinition Width="auto" />
                    <ColumnDefinition Width="auto" />
```

```
                </Grid.ColumnDefinitions>
                <DataGrid
                    Grid.Row="1"
                    Grid.Column="0"
                    ItemsSource="{Binding Source, Mode=TwoWay}"
                    Style="{StaticResource TableDataGridStyle}"
                    x:Name="DataGridPar"

                    >
                    <DataGrid.Columns>
                        <DataGridTextColumn Header="Id" Binding="{Binding Id}"/>
                        <DataGridTextColumn Header="Full name" Binding="{Binding
FullName}"/>
                        <DataGridTextColumn Header="Age" Binding="{Binding Age}"/>
                        <DataGridCheckBoxColumn Header="Gender" Binding="{Binding
Gender}"/>
                        <DataGridTextColumn Header="Address" Binding="{Binding
Address}"/>
                        <DataGridTextColumn Header="Phone" Binding="{Binding
Phone}"/>
                        <DataGridTextColumn Header="Passport" Binding="{Binding
PassportData}"/>
                        <DataGridTextColumn Header="Position" Binding="{Binding
PositionId}"/>
                    </DataGrid.Columns>
                    <i:Interaction.Triggers>
                        <i:EventTrigger EventName="RowEditEnding">
                            <i:InvokeCommandAction Command="{Binding
RowEditEndingCommand}"
                                                   PassEventArgsToCommand="True"/>
                        </i:EventTrigger>
                        <i:EventTrigger EventName="PreviewKeyDown">
                            <i:InvokeCommandAction
                                        Command="{Binding DeleteRowsCommand}"
                                        PassEventArgsToCommand="True" />
                        </i:EventTrigger>
                    </i:Interaction.Triggers>
                    <DataGrid.ContextMenu>
                        <ContextMenu>
                            <MenuItem Header="Delete Table"></MenuItem>
                        </ContextMenu>
                    </DataGrid.ContextMenu>
                </DataGrid>
                <Expander Grid.Column="1"
                          Grid.Row="1"
                          Style="{StaticResource QueryExpanderStyle}">
                    <Border
                            Style="{StaticResource QueryContainerBorderStyle}">
                        <Grid>
                            <Grid.RowDefinitions>
                                <RowDefinition Height="auto"/>
                                <RowDefinition/>
                                <RowDefinition/>
                                <RowDefinition/>
                                <RowDefinition/>
                            </Grid.RowDefinitions>
                            <TextBlock
                                Grid.Row="0"
                                Grid.ColumnSpan="2"
                                Text="Queries"
                                Style="{StaticResource QueryTitleTextBlockStyle}" />
                            <Border
                                   Grid.Row="1"
                                   Grid.Column="0"
                                   Style="{StaticResource QueryGroupBorderStyle}">
```

```
                        <Grid>
                            <Grid.ColumnDefinitions>
                                <ColumnDefinition/>
                                <ColumnDefinition/>
                            </Grid.ColumnDefinitions>
                            <Grid.RowDefinitions>
                                <RowDefinition/>
                                <RowDefinition/>
                                <RowDefinition/>
                                <RowDefinition/>
                                <RowDefinition/>
                            </Grid.RowDefinitions>
                            <TextBlock
                                Grid.Row="0"
                                Text="Id"
                                Style="{StaticResource
QueryLabelTextBlockStyle}" />
                            <Button
                                Grid.Column="1"
                                Content="+"
                                Style="{StaticResource QueryAddButtonStyle}"
                                Command="{Binding UseIdQueryCommand}" />
                            <TextBlock
                                TextAlignment="Right"
                                Grid.Row="1"
                                Text="from"
                                Style="{StaticResource
QueryLabelTextBlockStyle}" />
                            <TextBox
                                Grid.Row="1"
                                Grid.Column="1"
                                Text="{Binding IdFrom, Mode=TwoWay}" />
                            <TextBlock
                                TextAlignment="Right"
                                Grid.Row="2"
                                Text="to"
                                Style="{StaticResource
QueryLabelTextBlockStyle}" />
                            <TextBox
                                Grid.Row="2"
                                Grid.Column="1"
                                Text="{Binding IdTo, Mode=TwoWay}" />
                        </Grid>
                    </Border>
                    <Border
                           Grid.Row="2"
                           Grid.Column="0"
                           Style="{StaticResource QueryGroupBorderStyle}">
                        <Grid>
                            <Grid.ColumnDefinitions>
                                <ColumnDefinition/>
                                <ColumnDefinition/>
                            </Grid.ColumnDefinitions>
                            <Grid.RowDefinitions>
                                <RowDefinition/>
                                <RowDefinition/>
                                <RowDefinition/>
                            </Grid.RowDefinitions>
                            <Button
                                Grid.Row="2"
                                Grid.ColumnSpan="2"
                                Content="Join by salary"
                                Style="{StaticResource QueryAddButtonStyle}"
                                FontSize="12"
```

```
                                        CommandParameter="{Binding
ElementName=DataGridPar}"
                                        Command="{Binding
PositionSalaryQueryCommand}" />
                                </Grid>
                            </Border>
                        </Grid>

                    </Border>
                </Expander>
            </Grid>
        </Border>
    </Grid>
</Page>
```

PositionPage.xaml

Страница WPF для таблицы должностей. Отображает должности и результаты запросов (минимальный/максимальный оклад), а также предоставляет элементы управления для выполнения команд Position_VM.

```
<Page x:Class="ORM_Individual.Views.TablePages.PositionPage"
      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:viewModels="clr-namespace:ORM_Individual.ViewModels.TableViewModels"
      xmlns:behaviors="clr-namespace:ORM_Individual.WPFControls.AttachedProperties"
      xmlns:i="http://schemas.microsoft.com/xaml/behaviors"
      mc:Ignorable="d"
      d:DesignHeight="450" d:DesignWidth="800"
      Height="auto"
      Width="auto"
      Title="PositionPage">

    <Page.DataContext>
        <viewModels:Position_VM />
    </Page.DataContext>

    <Grid>
        <Grid.RowDefinitions>
            <RowDefinition Height="23"/>
            <RowDefinition Height="Auto"/>
        </Grid.RowDefinitions>
        <TextBlock Text="Position Table"
                   Grid.ColumnSpan="2"
                   Style="{StaticResource PageHeaderTextBlockStyle}"
                   Background="AliceBlue"
                   />

        <Border Grid.Row="1"
                Style="{StaticResource TableBorderStyle}">
            <Grid>
                <Grid.ColumnDefinitions>
                    <ColumnDefinition Width="auto" />
                    <ColumnDefinition Width="auto" />
                </Grid.ColumnDefinitions>
                <DataGrid
                    Grid.Row="1"
                    Grid.Column="0"
                    ItemsSource="{Binding Source, Mode=TwoWay}"
```

```
                    Style="{StaticResource TableDataGridStyle}">
                    <DataGrid.Columns>
                        <DataGridTextColumn Header="Id" Binding="{Binding Id}"/>
                        <DataGridTextColumn Header="Name" Binding="{Binding Name}"/>
                        <DataGridTextColumn Header="Salary" Binding="{Binding
Salary}"/>
                        <DataGridTextColumn Header="Duties" Binding="{Binding
Duties}"/>
                        <DataGridTextColumn Header="Requirements" Binding="{Binding
Requirements}"/>
                    </DataGrid.Columns>
                    <i:Interaction.Triggers>
                        <i:EventTrigger EventName="RowEditEnding">
                            <i:InvokeCommandAction Command="{Binding
RowEditEndingCommand}"
                                                   PassEventArgsToCommand="True"/>
                        </i:EventTrigger>
                        <i:EventTrigger EventName="PreviewKeyDown">
                            <i:InvokeCommandAction
                                        Command="{Binding DeleteRowsCommand}"
                                        PassEventArgsToCommand="True" />
                        </i:EventTrigger>
                    </i:Interaction.Triggers>
                    <DataGrid.ContextMenu>
                        <ContextMenu>
                            <MenuItem Header="Delete Table"></MenuItem>
                        </ContextMenu>
                    </DataGrid.ContextMenu>
                </DataGrid>
                <Expander Grid.Column="1"
                          Grid.Row="1"
                          Style="{StaticResource QueryExpanderStyle}">
                    <Border
                            Style="{StaticResource QueryContainerBorderStyle}">
                        <Grid>
                            <Grid.RowDefinitions>
                                <RowDefinition Height="auto"/>
                                <RowDefinition/>
                                <RowDefinition/>
                                <RowDefinition/>
                                <RowDefinition/>
                            </Grid.RowDefinitions>
                            <TextBlock
                                Grid.Row="0"
                                Grid.ColumnSpan="2"
                                Text="Queries"
                                Style="{StaticResource QueryTitleTextBlockStyle}" />
                            <Border
                                   Grid.Row="1"
                                   Grid.Column="0"
                                   Style="{StaticResource QueryGroupBorderStyle}">
                                <Grid>
                                    <Grid.ColumnDefinitions>
                                        <ColumnDefinition/>
                                        <ColumnDefinition/>
                                    </Grid.ColumnDefinitions>
                                    <Grid.RowDefinitions>
                                        <RowDefinition/>
                                        <RowDefinition/>
                                        <RowDefinition/>
                                    </Grid.RowDefinitions>
                                    <TextBlock
                                        Grid.Row="0"
                                        Text="Id"
```

```
                    Style="{StaticResource
QueryLabelTextBlockStyle}" />
                <Button
                    Grid.Column="1"
                    Content="+"
                    Style="{StaticResource QueryAddButtonStyle}"
                    Command="{Binding UseIdQueryCommand}" />
                <TextBlock
                    TextAlignment="Right"
                    Grid.Row="1"
                    Text="from"
                    Style="{StaticResource
QueryLabelTextBlockStyle}" />
                <TextBox
                    Grid.Row="1"
                    Grid.Column="1"
                    Text="{Binding IdFrom, Mode=TwoWay}" />
                <TextBlock
                    TextAlignment="Right"
                    Grid.Row="2"
                    Text="to"
                    Style="{StaticResource
QueryLabelTextBlockStyle}" />
                <TextBox
                    Grid.Row="2"
                    Grid.Column="1"
                    Text="{Binding IdTo, Mode=TwoWay}" />
            </Grid>
        </Border>
        <Border
               Grid.Row="2"
               Grid.Column="0"
               Style="{StaticResource QueryGroupBorderStyle}">
            <Grid>
                <Grid.ColumnDefinitions>
                    <ColumnDefinition/>
                    <ColumnDefinition/>
                </Grid.ColumnDefinitions>
                <Grid.RowDefinitions>
                    <RowDefinition/>
                    <RowDefinition/>
                    <RowDefinition/>
                </Grid.RowDefinitions>
                <TextBlock
                    Grid.Row="0"
                    Text="Salary"
                    Style="{StaticResource
QueryLabelTextBlockStyle}" />
                <Button
                    Grid.Row="1"
                    Style="{StaticResource QueryAddButtonStyle}"
                    Command="{Binding MaxSalaryQueryCommand}"
                    FontSize="13"
                    Background="White"
                    Grid.ColumnSpan="2"
                    Content="Max"
                    />
                <Button
                    Grid.Row="2"
                    Style="{StaticResource QueryAddButtonStyle}"
                    Command="{Binding MinSalaryQueryCommand}"
                    FontSize="13"
                    Background="White"
                    Grid.ColumnSpan="2"
                    Content="Min"
```

```
                                    />
                            </Grid>
                        </Border>
                        <Border
                            Grid.Row="3"
                            Grid.Column="0"
                            Style="{StaticResource QueryGroupBorderStyle}">
                            <Grid>
                                <Grid.ColumnDefinitions>
                                    <ColumnDefinition/>
                                    <ColumnDefinition/>
                                </Grid.ColumnDefinitions>
                                <Grid.RowDefinitions>
                                    <RowDefinition/>
                                    <RowDefinition/>
                                    <RowDefinition/>
                                </Grid.RowDefinitions>
                                <TextBlock
                                    Grid.Row="0"
                                    Text="Serialised"
                                    Style="{StaticResource
QueryLabelTextBlockStyle}" />

                                <TextBlock
                                    Grid.Row="1"
                                    Grid.ColumnSpan="2"
                                    Background="White"
                                    TextWrapping="Wrap"
                                    Text="{Binding Serialised}"
                                    Width="auto"
                                    />
                            </Grid>
                        </Border>
                    </Grid>
                </Border>
            </Expander>
        </Grid>
    </Border>
</Grid>
</Page>
```

Папка WPFControls

EnableDragHelper.cs

Вспомогательный класс для поведения интерфейса WPF. Реализует attached-property/обработчики для поддержки перетаскивания или дополнительного взаимодействия с элементами окна.

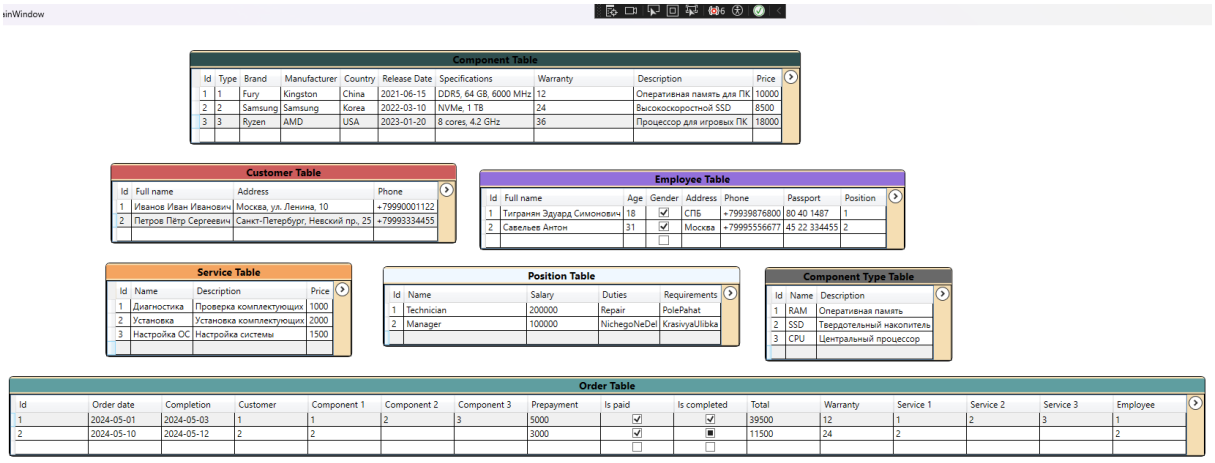
```
using System.Windows;
using System.Windows.Controls;
using System.Windows.Input;
using System.Windows.Media;

namespace ORM_Individual.WPFControls.AttachedProperties
{
    public static class DragBehavior
    {
        public static readonly DependencyProperty IsDraggableProperty =
            DependencyProperty.RegisterAttached(
```

```
                "IsDraggable",
                typeof(bool),
                typeof(DragBehavior),
                new PropertyMetadata(false, OnIsDraggableChanged));

        public static void SetIsDraggable(DependencyObject element, bool value) =>
            element.SetValue(IsDraggableProperty, value);

        public static bool GetIsDraggable(DependencyObject element) =>
            (bool)element.GetValue(IsDraggableProperty);

        private static void OnIsDraggableChanged(DependencyObject d,
DependencyPropertyChangedEventArgs e)
        {
            if (d is UIElement element)
            {
                element.MouseLeftButtonDown -= StartDrag;
                element.MouseLeftButtonUp -= StopDrag;
                element.MouseMove -= DoDrag;

                if ((bool)e.NewValue)
                {
                    element.MouseLeftButtonDown += StartDrag;
                    element.MouseLeftButtonUp += StopDrag;
                    element.MouseMove += DoDrag;
                }
            }
        }

        private static Point _offset;
        private static bool _isDragging = false;

        private static void StartDrag(object sender, MouseButtonEventArgs e)
        {
            if (sender is UIElement element)
            {
                _isDragging = true;
                _offset = e.GetPosition(element);
                element.CaptureMouse();
            }
        }

        private static void DoDrag(object sender, MouseEventArgs e)
        {
            if (_isDragging && sender is UIElement element)
            {
                var parent = VisualTreeHelper.GetParent(element) as Canvas;
                if (parent != null)
                {
                    Point currentPosition = e.GetPosition(parent);
                    Canvas.SetLeft(element, currentPosition.X - _offset.X);
                    Canvas.SetTop(element, currentPosition.Y - _offset.Y);
                }
            }
        }

        private static void StopDrag(object sender, MouseButtonEventArgs e)
        {
            _isDragging = false;
            if (sender is UIElement element)
                element.ReleaseMouseCapture();
        }
    }
}
```

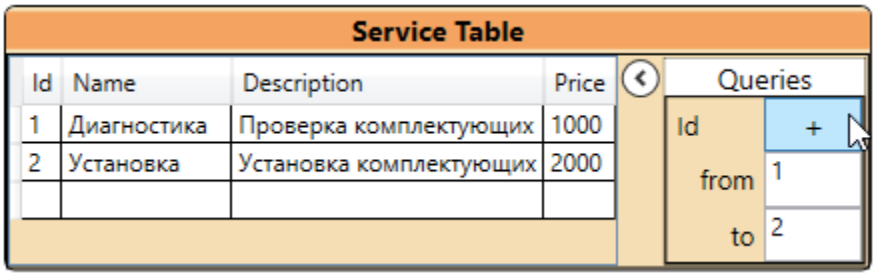
Описание кода

В индивидуальном задании таблицы отображаются через DataGrid, а логика вынесена в BaseTable_VM. При изменении строки/добавлении записи срабатывают команды ViewModel и выполняются CRUD-операции через репозиторий (IRepository<T>). Отдельно реализованы запросы: выборка по диапазону ID, а также примеры статистических функций Max и Min, и запрос соединения таблиц (JOIN) — результаты отображаются в интерфейсе. Также предусмотрен вывод данных таблицы в JSON через Serialised (сериализация Source библиотекой Newtonsoft.Json).

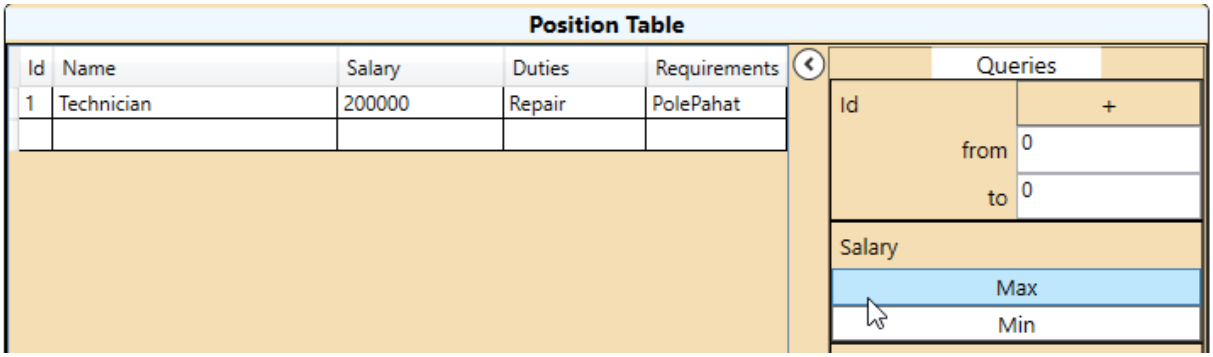
Результат



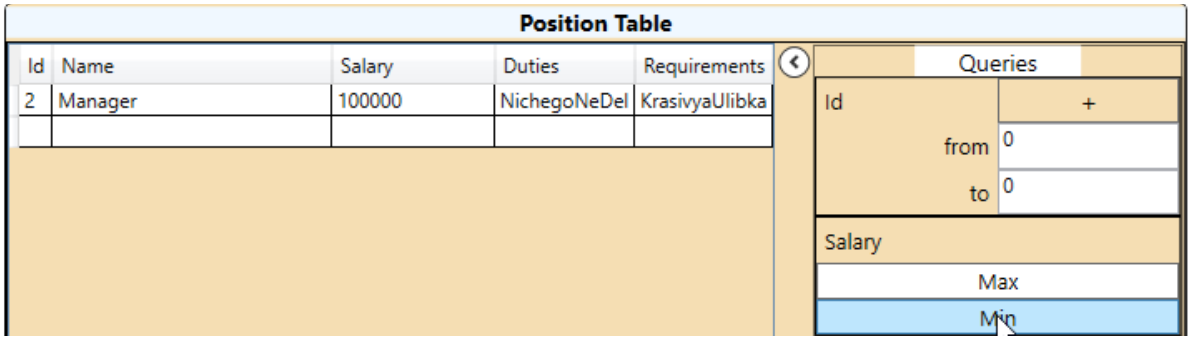
Пример заполненных данных с таблицами



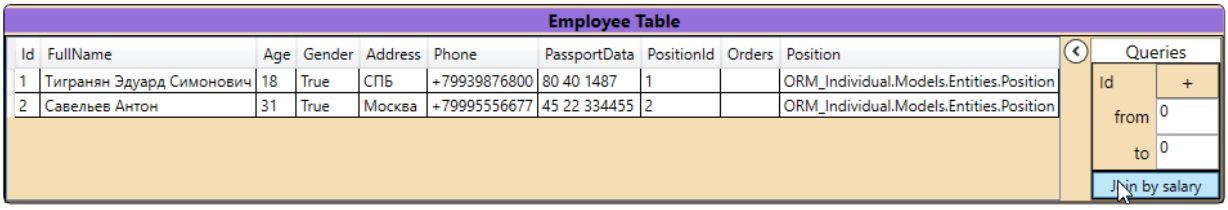
Применение запроса на выборку по ID



Применение статической функции Max

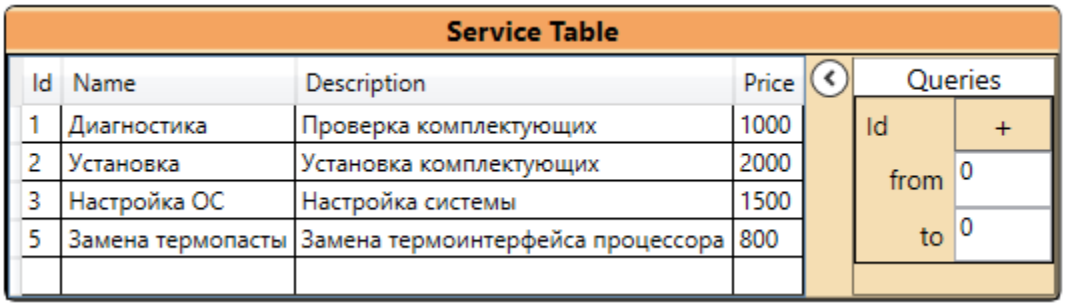


Применение статической функции Min



Выполнение запроса на соединение таблиц

Добавление 2 записей в таблицу Service



Удаление записи из таблицы

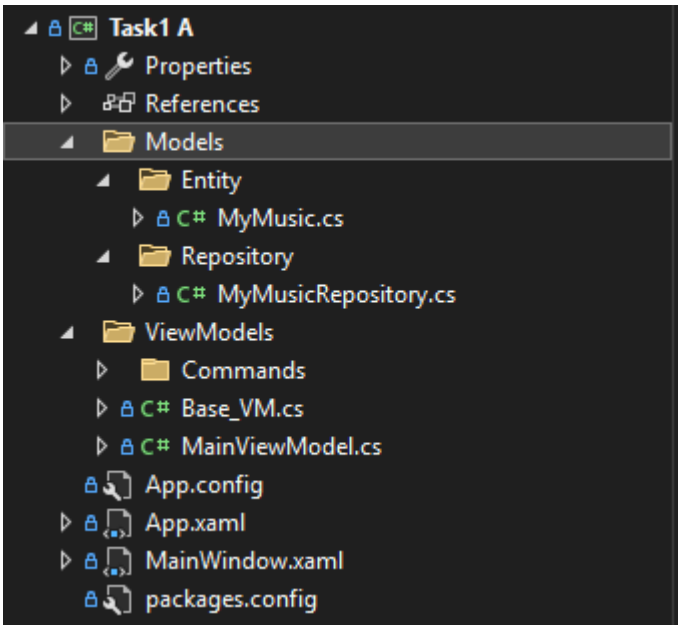
Лабораторная работа 3

Тема: работа с данными в формате json. Сериализация и десериализация объектов.

Цель работы: получение практических навыков при работе с форматом json.

Задание 1.1

Представьте данные таблицы Auths в формате json(используйте библиотеку Newtonsoft.Json)



Структура проекта

Папка Models

MyMusic.cs

Описывает структуру объектов (свойства), которые сериализуются в JSON и затем восстанавливаются при десериализации для вывода в интерфейс.

```
namespace Task1.Models.Entity
{
    public class MyMusic
    {
        public Track[] Tracks { get; set; }
    }

    public class Track
    {
        public string Artist;
        public string Album;
        public string Title;
        public string Year;
    }
}
```

MyMusicRepository.cs

Репозиторий JSON-данных. Формирует коллекции объектов MyMusic, выполняет сериализацию/десериализацию (Newtonsoft.Json), читает/сохраняет данные и возвращает результат в ViewModel.

```
using Newtonsoft.Json;
using System;
using Task1.Models.Entity;

namespace Task1.Models.Repository
{
    public class MyMusicRepository
    {
        private bool _disposed = false;
        private MyMusic _music;

        public MyMusicRepository(MyMusic music = null)
        {
            _music = music ?? new MyMusic();
        }

        public string Serialize()
        {
            return JsonConvert.SerializeObject(_music);
        }
    }
}
```

Папка ViewModels

RelayCommand.cs

```
using System;
using System.Windows.Input;

namespace Task1.ViewModels.Commands
{
    public class RelayCommand : ICommand
    {
        public event EventHandler CanExecuteChanged;
        private readonly Action _execute;
        public RelayCommand(Action execute, Func<bool> canExecute = null)
        {
            _execute = execute;
        }
        public bool CanExecute(object parameter)
        {
            return true;
        }

        public void Execute(object parameter)
        {
            _execute();
        }
    }
}
```

Реализация ICommand

Base_VM.cs

Реализует INotifyPropertyChanged, обеспечивает обновление привязанных данных при изменении свойств и поддерживает MVVM-структуру приложения.

```
using System.ComponentModel;
using System.Runtime.CompilerServices;

namespace Task1.ViewModels
{
    public class Base_VM : INotifyPropertyChanged
    {
        public event PropertyChangedEventHandler PropertyChanged;
        protected void OnPropertyChanged([CallerMemberName] string propertyName =
null)
        {
            PropertyChanged?.Invoke(this, new
PropertyChangedEventArgs(propertyName));
        }
    }
}
```

MainViewModel.cs

Содержит команды для работы с JSON, инициирует вызовы MyMusicRepository, принимает результаты десериализации и передаёт данные в коллекции для отображения.

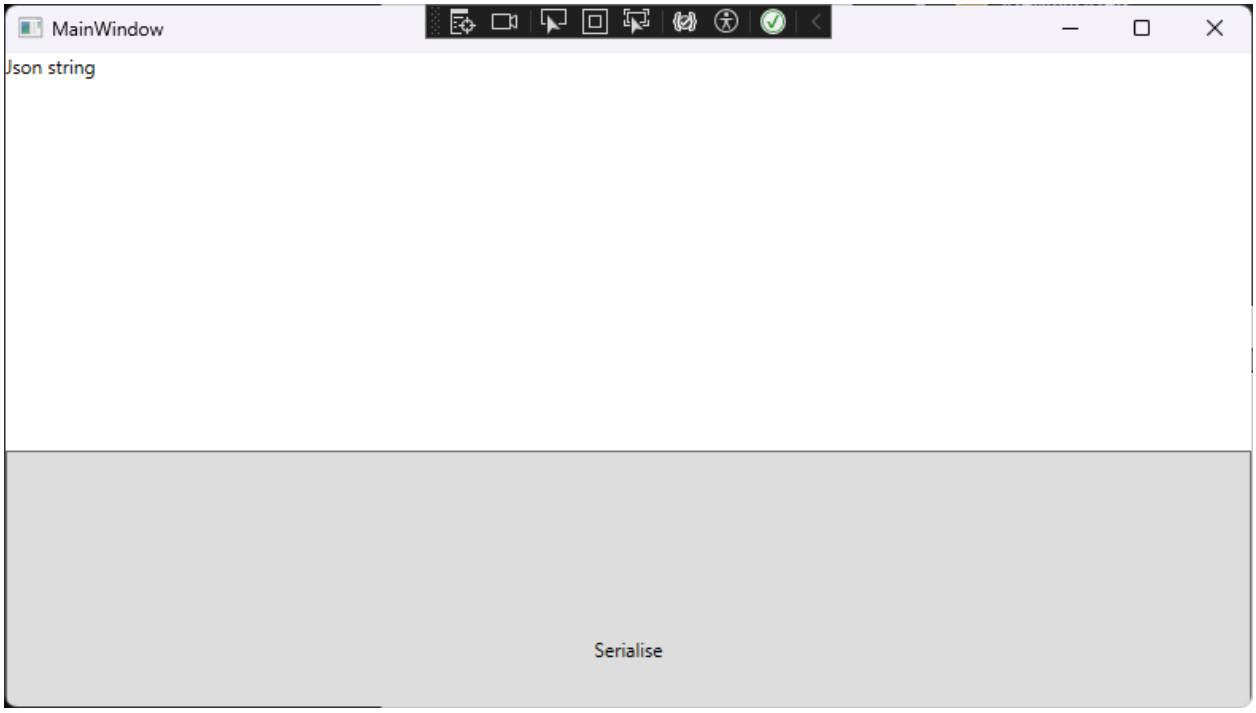
```
using System.Windows.Input;
using Task1.Models.Entity;
using Task1.Models.Repository;
using Task1.ViewModels.Commands;

namespace Task1.ViewModels
{
    public class MainViewModel : Base_VM
    {
        MyMusic myCollection;
        private string _serializedString;
        public string SerializedString { get => _serializedString; set {
_serializedString = value; OnPropertyChanged(); } }

        public ICommand SerializeCommand {  get; }
        public MainViewModel()
        {
            myCollection = new MyMusic();
            myCollection.Tracks = new Track[3];
            myCollection.Tracks[0] = new Track()
            {
                Artist = "Artist1",
                Album = "Album1",
                Title = "Title1",
                Year = "2015"
            };
```

```
            myCollection.Tracks[1] = new Track()
            {
                Artist = "Artist2",
                Album = "Album2",
                Title = "Title2",
                Year = "2015"
            };
            myCollection.Tracks[2] = new Track()
            {
                Artist = "Artist3",
                Album = "Album3",
                Title = "Title3",
                Year = "2015"
            };

            SerializeCommand = new RelayCommand(Serialize);
        }

        public void Serialize()
        {
            var _repo = new MyMusicRepository(myCollection);
            SerializedString =  _repo.Serialize();
        }
    }
}
```

Описание кода

Пользователь нажимает кнопку в интерфейсе, команда через RelayCommand вызывает метод во MainViewModel. MainViewModel получает данные (через MyMusicRepository) и сериализует их в JSON с помощью Newtonsoft.Json. Полученная JSON-строка записывается в свойство ViewModel и автоматически отображается в окне благодаря INotifyPropertyChanged.

Результат

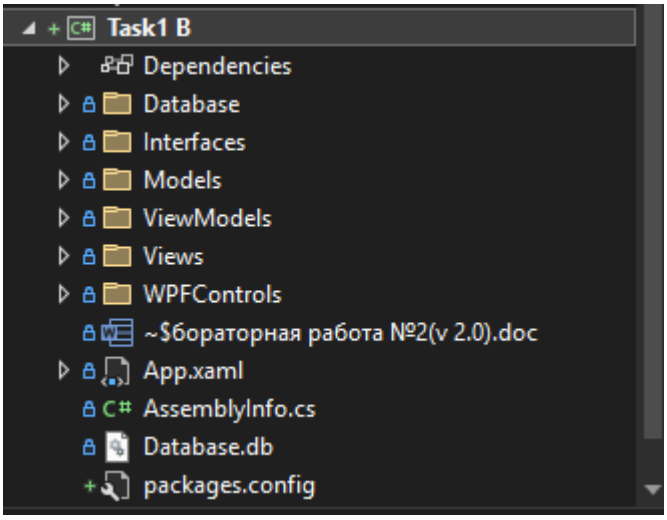


Главный экран приложения

Результат нажатия на кнопку сериализации

Задание 1.2

Представьте данные одной из таблиц индивидуального задания в формате json(используйте библиотеку Newtonsoft.Json)



Структура проекта

BaseTable_VM.cs

Хранит ObservableCollection, обеспечивает обновление таблицы при загрузке из JSON и предоставляет команды управления данными/отображением в интерфейсе.

```
using ORM_Individual.Interfaces;
using ORM_Individual.Models.Database;
using ORM_Individual.ViewModels.Commands;
using System.Collections.ObjectModel;
using System.Data;
using System.Windows.Controls;
using System.Windows.Input;
using Newtonsoft.Json;

namespace ORM_Individual.ViewModels.TableViewModels
{
    public abstract class BaseTable_VM<T> : Base_VM, IDisposable where T : IEntity
    {
        private static bool _databaseInitialized;
        public ObservableCollection<T> _source = new();
        protected IRepository<T> Repository { get; }
        public ICommand RowEditEndingCommand { get; }
        public ICommand SaveRowCommand { get; }
        public ICommand DeleteRowsCommand { get; }
        public virtual ObservableCollection<T> Source
        {
            get => _source;
            set
            {
                _source = value;
                OnPropertyChanged();
            }
        }

        private string _serialised;
        public string Serialised
        {
            get { return _serialised; }
            set { _serialised = value;
                OnPropertyChanged();
            }
```

```
        }

        private string temp {  get; set; }

        protected BaseTable_VM(IRepository<T> repository)
        {
            Repository = repository ?? throw new
ArgumentNullException(nameof(repository));
            EnsureDatabase();
            LoadSource();
            IsIdQuery = false;
            RowEditEndingCommand = new RelayCommand(RowEditEnding);
            DeleteRowsCommand = new RelayCommand(DeleteRows);
            UseIdQueryCommand = new RelayCommand(UserIdQuery);
            UpdateSerString();
        }
        public void RowEditEnding(object parameter)
        {
            if (parameter is DataGridRowEditEndingEventArgs e)
            {
                e.Row.BindingGroup?.CommitEdit();
                if ((e.EditAction == DataGridEditAction.Commit) &&
(e.Row.DataContext is IEntity entity))
                {
                    try
                    {
                        foreach (IEntity row in Repository.GetAll())
                        {
                            if (row.Id == entity.Id)
                            {
                                Repository.Update((T)entity);
                                return;
                            }
                        }
                        Repository.Add((T)entity);
                        UpdateSerString();
                    }
                    catch (Exception ex) {
                        LoadSource();
                    }
                }
            }
        }
        private void UpdateSerString()
        {
            Serialised = string.Empty;
            temp = JsonConvert.SerializeObject(Source);
            foreach (var str in temp.Split(',')) {
                if (!(str[0] == '{' || str[1] == '{'))
                {
                    Serialised += ('\t' + str + "\n");
                }
                else
                {
                    Serialised += (str + "\n");
                }
            }
        }
        public void DeleteRows(object parameter)
        {
            if (parameter is KeyEventArgs e)
            {
                if (e.Key != Key.Delete)
                    return;
                if (e.OriginalSource is TextBox)
```

```
            return;
        var grid = (DataGrid)e.Source;

        try
        {
            var toDelete = grid.SelectedItems.Cast<T>().ToList();
            foreach (T item in toDelete)
            {
                Source.Remove(item);
                if (item is IEntity entity)
                {
                    Repository.Remove(entity.Id);
                    UpdateSerString();
                }
            }
        }
        catch (Exception ex) {
            LoadSource();
        }
    }
}
protected void LoadSource()
{
    Source = Repository.GetAll();
}
private static void EnsureDatabase()
{
    var context = DatabaseContext.GetContext();
    context.Database.EnsureCreated();
    _databaseInitialized = true;
}

#region Queries
public ICommand UseIdQueryCommand { get; }
public bool IsIdQuery;
private void UserIdQuery(object parameter)
{
    if(IsIdQuery == false)
    {
        IsIdQuery = true;
        IdQuerySelect();
    }
    else
    {
        IsIdQuery = false;
        LoadSource();
    }
}
private void IdQuerySelect()
{
    if (IsIdQuery == true) {
        Source = Repository.IdQueries(Source, IdFrom, IdTo);
    }
}

public void Dispose()
{
    throw new NotImplementedException();
}

private int _idFrom;
public int IdFrom
{
    get { return _idFrom; }
    set { _idFrom = value;
```

```
                OnPropertyChanged();
                IdQuerySelect();
            }
        }
        private int _idTo;
        public int IdTo
        {
            get { return _idTo; }
            set
            {
                _idTo = value;
                OnPropertyChanged();
                IdQuerySelect();
            }
        }
        #endregion
    }
}
```

PositionPage.xaml

Использует DataGrid и привязку к ViewModel для вывода десериализованных объектов и результатов выполнения команд пользователя.

```
<Page x:Class="ORM_Individual.Views.TablePages.PositionPage"
      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:viewModels="clr-namespace:ORM_Individual.ViewModels.TableViewModels"
      xmlns:behaviors="clr-namespace:ORM_Individual.WPFControls.AttachedProperties"
      xmlns:i="http://schemas.microsoft.com/xaml/behaviors"
      mc:Ignorable="d"
      d:DesignHeight="450" d:DesignWidth="800"
      Height="auto"
      Width="auto"
      Title="PositionPage">

    <Page.DataContext>
        <viewModels:Position_VM />
    </Page.DataContext>

    <Grid>
        <Grid.RowDefinitions>
            <RowDefinition Height="23"/>
            <RowDefinition Height="Auto"/>
        </Grid.RowDefinitions>
        <TextBlock Text="Position Table"
                   Grid.ColumnSpan="2"
                   Style="{StaticResource PageHeaderTextBlockStyle}"
                   Background="AliceBlue"
                   />

        <Border Grid.Row="1"
                Style="{StaticResource TableBorderStyle}">
            <Grid>
                <Grid.ColumnDefinitions>
                    <ColumnDefinition Width="auto" />
                    <ColumnDefinition Width="auto" />
                </Grid.ColumnDefinitions>
                <DataGrid
                    Grid.Row="1"
                    Grid.Column="0"
```

```
                    ItemsSource="{Binding Source, Mode=TwoWay}"
                    Style="{StaticResource TableDataGridStyle}">
                    <DataGrid.Columns>
                        <DataGridTextColumn Header="Id" Binding="{Binding Id}"/>
                        <DataGridTextColumn Header="Name" Binding="{Binding Name}"/>
                        <DataGridTextColumn Header="Salary" Binding="{Binding
Salary}"/>
                        <DataGridTextColumn Header="Duties" Binding="{Binding
Duties}"/>
                        <DataGridTextColumn Header="Requirements" Binding="{Binding
Requirements}"/>
                    </DataGrid.Columns>
                    <i:Interaction.Triggers>
                        <i:EventTrigger EventName="RowEditEnding">
                            <i:InvokeCommandAction Command="{Binding
RowEditEndingCommand}"
                                                   PassEventArgsToCommand="True"/>
                        </i:EventTrigger>
                        <i:EventTrigger EventName="PreviewKeyDown">
                            <i:InvokeCommandAction
                                        Command="{Binding DeleteRowsCommand}"
                                        PassEventArgsToCommand="True" />
                        </i:EventTrigger>
                    </i:Interaction.Triggers>
                    <DataGrid.ContextMenu>
                        <ContextMenu>
                            <MenuItem Header="Delete Table"></MenuItem>
                        </ContextMenu>
                    </DataGrid.ContextMenu>
                </DataGrid>
                <Expander Grid.Column="1"
                          Grid.Row="1"
                          Style="{StaticResource QueryExpanderStyle}">
                    <Border
                            Style="{StaticResource QueryContainerBorderStyle}">
                        <Grid>
                            <Grid.RowDefinitions>
                                <RowDefinition Height="auto"/>
                                <RowDefinition/>
                                <RowDefinition/>
                                <RowDefinition/>
                                <RowDefinition/>
                            </Grid.RowDefinitions>
                            <TextBlock
                                Grid.Row="0"
                                Grid.ColumnSpan="2"
                                Text="Queries"
                                Style="{StaticResource QueryTitleTextBlockStyle}" />
                            <Border
                                   Grid.Row="1"
                                   Grid.Column="0"
                                   Style="{StaticResource QueryGroupBorderStyle}">
                                <Grid>
                                    <Grid.ColumnDefinitions>
                                        <ColumnDefinition/>
                                        <ColumnDefinition/>
                                    </Grid.ColumnDefinitions>
                                    <Grid.RowDefinitions>
                                        <RowDefinition/>
                                        <RowDefinition/>
                                        <RowDefinition/>
                                    </Grid.RowDefinitions>
                                    <TextBlock
                                        Grid.Row="0"
                                        Text="Id"
```

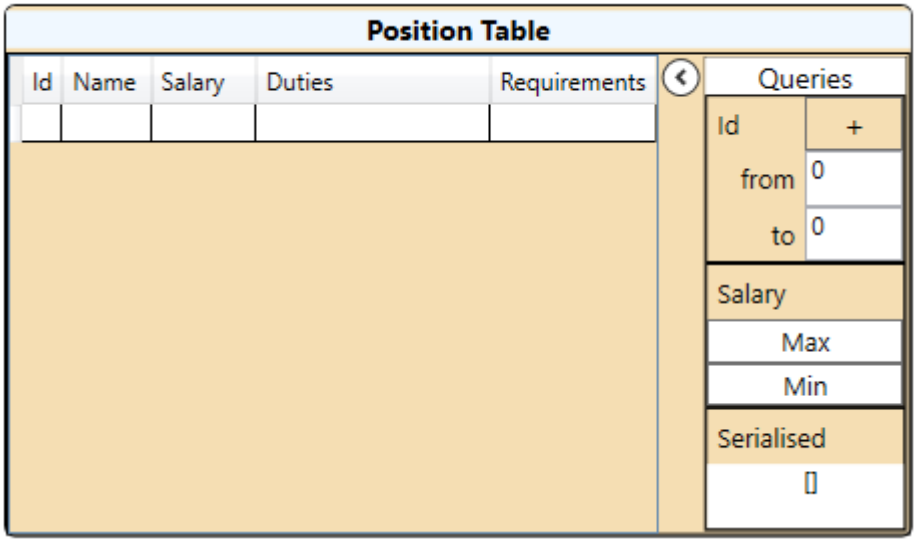
```
                    Style="{StaticResource
QueryLabelTextBlockStyle}" />
                <Button
                    Grid.Column="1"
                    Content="+"
                    Style="{StaticResource QueryAddButtonStyle}"
                    Command="{Binding UseIdQueryCommand}" />
                <TextBlock
                    TextAlignment="Right"
                    Grid.Row="1"
                    Text="from"
                    Style="{StaticResource
QueryLabelTextBlockStyle}" />
                <TextBox
                    Grid.Row="1"
                    Grid.Column="1"
                    Text="{Binding IdFrom, Mode=TwoWay}" />
                <TextBlock
                    TextAlignment="Right"
                    Grid.Row="2"
                    Text="to"
                    Style="{StaticResource
QueryLabelTextBlockStyle}" />
                <TextBox
                    Grid.Row="2"
                    Grid.Column="1"
                    Text="{Binding IdTo, Mode=TwoWay}" />
            </Grid>
        </Border>
        <Border
               Grid.Row="2"
               Grid.Column="0"
               Style="{StaticResource QueryGroupBorderStyle}">
            <Grid>
                <Grid.ColumnDefinitions>
                    <ColumnDefinition/>
                    <ColumnDefinition/>
                </Grid.ColumnDefinitions>
                <Grid.RowDefinitions>
                    <RowDefinition/>
                    <RowDefinition/>
                    <RowDefinition/>
                </Grid.RowDefinitions>
                <TextBlock
                    Grid.Row="0"
                    Text="Salary"
                    Style="{StaticResource
QueryLabelTextBlockStyle}" />
                <Button
                    Grid.Row="1"
                    Style="{StaticResource QueryAddButtonStyle}"
                    Command="{Binding MaxSalaryQueryCommand}"
                    FontSize="13"
                    Background="White"
                    Grid.ColumnSpan="2"
                    Content="Max"
                    />
                <Button
                    Grid.Row="2"
                    Style="{StaticResource QueryAddButtonStyle}"
                    Command="{Binding MinSalaryQueryCommand}"
                    FontSize="13"
                    Background="White"
                    Grid.ColumnSpan="2"
                    Content="Min"
```

```
                                />
                            </Grid>
                        </Border>
                        <Border
                            Grid.Row="3"
                            Grid.Column="0"
                            Style="{StaticResource QueryGroupBorderStyle}">
                            <Grid>
                                <Grid.ColumnDefinitions>
                                    <ColumnDefinition/>
                                    <ColumnDefinition/>
                                </Grid.ColumnDefinitions>
                                <Grid.RowDefinitions>
                                    <RowDefinition/>
                                    <RowDefinition/>
                                    <RowDefinition/>
                                </Grid.RowDefinitions>
                                <TextBlock
                                    Grid.Row="0"
                                    Text="Serialised"
                                    Style="{StaticResource
QueryLabelTextBlockStyle}" />
                                <TextBlock
                                    Grid.Row="1"
                                    Grid.ColumnSpan="2"
                                    Background="White"
                                    TextWrapping="Wrap"
                                    Text="{Binding Serialised}"
                                    Width="auto"
                                    />
                            </Grid>
                        </Border>
                    </Grid>
                </Border>
            </Expander>
        </Grid>
    </Border>
</Grid>
</Page>
```

Описание кода

Данные таблицы загружаются в Source внутри BaseTable_VM. При выполнении сериализации коллекция преобразуется в JSON, и результат сохраняется в Serialised. PositionPage.xaml выводит этот JSON в интерфейс, поэтому пользователь видит текстовое представление таблицы в формате json.

Результат



Пустая таблица Position

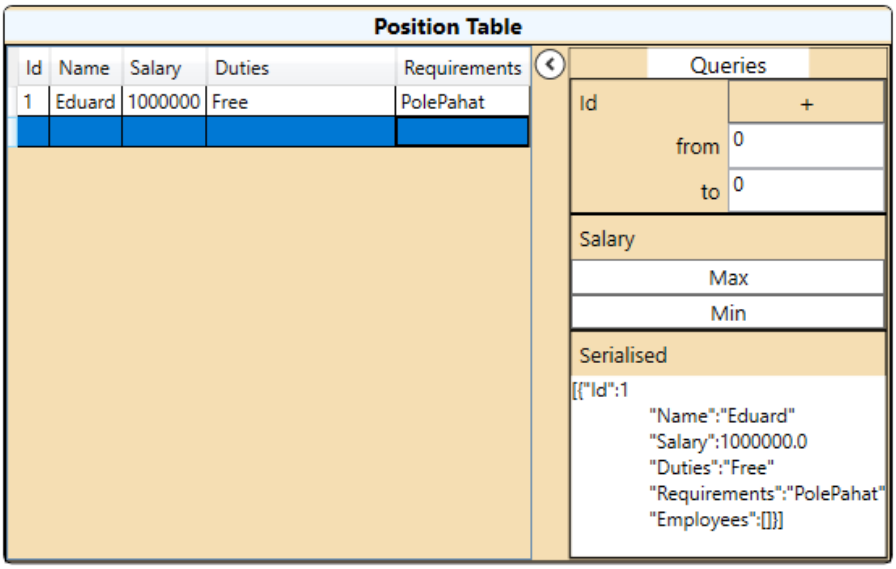


Таблица Position с записью

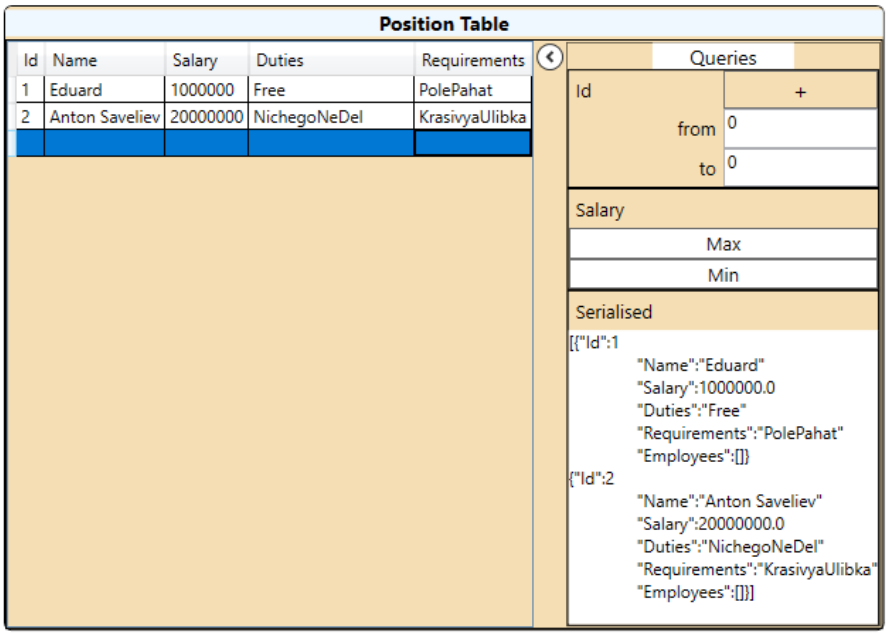
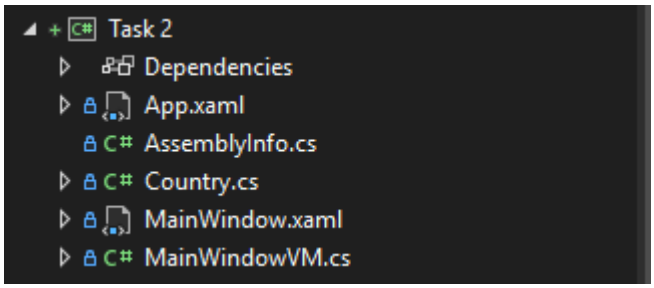


Таблица Position с новой записью

Задание 2

Используя различные ресурсы в сети, десериализовать данные, полученные в формате json. Самостоятельно найти бесплатный ресурс, который может предоставить различные данные в формате json(сведения о погоде, странах, фильмах, валютах и т.д.).



Структура проекта

Country.cs

Модель страны, используемая для десериализации JSON в объект с вложенными полями (название, столица, валюта, языки и т.д.).

```
namespace Task2
{
    public class Rootobject
    {
        public Country[] Property1 { get; set; }
    }

    public class Country
    {
```

```
    public Name name { get; set; }
    public string[] tld { get; set; }
    public string cca2 { get; set; }
    public string ccn3 { get; set; }
    public string cioc { get; set; }
    public bool independent { get; set; }
    public string status { get; set; }
    public bool unMember { get; set; }
    public Currencies currencies { get; set; }
    public Idd idd { get; set; }
    public string[] capital { get; set; }
    public string[] altSpellings { get; set; }
    public string region { get; set; }
    public string subregion { get; set; }
    public Languages languages { get; set; }
    public float[] latlng { get; set; }
    public bool landlocked { get; set; }
    public string[] borders { get; set; }
    public float area { get; set; }
    public Demonyms demonyms { get; set; }
    public string cca3 { get; set; }
    public Translations translations { get; set; }
    public string flag { get; set; }
    public Maps maps { get; set; }
    public int population { get; set; }
    public Gini gini { get; set; }
    public string fifa { get; set; }
    public Car car { get; set; }
    public string[] timezones { get; set; }
    public string[] continents { get; set; }
    public Flags flags { get; set; }
    public Coatofarms coatOfArms { get; set; }
    public string startOfWeek { get; set; }
    public Capitalinfo capitalInfo { get; set; }
    public Postalcode postalCode { get; set; }
}

public class Name
{
    public string common { get; set; }
    public string official { get; set; }
    public Nativename nativeName { get; set; }
}

public class Nativename
{
    public Rus rus { get; set; }
}

public class Rus
{
    public string official { get; set; }
    public string common { get; set; }
}

public class Currencies
{
    public RUB RUB { get; set; }
}

public class RUB
{
    public string symbol { get; set; }
    public string name { get; set; }
}
```

```
public class Idd
{
    public string root { get; set; }
    public string[] suffixes { get; set; }
}

public class Languages
{
    public string rus { get; set; }
}

public class Demonyms
{
    public Eng eng { get; set; }
    public Fra fra { get; set; }
}

public class Eng
{
    public string f { get; set; }
    public string m { get; set; }
}

public class Fra
{
    public string f { get; set; }
    public string m { get; set; }
}

public class Translations
{
    public Ara ara { get; set; }
    public Bre bre { get; set; }
    public Ces ces { get; set; }
    public Cym cym { get; set; }
    public Deu deu { get; set; }
    public Est est { get; set; }
    public Fin fin { get; set; }
    public Fra1 fra { get; set; }
    public Hrv hrv { get; set; }
    public Hun hun { get; set; }
    public Ind ind { get; set; }
    public Ita ita { get; set; }
    public Jpn jpn { get; set; }
    public Kor kor { get; set; }
    public Nld nld { get; set; }
    public Per per { get; set; }
    public Pol pol { get; set; }
    public Por por { get; set; }
    public Rus1 rus { get; set; }
    public Slk slk { get; set; }
    public Spa spa { get; set; }
    public Srp srp { get; set; }
    public Swe swe { get; set; }
    public Tur tur { get; set; }
    public Urd urd { get; set; }
    public Zho zho { get; set; }
}

public class Ara
{
    public string official { get; set; }
    public string common { get; set; }
}
```

```
public class Bre
{
    public string official { get; set; }
    public string common { get; set; }
}

public class Ces
{
    public string official { get; set; }
    public string common { get; set; }
}

public class Cym
{
    public string official { get; set; }
    public string common { get; set; }
}

public class Deu
{
    public string official { get; set; }
    public string common { get; set; }
}

public class Est
{
    public string official { get; set; }
    public string common { get; set; }
}

public class Fin
{
    public string official { get; set; }
    public string common { get; set; }
}

public class Fra1
{
    public string official { get; set; }
    public string common { get; set; }
}

public class Hrv
{
    public string official { get; set; }
    public string common { get; set; }
}

public class Hun
{
    public string official { get; set; }
    public string common { get; set; }
}

public class Ind
{
    public string official { get; set; }
    public string common { get; set; }
}

public class Ita
{
    public string official { get; set; }
    public string common { get; set; }
```

```
}

public class Jpn
{
    public string official { get; set; }
    public string common { get; set; }
}

public class Kor
{
    public string official { get; set; }
    public string common { get; set; }
}

public class Nld
{
    public string official { get; set; }
    public string common { get; set; }
}

public class Per
{
    public string official { get; set; }
    public string common { get; set; }
}

public class Pol
{
    public string official { get; set; }
    public string common { get; set; }
}

public class Por
{
    public string official { get; set; }
    public string common { get; set; }
}

public class Rus1
{
    public string official { get; set; }
    public string common { get; set; }
}

public class Slk
{
    public string official { get; set; }
    public string common { get; set; }
}

public class Spa
{
    public string official { get; set; }
    public string common { get; set; }
}

public class Srp
{
    public string official { get; set; }
    public string common { get; set; }
}

public class Swe
{
    public string official { get; set; }
```

```
        public string common { get; set; }
    }

    public class Tur
    {
        public string official { get; set; }
        public string common { get; set; }
    }

    public class Urd
    {
        public string official { get; set; }
        public string common { get; set; }
    }

    public class Zho
    {
        public string official { get; set; }
        public string common { get; set; }
    }

    public class Maps
    {
        public string googleMaps { get; set; }
        public string openStreetMaps { get; set; }
    }

    public class Gini
    {
        public float _2018 { get; set; }
    }

    public class Car
    {
        public string[] signs { get; set; }
        public string side { get; set; }
    }

    public class Flags
    {
        public string png { get; set; }
        public string svg { get; set; }
        public string alt { get; set; }
    }

    public class Coatofarms
    {
        public string png { get; set; }
        public string svg { get; set; }
    }

    public class Capitalinfo
    {
        public float[] latlng { get; set; }
    }

    public class Postalcode
    {
        public string format { get; set; }
        public string regex { get; set; }
    }

}
```

MainWindow.xaml

Интерфейс окна для отображения информации о стране. Содержит элементы вывода текста, который формирует ViewModel.

```
<Window x:Class="Task2.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:Task2"
        mc:Ignorable="d"
        Title="MainWindow"
        Height="300"
        Width="500"
        >
    <Grid>
        <TextBlock
            Margin="40"
            FontSize="14"
            Text="{Binding Data}"
            ></TextBlock>
    </Grid>
</Window>
```

MainWindowVM.cs

ViewModel, которая получает данные о стране в JSON, преобразует их в объект Country, а затем собирает итоговую строку Data (страна, столица, регион, население, валюта и т.п.). При ошибке записывает в Data текст ошибки.

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Net.Http;
using System.Runtime.CompilerServices;
using System.Threading.Tasks;
using Newtonsoft.Json;

namespace Task2
{
    public class MainWindowVM : INotifyPropertyChanged
    {
        public MainWindowVM()
        {
            Initialize();
        }

        public event PropertyChangedEventHandler? PropertyChanged;

        private void OnPropertyChanged([CallerMemberName] string propertyName =
null)
        {
            PropertyChanged?.Invoke(this, new
PropertyChangedEventArgs(propertyName));
        }
```

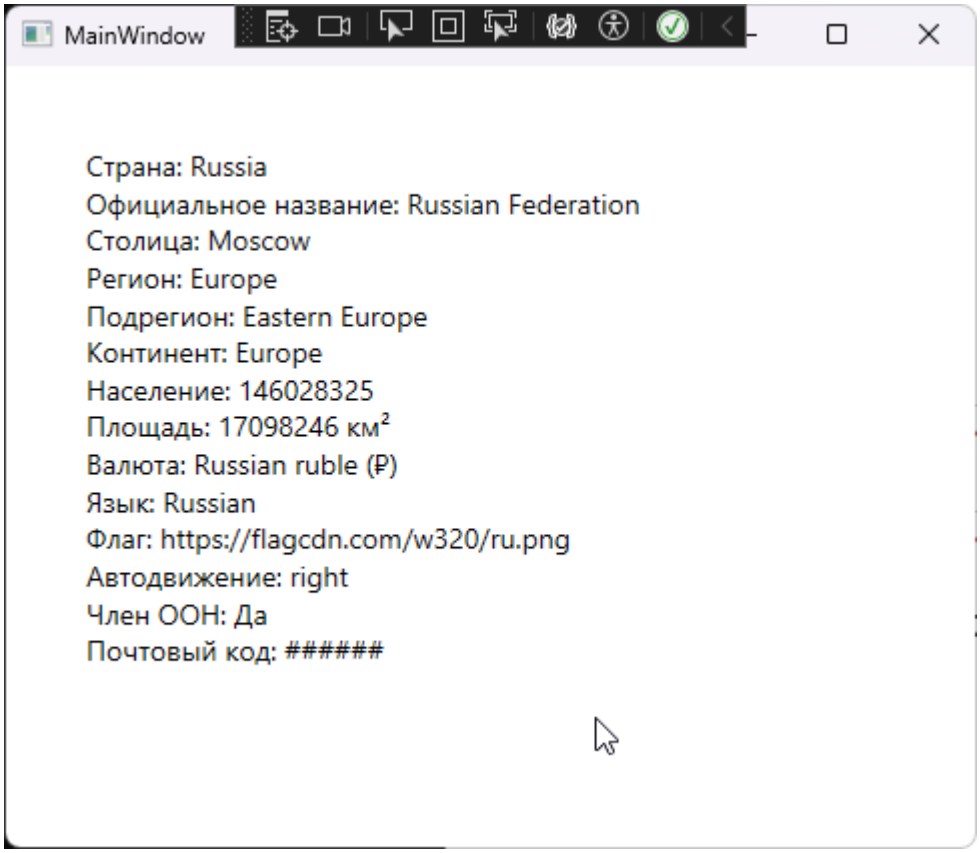
```
        private string _data;
        public string Data
        {
            get => _data;
            set
            {
                _data = value;
                OnPropertyChanged();
            }
        }

        private async Task<List<Country>> DeserializationAsync()
        {
            try
            {
                string url = "https://restcountries.com/v3.1/name/russia";

                using HttpClient client = new HttpClient();
                HttpResponseMessage response = await client.GetAsync(url);

                if (!response.IsSuccessStatusCode)
                    throw new HttpRequestException($"HTTP Error:
{response.StatusCode}");

                string json = await response.Content.ReadAsStringAsync();

                try
                {
                    return JsonConvert.DeserializeObject<List<Country>>(json);
                }
                catch (JsonException jex)
                {
                    Data = "Ошибка десериализации: " + jex.Message;
                    return null;
                }
            }
            catch (HttpRequestException httpEx)
            {
                Data = "Ошибка HTTP: " + httpEx.Message;
                return null;
            }
            catch (TaskCanceledException)
            {
                Data = "Запрос тайм-аут";
                return null;
            }
            catch (Exception ex)
            {
                Data = "Неизвестная ошибка: " + ex.Message;
                return null;
            }
        }

        public async void Initialize()
        {
            List<Country> countries = await DeserializationAsync();

            if (countries == null || countries.Count == 0)
            {
                Data = Data ?? "Данные не найдены";
                return;
            }

            try
            {
```

```
                Country russia = countries[0];

                Data =
                    $"Страна: {russia.name.common}\n" +
                    $"Официальное название: {russia.name.official}\n" +
                    $"Столица: {russia.capital[0]}\n" +
                    $"Регион: {russia.region}\n" +
                    $"Подрегион: {russia.subregion}\n" +
                    $"Континент: {russia.continents[0]}\n" +
                    $"Население: {russia.population}\n" +
                    $"Площадь: {russia.area} км²\n" +
                    $"Валюта: {russia.currencies.RUB.name}
({russia.currencies.RUB.symbol})\n" +
                    $"Язык: {russia.languages.rus}\n" +
                    $"Флаг: {russia.flags.png}\n" +
                    $"Автодвижение: {russia.car.side}\n" +
                    $"Член ООН: {(russia.unMember ? "Да" : "Нет")}\n" +
                    $"Почтовый код: {russia.postalCode.format}";
            }
            catch (Exception ex)
            {
                Data = "Ошибка при обработке данных страны: " + ex.Message;
            }
        }
    }
}
```

Описание кода

После получения JSON-данных ViewModel выполняет десериализацию в объект Country. Затем из полей объекта формируется текст Data (например: название, столица, регион, валюта, язык, флаг и т.д.) и выводится в интерфейс. Если десериализация или доступ к полям вызывает ошибку, ViewModel перехватывает исключение и выводит сообщение об ошибке в Data.

Результат

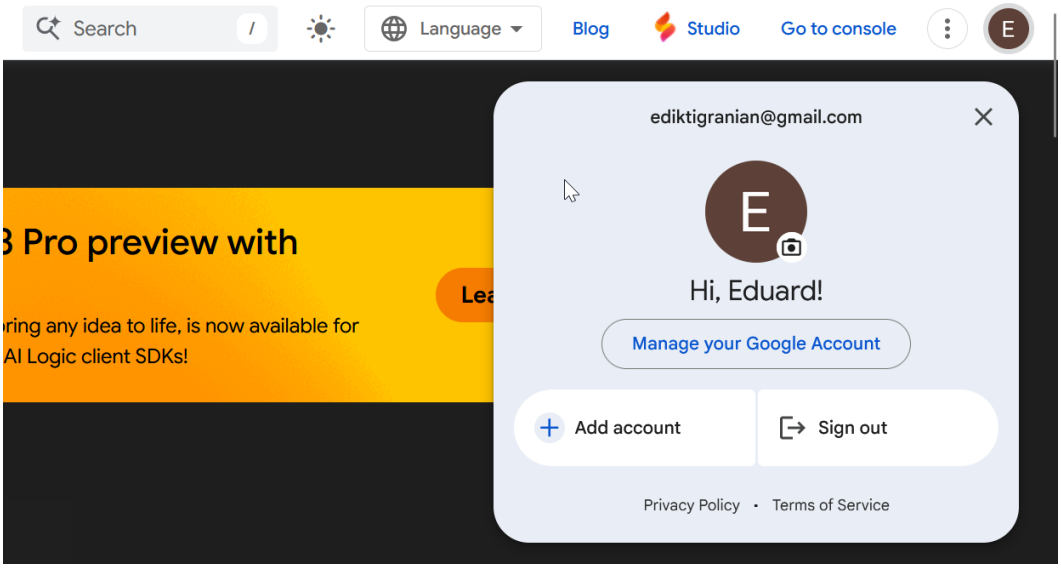


Главная окно

Лабораторная работа 4

Задание 1, 2

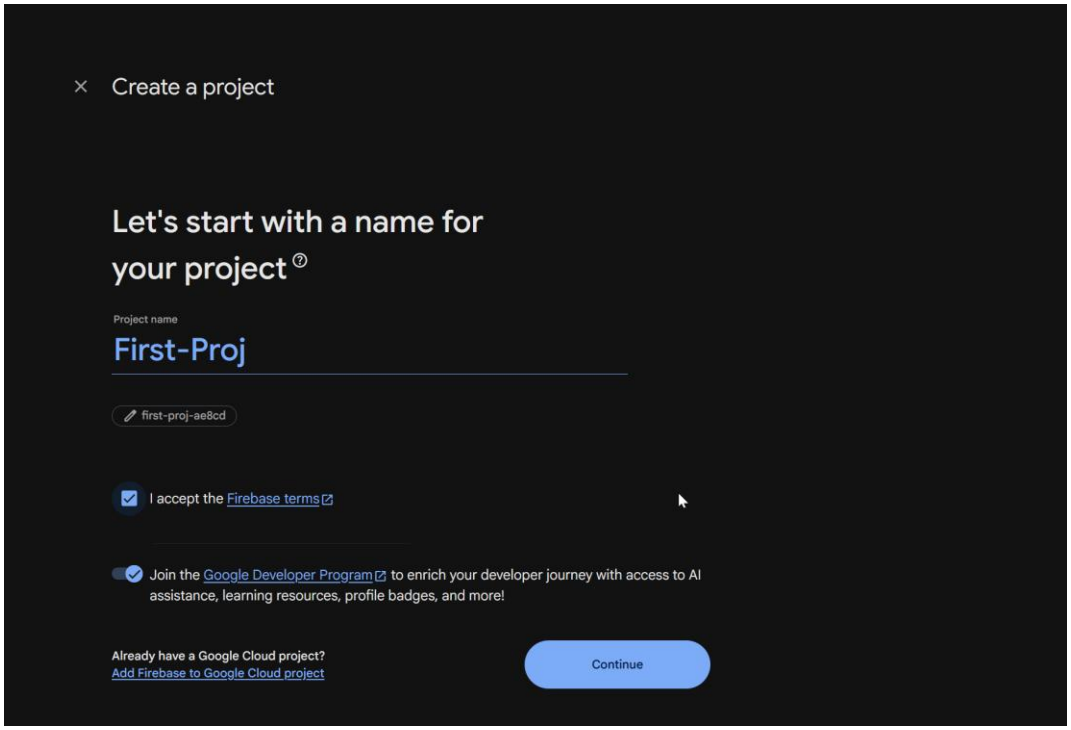
Создать коллекцию и документы firestore



Осуществление входа в аккаунт на сайте Firebase

Был выполнен вход в Google-аккаунт, необходимый для работы с Firebase. Панель аккаунта отображает активного пользователя и предоставляет доступ

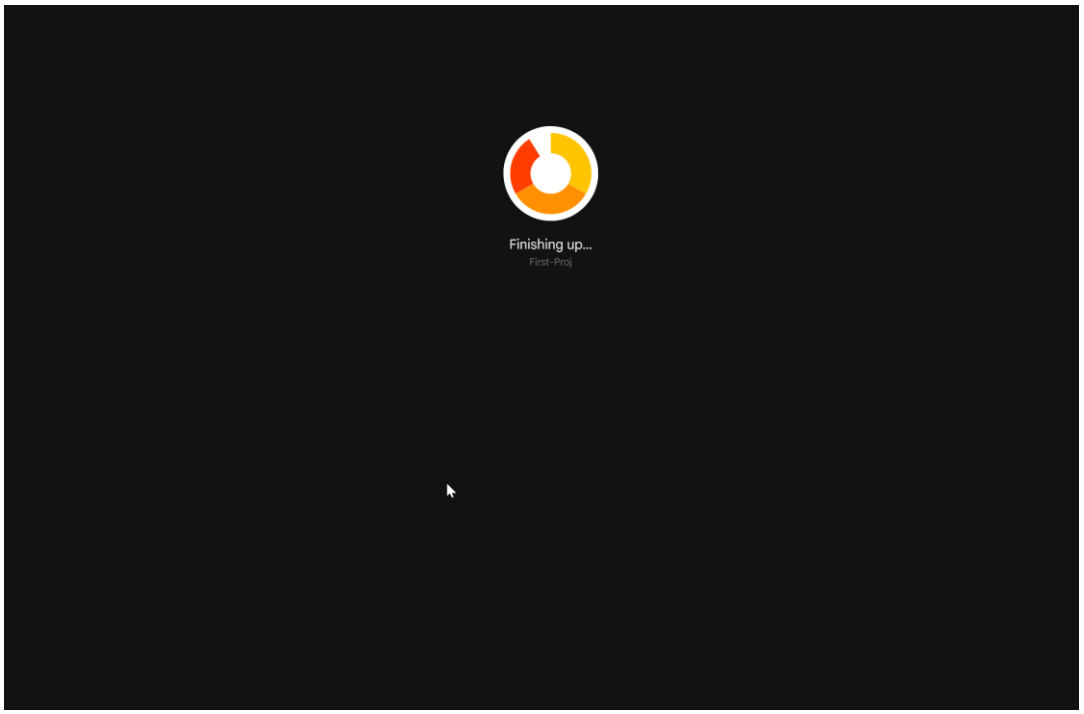
к управлению аккаунтом Google, добавлению новых аккаунтов или выходу из системы.



Создание проекта

В поле Project name введено имя проекта — First-Proj.

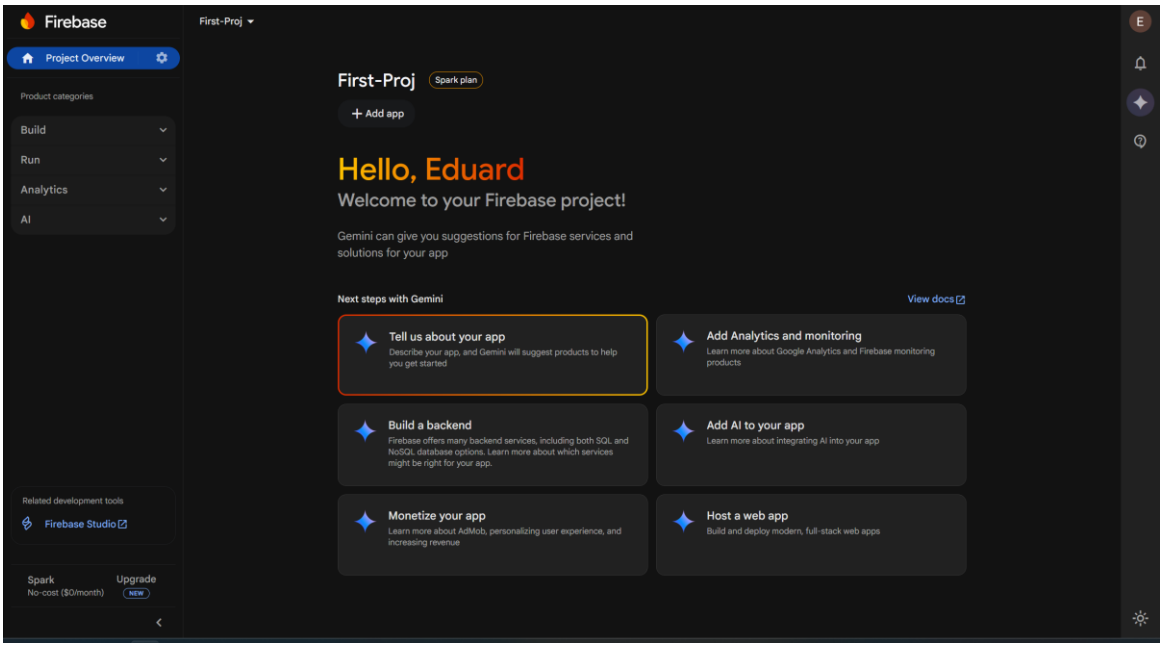
Также принято пользовательское соглашение Firebase и подтверждено участие в Google Developer Program.



Загрузка проекта

Отображается индикатор выполнения с надписью Finishing up ….

Firebase завершает автоматическую настройку инфраструктуры проекта.

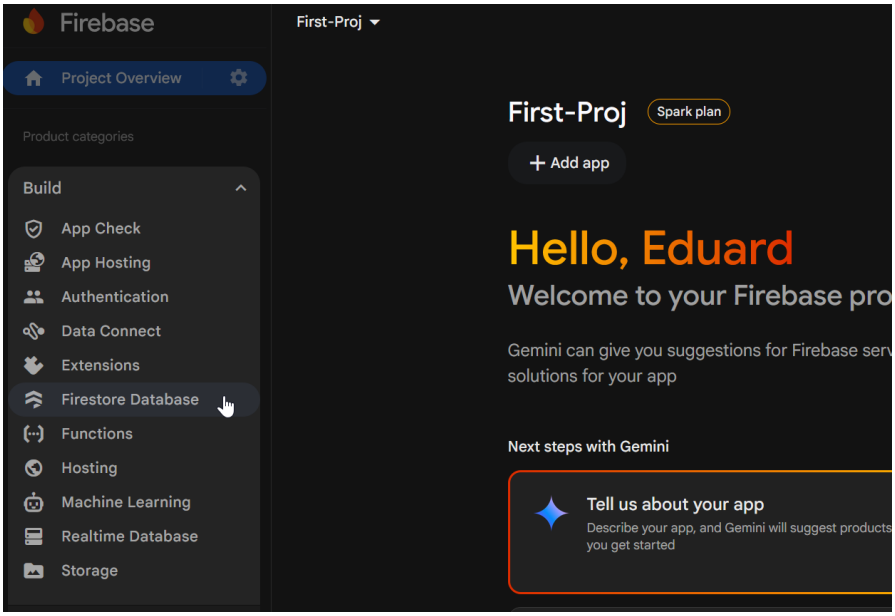


Главная страница проекта Firebase

Открыта страница Project Overview проекта First-Proj.

Отображается приветственное сообщение и доступные шаги:

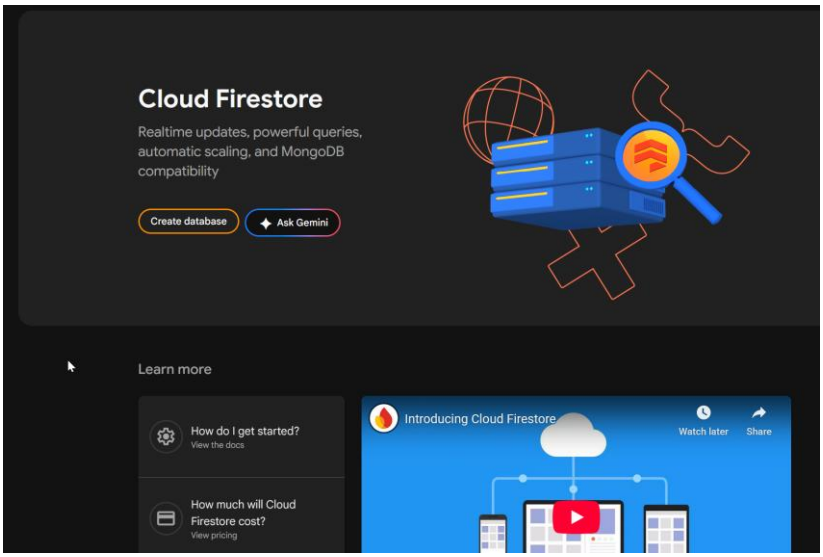
- добавление приложения,
- подключение аналитики,
- настройка backend-сервисов,
- добавление AI-функций.



Переход к Firestore Database

В левом боковом меню выбран пункт Firestore Database в разделе Build.

Данный сервис используется для хранения данных в формате NoSQL.

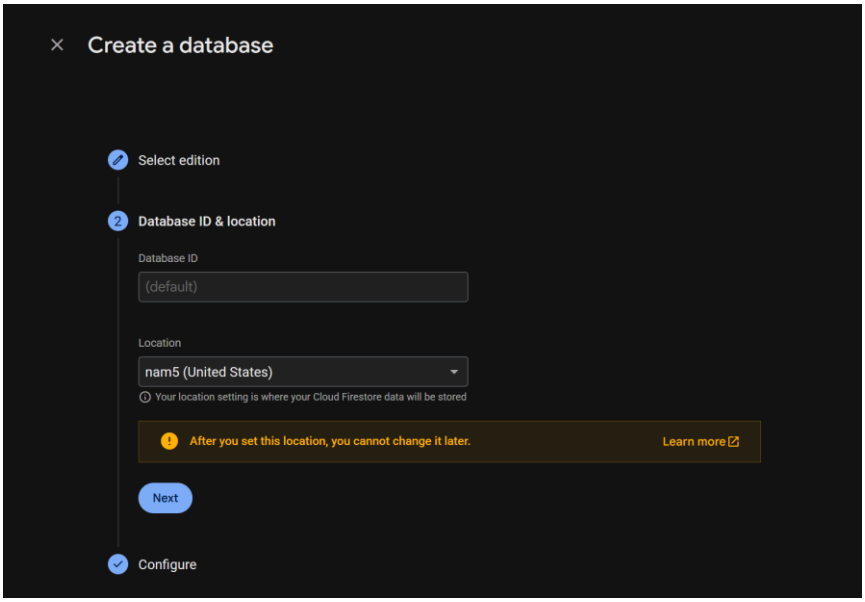


Страница Cloud Firestore

Отображается экран Cloud Firestore с кнопкой Create database.

Firestore поддерживает:

- обновления в реальном времени,
- масштабирование,
- гибкую структуру данных.

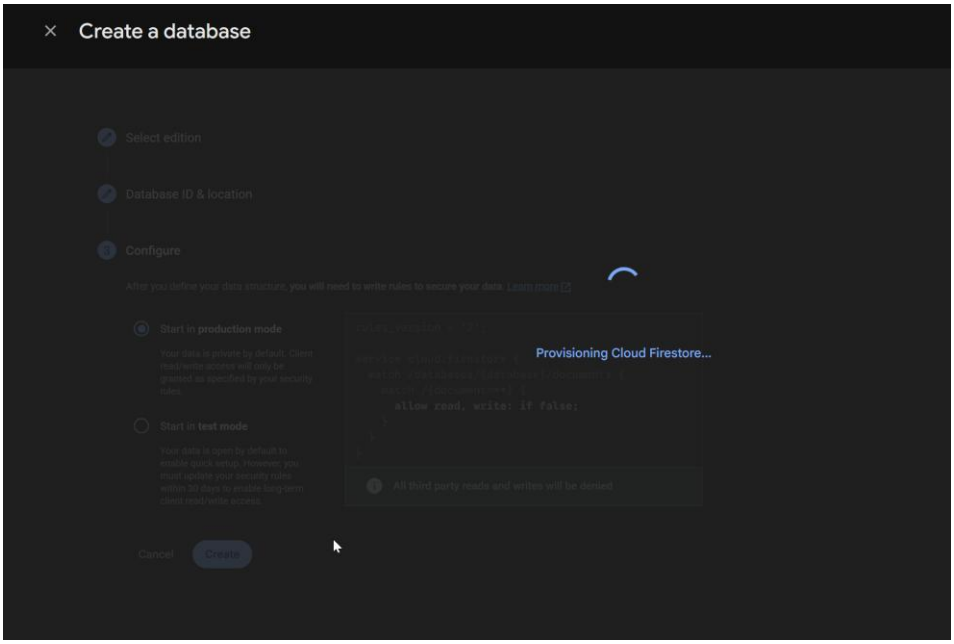


Выбор параметров базы данных

В окне Create a database указаны параметры:

- Database ID: (default)
- Location: nam5 (United States)

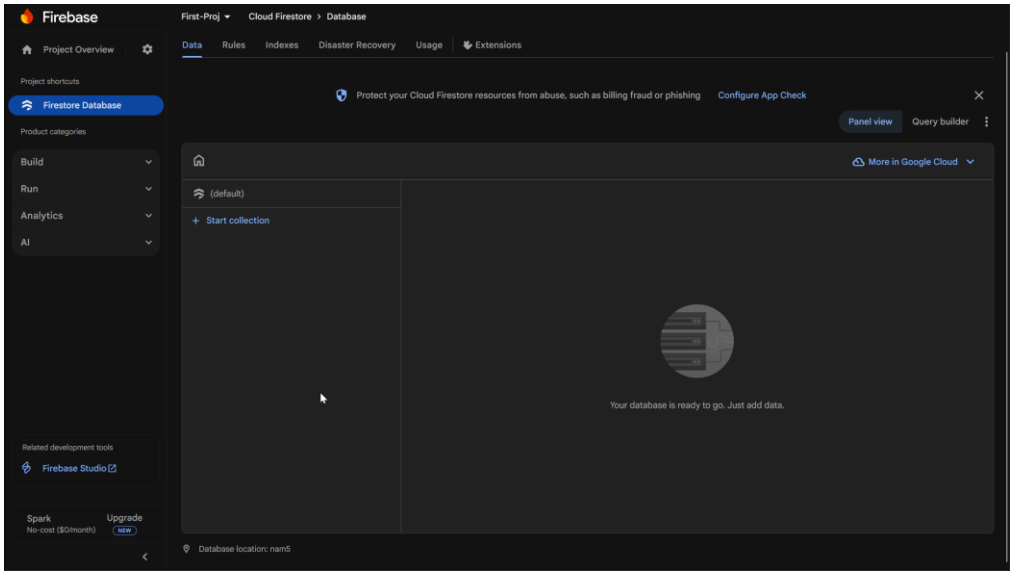
Также отображается предупреждение о невозможности изменения региона после создания базы данных.



Процесс создания базы данных

Отображается окно Provisioning Cloud Firestore….

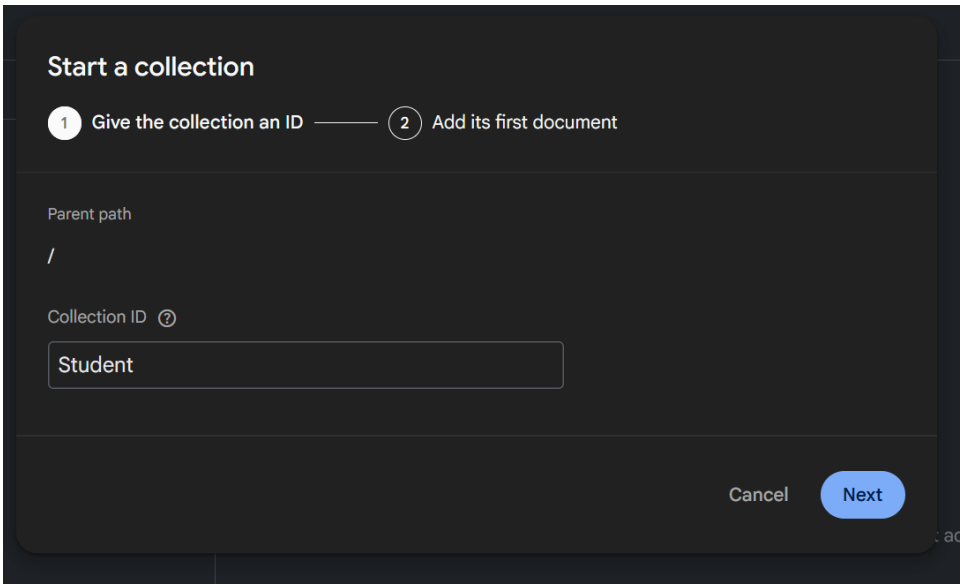
Firebase выполняет инициализацию базы данных и настройку серверной части.



Пустая база данных Firestore

Открыта вкладка Data Firestore Database.

Отображается сообщение о том, что база данных готова к работе, но данные ещё не добавлены.

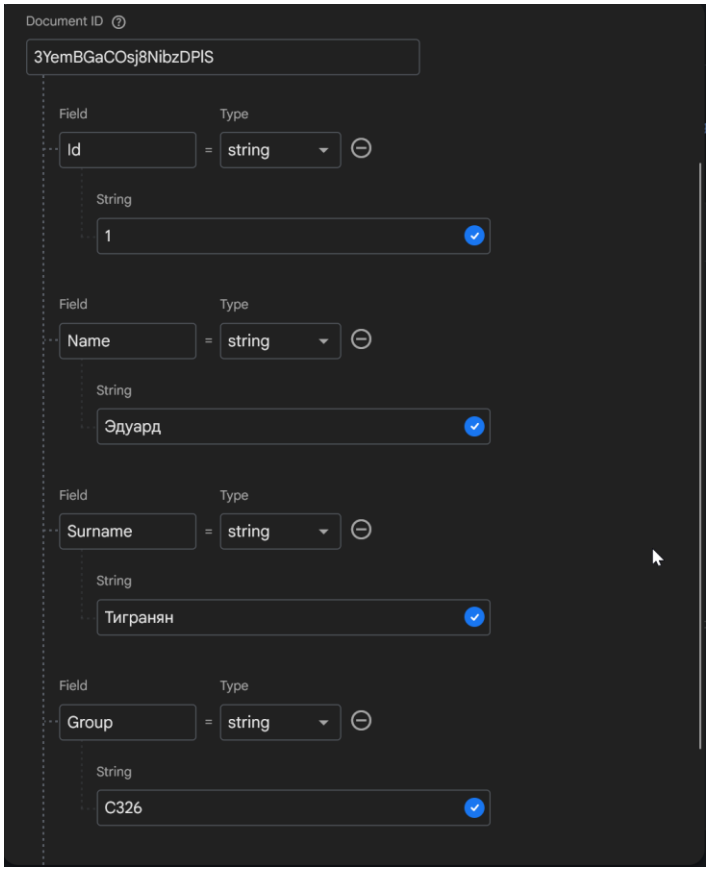


Создание первой коллекции

Открыто окно Start a collection.

В поле Collection ID введено название коллекции — Student.

Коллекция будет использоваться для хранения данных студентов.



Добавление документа в коллекцию Student

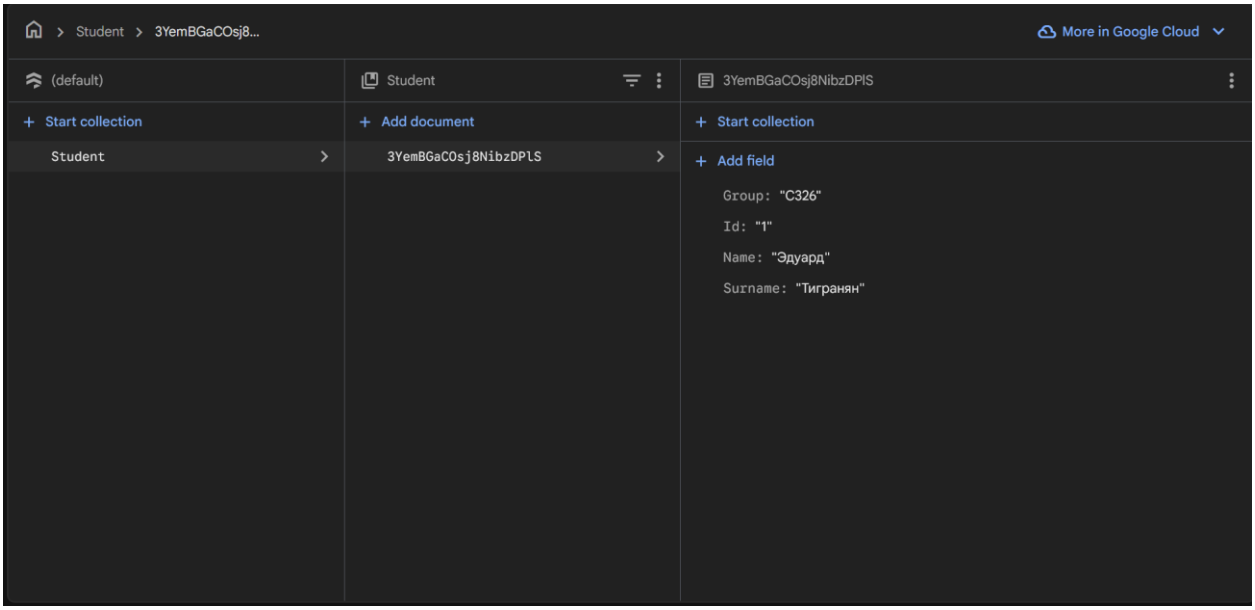
Добавление первого документа в коллекцию Student базы данных Cloud Firestore.

Документу был автоматически присвоен уникальный идентификатор (Document ID).

Для документа заданы следующие поля:

- Id (string) — идентификатор студента;
- Name (string) — имя студента;
- Surname (string) — фамилия студента;
- Group (string) — учебная группа.

Все поля имеют строковый тип данных (string).

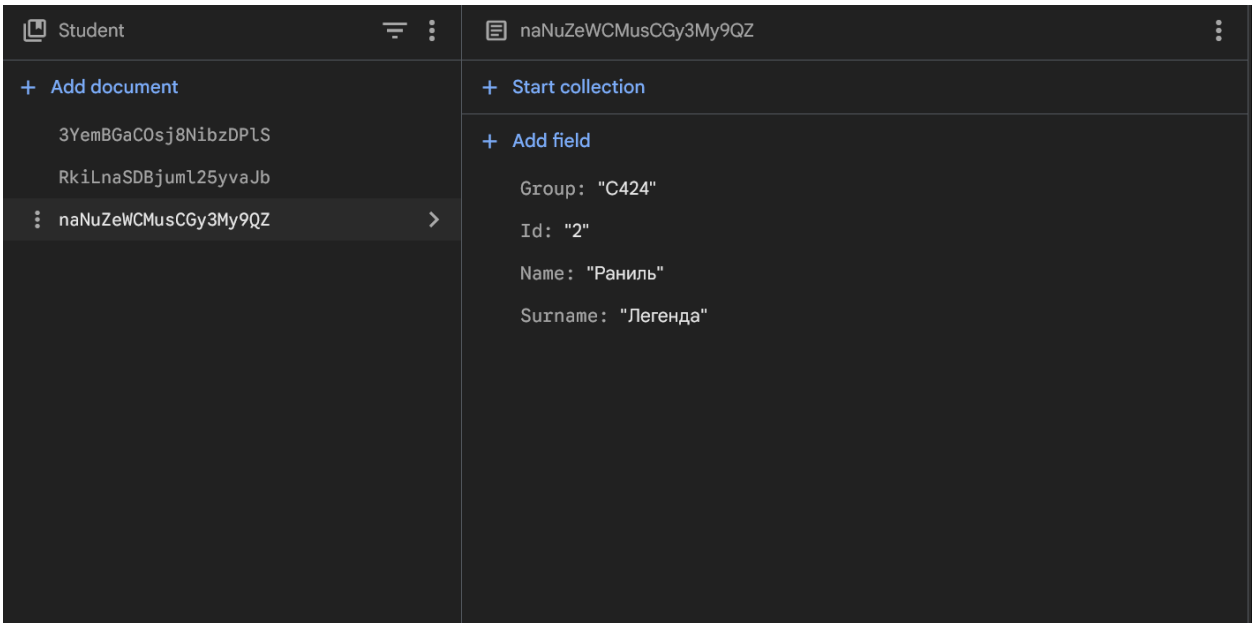


Просмотр структуры коллекции Student

Отображается структура базы данных Firestore:

- коллекция Student;
- документ с автоматически сгенерированным ID;
- список полей и их значений (Id, Name, Surname, Group).

Интерфейс Firestore позволяет визуально просматривать структуру NoSQL-базы.



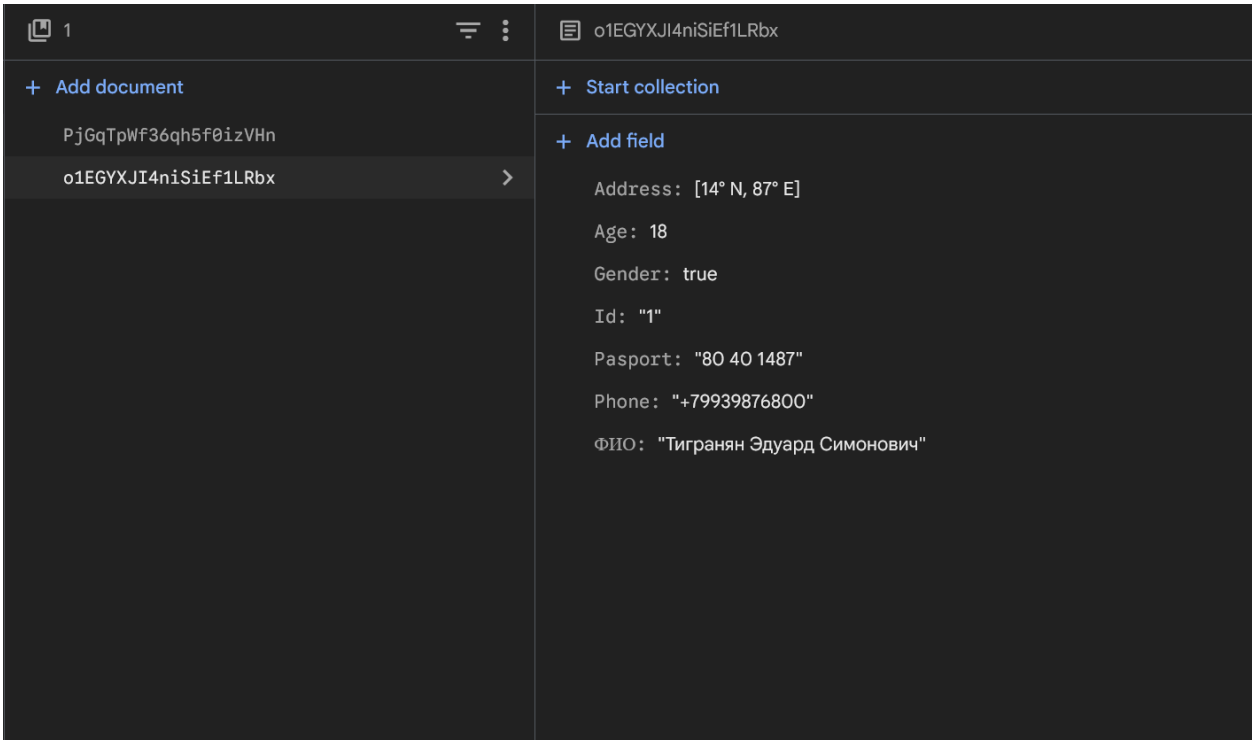
Добавление второго документа в коллекцию Student

В коллекцию Student добавлен ещё один документ с новыми данными:

- Id — «2»;
- Name — имя второго студента;
- Surname — фамилия второго студента;
- Group — другая учебная группа.

Каждый документ имеет уникальный идентификатор, что обеспечивает независимое хранение данных.

Задание 3



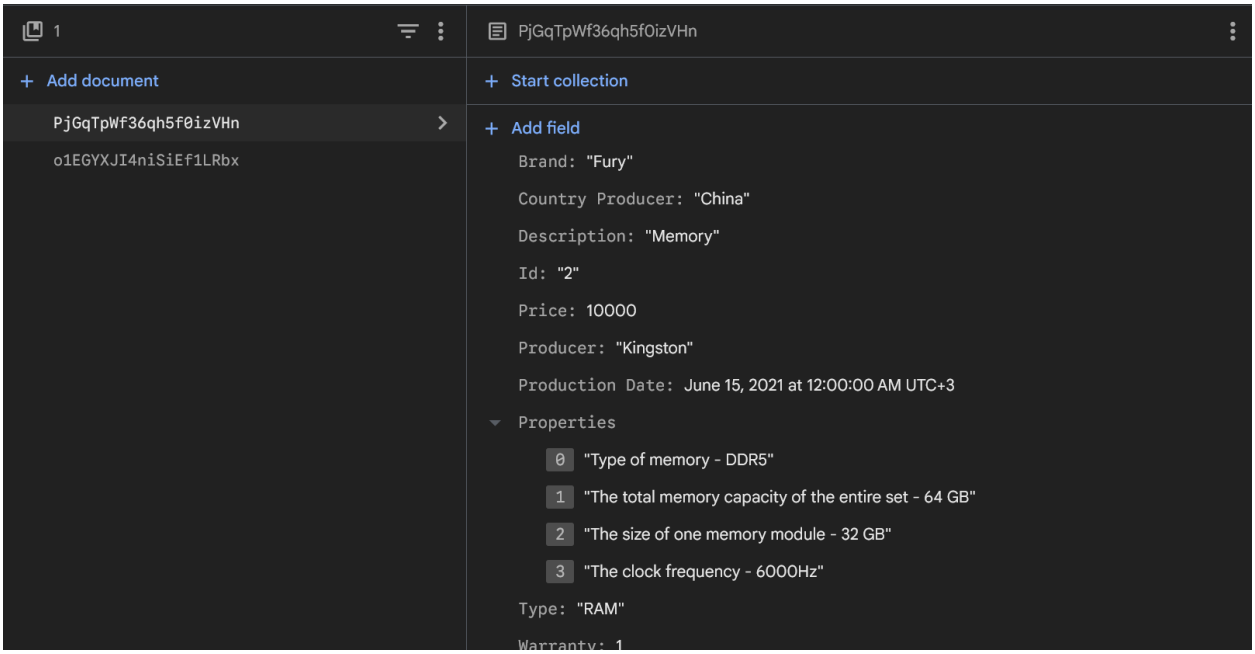
Создание вложенной коллекции (подколлекции)

Показан пример структуры документа с расширенными данными студента. Добавлены дополнительные поля:

- Age — возраст;
- Gender — пол;
- Phone — номер телефона;
- Passport — паспортные данные;
- ФИО — полное имя;
- Address — координаты (геолокация).

Также используются разные типы данных:

- string,
- number,
- boolean,
- array,
- geopoint.



Коллекция с данными о товаре

В базе данных создан документ, содержащий информацию о товаре (оперативная память):

Brand — производитель;

Country Producer — страна производства;

Price — цена;

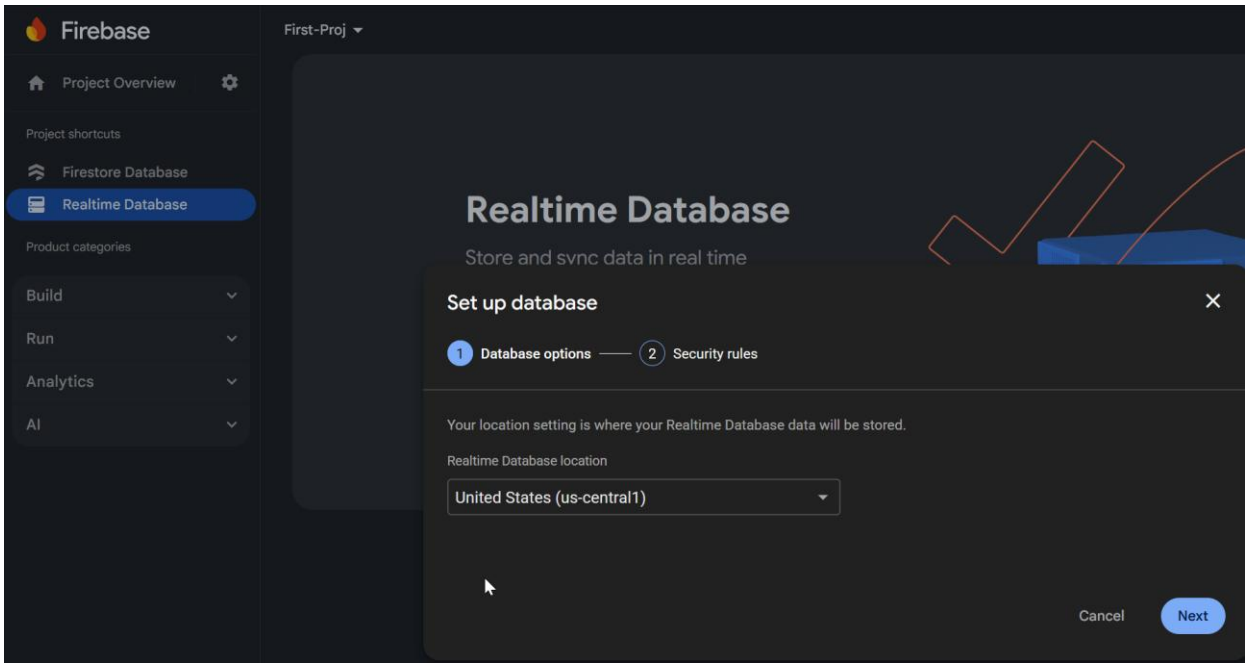
Production Date — дата выпуска;

Type — тип устройства;

Properties — массив характеристик (тип памяти, объём, частота).

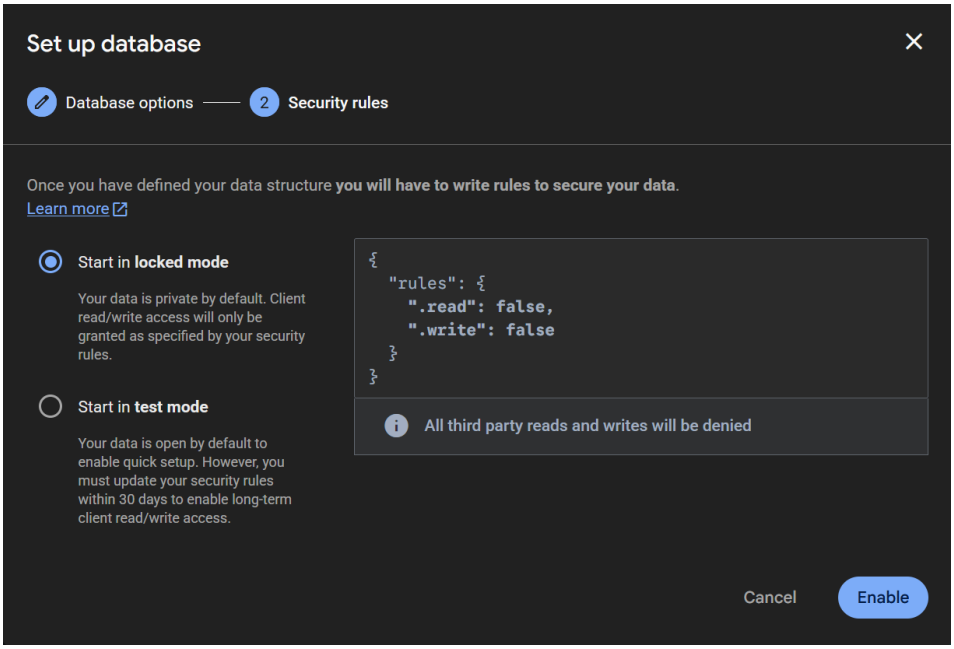
Используются вложенные структуры и массивы данных.

Задание 4



Создание Realtime Database

В разделе Realtime Database выполнен запуск мастера создания базы данных. Выбран регион хранения данных — United States (us-central1).

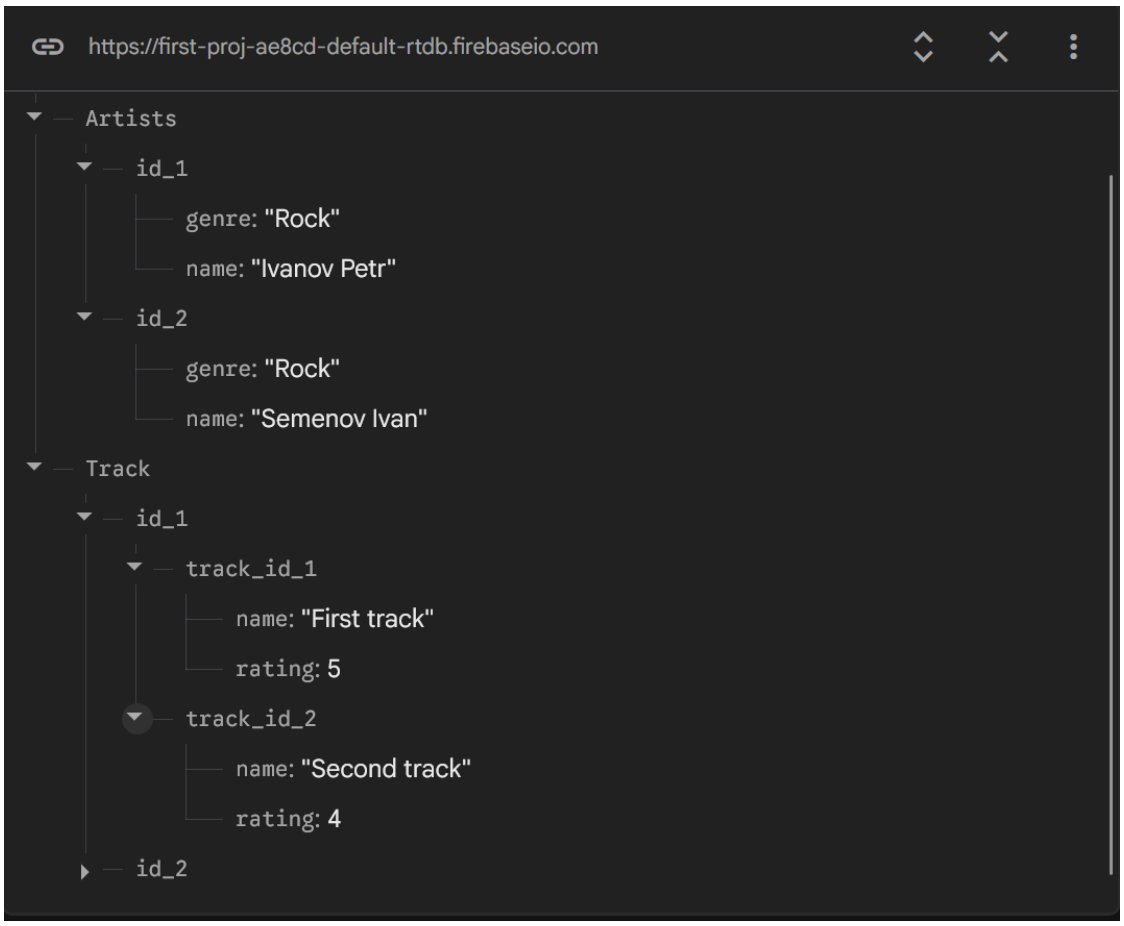


Настройка правил безопасности Realtime Database

Отображается окно выбора режима безопасности:

- Locked mode — доступ к данным запрещён по умолчанию;
- Test mode — временный открытый доступ.

Для лабораторной работы выбран режим Start in locked mode, обеспечивающий максимальную безопасность.



Структура данных Realtime Database

Отображается веб-интерфейс Realtime Database с иерархической структурой данных:

- узел Artists, содержащий данные об исполнителях (имя, жанр);
- узел Track, содержащий треки, их названия и рейтинги.

Данные представлены в формате JSON.