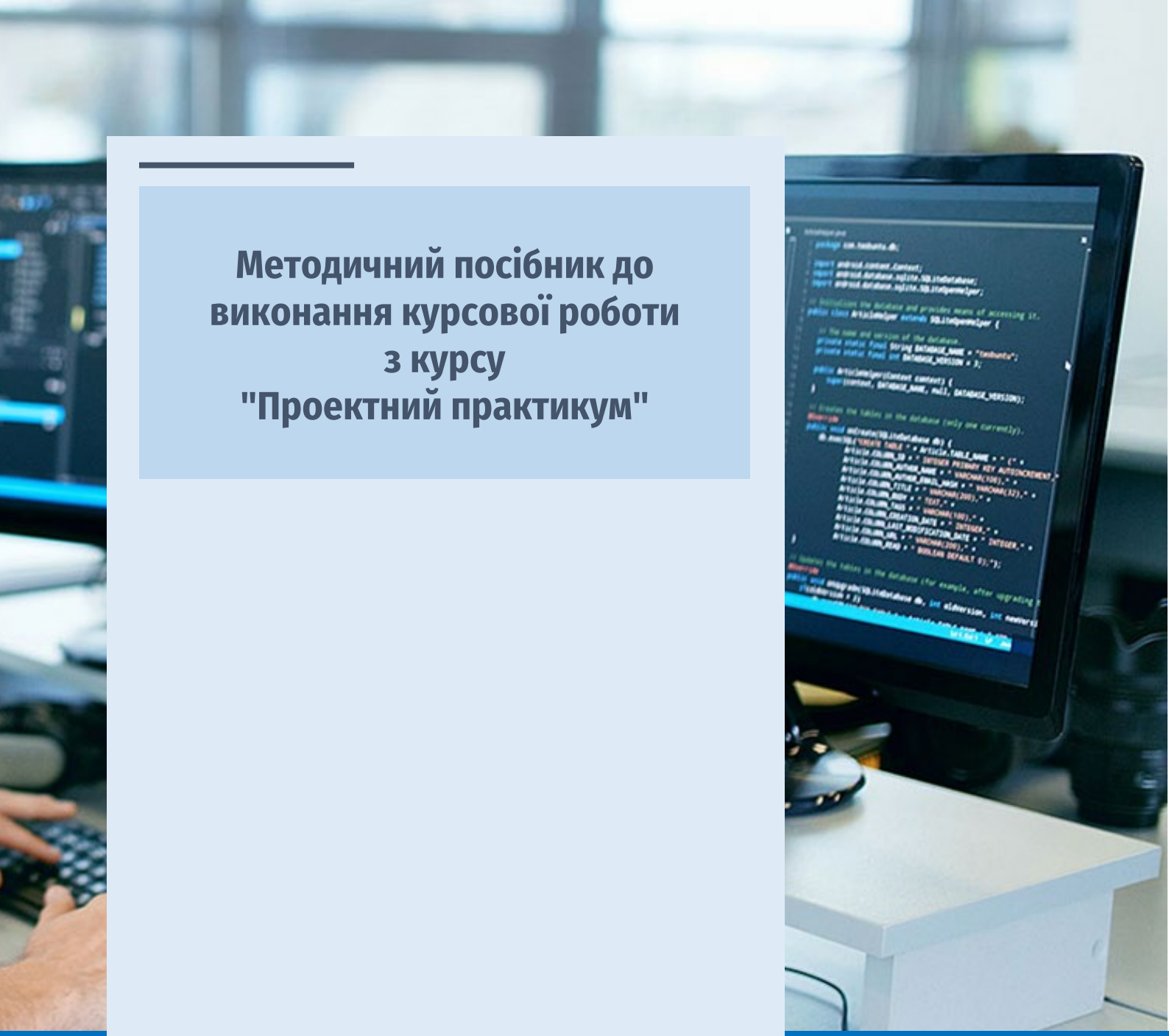


Методичний посібник до виконання курсової роботи з курсу "Проектний практикум"

2022

ТНТУ, кафедра програмної інженерії

Методичний посібник до виконання курсової роботи з курсу “Проектний практикум”/ Уклад. Коноваленко І.В., Петрик М.Р.– Тернопіль: ТНТУ, 2022.

Укладачі: канд. техн. наук, доцент кафедри автоматизації технологічних процесів і виробництв Коноваленко І.В.

докт. техн. наук, професор кафедри програмної інженерії Петрик М.Р.

Методичні вказівки розглянуті та затверджені на засіданні кафедри програмної інженерії.

1. Завдання на курсову роботу

Використовуючи принципи об'єктно-орієнтованого програмування та принципи модульного тестування, розробити застосунок WPF для платформи .NET 5.0, який дозволяє працювати з даними про об'єкти заданої предметної області. Для цього розробити клас, який описує об'єкт предметної області. Клас повинен містити не менше 10 властивостей різних типів і не менше двох методів (крім конструкторів), які забезпечать обробку даних об'єкта. Вид властивостей та методів слід вибрати самостійно шляхом аналізу заданої предметної області.

Дані про об'єкти слід зберігати у базі даних Microsoft SQL Server.

Застосунок, зокрема, повинен забезпечити:

- збереження даних про об'єкти у БД Microsoft SQL Server;
- структурне розділення коду на шар доступу до даних (Data Access Layer) та користувацький шар (Application Layer);
- модульні тести для методів DAL;
- клас, що інкапсулює логіку роботи з об'єктом вибраного виду;
- відображення даних про перелік об'єктів у табличному виді у вікні WPF;
- ввід даних про новий об'єкт;
- зміну даних про попередньо введені об'єкти;
- видалення попередньо введених об'єктів по одному;
- відображення результату роботи обох методів класу;
- експорт даних про перелік об'єктів у текстовий файл формату CSV;
- сортування переліку даних за кількома найважливішими атрибутами;
- фільтрування даних у переліку за одним з найважливіших атрибутів;
- збереження базових налаштувань між сесіями роботи.

Для взаємодії з користувачем у вікні WPF використати елементи керування "панель інструментів" (елемент керування ToolBar), головне меню (Menu) та контекстне меню (ContextMenu).

Застосунок повинен містити не менше двох форм.

Застосунок повинен забезпечити збереження базових налаштувань між сесіями роботи, до яких віднесемо: розміри (ширина, висота) головного вікна та порядок сортування даних.

Вид об'єкта (його предметну область) слід обрати згідно варіанту, або самостійно, узгодивши з викладачем. Можна також вибрати інше завдання, узгодивши його з викладачем.

Таблиця №1. Варіанти завдань на курсову роботу

№	Об'єкт	№	Об'єкт
1	Комп'ютер	16	Годинник
2	Принтер	17	Книга
3	Автомобіль	18	Будівля
4	Літак	19	Планета
5	Фотоапарат	20	Футбольна команда
6	Жорсткий диск до комп'ютера	21	Рослина
7	Процесор	22	Тварина
8	Телефон	23	Університет
9	Телевізор	24	Фільм
10	Велосипед	25	Музичний виконавець
11	Планшет	26	
12	Ноутбук	27	
13	Абонент телефонної мережі	28	
14	Студент	29	
15	Персона	30	

2. Приклад вирішення завдання

2.1. Уточнення завдання

Розглядатимемо предметну область: "місто". Проаналізувавши предметну область, вибираємо для об'єкта "місто" такі властивості:

1. Назва (тип даних – рядок).
2. Країна, у якій розташоване місто (тип даних – рядок).
3. Регіон, у якому розташоване місто (тип даних – рядок).
4. Кількість населення (тип даних – ціле число).
5. Річний бюджет (тип даних – число з плаваючою комою).
6. Площа (тип даних – число з плаваючою комою).
7. Географічна широта (тип даних – число з плаваючою комою).
8. Географічна довгота (тип даних – число з плаваючою комою).
9. Наявність порта (тип даних – логічний).
10. Наявність аеропорта (тип даних – логічний).

На основі даних про об'єкт "місто" слід розрахувати (сформувати):

1. Суму річного бюджету, яка припадає на одного мешканця (річний бюджет поділити на кількість мешканців).
2. Щільність населення у місті (кількість мешканців поділити на площу міста).

Застосунок має виконувати сортування списку з об'єктами "місто" за назвою, країною, регіоном, кількістю населення та площею.

Слід реалізувати можливість фільтрування даних у списку за параметром "кількість населення". Користувач має задати мінімальне значення кількості населення, після чого застосунок відобразить у списку тільки ті записи, для яких об'єкти "місто" мають чисельність, більшу за вказану.

2.2. Архітектура рішення

Архітектуру рішення проілюстровано на рис. 1. Як довготривале та надійне сховище зберігання даних використано базу даних Microsoft SQL Server. Для низькорівневої роботи з сутностями БД використано шар доступу до даних (Data Access Layer, DAL). Він міститиме методи для виконання операцій з БД (для читання даних, додання, редагування та видалення записів тощо). Виокремлення таких дій в окремий програмний шар дозволяє їх використання

настільними, мобільними чи web-застосунками. У даних методичних вказівках розглянуто розробку лише настільного WPF-застосунку, але застосунки інших видів також можуть бути побудовані на основі розроблених БД та DAL.

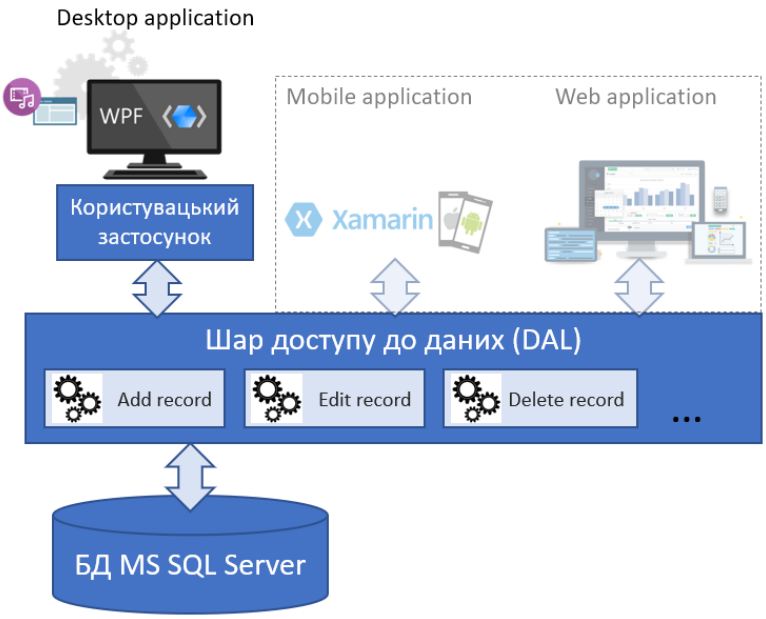


Рис. 1. Архітектура рішення

Таким чином, рішення[1] Microsoft Visual Studio міститиме такі проекти:

1. Проект бази даних.
2. Проект бібліотеки для шару доступу до даних.
3. Проект модульних тестів для шару доступу до даних.
4. Проект клієнтського WPF-застосунку.

Головне вікно застосунку, розробку якого розглядатимемо в наступних розділах, приведено на рис. 2.

[1] Проект Microsoft Visual Studio знаходиться всередині "рішення" (solution). Незважаючи на назву, під рішенням не мається на увазі "вирішення питання". Це просто контейнер для одного або декількох пов'язаних проектів разом з інформацією про збірку, параметрами середовища Visual Studio та іншими файлами, які не належать до якогось конкретного проекту.

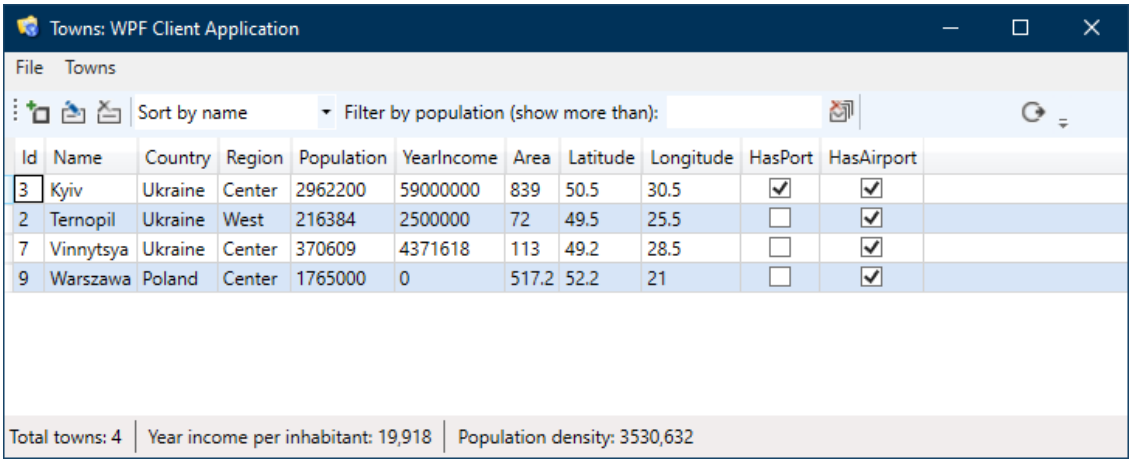


Рис. 2. Головне вікно застосунку

2.3. Розробка бази даних Microsoft SQL Server

2.3.1. Створення проекту SQL Server Database Project

Щоб створити новий проект для бази даних Microsoft SQL Server, слід у середовищі Visual Studio вибрати команду File/New/Project, далі вказати шаблон проекту SQL Server Database Project (рис. 3) і задати назву проекту (в нашому прикладі він називатиметься TownsDBDemo). Слід також звернути увагу, що так як це перший проект, що створюється, потрібно вказати назву рішення та шлях, де воно зберігатиметься.

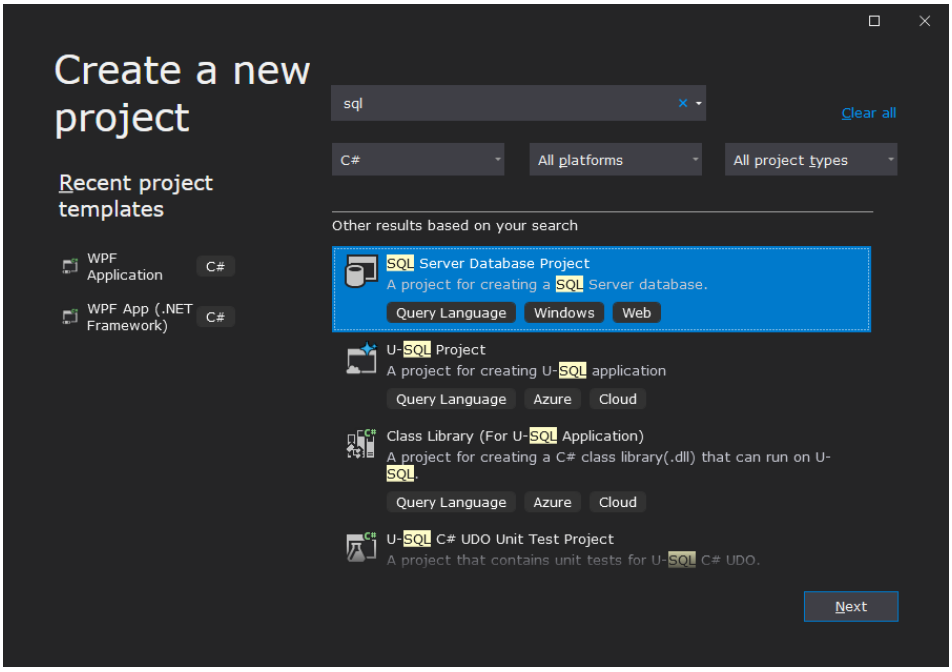


Рис. 3. Створення проекту SQL Server Database Project в середовищі MS Visual Studio

2.3.2. Створення таблиці для збереження даних

Дані про об'єкти відповідно до завдання зберігатимуться у таблиці Towns. Щоб її створити, слід викликати контекстне меню на проекті TownsDBDemo у вікні Solution Explorer і вибрати команду Add/Table, після чого задати назву таблиці та підтвердити створення SQL-файлу з її описом.

Далі за допомогою дизайнера необхідно задати всі поля (стопці) таблиці (рис. 4). Відповідно до першої нормальної форми, таблиця повинна містити первинний ключ. Його назва – Id. Для цього поля слід задати тип int, а також задати для нього властивості:

- Is Identity = true;
- Identity Increment = 1;
- Identity Seed = 1.

Такі налаштування дозволять при доданні нового запису у таблицю автоматично створювати для первинного ключа унікальні цілі значення.

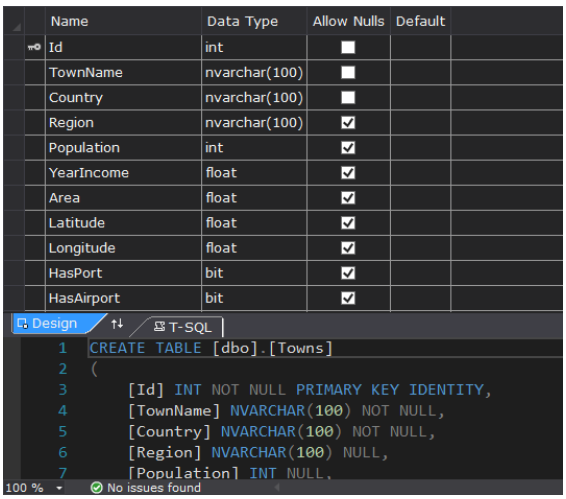


Рис. 4. Створення таблиці Towns у дизайнері Visual Studio

Назви і типи всіх інших полів таблиці необхідно вибрати відповідно до завдання. Призначення полів описано у таблиці 2.

Таблиця 2. Поля таблиці для зберігання даних про місто

Назва поля	Тип SQL	Опис
Id	int	Ідентифікатор міста, первинний ключ
TownName	nvarchar(100)	Назва міста

Country	nvarchar(100)	Країна, у якій розташоване місто
Region	nvarchar(100)	Регіон, у якому розташоване місто
Population	int	Кількість населення
YearIncome	float	Річний бюджет
Area	float	Площа
Latitude	float	Географічна широта
Longitude	float	Географічна довгота
HasPort	bit	Наявність порта
HasAirport	bit	Наявність аеропорта

Варто звернути увагу, що для полів таблиці також можна задати обов'язкові та необов'язкові поля. Для цього у стовпці Allow nulls дизайнера (рис. 4) слід поставити маркер (маркер вказує на те, що запис таблиці може містити значення null для цього поля). При цьому первинний ключ обов'язково повинен містити значення.

Зауважимо, що коли у дизайнері міняється структура таблиці, у нижній його частині на вкладці T-SQL Visual Studio автоматично генеруватиме SQL-код, який описує таблицю із заданими властивостями (рис. 4). Можливий і зворотній порядок дій: через введення SQL-коду мінятиметься подання таблиці у дизайнері. У лістингу 1 приведено SQL-код, що описує розроблену таблицю.

Лістинг 1

```
CREATE TABLE [dbo].[Towns]
(
    [Id] INT NOT NULL PRIMARY KEY IDENTITY,
    [TownName] NVARCHAR(100) NOT NULL,
    [Country] NVARCHAR(100) NOT NULL,
    [Region] NVARCHAR(100) NULL,
    [Population] INT NULL,
    [YearIncome] FLOAT NULL,
    [Area] FLOAT NULL,
    [Latitude] FLOAT NULL,
    [Longitude] FLOAT NULL,
    [HasPort] BIT NULL,
    [HasAirport] BIT NULL
)
```

2.3.3. Опублікування бази даних

Після того, як розробка бази даних закінчена, базу даних необхідно опублікувати (фізично створити у середовищі Microsoft SQL Server). Для цього

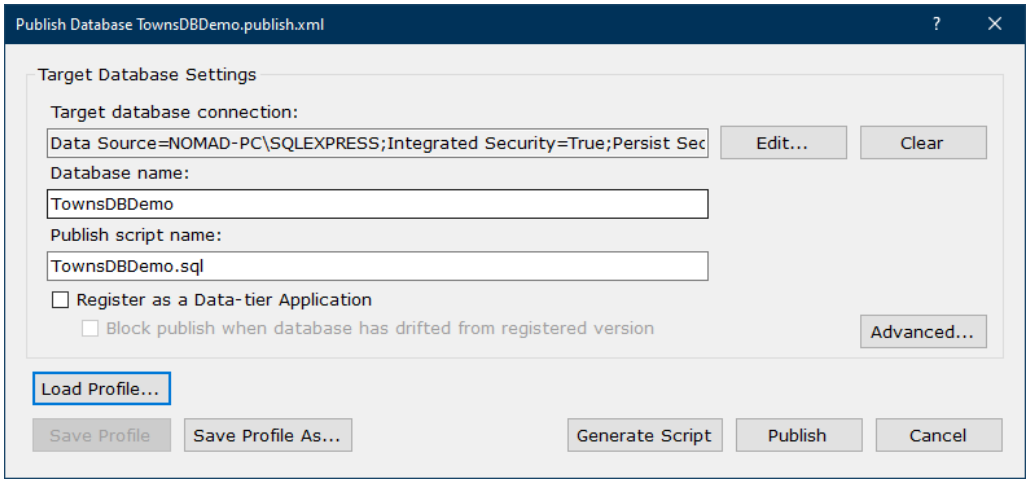
сервіс MS SQL Server повинен бути встановлений і запущений на машині, де ведеться розробка. При цьому зручно користуватись SQL Server Express LocalDB (це функція SQL Server Express, призначена спеціально для розробників). Його можна встановити безпосередньо при встановленні Microsoft Visual Studio. Інсталяція LocalDB копіює мінімальний набір файлів, необхідних для запуску сервісу SQL Server.

Щоб опублікувати БД, необхідно з контекстного меню проекту TownsDBDemo у Solution Explorer вибрати команду Publish… Далі у вікні Publish Database слід ввести рядок з'єднання, назву БД та скрипт, який містить SQL-код, з якого буде розгорнуто БД (рис. 5,а).

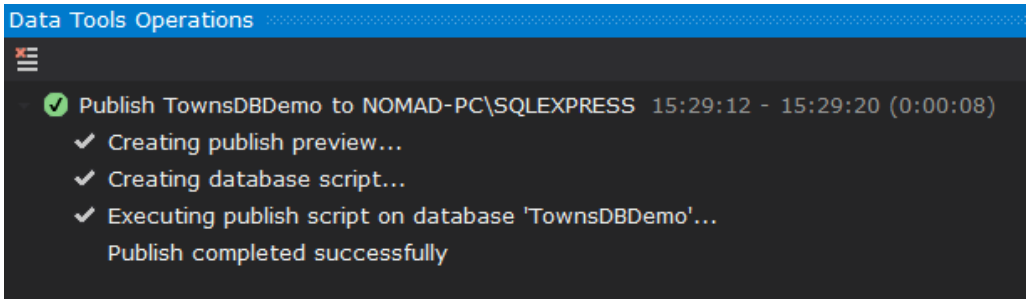
Найважливішим параметром є рядок з'єднання. Його вид може бути різним, залежно від налаштувань SQL Server. Якщо використовувати аутентифікацію на основі користувачів Windows, рядок з'єднання може мати вид:

```
Data Source=NOMAD-PC\SQLEXPRESS;Integrated Security=True;
Persist Security Info=False;Pooling=False;
MultipleActiveResultSets=False;Connect Timeout=60;
Encrypt=False;TrustServerCertificate=False
```

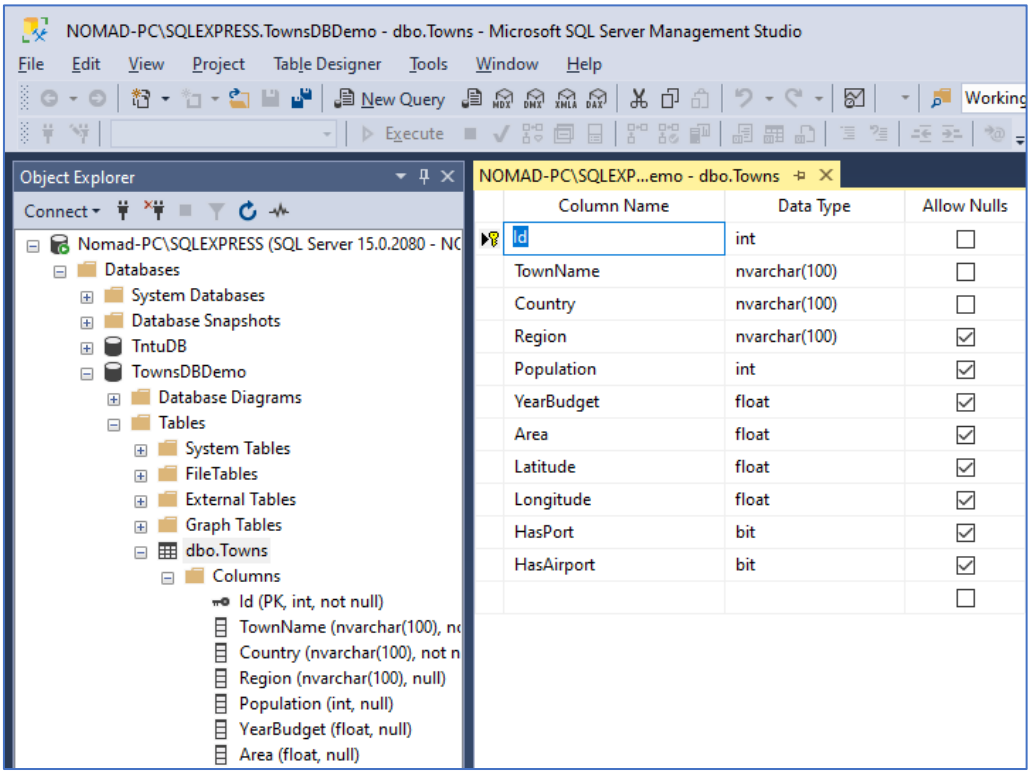
Важливо правильно задати назву машини з працюючим SQL Server (у прикладі вище – NOMAD-PC), назву сервісу (у приклад – SQLEXPRESS) та параметри аутентифікації.



а



б



в

Рис. 5. Опублікування бази даних

Після задання рядка з'єднання та назви бази даних слід натиснути кнопку Publish (рис. 5,а). Про успішне опублікування свідчитиме відповідне повідомлення у вікні Data Tools Operations (рис. 5,б). Після цього база даних буде доступною для клієнтських застосунків, наприклад, Microsoft SQL Server Management Studio. Якщо з'єднатись із сервером за допомогою такого застосунку, можна побачити відповідну БД TownsDBDemo та таблицю Towns у ній (рис. 5,в).

2.4. Розробка шару доступу до даних

2.4.1. Створення проекту Class Library

Рівень доступу до даних призначений для виконання низькорівневих SQL-запитів до бази даних. Він має бути цілком незалежним від інтерфейсу та логіки роботи користувацького застосунку. Код DAL зручно оформити як бібліотеку класів .NET. Така відокремлена структура дозволяє використовувати розроблену бібліотеку без змін у застосунках різних видів (настільних, мобільних, web тощо, див. рис. 1).

Щоб додати до існуючого рішення Microsoft Visual Studio новий проект бібліотеки класів, слід вибрати команду меню File/New/Project, і вказати шаблон Class Library (рис. 6) з мовою програмування C#. Далі слід вказати назву проекту (в розглянутому прикладі це DataAccessLibrary), і цільову платформу (.NET 5.0). Після створення нового проекту бібліотеки у вікні Solution Explorer має відображуватись вже два проекти (проект БД і проект бібліотеки).

2.4.2. Проектування класу Town для опису об'єкта "місто"

2.4.2.1. Опис елементів класу Town

Назвемо клас, який інкапсулює об'єкт "місто" – Town. Повернувшись до уточненого завдання та проаналізувавши предметну область, складемо список елементів класу, необхідних для реалізації поведінки об'єкта "місто" згідно завдання.

Щоб зберігати необхідні (перелічені у завданні) дані про місто, клас Town повинен мати властивості відповідно до таблиці 3:

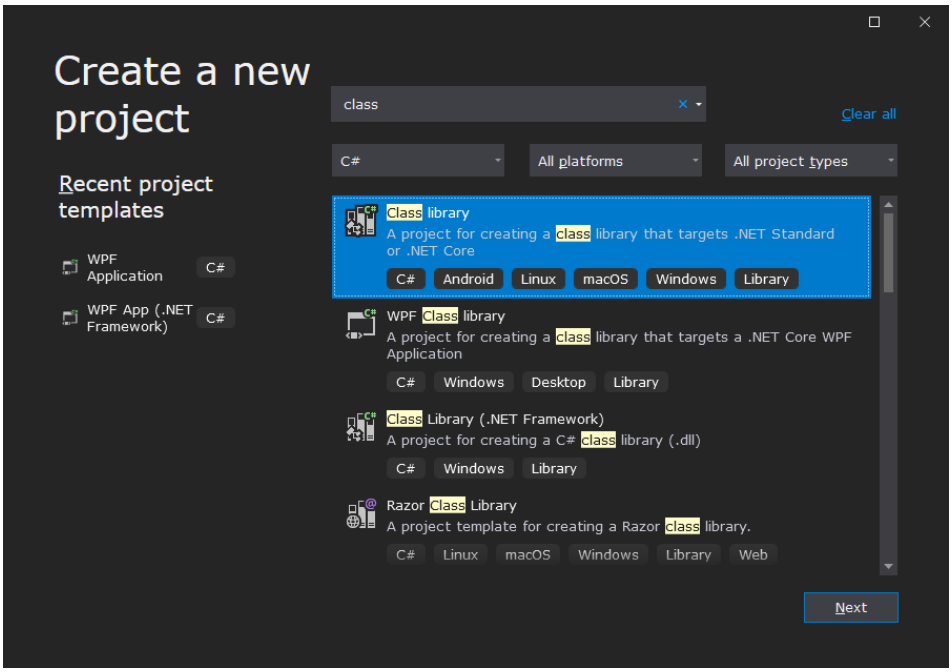


Рис. 6. Створення проекту Class Library в середовищі MS Visual Studio

Таблиця 3. Властивості класу Town

Назва	Опис назви	Тип	Опис типу
Name	Назва міста	string	Рядок
Country	Назва країни, у якій розташоване місто	string	Рядок
Region	Назва регіону, у якому розташоване місто	string	Рядок
Population	Кількість мешканців	int	Ціле число
YearIncome	Річний бюджет	double	Число з плаваючою комою
Area	Площа міста	double	Число з плаваючою комою
Latitude	Географічна широта	double	Число з плаваючою комою
Longitude	Географічна довгота	double	Число з плаваючою комою
HasPort	Наявність порта	bool	Логічний
HasAirport	Наявність аеропорта	bool	Логічний

Для створення екземпляра об'єкта "місто" розробимо три конструктори: один з них створюватиме повністю "порожній" об'єкт з неініціалізованими даними, а два інші – на основі значень всіх вищезгаданих властивостей створюватиме екземпляр, який представлятиме реальний об'єкт "місто". Вид цих конструкторів (їх реалізацію детально розглянемо далі):

Конструктор	Опис
`public Town()`	Створює "порожній" об'єкт
`public Town(string name, string country, string region, int population, double yearIncome, double area, double latitude, double longitude, bool hasPort, bool hasAirport)`	Створює об'єкт, який представляє конкретне місто, без ідентифікатора міста
`public Town(int id, string name, string country, string region, int population, double yearIncome, double area, double latitude, double longitude, bool hasPort, bool hasAirport)`	Створює об'єкт, який представляє конкретне місто, з ідентифікатором міста

Для реалізації необхідних згідно завдання обчислень над даними об'єкта "місто" введемо у клас такі методи:

Метод	Опис методу
`public double GetYearIncomePerInhabitant()`	Розраховує суму річного бюджету на одного мешканця

`public double PopulationDensity()`	Розраховує щільність населення

2.4.2.2. Створення класу Town

Щоб додати у проект новий файл, де буде описано клас Town, слід з контекстного меню проекту в Solution Explorer вибрати команду Add\ New Item…, вказати шаблон Visual C# Items\ Class та задати назву файлу для опису класу – Town.cs (розширення можна не вказувати). Далі слід натиснути кнопку Add. Новий файл – Town.cs – з'явиться у списку Solution Explorer, і відкриється у редакторі коду.

У стандартній заготовці, яку Visual Studio створить для класу, слід ввести опис всіх властивостей згідно лістингу 2. У лістингах код, який автоматично генерується середовищем, відзначено сірим. Цей код не повинен набиратися вручну. Якщо його немає, щось виконано не так. У цьому випадку слід перевірити попередні дії над проектуванням застосунку. Крім властивостей класу, які представляють атрибути сутності "місто", клас Town містить ще одну властивість – Id. Вона призначена для збереження ідентифікатора міста з бази даних (див. табл. 2). Всі властивості мають автоматичні підтримуючі поля (на це вказують порожні методи доступу).

Зауважимо, що код у файлі Town.cs використовує простір імен System, тобто з самого початку тексту має бути:

```
using System;
```

Лістинг 2

```
public class Town
{
    public int Id { get; set; }
    public string Name { get; set; }
    public string Country { get; set; }
    public string Region { get; set; }
    public int Population { get; set; }
    public double YearIncome { get; set; }
    public double Area { get; set; }
    public double Latitude { get; set; }
    public double Longitude { get; set; }
    public bool HasPort { get; set; }
    public bool HasAirport { get; set; }
```

Тепер введемо код конструкторів (лістинг 3). Перший конструктор "порожній". Його виклик приведе до створення екземпляра класу Town з неініціалізованими властивостями. Їх значення у подальшому потрібно програмно задати. Другий конструктор отримує значення всіх властивостей, які містять дані про об’єкт "місто", і ініціалізує ними відповідні властивості. Таким чином, цей конструктор створює екземпляр, який відповідає конкретному місту. Третій конструктор виконує те ж, що і попередній, але крім інформації про саме місто, отримує також ідентифікатор міста.

Лістинг 3

```
public Town()
{

}

public Town(string name, string country, string region,
    int population, double yearIncome, double area,
    double latitude, double longitude, bool hasPort,
    bool hasAirport)
{
    Name = name;
    Country = country;
    Region = region;
    Population = population;
    YearIncome = yearIncome;
    Area = area;
    Latitude = latitude;
    Longitude = longitude;
    HasPort = hasPort;
    HasAirport = hasAirport;
}

public Town(int id, string name, string country, string region,
    int population, double yearIncome, double area,
    double latitude, double longitude, bool hasPort,
    bool hasAirport)
{
    Id = id;
    Name = name;
    Country = country;
    Region = region;
    Population = population;
    YearIncome = yearIncome;
    Area = area;
    Latitude = latitude;
    Longitude = longitude;
```

```
        HasPort = hasPort;
        HasAirport = hasAirport;
    }
```

Тепер додамо методи, які працюють з властивостями об'єкта (лістинг 4). Метод GetYearIncomePerInhabitant обчислює долю міського бюджету, яка припадає на одного мешканця. Для цього сума міського бюджету ділиться на кількість мешканців. Але інколи кількість мешканців може бути не задана (тоді відповідна властивість класу дорівнюватиме нулю). Щоб запобігти помилці ділення на нуль, метод виконує перевірку, чи властивість Population відмінна від нуля. Якщо Population дорівнюватиме нулю, метод поверне нуль. Метод GetYearIncomePerInhabitant повертає дійсне число.

Лістинг 4

```
public double GetYearIncomePerInhabitant()
{
    return Population != 0 ? YearIncome / Population : 0;
}

public double PopulationDensity()
{
    return Area != 0 ? Population / Area : 0;
}
```

Метод PopulationDensity обчислює щільність (густину) населення, ділячи загальну кількість мешканців на площу міста. Як і в попередньому випадку, перевіряємо, чи знаменник виразу не дорівнює нулю. Метод PopulationDensity також повертає дійсне число.

Цілісно текст файлу Town.cs у складі проекту DBAppDemo можна завантажити з матеріалів електронного навчального курсу "Проектний практикум" (файл DBAppDemo.zip).

2.4.2.3. Розробка шару доступу до даних

Для розміщення логіки роботи із даними до проекту слід додати ще один клас (TownDAL). Цей клас, як і раніше розроблений клас Town, розмістимо у просторі імен DataAccessLibrary. Клас TownDAL міститиме методи для роботи з даними у БД. У повному обсязі код класу TownDAL можна проглянути у

матеріалах електронного навчального курсу "Проектний практикум". Тут розглянемо тільки деякі операції щодо роботи із базою даних.

Щоб можна було виконувати операції з БД, до рішення слід додати пакет System.Data.SqlClient. Для цього у MS Visual Studio використовується менеджер пакетів NuGet (його можна відкрити через головне меню Tools/Nuget Package Manager/Manage NuGet Packages for Solution). Додамо до проекту DataAccessLibrary необхідний пакет. Для цього з контекстного меню проекту у вікні Solution Explorer слід вибрати команду Manage NuGet Packages… Далі у вікні NuGet: DataAccessLibrary слід перейти на вкладку Browse, в полі пошуку ввести назву SqlClient, і знайти пакет з назвою System.Data.SqlClient (рис. 7). Після цього в інформаційній панелі з описом пакету справа від переліку слід натиснути кнопку Install і дочекатись встановлення.

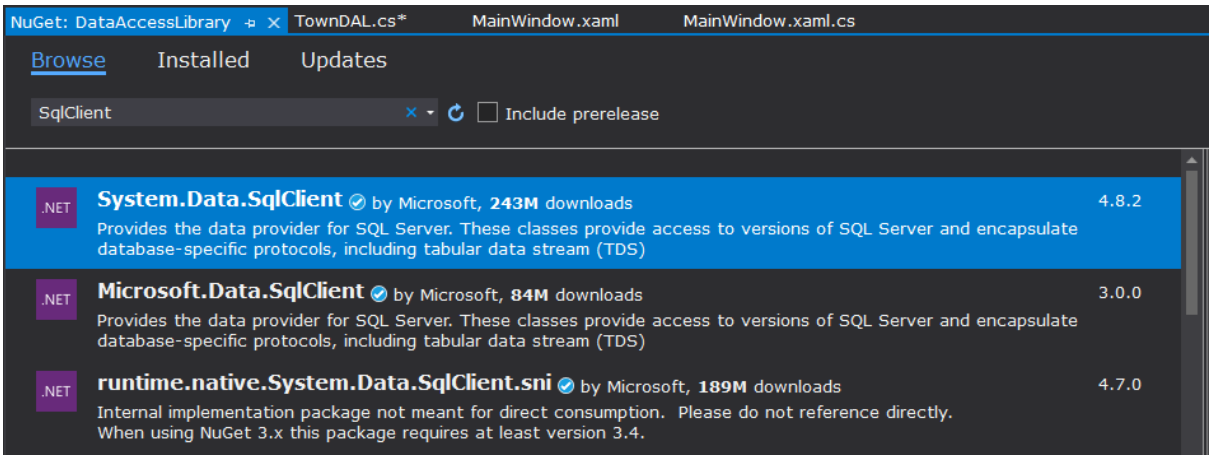


Рис. 7. Встановлення пакету System.Data.SqlClient через менеджер NuGet

Тепер можна перейти до програмування шару роботи із даними.

Клас TownDAL містить константу ConnectionString, яка є рядком з'єднання з БД:

```
Data Source=NOMAD-PC\SQLEXPRESS;Initial Catalog=TownsDBDemo;
Integrated Security=True;Persist Security Info=False;
Pooling=False;MultipleActiveResultSets=False;
Connect Timeout=60;Encrypt=False;TrustServerCertificate=False
```

У цьому рядку слід правильно задати назву сервера та сервісу MS SQL Server (параметр Data Source), назву БД (параметр Initial Catalog) та параметри аутентифікації (тут використано аутентифікацію на основі користувачів Windows).

Розглянемо операцію додання нових даних про місто у БД. Для цього призначений метод InsertTown класу, який отримує один параметр – екземпляр

раніше розробленого класу Town, дані з якого будуть перенесені у таблицю для тривалого зберігання.

Лістинг 5

```
public int InsertTown(Town town)
{
    using (SqlConnection connection = new(ConnectionString))
    {
        try
        {
            string insertSQL =
                "INSERT INTO Towns " +
                "(TownName, Country, Region, Population, " +
                "YearIncome, Area, Latitude, Longitude, " +
                "HasPort, HasAirport) " +
                "OUTPUT INSERTED.ID " +
                "VALUES " +
                "(@TownName, @Country, @Region, @Population, " +
                "@YearIncome, @Area, @Latitude, @Longitude, " +
                "@HasPort, @HasAirport)";

            using (SqlCommand insertQuery = new(insertSQL))
            {
                insertQuery.Connection = connection;
                insertQuery.Parameters.Add(
                    "@TownName",
                    SqlDbType.NVarChar,
                    100
                ).Value = town.Name;
                insertQuery.Parameters.Add(
                    "@Country",
                    SqlDbType.NVarChar,
                    100
                ).Value = town.Country;
                insertQuery.Parameters.Add(
                    "@Region",
                    SqlDbType.NVarChar,
                    100
                ).Value = town.Region;
                insertQuery.Parameters.Add(
                    "@Population",
                    SqlDbType.Int
                ).Value = town.Population;
                insertQuery.Parameters.Add(
                    "@YearIncome",
                    SqlDbType.Float
                ).Value = town.YearIncome;
                insertQuery.Parameters.Add(
                    "@Area",
                    SqlDbType.Float
```

```
                    ).Value = town.Area;
                    insertQuery.Parameters.Add(
                        "@Latitude",
                        SqlDbType.Float
                    ).Value = town.Latitude;
                    insertQuery.Parameters.Add(
                        "@Longitude",
                        SqlDbType.Float
                    ).Value = town.Longitude;
                    insertQuery.Parameters.Add(
                        "@HasPort",
                        SqlDbType.Bit
                    ).Value = town.HasPort;
                    insertQuery.Parameters.Add(
                        "@HasAirport",
                        SqlDbType.Bit
                    ).Value = town.HasAirport;

                    connection.Open();

                    int insertedId = Convert.ToInt32(
                        insertQuery.ExecuteScalar()
                    );

                    return insertedId;
                }
            }
            catch
            {
                return ERROR_CODE_COMMON;
            }
        }
    }
```

Для виконання SQL-запиту на додання нових даних у таблицю Towns у методі InsertTown спочатку встановлюється з'єднання з MS SQL Server за допомогою змінної connection класу SqlConnection. При створенні екземпляра цього класу йому передається рядок з'єднання ConnectionString, який буде використано при виконанні SQL-запитів.

Локальна змінна insertSQL містить код SQL-запиту, який буде виконано. У цьому запиті змінні параметри представлено мітками заповнення, назви яких починаються з символа @.

За допомогою екземпляра класу SqlCommand (змінна insertQuery) заданий SQL-запит виконується, і (у випадку успішного виконання) новий запис додається у базу даних. Використаний SQL-запит повертає також ідентифікатор

(значення первинного ключа Id таблиці Towns) нового запису про місто. Він зберігається у змінній insertedId. При виконанні операцій з'єднання з БД та виконання SQL-запиту відслідковуються виняткові ситуації. Тому, якщо станеться помилка, метод InsertTown поверне значення коду помилки, яке задане змінною ERROR_CODE_COMMON = -1. У випадку успішного виконання всіх операцій метод поверне ідентифікатор (додатне число) нового доданого запису таблиці Towns.

По закінченню роботи екземпляри класів SqlConnection та SqlCommand автоматично вивільняються, тому ніяких додаткових дій для цього виконувати не потрібно. Це забезпечується використання інструкції using.

Методи UpdateTown (міняє запис про задане місто) та DeleteTown (видаляє місто) працюють за аналогічною схемою. Різними є лише SQL-запити для виконання відповідних дій, та параметри, які при цьому використовуються.

Для читання всіх записів про місто використано метод ReadTowns. Він читає всі записи з таблиці в БД (виконує SQL-запит `SELECT * FROM Towns`), для кожного прочитаного міста створює екземпляр класу Town, і додає його у список міст TownsBuffer, який описаний як поле класу TownDAL:

```
public List<Town> TownsBuffer = new();
```

Метод ReadTowns повертає кількість прочитаних з таблиці записів.

На цьому розробка шару для роботи із даними (DAL) завершена (але ще не перевірена). Щоб скомпонувати написаний код у бібліотеку .NET (файл із розширенням dll), слід у контекстному меню проекту бібліотеки (у вікні Solution Explorer) вибрати команду Build. Якщо код не містить синтаксичних помилок, Visual Studio повідомить про успішне формування збірки.

Цілісно код проекту бібліотеки DAL можна завантажити з матеріалів електронного навчального курсу "Проектний практикум" (файл DBAppDemo.zip).

2.5. Розробка модульних тестів

2.5.1. Створення проекту MSTest Test Project

При розробці backend-засобів важливо переконатись, що вони функціонують коректно. Для перевірки роботоспроможності розробленого технічного коду, навіть при відсутності будь-якого користувацького інтерфейсу, використовують модульні тести. Вони дозволяють перевірити коректність роботи запрограмованої логіки.

Щоб додати до існуючого рішення Microsoft Visual Studio новий проект модульних тестів, слід вибрати команду меню File/New/Project, і вказати шаблон MSTest Test Project (рис. 7) з мовою програмування C#.

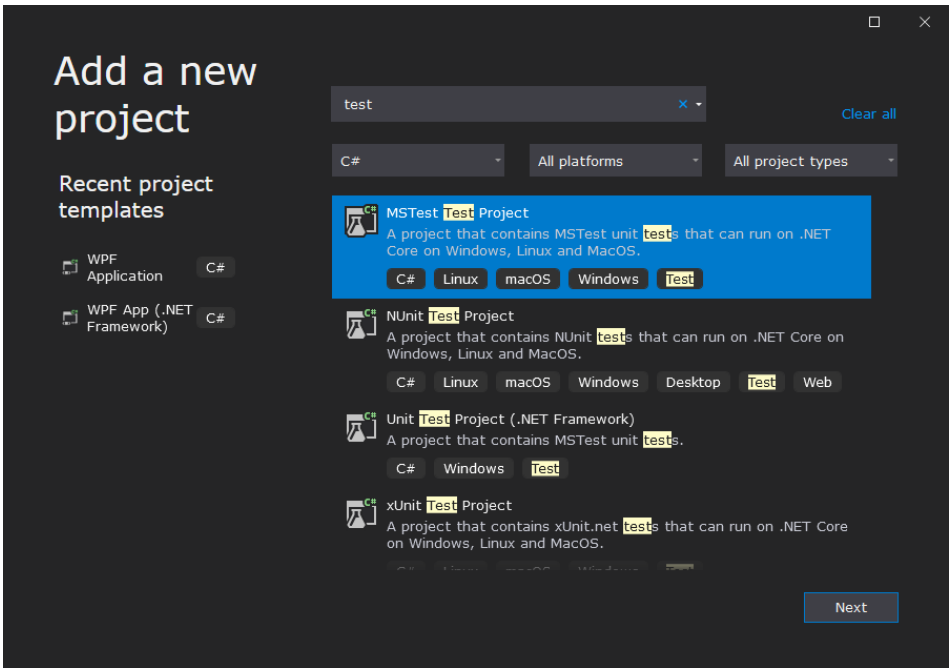


Рис. 8. Створення проекту MSTest Test Project в середовищі MS Visual Studio

Модульні тести повинні використовувати попередньо розроблену бібліотеку DataAccessLibrary. Тому її проект слід додати у посилання для проекту тестів. Для цього у вікні Solution Explorer з контекстного меню проекту тестів слід вибрати команду Add/Project Reference, і у вікні Reference Manager поставити маркер біля проекту DataAccessLibrary (рис. 9).

2.5.2. Розробка тестів

У cs-файлі тестів слід створити клас DALTest з атрибутом TestClass. Цей атрибут позначає клас, який містить модульні тести. Методи цього класу,

відмічені атрибутом TestMethod, міститимуть саме тести і виконуватимуться після запуску тестів.

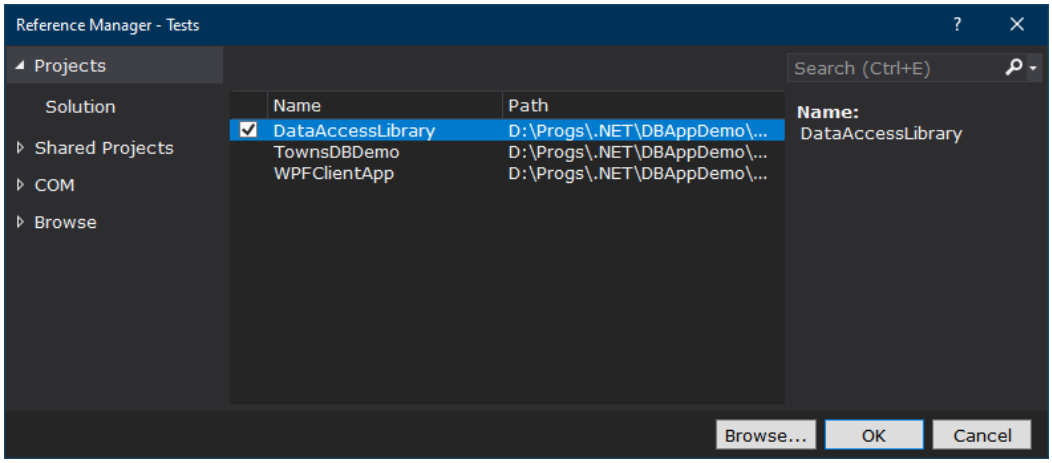


Рис. 9. Додання посилання на проект бібліотеки

2.5.2.1. Тест з'єднання з базою даних

Перший тест, який потрібно створити – це тест з'єднання з БД Microsoft SQL Server. Якщо з'єднання не встановлюватиметься, інші операції виконати буде неможливо.

Так як клас TownDAL бібліотеки описано в просторі імен DataAccessLibrary, її слід задекларувати у коді тестів через оператор using:

```
using DataAccessLibrary;
```

Після цього сутності простору імен DataAccessLibrary можна використовувати.

Метод класу DALTest для перевірки з'єднання називається CheckDBConnection (лістинг 6). У ньому створюється екземпляр класу SqlConnection, в який передається рядок з'єднання TownDAL.ConnectionString (див. п.2.4.2.3). Головне призначення цього тест-метода – перевірити рядок з'єднання з Microsoft SQL Server.

Лістинг 6

```
[TestMethod]
public void CheckDBConnection()
{
    bool connectionResult;
    using (SqlConnection connection =
              new(TownDAL.ConnectionString))
    {
        try
```

```
            {
                connection.Open();
                connectionResult = true;
            }
            catch (SqlException)
            {
                connectionResult = false;
            }
        }
        Assert.AreEqual(true, connectionResult);
    }
```

У методі CheckDBConnection робиться спроба встановити з'єднання з базою даних MS SQL Server. Якщо з'єднання встановлено, змінній connectionResult присвоюється значення true, а в протилежному випадку – false.

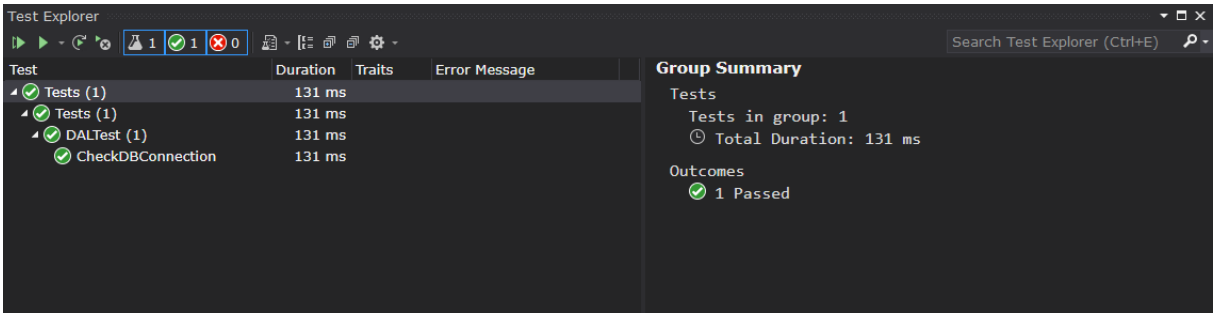
Для перевірки результатів виконання коду на відповідність очікуваним результатам використовується клас Assert з простору імен Microsoft.VisualStudio.TestTools.UnitTesting. Він містить набір методів, які дозволяють перевіряти різні умови, залежно від виду тестованих значень[2].

За допомогою методу AreEqual класу Assert перевіряємо, чи connectionResult дорівнює true (тобто – чи вдалося встановити з'єднання). Першим параметром метод AreEqual отримує очікуване у випадку коректної роботи значення, а другим – реальний результат роботи коду.

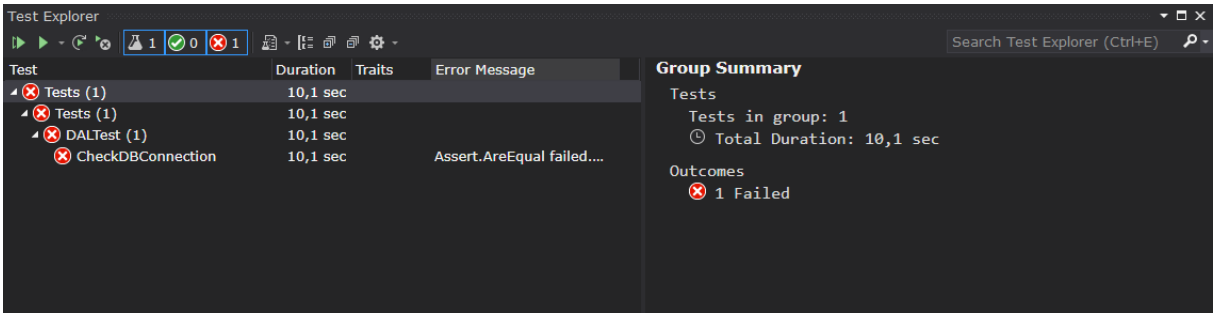
Коли тестовий метод готовий, запустити його виконання можна, вибравши команду Run Tests із контекстного меню проекту модульних тестів у вікні Solution Explorer. З'явиться вікно Test Explorer, у якому будуть відображені тести і результат їх виконання (рис. 10).

У вікні (рис. 10) показано результати виконання проекта модульних тестів Test, у якому є простір імен Test із тест-класом DALTest. Цей клас поки-що містить тільки один метод CheckDBConnection. Слід добитись, щоб тест проходив успішно (рис. 10,а). Якщо тест завершується невдачею, потрібно проаналізувати помилку і виправити її.

[2] Опис класу Assert можна детально проглянути у MSDN.



а



б

Рис. 10. Успішне (а) і помилкове (б) виконання тестів

2.5.2.2. Тест читання даних з БД

Наступний тест – для читання даних із таблиці у БД (лістинг 7). У цьому тест-методі створюється екземпляр класу TownDAL і викликається його метод ReadTowns, у якому реалізовано запит SELECT до БД. Метод ReadTowns повертає кількість прочитаних рядків з таблиці (або нуль, якщо таблиця порожня). Тому в кінці за допомогою методу AreNotEqual класу Assert перевіряємо, щоб результат отримання даних не дорівнював від'ємному коду помилки TownDAL.ERROR_CODE_COMMON.

Лістинг 7

```
[TestMethod]
public void ReadTownData()
{
    TownDAL townDAL = new();
    int readRows = townDAL.ReadTowns();

    Assert.AreNotEqual(TownDAL.ERROR_CODE_COMMON, readRows);
}
```

2.5.2.3. Тести додання, редагування та видалення даних

Тепер необхідно створити тест-методи для перевірки операцій додання нового запису в БД а також редагування та видалення існуючого запису.

Для додавання нового запису в таблицю Towns бази даних у тест-класі DALTest описано поле TernopilTown типу Town з даними про місто, яке має бути додане у таблицю (лістинг 8). Для перевірки функціональності методу InsertTown класу TownDAL створено тест-метод InsertRecord. У ньому створюється екземпляр townDAL, і викликається його мето InsertTown. Він повертає додатній ідентифікатор доданого у таблицю запису, або від'ємний код помилки, якщо операцію не вдалось виконати. Тому в кінці тесту перевіряємо, чи результат виконання методу InsertTown не дорівнює коду помилки.

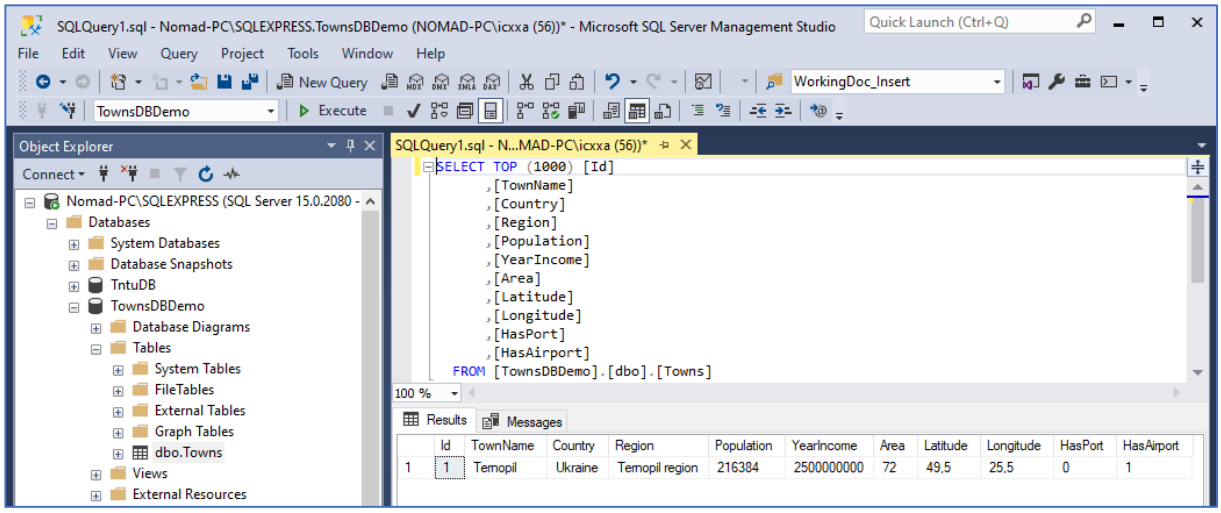
Лістинг 8

```
public static Town TernopilTown = new(
    name: "Ternopil",
    country: "Ukraine",
    region: "Ternopil region",
    population: 216384,
    yearIncome: 2500000000,
    area: 72,
    longitude: 25.5,
    latitude: 49.5,
    hasPort: false,
    hasAirport: true
    );

...

[TestMethod]
public void InsertRecord()
{
    TownDAL townDAL = new();
    int insertedId = townDAL.InsertTown(TernopilTown);

    Console.WriteLine("Inserted record Id = {0}", insertedId);
    Assert.AreNotEqual(TownDAL.ERROR_CODE_COMMON, insertedId);
}
```

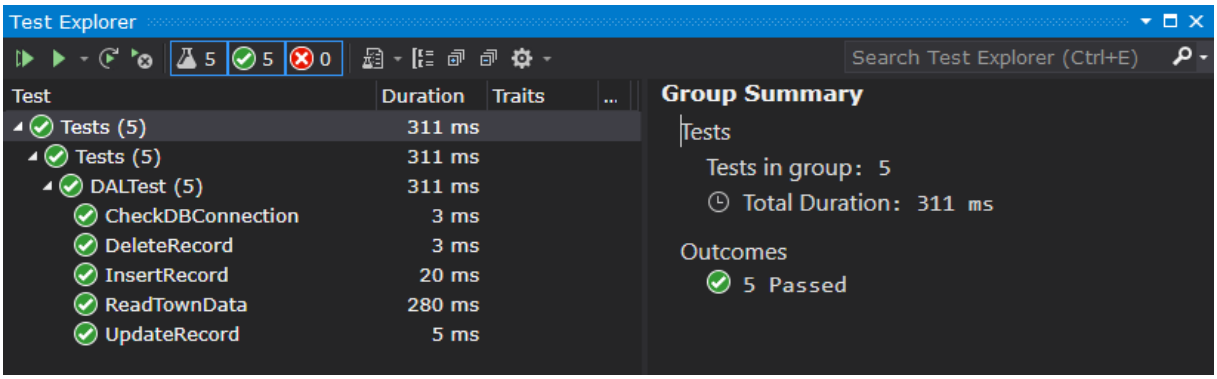
Якщо виконати розглянуті перед цим тести, і вони завершаться успішно, у БД буде новий запис про місто зі значеннями, визначеними у полі TernopilTown (див. лістинг 8). У цьому також можна переконатись, якщо з'єднатись із БД за допомогою клієнтського застосунку і перевірити вміст таблиці Towns (рис. 11,а).

Методи для редагування та видалення даних мають схожу структуру. Їх код приведено у проекті DBAppDemo в електронному навчальному курсі "Проектний практикум" (файл DBAppDemo.zip).

Ознакою правильної реалізації коду бібліотеки рівня доступу до даних є успішне виконання всіх доданих тестів (рис. 11,б). Якщо якийсь із них завершується невдачею, необхідно знайти причину і виправити її.



а



б

Рис. 11. Перевірка доданих даних за допомогою Microsoft SQL Server Management Console (а); результати успішного виконання всіх тест-методів для перевірки роботи DAL (б)

2.6. Розробка клієнтського застосунку

Попередні проекти формують бекенд (backend) – невидиму для користувача технічну частину застосунку, яка містить його внутрішній функціонал. Після того як цей функціонал готовий, можна приступати до розробки клієнтського застосунку, який і надаватиме користувачеві засоби для виконання дій, перелічених у завданні на проектування програмного забезпечення.

Настільний клієнтський застосунок (фронтенд) розробимо на основі технології Windows Presentation Foundation (WPF). Технологія WPF є частиною екосистеми платформи .NET і призначена для побудови графічних користувацьких інтерфейсів.

При створенні застарілих застосунків на основі Windows Forms за промальовування елементів управління і графіки (вікон) відповідали такі частини Windows, як User32 і GDI+. На відміну від них, застосунки WPF використовують бібліотеку DirectX, яка дозволяє працювати із графікою значно ефективніше, і до того ж може використовувати всі можливості сучасних відеоадаптерів. Завдяки WPF значна частина роботи по відображенні графіки, як найпростіших кнопок, так і складних 3D-моделей, передається на графічний процесор відеокарти, що також дозволяє використовувати апаратне прискорення графіки.

Однією з важливих особливостей WPF є використання мови декларативної розмітки інтерфейсу XAML (eXtensible Application Markup Language), заснованої на XML. Багатий графічний інтерфейс можна створювати через декларативне оголошення, або у коді мовами C# чи VB.NET, або поєднувати обидва способи. Використання XAML у певній мірі нагадує застосування HTML для формування вмісту web-сторінок. Це, порівняно з технологією Windows Forms, значно розширює можливості формування візуального контенту. При цьому мову C# використовують для задання активної частини програми та реалізації її поведінки (зокрема, й реакції на зміну стану описаних за допомогою XAML елементів інтерфейсу). Наприклад, за допомогою XAML можемо розмістити у вікні кнопку та описати її вигляд. Але дії, які виконуватимуться після її натискання, розміщують у файлі C#.

2.6.1. Створення проекту WPF Application

Щоб додати до існуючого рішення Microsoft Visual Studio новий проект WPF-застосунку, слід вибрати команду меню File/New/Project, і вказати шаблон WPF Application (рис. 12) з мовою програмування C#. Далі потрібно вказати назву (у прикладі з електронного навчального курсу назва проекту – WPFClientApp) та вибрати платформу .NET 5.0.

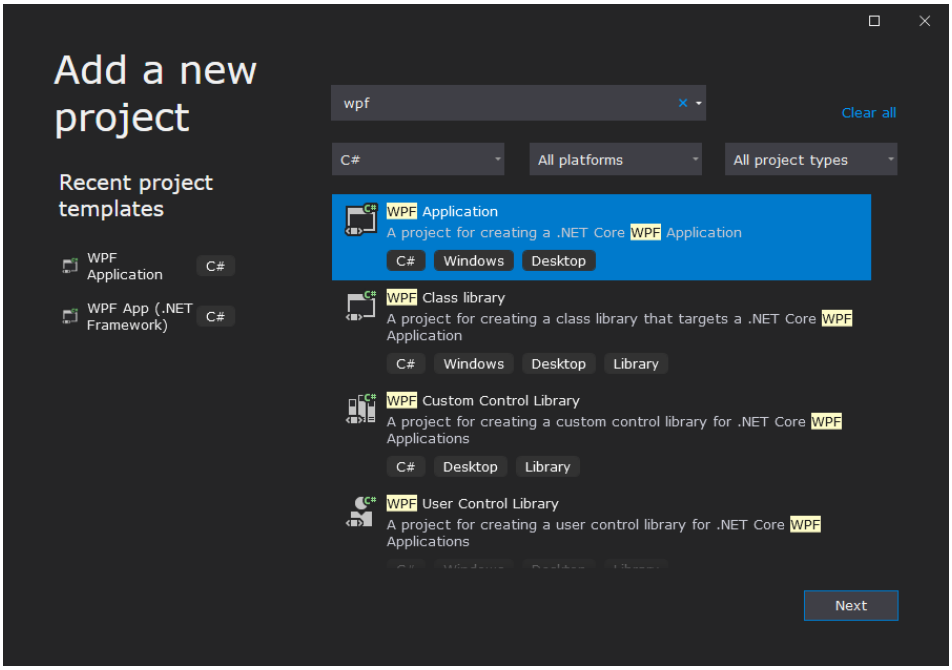


Рис. 12. Створення проекту WPF Application у середовищі MS Visual Studio

Клієнтський застосунок використовуватиме попередньо розроблену бібліотеку DataAccessLibrary. Тому її проект слід додати у посилання для проекту клієнтського застосунку (так само як і для проекту модульних тестів). Для цього у вікні Solution Explorer з контекстного меню проекту слід вибрати команду Add/Project Reference, і у вікні Reference Manager поставити маркер біля проекту DataAccessLibrary (рис. 13).

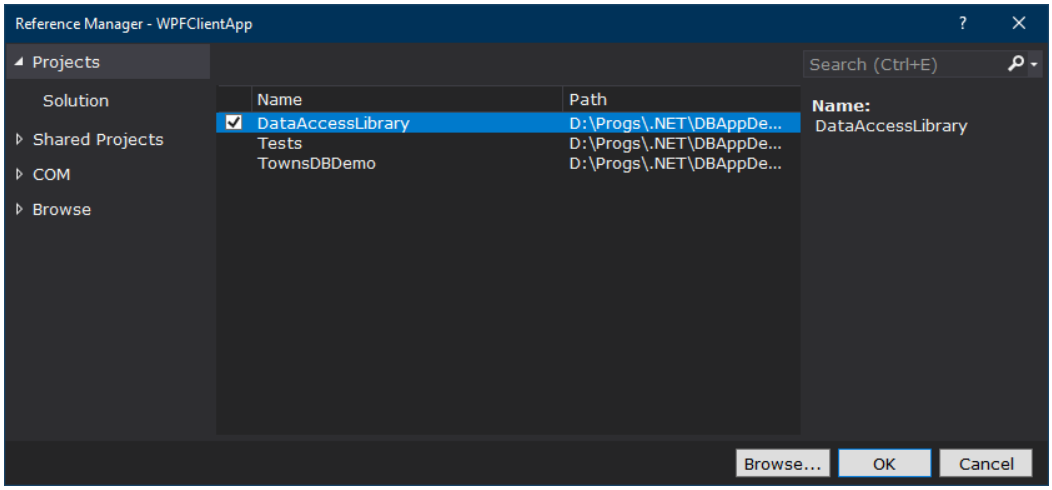


Рис. 13. Створення проекту WPF Application у середовищі MS Visual Studio

В середовищі Microsoft Visual Studio буде створене порожнє WPF-вікно (рис. 14). Зліва зверху у дизайнері відображене само вікно, а під ним – його опис мовою XAML. Код розмітки, який відповідатиме елементам вікна, слід вводити безпосередньо в редакторі XAML.

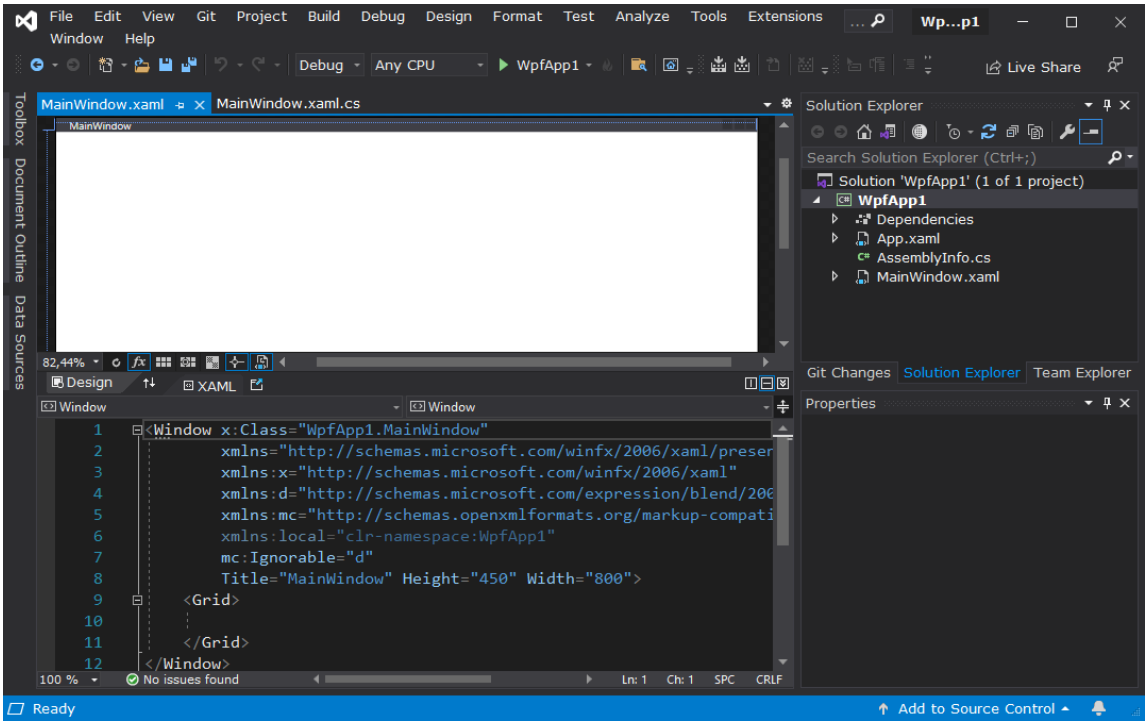


Рис. 14. Новостворене вікно WPF та його розмітка XAML у середовищі MS Visual Studio

2.6.2. Проектування головного вікна застосунку

Інтерфейс користувача визначає загальний вигляд вікон та розміщення і використання елементів керування, за допомогою яких користувач взаємодіє із застосунком. Ключовим елементом інтерфейсу нашого застосунку буде головне вікно, яке міститиме елементи керування для виконання всіх операцій згідно завдання.

2.6.3. Проектування структури головного вікна

Сформуємо головне вікно застосунку за типовою схемою, широко розповсюдженою серед різноманітного програмного забезпечення. Воно міститиме: смугу меню, панель інструментів, область з даними, і рядок стану (рис. 15).

Рис. 15. Структура головного вікна

Таку структуру використовують дуже часто, і вона забезпечує зручну взаємодію користувача із застосунком та швидкий доступ до його функціоналу. Команди панелі інструментів та меню зазвичай дублюють. Меню містить зручний для сприйняття тестовий перелік команд, а панель інструментів надає швидкий доступ до них за допомогою набору кнопок (та інших елементів керування) з піктограмами. Область даних міститиме таблицю з введеними даними про об'єкти "місто". Для подання даних у табличному виді використаємо елемент керування DataGrid.

Панель інструментів забезпечуватиме засоби для виконання всіх операцій згідно завдання (додавання, редагування та видалення даних, їх збереження та читання, фільтрування, сортування і пошук). Панель інструментів сформуємо за допомогою елемента керування ToolBarTray.

Меню міститиме доступ до всіх операцій, які реалізує панель інструментів, крім сортування та пошуку. Меню сформуємо за допомогою елемента керування Menu.

Рядок стану міститиме додаткову інформацію: назву відкритого файлу, а також додаткові дані про вибраний в області даних об'єкт переліку. Рядок стану сформуємо за допомогою елемента керування StatusBar.

Деякі операції, які виконуватиме застосунок, потребують додаткового введення даних. Так, для додавання (чи редагування) запису про об'єкт "місто" потрібно ввести значення його властивостей. Це завдання виконаємо за допомогою додаткового вікна, структура якого відповідатиме переліку

властивостей об'єкта. Детальніше його розглянемо пізніше, під час реалізації відповідних операцій.

2.6.4. Проектування вигляду головного вікна

Кінцевий вигляд головного вікна застосунку, над яким будемо працювати, показано на рис. 16. Розглянемо проектування головного вікна детально.

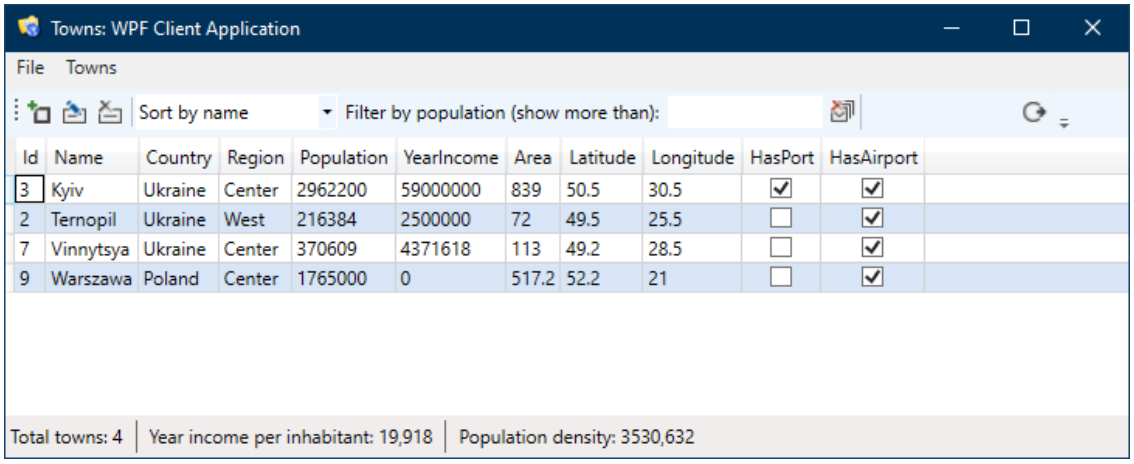


Рис. 16. Головне вікно застосунку

Як контейнер для розміщення елементів вікна використаємо панель DockPanel (лістинг 9). Її внутрішні елементи прив'язуються до одного з чотирьох країв панелі.

Лістинг 9

```
<DockPanel x:Name="MainPanel"
    ScrollViewer.CanContentScroll="True"
    ScrollViewer.HorizontalScrollBarVisibility="Visible"
    Margin="0,0,0,0" HorizontalAlignment="Stretch"
    VerticalAlignment="Stretch" Height="Auto">
...
</DockPanel>
```

2.6.4.1. Проектування головного меню

Головне меню призначене для виклику операцій відповідно до завдання (див. п.1). Код XAML-розмітки, яка описує команди головного меню приведено у лістингу 10.

Лістинг 10

```
<Menu DockPanel.Dock="Top" Height="25" VerticalAlignment="Top">
    <MenuItem x:Name="mniFile" Header="File">
```

```
        <MenuItem x:Name="mniSaveCsv" Header="Save As CSV"
            Click="mniSaveCsv_Click">
            <MenuItem.Icon>
                <Image Source="Resources\SaveAs_16x.png"/>
            </MenuItem.Icon>
        </MenuItem>
        <MenuItem x:Name="mniAbout" Header="About"
            Click="mniAbout_Click">
            <MenuItem.Icon>
                <Image
                   Source="Resources\InformationSymbol_16x.png"/>
            </MenuItem.Icon>
        </MenuItem>
        <Separator />
        <MenuItem x:Name="mniExit" Header="Exit"
            Click="BtnExit_Click">
            <MenuItem.Icon>
                <Image Source="Resources\Exit_16x.png"/>
            </MenuItem.Icon>
        </MenuItem>
    </MenuItem>
    <MenuItem x:Name="mniTowns" Header="Towns" >
        <MenuItem x:Name="mniAddTown" Header="Add new town"
            Click="BtnAddTown_Click">
            <MenuItem.Icon>
                <Image Source="Resources\AddItem_16x.png"/>
            </MenuItem.Icon>
        </MenuItem>
        <MenuItem x:Name="mniEditTown"
            Header="Edit selected town"
            Click="BtnEditTown_Click">
            <MenuItem.Icon>
                <Image
                    Source="Resources\UpdateListItem_16x.png"/>
            </MenuItem.Icon>
        </MenuItem>
        <Separator/>
        <MenuItem x:Name="mniDeleteTown" Header="Delete town"
            Click="BtnDeleteTown_Click">
            <MenuItem.Icon>
                <Image
                    Source="Resources\DeleteListItem_16x.png"/>
            </MenuItem.Icon>
        </MenuItem>
        <Separator/>
        <MenuItem x:Name="mniClearFilter" Header="Clear filter"
            Click="BtnClearFilter_Click">
            <MenuItem.Icon>
                <Image
                   Source="Resources\ClearMessageQueue_16x.png"/>
            </MenuItem.Icon>
```

```
        </MenuItem>
    </MenuItem>
</Menu>
```

Важливо для елемента розмітки Menu задати атрибут DockPanel.Dock="Top". Це забезпечить його розміщення біля верхнього краю контейнера DockPanel, у якому меню описане.

За допомогою тегів MenuItem описують елементи меню. Атрибути всередині них задають властивості відповідних елементів: назву (Name), текст (Header) тощо.

Щоб задати піктограму для елемента меню, використовують вкладені елементи XAML MenuItem.Icon. Їх атрибути Source вказують на графічний файл з піктограмою (такі файли слід додати у ресурси проекту). Для піктограм можна використати будь-який доречний графічний файл відповідного розміру. Велику безкоштовну бібліотеку піктограм можна також завантажити з сайту Microsoft в розділі Visual Studio Image Library.

Щоб додати у меню розділювач, використовують елемент розмітки Separator.

Зауважимо, що код розмітки XAML містить фінальний опис меню після завершення розробки. Тому для елементів меню задано також атрибути Click. Такий атрибут містить назву метода (його описують мовою C#), який виконуватиметься, якщо користувач вибере даний елемент меню під час роботи. На даному етапі розробки методи ще не створені, тому атрибути Click у розмітці поки-що не потрібно задавати. В іншому випадку проект неможливо буде скомпілювати та запустити. Тому спочатку слід ввести статичні елементи розмітки меню, а атрибути, пов'язані з динамікою вікна, вводити після розробки відповідних методів. У лістингу 10 і наступних ділянки розмітки, які посилаються на ще не відсутні методи, виділено синім.

Проміжна перевірка

Щоб перевірити правильність розробленого меню, після його розробки слід запустити застосунок і пересвідчитись, що меню має такий вигляд, як передбачено ідеєю розробки.

2.6.4.2. Проектування панелі інструментів

Панель інструментів (див. рис. 16) містить інструменти для швидкого виконання ряду дій, передбачених завданням. Код XAML панелі інструментів приведено у лістингу 11. Текст розмітки слід розмістити відразу після розмітки головного меню, в межах панелі DockPanel. Панель керування містить такі елементи:

- кнопка (елемент розмітки Button) "Add New Town" для додання нового запису про місто;
- кнопка "Edit Town" для редагування вибраного запису про місто;
- кнопка "Delete Town" для видалення вибраного запису про місто;
- розділювач (елемент розмітки Separator);
- випадний список (ComboBox) з переліком параметрів сортування. Цей елемент розмітки містить внутрішні (ComboBoxItem), якими задано елементи самого випадного списку;
- мітка (Label) – статичний текстовий напис ("Filter by population (show more than):") з описом призначення наступного елемента керування;
- поле текстового вводу TextBox для введення мінімальної кількості населення міста, яка використовується для фільтрування записів у таблиці;
- кнопка "Clear Filter" для очищення значення фільтрування у попередньому полі вводу;
- розділювач, який відділяє розглянуті елементи керування від останньої кнопки для виходу з програми;
- кнопка "Exit" для завершення роботи програми.

Лістинг 11

```
<ToolBarTray DockPanel.Dock="Top" Height="30"
    VerticalAlignment="Top" Margin="0,0,0,0" Focusable="False"
    KeyboardNavigation.IsTabStop="False">
    <ToolBar HorizontalAlignment="Left" Height="28"
        Margin="0,0,0,0" VerticalAlignment="Top">
        <Button x:Name="BtnAddTown" ToolTip="Add New Town"
            Click="BtnAddTown_Click">
            <Image Source="Resources\AddItem_16x.png" />
        </Button>
        <Button x:Name="BtnEditTown" ToolTip="Edit Town"
            Click="BtnEditTown_Click">
            <Image Source="Resources\UpdateListItem_16x.png" />
        </Button>
        <Button x:Name="BtnDeleteTown" ToolTip="Delete Town"
            Click="BtnDeleteTown_Click">
            <Image Source="Resources\DeleteListItem_16x.png" />
```

```
        </Button>
        <Separator />
        <ComboBox x:Name="cbSortBy" Height="23" Margin="0"
            VerticalAlignment="Top" Width="130" ToolTip="Sort by"
            SelectionChanged="CbSortBy_SelectionChanged">
            <ComboBoxItem Content="Sort by name"/>
            <ComboBoxItem Content="Sort by country"/>
            <ComboBoxItem Content="Sort by region"/>
            <ComboBoxItem Content="Sort by population"/>
            <ComboBoxItem Content="Sort by area"/>
        </ComboBox>
        <Label Margin="0 -2 0 0">
           Filter by population (show more than):
        </Label>
        <TextBox x:Name="txbFilter" Width="100"
            TextChanged="txbFilter_TextChanged">
        </TextBox>
        <Button x:Name="BtnClearFilter" ToolTip="Clear Filter"
            Click="BtnClearFilter_Click">
            <Image Source="Resources\ClearMessageQueue_16x.png"/>
        </Button>
        <Separator Margin="0 0 100 0" />
        <Button x:Name="BtnExit" ToolTip="Exit"
            Click="BtnExit_Click">
            <Image Source="Resources\Exit_16x.png" />
        </Button>
    </ToolBar>
</ToolBarTray>
```

Для елемента розмітки ToolBarTray задано атрибут DockPanel.Dock="Top", що забезпечує його розміщення біля верхнього краю контейнера DockPanel, після меню.

Також слід звернути увагу, що лістинг 11 відображує розмітку завершеної панелі інструментів. Так, для кнопок задано атрибути Click, для випадного списку – атрибут SelectionChanged, а для текстового поля – атрибут TextChanged. Всі вони прив'язують до певної події, пов'язаної з елементом керування, метод, що викликатиметься при настанні цієї події.

На даному етапі розробки згадані методи ще не створені, тому відповідні атрибути у розмітці поки-що не потрібно задавати, бо проект буде неможливо скомпілювати та запустити. Тому спочатку слід ввести статичні елементи розмітки меню, а атрибути, пов'язані з динамікою вікна, вводити після розробки відповідних методів.

Проміжна перевірка

Щоб перевірити правильність введеної розмітки, після її введення слід запустити застосунок і пересвідчитись, що панель інструментів і вікно в цілому мають потрібний вигляд.

2.6.4.3. Таблиця для відображення даних

Для відображення даних з таблиці БД використаємо елемент розмітки DataGrid (лістинг 12). У ньому налаштовано цілий ряд атрибутів, які задають різні аспекти поведінки та зовнішнього вигляду. Розглянемо найважливіші із них:

- IsReadOnly="True" – забороняє редагування даних безпосередньо у таблиці;
- CanUserAddRows="false" – забороняє додавати новий рядок безпосередньо у таблиці;
- Focusable="True" – дозволяє елементу керування отримувати фокус введення;
- IsTabStop="True" – дозволяє активувати елемент використанням клавіші Tab;
- AlternationCount="2" AlternatingRowBackground="#FFD7E5F7" – разом ці атрибути зафарбовують кожен другий рядок заданим кольором.

Лістинг 12

```
<StackPanel Width="Auto" Height="Auto" DockPanel.Dock="Top">
<DataGrid x:Name="TownsGrid" Margin="0 0 0 0" IsReadOnly="True"
    CanUserAddRows="false" Focusable="True"
    IsTabStop="True" AlternationCount="2"
    AlternatingRowBackground="#FFD7E5F7"
    HorizontalGridLinesBrush="LightGray"
    VerticalGridLinesBrush="LightGray"
    SelectionChanged="TownsGrid_SelectionChanged"
    BorderThickness="0,0,0,0">
    <DataGrid.Resources>
        <Style TargetType="DataGridRow">
            <EventSetter Event="MouseDoubleClick"
                Handler="BtnEditTown_Click"/>
        </Style>
    </DataGrid.Resources>
    <DataGrid.ContextMenu>
        <ContextMenu>
            <MenuItem x:Name="miAddTown" Header="Add new town"
                Click="BtnAddTown_Click">
                <MenuItem.Icon>
                    <Image Source="Resources\AddItem_16x.png"/>
                </MenuItem.Icon>
            </MenuItem>
            <MenuItem x:Name="miEditTown"
```

```
                Header="Edit selected town"
                Click="BtnEditTown_Click">
                <MenuItem.Icon>
                    <Image
                    Source="Resources\UpdateListItem_16x.png"/>
                </MenuItem.Icon>
            </MenuItem>
            <Separator/>
            <MenuItem x:Name="miDeleteTown" Header="Delete town"
                Click="BtnDeleteTown_Click">
                <MenuItem.Icon>
                    <Image
                    Source="Resources\DeleteListItem_16x.png"/>
                </MenuItem.Icon>
            </MenuItem>
            <Separator/>
            <MenuItem x:Name="miClearFilter"
                Header="Clear filter"
                Click="BtnClearFilter_Click">
                <MenuItem.Icon>
                    <Image
                   Source="Resources\ClearMessageQueue_16x.png"/>
                </MenuItem.Icon>
            </MenuItem>
        </ContextMenu>
    </DataGrid.ContextMenu>
</DataGrid>
</StackPanel>
```

Таблицю розміщено всередині панелі StackPanel. Це забезпечує її автоматичне розтягування по вертикалі на всю доступну у вікні висоту. Для панелі слід задати прив'язування до верхнього краю (DockPanel.Dock="Top") і автоматичне розтягування по вертикалі та горизонталі Width="Auto" Height="Auto".

Лістинг 12 описує DataGrid у її фінальному вигляді, тому містить посилання на методи, які на даному етапі розробки ще відсутні. Один з таких методів використовуєтсья атрибутом SelectionChanged елемента DataGrid, інший описаний внутрішнім елементом розмітки DataGrid.Resources. Тому як згаданий атрибут, так і весь код елемента DataGrid.Resources поки-що має бути відсутнім.

Для швидкого виклику операцій, пов'язаних із модифікацією вмісту таблиці даних, використано контекстне меню. Таке меню задають за допомогою внутрішнього атрибута DataGrid.ContextMenu. Елементи меню описують за допомогою елементів XAML MenuItem, так само як і для раніше розглянутого головного меню вікна (лістинг 10). Атрибути кожного елемента меню, які

посилаються на методи – оброблювачі події Click, поки-що також слід пропустити.

2.6.4.4. Розробка рядка стану

Для відображення додаткової інформації щодо вибраного у таблиці запису використаємо елемент розмітки StatusBar (лістинг 13). Він містить кілька елементів зі статичним текстом TextBlock, розділених за допомогою елементів Separator. Щоб прив'язати рядок стану до нижнього краю вікна, слід задати його атрибут DockPanel.Dock="Bottom".

Лістинг 13

```
<StatusBar DockPanel.Dock="Bottom" Height="27"
    VerticalAlignment="Bottom">
    <StatusBarItem>
        <TextBlock x:Name="lblTotalTowns" Text="Total Towns:" />
    </StatusBarItem>
    <Separator Grid.Column="1" Margin="5 3 5 3" />
    <StatusBarItem>
        <TextBlock x:Name="lblYearIncome" Text="Year Income:" />
    </StatusBarItem>
    <Separator Grid.Column="1" Margin="5 3 5 3" />
    <StatusBarItem>
        <TextBlock x:Name="lblPopulationDensity"
            Text="Population Density:" />
    </StatusBarItem>
</StatusBar>
```

Рядок стану містить три панелі з текстовими блоками:

- lblTotalTowns – для відображення кількості міст;
- lblYearIncome – для відображення значення річного бюджету на мешканця;
- lblPopulationDensity – для відображення щільності населення.

На цьому дизайн вигляду головного вікна застосунку завершено.

Проміжна перевірка

Щоб перевірити правильність введеної розмітки, слід запустити застосунок у середовищі Microsoft Visual Studio та пересвідчитись, що вікно в цілому має потрібний вигляд.

2.6.5. Проектування вікна для введення даних про місто

Для введення та редагування даних про об’єкт "місто" потрібно передбачити засіб користувацького інтерфейсу для введення/виведення даних про властивості об'єкта (назва, країна, регіон тощо). Тому спроектуємо ще одне вікно, яке матиме поля, що відповідають структурі властивостей для даних про місто (рис. 17).

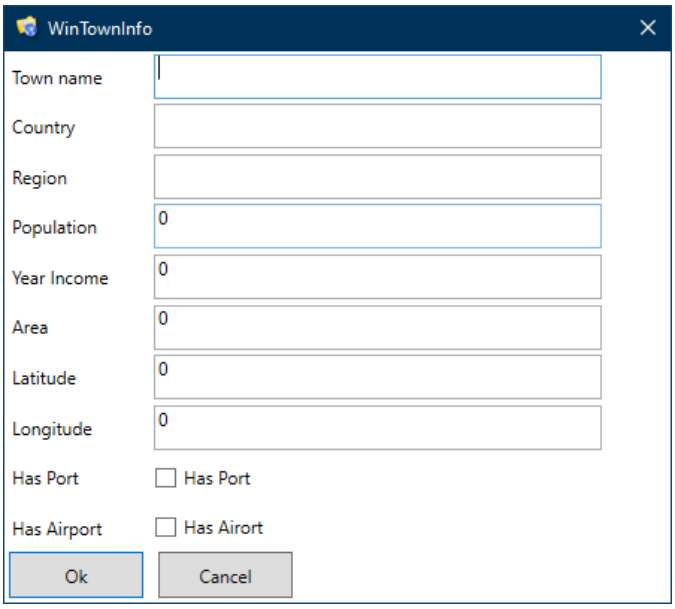


Рис. 17. Загальний вигляд вікна для введення інформації про місто

2.6.5.1. Розробка вигляду вікна

Щоб додати до проекту ще одну форму WPF, слід з контекстного меню проекта WPFClientApp у вікні Solution Explorer викликати команду Add/ Window (WPF)… Далі потрібно задати назву вікна (у нашому випадку це WinTownInfo).

Розмітку XAML вікна у її кінцевому вигляді показано у лістингу 14. Для компонування елементів керування у вікні використано сітку Grid, в якій за допомогою внутрішніх елеметів Grid.ColumnDefinitions та Grid.RowDefinitions задано 2 стовпця і 11 рядків.

Лістинг 14

```
<Window x:Class="WPFClientApp.WinTownInfo"
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:WPFClientApp"
mc:Ignorable="d"
```

```
Title="WinTownInfo" Height="408" Width="462" MinHeight="300"
WindowStartupLocation="CenterScreen" ResizeMode="NoResize"
FocusManager.FocusedElement="{Binding ElementName=txbName}">
    <Grid>
        <Grid.ColumnDefinitions>
            <ColumnDefinition Width="100" />
            <ColumnDefinition Width="300" />
        </Grid.ColumnDefinitions>
        <Grid.RowDefinitions>
            <RowDefinition MaxHeight="35" Height="35*" />
            <RowDefinition MaxHeight="35" Height="35*" />
            <RowDefinition MaxHeight="35" Height="35*" />
            <RowDefinition MaxHeight="35" Height="35*" />
            <RowDefinition MaxHeight="35" Height="35*" />
            <RowDefinition MaxHeight="35" Height="35*" />
            <RowDefinition MaxHeight="35" Height="35*" />
            <RowDefinition MaxHeight="35" Height="35*" />
            <RowDefinition MaxHeight="35" Height="35*" />
            <RowDefinition MaxHeight="35" Height="35*" />
            <RowDefinition MaxHeight="35" Height="35*" />
        </Grid.RowDefinitions>
        <Label x:Name="lName" Grid.Column="0"
            VerticalAlignment="Center">Town name</Label>
        <Label x:Name="lCountry" Grid.Column="0" Grid.Row="1"
            VerticalAlignment="Center">Country</Label>
        <Label x:Name="lRegion" Grid.Column="0" Grid.Row="2"
            VerticalAlignment="Center">Region</Label>
        <Label x:Name="lPopulation" Grid.Column="0" Grid.Row="3"
            VerticalAlignment="Center">Population</Label>
        <Label x:Name="lYearIncome" Grid.Column="0" Grid.Row="4"
            VerticalAlignment="Center">Year Income</Label>
        <Label x:Name="lArea" Grid.Column="0" Grid.Row="5"
            VerticalAlignment="Center">Area</Label>
        <Label x:Name="lLatitude" Grid.Column="0" Grid.Row="6"
            VerticalAlignment="Center">Latitude</Label>
        <Label x:Name="lLongitude" Grid.Column="0" Grid.Row="7"
            VerticalAlignment="Center">Longitude</Label>
        <Label x:Name="lHasPort" Grid.Column="0" Grid.Row="8"
            VerticalAlignment="Center">Has Port</Label>
        <Label x:Name="lHasAirport" Grid.Column="0" Grid.Row="9"
            VerticalAlignment="Center">Has Airport</Label>

        <TextBox x:Name="txbName" Grid.Column="1" Grid.Row="0"
            MaxHeight="30" >Ternopil</TextBox>
        <TextBox x:Name="txbCountry" Grid.Column="1" Grid.Row="1"
            MaxHeight="30"></TextBox>
        <TextBox x:Name="txbRegion" Grid.Column="1" Grid.Row="2"
            MaxHeight="30"></TextBox>
        <TextBox x:Name="txbPopulation" Grid.Column="1"
            Grid.Row="3" MaxHeight="30"></TextBox>
        <TextBox x:Name="txbYearIncome" Grid.Column="1"
```

```
            Grid.Row="4" MaxHeight="30"></TextBox>
        <TextBox x:Name="txbArea" Grid.Column="1" Grid.Row="5"
            MaxHeight="30"></TextBox>
        <TextBox x:Name="txbLatitude" Grid.Column="1"
            Grid.Row="6" MaxHeight="30"></TextBox>
        <TextBox x:Name="txbLongitude" Grid.Column="1"
            Grid.Row="7" MaxHeight="30"></TextBox>

        <CheckBox x:Name="chbHasPort" Grid.Column="1"
            Grid.Row="8" VerticalAlignment="Center">
            Has Port
        </CheckBox>
        <CheckBox x:Name="chbHasAirport" Grid.Column="1"
            Grid.Row="9" VerticalAlignment="Center">
            Has Airort
        </CheckBox>

        <Button x:Name="btnOk" Grid.Column="0" Grid.Row="10"
            MinWidth="90" MaxWidth="150"
            HorizontalAlignment="Left" IsDefault="True"
            Margin="3 0 0 3" Click="BtnOk_Click" >Ok</Button>
        <Button x:Name="btnCancel" Grid.Column="1" Grid.Row="10"
            MinWidth="90" MaxWidth="150"
            HorizontalAlignment="Left" IsCancel="True"
            Margin="3 0 0 3" >Cancel</Button>
    </Grid>
</Window>
```

У першому (лівому) стовпці таблиці Grid розміщено текстові мітки Label, які описують призначення розташованих після них елементів керування (для кожної мітки задано Grid.Column="0").

У другому (правому) стовпці таблиці Grid розміщено елементи керування, за допомогою яких можна відобразити чи задати значення полів об'єкта. Для значень, які передбачають введення текстових чи числових даних з клавіатури (назва міста, країна тощо), використано елементи розмітки TextBox. Для полів логічного типу (HasPort, HasAirport) використано елементи CheckBox. Всі ці елементи розмітки мають атрибут Grid.Column="1".

У нижній частині вікна розміщено дві кнопки Button. Перша з них (кнопка Ok) призначена для підтвердження введених даних, друга (Cancel) – для скасування операції. Для кнопки Ok задано атрибут IsDefault="True", який встановлює її як кнопку за замовчуванням. Така кнопка автоматично спрацьовує, коли користувач у вікні натисне клавішу Enter. Кнопка Cancel містить атрибут

IsCancel="True", який спричиняє автоматичне спрацювання кнопки при натисканні клавіші Esc (і закриття вікна).

Кнопка Ok містить також атрибут Click з посиланням на поки-що не розроблений метод. Цей атрибут поки-що не потрібно вводити.

2.6.5.2. Програмування поведінки вікна

Вікно WinTownInfo будемо використовувати для введення даних про нове місто, а також для редагування даних про вже існуюче місто. При створенні у вікно передаватимемо екземпляр класу Town, у який записуватимемо введені користувачем дані. При редагуванні через цей екземпляр будемо передавати дані, які слід відобразити у вікні.

Код у файлі WinTownInfo.xaml.cs використовує такі простори імен:

```
using System;
using System.Windows;
using DataAccessLibrary;
```

Якщо якогось із них немає, його потрібно додати. Простір імен DataAccessLibrary використовується, бо у ньому оголошено клас Town.

До класу форми додамо поле TownInfo класу Town (лістинг 9)[3]:

Лістинг 15

```
public partial class WinTownInfo : Window
{
    public Town TownInfo;
```

Конструктор класу перепишемо згідно лістингу 16. У конструктор передається посилання на екземпляр класу Town, яке запам’ятовується у щойно описаному полі TownInfo. При створенні вікна дані про передане місто слід відобразити у полях форми. Кожному полю присвоюється відповідне значення об’єкта TownInfo. Елементи керування txbName, txbCountry та інші використані в лістингу 16, задані у розмітці вікна (лістинг 14, див. атрибути x:Name елементів розмітки).

Лістинг 16

```
public WinTownInfo(Town town)
```

[3] Нагадаємо: у лістингах виділений сірим код має бути згенерований середовищем Microsoft Visual Studio. Його не слід вводити вручну.

```
{
    InitializeComponent();

    TownInfo = town;

    txbName.Text = town.Name;
    txbCountry.Text = town.Country;
    txbRegion.Text = town.Region;
    txbPopulation.Text = town.Population.ToString();
    txbYearIncome.Text = town.YearIncome.ToString();
    txbArea.Text = town.Area.ToString();
    txbLatitude.Text = town.Latitude.ToString();
    txbLongitude.Text = town.Longitude.ToString();
    chbHasPort.IsChecked = town.HasPort;
    chbHasAirport.IsChecked = town.HasAirport;
}
```

При натисканні на кнопку "Ok" у вікні (рис. 17) слід перевірити введені дані і записати їх в об’єкт TownInfo. Ці операції реалізовано в приватному методі CheckTownInfo (лістинг 17). Перевірку правильності вводу числових значень виконуємо за допомогою методу TryParse() відповідного класу. Цей метод робить спробу перетворення рядка у числовий тип даних та повертає true у випадку успішного перетворення і false – при неможливості перетворення. В разі успіху результат повертається у вихідному параметрі методу.

Якщо введено помилкові дані, то за допомогою класу MessageBox показуємо користувачеві повідомлення про помилку, активуємо елемент керування з помилковими даними (за допомогою його методу Focus), і припиняємо роботу методу, повертаючи значення false.

Лістинг 17

```
private bool CheckTownInfo()
{
    int population;
    double yearIncome, area, latitude, longitude;
    if (!Int32.TryParse(txbPopulation.Text, out population))
    {
        MessageBox.Show("Incorrect population value!",
        "Incorrect value", MessageBoxButton.OK,
        MessageBoxImage.Warning);
        txbPopulation.Focus();
        return false;
    }
    if (!Double.TryParse(txbYearIncome.Text, out yearIncome))
    {
        MessageBox.Show("Incorrect year income value!",
        "Incorrect value", MessageBoxButton.OK,
```

```
            MessageBoxImage.Warning);
            txbYearIncome.Focus();
            return false;
        }
        if (!Double.TryParse(txbArea.Text, out area))
        {
            MessageBox.Show("Incorrect area value!",
            "Incorrect value", MessageBoxButton.OK,
            MessageBoxImage.Warning);
            txbArea.Focus();
            return false;
        }
        if (!Double.TryParse(txbLatitude.Text, out latitude))
        {
            MessageBox.Show("Incorrect latitude value!",
            "Incorrect value", MessageBoxButton.OK,
            MessageBoxImage.Warning);
            txbLatitude.Focus();
            return false;
        }
        if (!Double.TryParse(txbLongitude.Text, out longitude))
        {
            MessageBox.Show("Incorrect longitude value!",
            "Incorrect value", MessageBoxButton.OK,
            MessageBoxImage.Warning);
            txbLongitude.Focus();
            return false;
        }

        TownInfo = new(
            id: TownInfo.Id,
            name: txbName.Text,
            country: txbCountry.Text,
            region: txbRegion.Text,
            population: population,
            yearIncome: yearIncome,
            area: area,
            latitude: latitude,
            longitude: longitude,
            hasPort: chbHasPort.IsChecked.Value,
            hasAirport: chbHasAirport.IsChecked.Value
        );

        return true;
    }
```

Після всіх перевірок на основі введених у вікні даних створюється екземпляр класу Town, який присвоюється полю класу TownInfo.

Оброблювач події Click кнопки btnOk просто викликає метод CheckTownInfo для перевірки введених значень, і якщо він поверне true, присвоюємо властивості DialogResult вікна значення true. Такий результат означає підтвердження даних користувачем, а вікно при цьому автоматично закриється.

Лістинг 18

```
private void BtnOk_Click(object sender, RoutedEventArgs e)
{
    if (CheckTownInfo()) DialogResult = true;
}
```

Таким чином, дізнатися, чи користувач підтвердив введені дані, можна за допомогою результату діалогового вікна. Якщо він дорівнює true, то користувач ввів дані і схвалює продовження операції (додавання запису чи його редагування). Значення false вказує на те, що операцію слід скасувати.

Після того, як метод – оброблювач події Click для кнопки Ok створено, у розмітці XAML цієї кнопки потрібно додати елемент Click="BtnOk_Click" (у лістингу 14 він виділений синім). Назва методу (тут – BtnOk_Click має бути така ж, як при його створенні (лістинг 18)).

На цьому розробка допоміжного вікна для введення та відображення даних про об'єкт "місто" повністю завершена. Тому повернемось до розробки головного вікна застосунку, а саме – до програмування операцій, які мають бути реалізовані відповідно до завдання.

2.6.6. Програмування операцій головного вікна

Для програмування поведінки головного вікна застосунку будемо вносити зміни у файл MainWindow.xaml.cs. В подальшому коді використовуватимуться сутності з таких просторів імен:

```
using System;
using System.IO;
using System.Linq;
using System.Windows;
using System.Windows.Controls;
using System.Windows.Input;
using Microsoft.Win32;
using DataAccessLibrary;
```

```
using System.Text;
```

Тому слід переконатись, що всі вони перелічені на початку. Простір імен DataAccessLibrary містить класи шару доступу до даних (DAL), розробка якого описана перед цим (див. п.2.4).

2.6.6.1. Ініціалізація головного вікна

Під час запуску застосунку слід виконати ряд підготовчих дій, головна з яких – читання інформації про міста з бази даних та відображення їх у таблиці головного вікна. Спочатку в класі головного вікна слід оголосити два приватних поля:

```
private readonly TownDAL DAL = new();
private int TotalTowns;
```

Через приватне поле з назвою DAL класу TownDAL матимемо доступ до раніше розробленого функціоналу рівня доступу до даних. TotalTowns просто міститиме кількість прочитаних з БД міст.

Головне вікно застосунку зручно ініціалізувати в конструкторі його класу (лістинг 19). У автоматично згенерованому коді конструктора вже присутній виклик метода InitializeComponent. Він завантажує файл розмітки вікна XAML та формує вікно відповідно до розмітки.

Лістинг 19

```
public MainWindow()
{
    InitializeComponent();

    TotalTowns = DAL.ReadTowns();
    TownsGrid.ItemsSource = DAL.TownsBuffer;
    cbSortBy.SelectedIndex = 0;
}
```

Першим рядком після цього викликаємо метод ReadTowns бібліотеки доступу до даних, який читає перелік всіх наявних міст із БД, розміщує їх у буферному списку TownsBuffer, і повертає кількість прочитаних записів.

Далі як джерело даних для таблиці DataGrid встановлюється список TownsBuffer. При цьому внутрішній механізм класу DataGrid автоматично завантажить дані і відобразить їх.

Остання дія у лістингу 19 – встановлення порядку сортування за замовчуванням, який відповідає першому елементу зі списку cbSortBy.

Проміжна перевірка

Запустіть застосунок, натиснувши клавішу F5. Якщо попередні дії виконані вірно, має з'явитися головне вікно з таблицею. Таблиця може бути порожньою, або містити запис (-и) про місто, яке може бути додане при виконанні тестів. Щоб повернутися до середовища Visual Studio для подальшої роботи над проектом, слід закрити застосунок. Для цього слід скористатися кнопкою з хрестиком у правому верхньому кутку – адже вихід з програми все ще не запрограмовано.

2.6.6.2. Внесення нових даних про місто

Операцію додання даних про нове місто реалізуємо при натисканні кнопки BtnAdd на панелі інструментів головного вікна (в оброблювачі події Click кнопки). Загальна послідовність дій при цьому така (лістинг 20).

Спочатку створюємо вікно wTownInfo класу WinTownInfo (і передаємо йому посилання на новий об’єкт Town), показуємо вікно і чекаємо на його закриття. Далі аналізуємо результат вікна: якщо він дорівнює true (це означає, що в об’єкт TownInfo записано введені користувачем дані), то викликаємо метод InsertTown шару доступу до даних DAL, передавши йому об'єкт TownInfo вікна wTownInfo, який містить введену користувачем інформацію. Якщо метод InsertTown повертає додатне значення (відповідно до запрограмованої логіки), це свідчить про успішну операцію запису. Тому далі викликом методу DAL ReadTowns перечитуємо дані з БД і викликом метода RefreshGrid оновлюємо дані у таблиці. Якщо ж метод InsertTown поверне нуль чи від'ємне значення, відображується повідомлення про невдалу операцію.

Лістинг 20

```
private void BtnAddTown_Click(object sender, RoutedEventArgs e)
{
    WinTownInfo wTownInfo = new(new Town());
    if (wTownInfo.ShowDialog().Equals(true))
    {
        if (DAL.InsertTown(wTownInfo.TownInfo) > 0)
        {
            TotalTowns = DAL.ReadTowns();
            RefreshGrid();
        }
        else
        {
```

```
                MessageBox.Show("Record not insert!",
                "Error occured", MessageBoxButton.OK,
                MessageBoxImage.Error);
            }
        }
}

private void RefreshGrid()
{
    TownsGrid.ItemsSource = null;
    TownsGrid.ItemsSource = DAL.TownsBuffer;

    lblTotalTowns.Text = String.Format("Total towns: {0}",
        TotalTowns);
}
```

Метод RefreshGrid просто перевстановлює джерело даних для таблиці, а також записує у відповідне поле рядка стану кількість міст, які прочитані з таблиці БД.

Після того, як метод BtnAddTown_Click створено, посилання на нього слід додати у розмітку XAML для кнопки BtnAddTown (лістинг 11), елемента головного меню mniAddTown (лістинг 10), та елемента контекстного меню miAddTown (лістинг 12). У всіх приведених лістингах відповідні елементи розмітки виділені синім. Таким чином, операція додання нового міста може бути виконана різними способами. При цьому використовується один і той же метод класу.

Проміжна перевірка

Запустіть застосунок, натисніть на кнопку "Add New Town", введіть дані і підтвердіть їх. У головному вікні у таблиці, яка представлена компонентом DataGrid, має з’явитися запис з введеними даними. Додайте таким чином кілька записів і перевірте, чи правильно відображуються дані.

2.6.6.3. Зміна даних про місто

Для зміни даних про місто призначена кнопка "Edit Town" на панелі інструментів головного вікна. Загальна послідовність дій при цьому така (лістинг 21). Спочатку перевіряємо, чи є у джерелі даних якісь дані. Якщо даних немає, припиняємо роботу методу. Далі отримуємо посилання на виділений елемент таблиці та зберігаємо його дані у змінній town класу Town. При цьому відразу перевіряємо, чи ці дані можуть бути приведені до класу Town. Якщо ні (а

це буває у випадку, коли в таблиці не виділено ніяких записів), виводимо відповідне повідомлення.

Далі створюємо раніше розроблене вікно wTownInfo класу WinTownInfo (так само, як і при доданні нового запису), передавши в його конструктор попередньо прочитану змінну town з даними про місто у виділеному рядку таблиці. Після цього показуємо вікно (воно відображує всі поточні властивості town) і чекаємо на його закриття. Аналізуємо результат вікна: якщо він дорівнює true (це означає, що користувач підтвердив операцію зміни запису після редагування даних у вікні), викликаємо метод DAL UpdateTown, передавши йому об'єкт з новими даними про місто (вони представлені змінною TownInfo). В кінці, як і після додання нового запису, оновлюємо дані в таблиці.

Лістинг 21

```
private void BtnEditTown_Click(object sender, RoutedEventArgs e)
{
    if (DAL.TownsBuffer.Count == 0) return;
    if (TownsGrid.SelectedItem is not Town town)
    {
        MessageBox.Show("Datagrid row contains town data " +
            "have to be selected first.", "Please select town",
            MessageBoxButton.OK, MessageBoxImage.Information);
        return;
    }

    WinTownInfo wTownInfo = new(town);
    if (wTownInfo.ShowDialog().Equals(true))
    {
        if (DAL.UpdateTown(wTownInfo.TownInfo) > 0)
        {
            TotalTowns = DAL.ReadTowns();
            RefreshGrid();
        }
        else
        {
            MessageBox.Show("Record not updated!",
                "Error occured", MessageBoxButton.OK,
                MessageBoxImage.Error);
        }
    }
}
```

Після того, як метод BtnEditTown_Click створено, посилання на нього потрібно додати у код розмітки XAML кнопки BtnEditTown (лістинг 11), елемента головного меню mniEditTown (лістинг 10), та елемента контекстного меню

miEditTown (лістинг 12). Крім цього, щоб додати можливість редагування запису таблиці подвійним кліком на ньому, у розмітку таблиці даних DataGrid слід додати такий код (див. лістинг 12):

```
<DataGrid.Resources>
    <Style TargetType="DataGridRow">
        <EventSetter Event="MouseDoubleClick"
            Handler="BtnEditTown_Click"/>
    </Style>
</DataGrid.Resources>
```

У всіх раніше згаданих лістингах відповідні елементи розмітки виділені синім.

Проміжна перевірка

Запустіть застосунок та відредагуйте раніше введені записи. Відредаговані дані мають відобразитися у таблиці. Перевірте, чи правильно змінюються дані.

2.6.6.4. Видалення запису про місто

Для видалення даних про місто призначена кнопка "Delete Town" на панелі інструментів головного вікна. Загальна послідовність дій при цьому наступна (лістинг 22). Як і при редагуванні, перевіряємо, чи є у джерелі даних якісь дані. Якщо їх немає, припиняємо роботу методу. Далі отримуємо посилання на виділений елемент таблиці та зберігаємо його дані у змінній town класу Town. При цьому перевіряємо, чи дані можуть бути приведені до класу Town. Якщо ні (а це буває у випадку, коли в таблиці не виділено ніяких записів), виводимо відповідне повідомлення. Далі перепитуємо, чи справді користувач хоче видалити запис. Якщо так, викликаємо метод DAL DeleteTown і передаємо йому ідентифікатор міста, яке слід видалити. При успішному виконанні метода DeleteTown оновлюємо дані, а у випадку помилки – відображуємо відповідне повідомлення користувачеві.

Лістинг 22

```
private void BtnDeleteTown_Click(object sender,
    RoutedEventArgs e)
{
    if (DAL.TownsBuffer.Count == 0) return;
    if (TownsGrid.SelectedItem is not Town town)
    {
        MessageBox.Show("Datagrid row contains town data have " +
        "to be selected first.", "Please select town",
            MessageBoxButton.OK, MessageBoxImage.Information);
        return;
```

```
    }

    if (MessageBox.Show(String.Format("Delete town {0} ?",
        town.Name), "Permanently delete record",
        MessageBoxButton.YesNo, MessageBoxImage.Question
       ) == MessageBoxResult.Yes)
    {
        if (DAL.DeleteTown(town.Id) > 0)
        {
            TotalTowns = DAL.ReadTowns();
            RefreshGrid();
        }
        else
        {
            MessageBox.Show("Record not deleted!",
            "Error occured", MessageBoxButton.OK,
            MessageBoxImage.Error);
        }
    }
}
```

Коли метод BtnDeleteTown_Click створено, посилання на нього слід додати у код розмітки XAML для кнопки BtnDeleteTown (лістинг 11), елемента головного меню mniDeleteTown (лістинг 10), та елемента контекстного меню miDeleteTown (лістинг 12). У всіх приведених лістингах відповідні елементи розмітки виділені синім.

Проміжна перевірка

Запустіть застосунок, введіть кілька довільних записів про місто, потім виділіть середній запис і видаліть його. Він має зникнути з таблиці. Послідовно видаліть всі введені записи. Перевірте, чи правильно видаляються дані.

2.6.6.5. Припинення роботи застосунку

Для припинення роботи застосунку викликаємо метод Shutdown класу Application (лістинг 23). Перед цим перепитуємо, чи справді користувач хоче припинити роботу.

Лістинг 23

```
private void BtnExit_Click(object sender, RoutedEventArgs e)
{
    if (MessageBox.Show("Close application?",
        "Close application", MessageBoxButton.YesNo,
        MessageBoxImage.Question) == MessageBoxResult.Yes)
        Application.Current.Shutdown();
```

```
}
```

Посилання на метод BtnExit_Click слід додати у код розмітки кнопки BtnExit панелі інструментів (лістинг 11) та елемента головного меню mniExit (лістинг 10).

Проміжна перевірка

Запустіть застосунок і спробуйте припинити його роботу за допомогою кнопки виходу і команди меню.

2.6.6.6. Збереження даних про міста у файл csv

Формат CSV (comma-separated values) – файловий формат для представлення табличних даних, у якому поля відокремлюються символом коми та переходу на новий рядок. Поля, які містять коми, декілька рядків, або лапки, мають обмежуватися з обох боків лапками. Формат CSV використовується зокрема для перенесення даних між базами даних та програмами – редакторами електронних таблиць.

Код для експорту даних про міста у файл формату csv реалізовано у методі mniSaveCsv_Click (лістинг 24). Після попередньої перевірки наявності даних, які можна було б записати, викликається стандартне діалогове вікно Windows для вибору назви та розміщення файлу, в який записуватимуться дані. З цією метою використовується екземпляр класу SaveFileDialog, який перед відображенням позиціюється на папці, де розміщено виконавчий файл застосунку. Для передачі даних у дисковий файл використано екземпляр класу StreamWriter.

За допомогою циклу foreach отримуємо дані всіх міст з буфера DAL.TownsBuffer і порядково передаємо їх у StreamWriter для запису в файл. В самому кінці закриваємо потік запису за допомогою його методу Close.

Лістинг 24

```
private void mniSaveCsv_Click(object sender, RoutedEventArgs e)
{
    if (DAL.TownsBuffer.Count == 0)
    {
        MessageBox.Show("Nothing to save. Please add " +
        "towns first.", "Nothing to save", MessageBoxButton.OK,
        MessageBoxImage.Information);
        return;
    }

    SaveFileDialog saveFileDialog = new();
    saveFileDialog.Filter =
```

```
            "CSV files (*.csv)|*.csv|All files (*.*)|*.*";
        saveFileDialog.Title = "Save As CSV";
        saveFileDialog.InitialDirectory =
            System.AppDomain.CurrentDomain.BaseDirectory;

        StreamWriter sw;

        if (saveFileDialog.ShowDialog() == true)
        {
            sw = new StreamWriter(saveFileDialog.FileName, false,
                Encoding.UTF8);
            sw.WriteLine("TownName,Country,Region,Population," +
                "Yea rIncome,Area,Latitude,Longitude,Has Port," +
                "Has Airport");
            try
            {
                foreach (Town town in DAL.TownsBuffer)
                {
                    string townInfo = String.Format(
$"\"{town.Name}\",\"{town.Country}\",\"{town.Region}\"," +
$"\"{town.Population}\",\"{town.YearIncome}\",\"{town.Area}\"," +
$"\"{town.Latitude}\",\"{town.Longitude}\"," +
$"\"{town.HasPort}\",\"{town.HasAirport}\"");
                    sw.WriteLine(townInfo);
                }
            }
            catch (Exception ex)
            {
                MessageBox.Show("Сталась помилка: \n{0}",
                ex.Message, MessageBoxButton.OKCancel,
                MessageBoxImage.Error);
            }
            finally
            {
                sw.Close();
            }
        }
}
```

Після закінчення розробки методу mniSaveCsv_Click його слід додати до розмітки XAML елемента меню mniSaveCsv (лістинг 10).

Проміжна перевірка

Запустіть застосунок, введіть не менше трьох записів і збережіть їх у файл csv. Далі імпортуйте дані з цього файлу в Excel і перевірте, чи правильно вони відображуються у табличному вигляді (рис. 18).

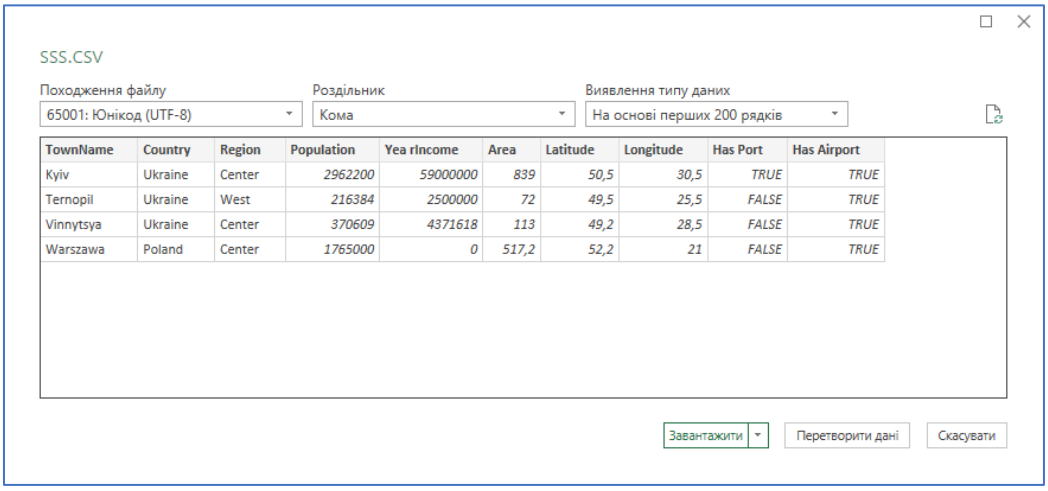


Рис. 18. Імпорт даних зі збереженого файлу csv в Excel

2.6.6.7. Сортування даних

Сортування записів про міста буде виконуватися при виборі користувачем елемента списку cbSortBy, розташованого на панелі інструментів головного вікна. Елементи цього списку (сортувати за: "назвою", "країною" тощо) задають параметр, за яким слід сортувати дані у таблиці. Саме сортування здійснюється в оброблювачі події SelectedIndexChanged компонента cbSortBy.

Для сортування переліку міст за зростанням у списку DAL TownsBuffer використано метод розширення OrderBy LINQ (Language-Integrated Query). Йому передається лямбда-вираз (типу o => o.Name), що вказує, за яким саме полем елемента списку слід сортувати. Зауважимо, що сортування не потребує запиту до БД, тому сортується лише список у пам'яті. Після цього оновлені дані відображуємо у таблиці вікна (лістинг 25).

Лістинг 25

```
private void CbSortBy_SelectionChanged(object sender,
SelectionChangedEventArgs e)
{
    DAL.TownsBuffer = cbSortBy.SelectedIndex switch
    {
        (int)TownsSortOrder.SortByName =>
            DAL.TownsBuffer.OrderBy(o => o.Name).ToList(),
        (int)TownsSortOrder.SortByCountry =>
            DAL.TownsBuffer.OrderBy(o => o.Country).ToList(),
        (int)TownsSortOrder.SortByRegion =>
            DAL.TownsBuffer.OrderBy(o => o.Region).ToList(),
        (int)TownsSortOrder.SortByPopulation =>
            DAL.TownsBuffer.OrderBy(o => o.Population).ToList(),
        (int)TownsSortOrder.SortByArea =>
            DAL.TownsBuffer.OrderBy(o => o.Area).ToList(),
```

```
            _ => DAL.TownsBuffer.OrderBy(o => o.Name).ToList(),
        };
        if (TownsGrid != null)
        {
            RefreshGrid();
            SetGridFocusByRow(0);
        }
    }

    private void SetGridFocusByRow(int rowNumber)
    {
        TownsGrid.SelectedIndex = rowNumber;
        DataGridRow row = (DataGridRow)TownsGrid
            .ItemContainerGenerator
            .ContainerFromIndex(rowNumber);
        if (row != null) row.MoveFocus(new TraversalRequest(
            FocusNavigationDirection.Next));
    }
```

Щоб після оновлення надати таблиці даних фокус вводу, використано метод SetGridFocusByRow.

Коли метод CbSortBy_SelectionChanged запрограмовано, посилання на нього слід додати у код розмітки випадного списку cbSortBy (лістинг 11).

Проміжна перевірка

Запустіть застосунок, введіть не менше п'яти записів і відсортуйте їх шляхом вибору різних умов сортування у випадному списку cbSortBy на панелі інструментів. Перевірте, чи сортування за всіма параметрами відбувається правильно.

2.6.6.8. Відображення додаткової інформації у рядку стану

При проектуванні рядка стану ми передбачили у ньому поля додаткових даних:

- для річного доходу, який припадає на одного мешканця міста (lblYearIncome);
- для щільності населення вибраного міста (lblPopulationDensity).

Оновлення рядка стану реалізовано у методі RefreshTownInfo (лістинг 26). Спочатку отримуємо об'єкт Town для вибраного елемента таблиці. Потім за допомогою методів GetYearIncomePerInhabitant (обчислює дохід на мешканця) та PopulationDensity (обчислює щільність населення) класу Town розраховуємо відповідні значення та записуємо їх у рядок стану. Щоб відобразити додаткову

інформацію про місто з виділеного у таблиці рядка, використаємо подію SelectionChanged таблиці TownsGrid. В оброблювачі цієї події спочатку просто викликаємо метод RefreshTownInfo.

Методи GetYearIncomePerInhabitant та PopulationDensity описані в розділі, присвяченому розробці класу Town. Вони повертають дійсне число, яке форматуємо і виводимо у відповідні елементи рядка стану.

Лістинг 26

```
private void TownsGrid_SelectionChanged(object sender,
SelectionChangedEventArgs e)
{
    RefreshTownInfo();
}

private void RefreshTownInfo()
{
    if (TownsGrid.SelectedItem is Town town)
    {
        lblYearIncome.Text = String.Format(
            "Year income per inhabitant: {0:0.000}",
            town.GetYearIncomePerInhabitant());
        lblPopulationDensity.Text = String.Format(
            "Population density: {0:0.000}",
            town.PopulationDensity());
    }
}
```

Метод TownsGrid_SelectionChanged потрібно додати у розмітку XAML елемента TownsGrid (лістинг 12).

Проміжна перевірка

Запустіть застосунок, і, активуючи різні записи у таблиці, прослідкуйте, як змінюються дані в рядку стану. Перевірте, чи правильно відображуються додаткові дані.

2.6.6.9. Фільтрування даних

Фільтрування даних має відбуватись, коли користувач вводить певне числове значення у текстове поле txbFilter на панелі інструментів (див. рис. 16 та лістинг 11). Код, який реалізує фільтрування даних, приведено у лістингу 27.

Саме фільтрування виконується за допомогою методу розширення Where LINQ, у якому за допомогою лямбда- виразу задаємо умову фільтрування (o => o.Population > minPopulation). Таким чином, у списку залишаться тільки ті міста, у яких значення поля Population більше за змінну minPopulation, у яку записується числове значення, введене користувачем у текстове поле txbFilter. При цьому фільтрування виконуватиметься тільки у тому випадку, якщо користувач введе додатне число.

Лістинг 27

```
private void txbFilter_TextChanged(object sender,
TextChangedEventArgs e)
{
    int minPopulation = 0;
    Int32.TryParse(txbFilter.Text, out minPopulation);

    if (minPopulation > 0)
    {
        DAL.ReadTowns();
        DAL.TownsBuffer =
            DAL.TownsBuffer.Where(
                o => o.Population > minPopulation
            ).ToList();
        TotalTowns = DAL.TownsBuffer.Count;
        RefreshGrid();
        RefreshTownInfo();
    }
    else
    {
        TotalTowns = DAL.ReadTowns();
        RefreshGrid();
        RefreshTownInfo();
    }
}
private void BtnClearFilter_Click(object sender,
    RoutedEventArgs e)
{
    txbFilter.Text = "";
}
```

Для очищення фільтру використовується кнопка BtnClearFilter панелі інструментів. Код, який виконуватиметься при її натисканні, реалізовано у методі BtnClearFilter_Click (лістинг 27).

Посилання на метод txbFilter_TextChanged потрібно додати у розмітку XAML елемента txbFilter панелі інструментів (лістинг 11). Аналогічно слід додати посилання на метод BtnClearFilter_Click для кнопки BtnClear (лістинг 11).

Проміжна перевірка

Запустіть застосунок. Подивіться на значення кількості населення для наявних у таблиці міст. Якщо вони для всіх записів одинакові, то відредагуйте їх так, щоб вони відрізнялися. Введіть значення мінімальної кількості населення у поле фільтра, вибравши його так, щоб частина записів містила кількість населення, меншу за введене значення. Перевірте, чи правильно відфільтровано дані.

2.6.6.10. Відображення інформації про програму

За допомогою елемента меню mniAbout викликається вікно з короткою інформацією про програму (лістинг 28). Повідомлення повинно містити дані про призначення та автора програми. Метод mniAbout_Click з цього лістингу слід додати до розмітки XAML відповідного елемента меню (лістинг 10).

Лістинг 28

```
private void mniAbout_Click(object sender, RoutedEventArgs e)
{
    MessageBox.Show("Application created as .NET & " +
        "MS SQL Server demo.\n\n" +
        "TNTU, Department of software engineering, 2021",
        "About", MessageBoxButton.OK,
        MessageBoxImage.Information);
}
```

Проміжна перевірка

Запустіть застосунок і перевірте роботу команди About.

2.6.6.11. Збереження налаштувань застосунку

При закінченні сесії роботи застосунку зберігатимемо такі налаштування:

- ширину головного вікна;
- висоту головного вікн;
- порядок сортування записів у таблиці.

C# пропонує для вирішення подібних завдань клас Settings. Спочатку слід створити параметри для налаштувань. Для цього у вікні Solution Explorer потрібно розгорнути елемент Properties і двічі клацнути на файлі Settings.settings. Саме у цьому файлі за замовчуванням зберігається опис параметрів, які слід зберігати. При проектуванні застосунку можна створити й інші файли .settings

для створення різних груп налаштувань. Але ми скористаємось вже наявним файлом.

Після подвійного клацання на файлі Settings.settings у Visual Studio відкриється редактор налаштувань (рис. 19). Кожен параметр має:

- назву (стовпчик Name);
- тип (стовпчик Type);
- область дії (стовпчик Scope);
- значення (стовпчик Value).

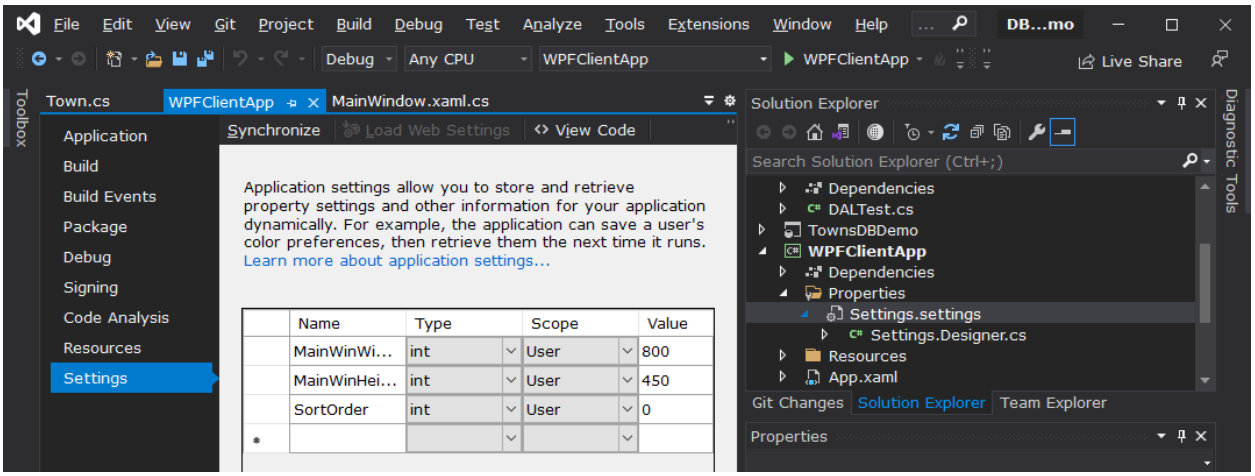


Рис. 19. Створення параметрів для налаштувань застосунку

Кожен параметр представлений окремим рядком таблиці (рис. 19). Створимо параметри відповідно до таблиці 4. Тип всіх параметрів – ціле число (int), а область дії – User. Ця область дії означає, що налаштування можуть бути записані та прочитані програмно. Можлива ще інша область дії – Application. Налаштування цього виду створюються при розробці застосунку і програмно можуть бути тільки прочитані. Записати у них якесь значення програмно неможливо.

Таблиця №4. Параметри для збереження

Назва (Name)	Тип (Type)	Область дії (Scope)
MainWinWidth	int	User
MainWinHeight	int	User
SortOrder	int	User

Всі створені таким чином налаштування супроводжуються додаванням до властивості Default класу Settings поля з заданою назвою та типом. Таким чином, доступ до параметру MyParameter у налаштуваннях здійснюється так: Settings.Default.MyParameter.

Зберігати налаштування потрібно при виході з програми. Для цього зручно використовувати подію Closed головного вікна. Отже, створимо оброблювач цієї події wpfMainWindow_Closed і приведемо його до виду згідно лістингу 29.

Лістинг 29

```
private void wpfMainWindow_Closed(object sender, EventArgs e)
{
    Properties.Settings.Default.MainWinWidth = (int)Width;
    Properties.Settings.Default.MainWinHeight = (int)Height;
    Properties.Settings.Default.SortOrder =
        cbSortBy.SelectedIndex;
    Properties.Settings.Default.Save();
}

private void Window_Loaded(object sender, RoutedEventArgs e)
{
    if (Properties.Settings.Default.MainWinWidth > 0)
        Width = Properties.Settings.Default.MainWinWidth;
    if (Properties.Settings.Default.MainWinHeight > 0)
        Height = Properties.Settings.Default.MainWinHeight;

    cbSortBy.SelectedIndex =
        Properties.Settings.Default.SortOrder;

    SetGridFocusByRow(0);
}
```

Читати налаштування зручно при створенні головного вікна. Тому в оброблювач події Loaded головного вікна (метод Window_Loaded) слід написати код, який читатиме збережені налаштування і присвоюватиме прочитані значення відповідним елементам головного вікна (лістинг 29). Крім налаштувань, цей код також викликає раніше розглянутий метод SetGridFocusByRow.

Налаштування, які зберігають описаним у цьому пункті способом, записуються у форматі xml у файл user.config, який розташований за шляхом:

[X]:\Users\[UserName]\AppData\Local\[ProjectName]\[ProjectName]_Url _[Hash]\[Version]\

Якщо відкрити його у текстовому редакторі (наприклад, у "Блокноті"), то можна побачити поточні значення параметрів.

Проміжна перевірка

Запустіть застосунок, змініть розміри головного вікна. Закрийте його, запустіть ще раз, і перевірте, чи розміри відповідають тим, які були в кінці минулої сесії.

2.6.6.12. Кінцеве тестування застосунку

Тестування є важливим етапом розробки програмного забезпечення.

Запустіть розроблений застосунок. Введіть не менше 10 записів з реальними (або правдоподібними) даними. Повторно протестуйте всі функції застосунку (для панелі інструментів, головного та контекстного меню). Якщо якась із них працює не так як слід, поверніться до відповідного розділу і перевірте, чи все зроблено вірно: чи правильно задано розмітку XAML, чи створено відповідні методи (оброблювачі подій), і чи правильно введено програмний код. На рис. 16 показано вигляд головного вікна застосунку з введеними даними.

Повністю робоче рішення зі всіма проектами, які описані у даних методичних вказівках, можна завантажити з матеріалів електронного навчального курсу "Проектний практикум". Воно розташоване в архіві DBAppDemo.zip. Його можна розпакувати, відкрити в середовищі Microsoft Visual Studio та детально розглянути роботу кожного проекту рішення.

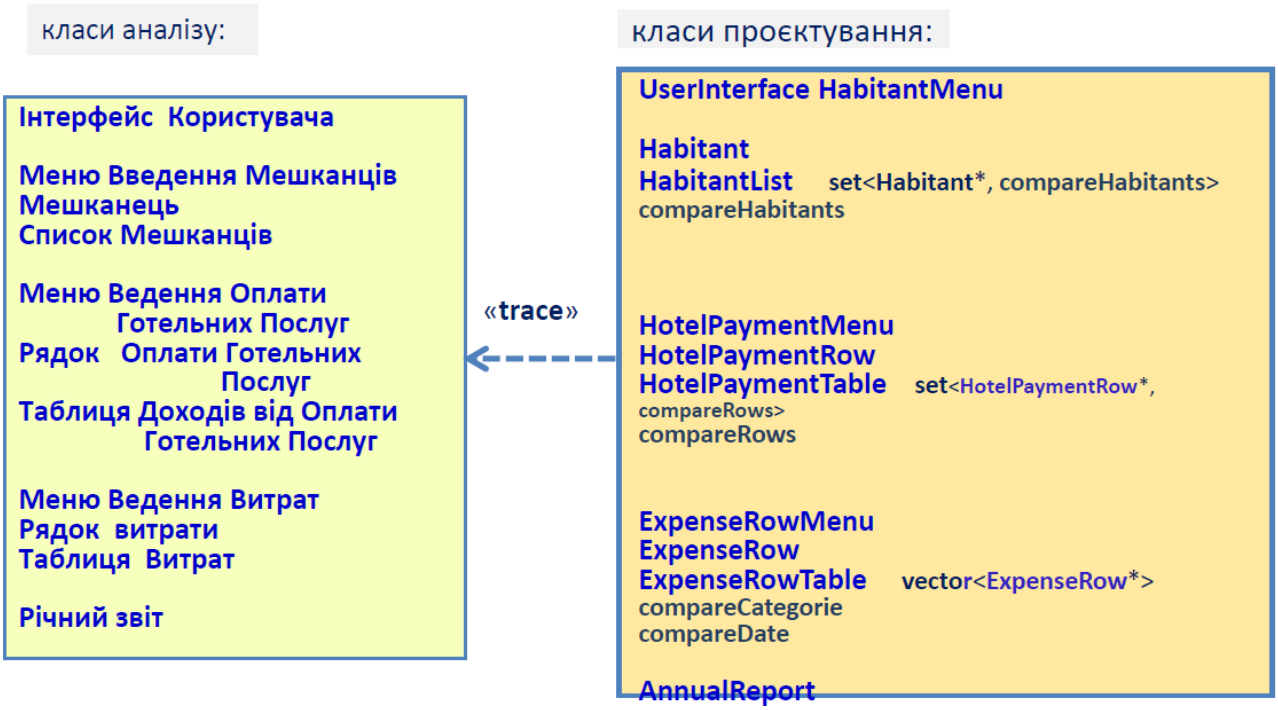
3. Додаток. Етапи проектування на прикладі ПЗ для підтримки готельної діяльності

3.1. Аналіз завдання, предметної області та виділення класів аналізу та класів проектування

Предметна область розробки. ПЗ розробляється згідно вимог замовника (для автоматизації технологічних операції по реєстрації даних і формуванню звітів про господарську та фінансову діяльність готельного бізнесу.

Детальний опис даних і дій клієнта. Реальні дані , з якими працює замовник (плата , отримувана за здачу в оренду кімнат, витрати, поточні платежі, дії над ними, механізми реєстрації, структури і форми зберігання даних.

Список мешканців. Містить номери кімнат та імена наймачів, що проживають в них впродовж 12 місяців поточного року. Це набір пар (2 взаємно пов'язаних сутностей ім'я мешканця – номер кімнати проживання. Реєструючи нового адміністратор вносить його ім'я 1 й стовпець списку мешканців і ставить проти нього номер кімнати у 2 й стовпець. Аналог при виселенні синхронно здійснюють дві зворотні операції.



Таблиця доходів від готельних послуг. Містить записи про платежі наймачів. Отримуючи платіж від мешканця, адміністратор заносить оплачену суму у відповідний елемент таблиці. Таблиця наочно показує, які суми і коли внесені за проживання/ оренду певного номера готелю.

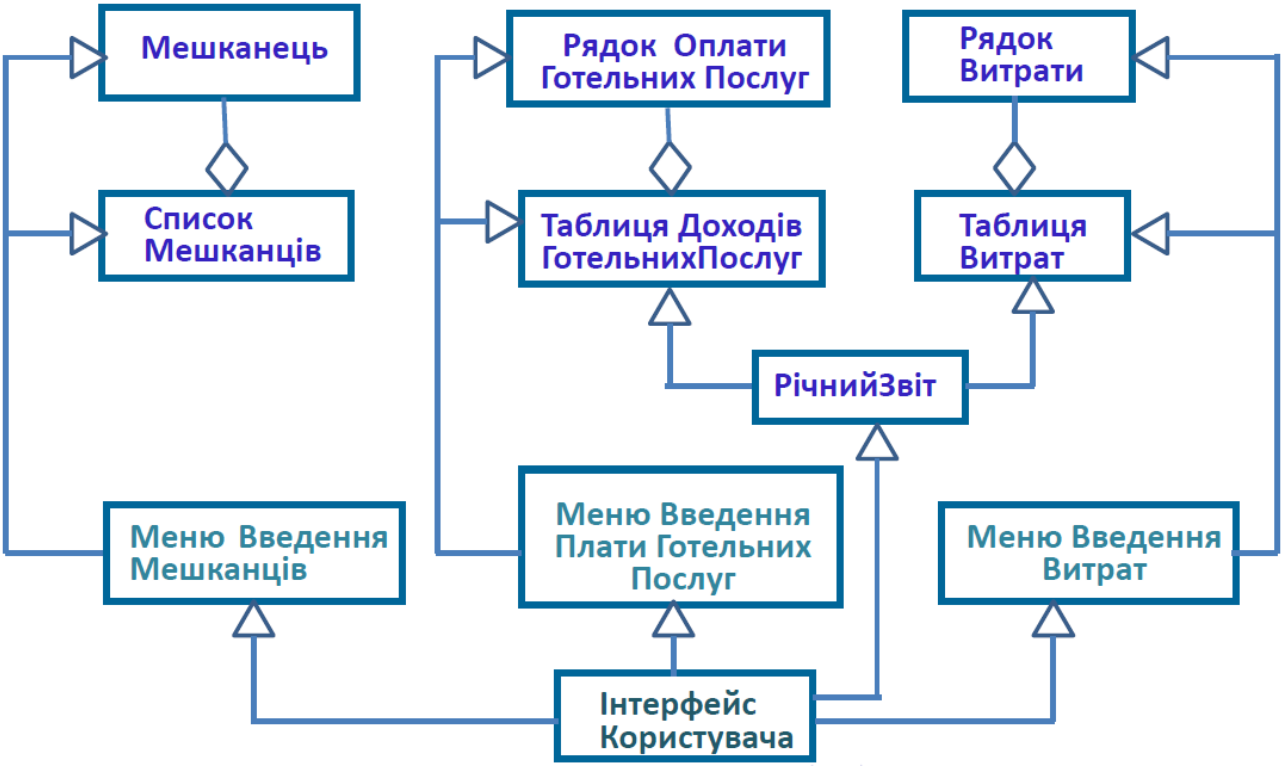
Номер кімнати	Вартість проживання (ааренди) кімнати (номера) в готелі, грн							
	Січень	Лютий	Берез	Квітень	Травень	Червень	Липень	Серп
101	60	60	60					…
102	50	50	50					…
103	80	80	80					…
104	70	70	70					…
201	60	60	60					…

3.2. Визначення відповідальності класів

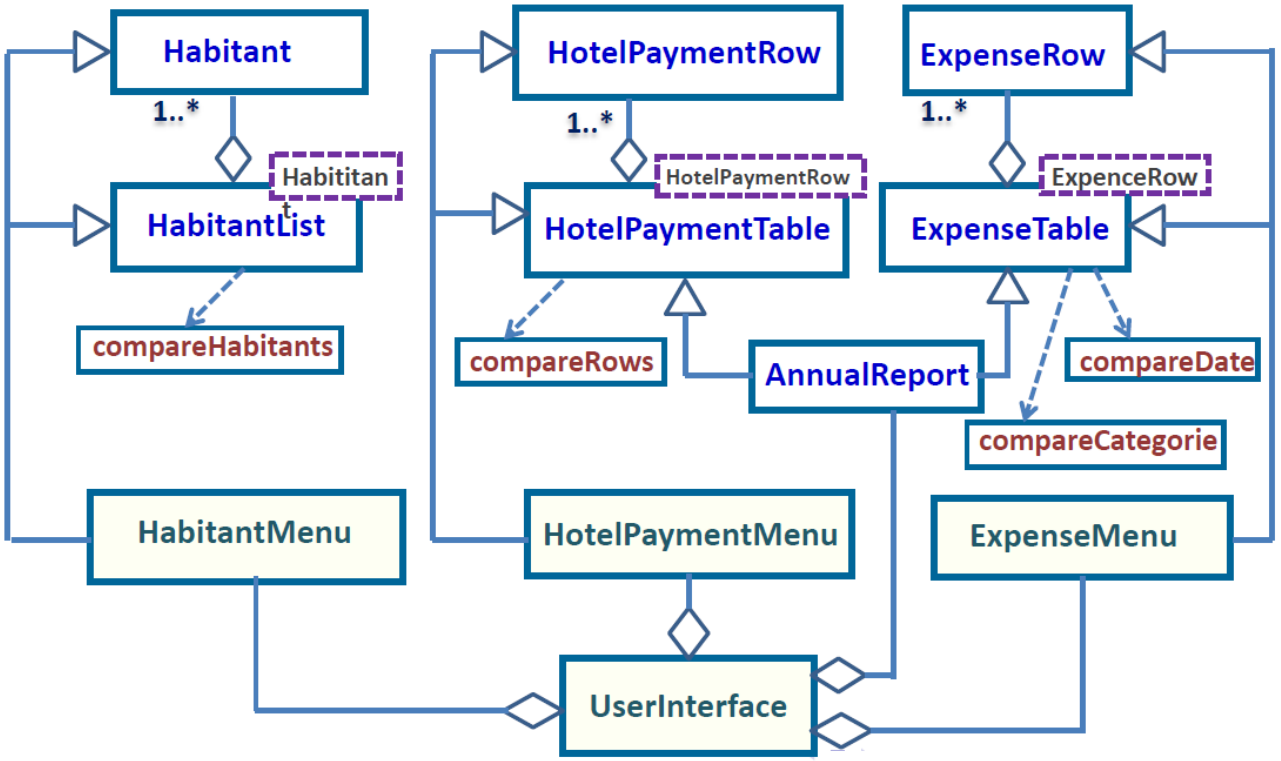
‘	класи	відповідальність
1	**Інтерфейс Користувача (UserInterface)**	забезпечує реалізацію користувачем сценіріїв ВВ системи; містить посилання на об’єкти класів **HabitantMenu, HotelPaymentMenu, ExpenseRowMenu, AnnualReport**
2	**Меню Введення Мешканців (HabitantMenu)**	забезпечує введення користувачем атрибутів Мешканця, включає посилання на об’єкт класу **HabitantList**
3	**Список Мешканців (HabitantList)**	Впорядкований set-список посилань на об’єкти класу **Habitant**, забезпечує вставку,зберігання, вставки і вилучення з ФПО та вивід атрибутів **Habitant**-об’єктів
4	Мешканнець (**Habitant**)	для створ.об’єктів пар «ім’я_мешканця-номер_кімнати», включаючи засоби їх порівняння
5	**Меню Введення Оплати Готельних Послуг (HotelPaymentMenu)**	забезпечує введення користувачем атрибутів оплати рядка готельних послуг **HotelPaymentRow**, по імені мешканця конкретного номера кімнати за вказ. місяць включає посилання на обєкт класів **HabitantList**, **HotelPaymentTable.**
6	**Таблиця Доходів від Оплати Готельних Послуг (HotelPaymentTable)**	Впорядкований set-список посилань на об’єкти класу **HotelPaymentRow**, забезпечує вставку,зберігання, вилучення, повернення, вставки і вилучення з ФПО та вивід атрибутів **HotelPaymentRow**-об’єктів
7	**Рядок Оплати Готельних Послуг (HotelPaymentRow)**	для створ . об’єктів пар «номер_кімнати-масив помісячних оплат», включ. встваку-вилучення з ФПО, Оп порівняння, та підсумовування оплат

3.3. Побудова діаграми класів

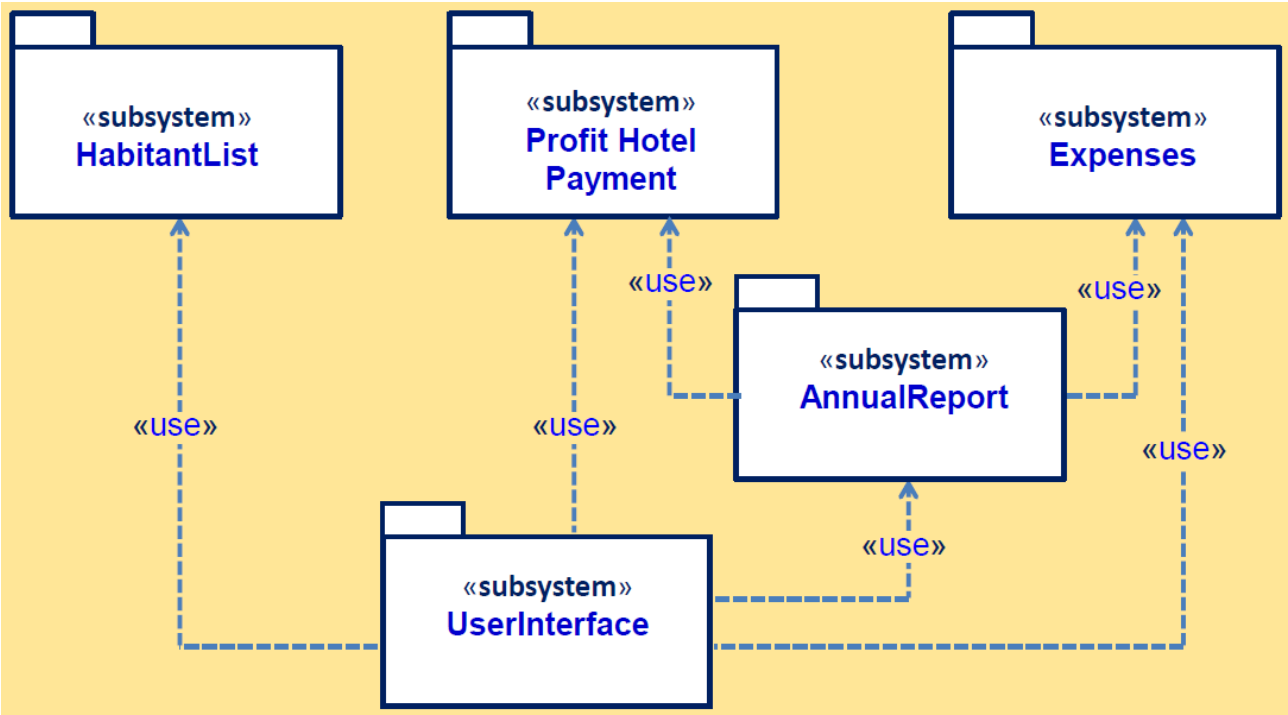
Класи аналізу



Класи проектування



3.4. Проектування загальної архітектури



3.5. Розробка коду

Клас "Мешканець" – Habitant

для створ.об'єктів - пар «ім'я_мешканця-номер_кімнати», включ.засоби порівняння, вставки і вилуч. Habitant-obj в ПО/ФПО

```
class Habitant      //      - клас  Мешканець

{
  private:
     string name;      // і'мя Мешканця
     int aptNumber;  //  номер кімнати
         // other Habitant info  (phone, etc.)
  public:
     Habitant(string n, int aNo);   // к-р  2 арг
     ~Habitant();
     int getAptNumber();   // return  номер кімнати

      bool operator < (const  Habitant&, const Habitant&);          // for use in set
      bool operator == (const Habitant&, const Habitant&);          // перевант.логічні Оп
      friend ostream& operator << (ostream&, const Habitant&);      //перевант Оп вставки
      friend istream& operator >> (istream&, Habitant &);           // вилучення в ПО/ФПО
};
```

Клас "Список мешканців" – HabitantList

для створ. впорядков. set-списку посилань на Habitant-obj, забезп. зберігання, вставки і вилучення в ПО/ ФПО списку Habitant-об'єктів

```
class HabitantList
  {
    private:          // set of pointers to Habitants
      set <Habitant*, compareHabitants>  setPtrsHabs;
      set <Habitant*, compareHabitants> :: iterator iter;  //  iterator  на елем set-*obj
    public:
      ~HabitantList();                    // destructor (deletes Habitants)
       void insertHabitant(Habitant*);  // вставляє  Habitant-obj  в set  SetPtrHabs
       int  getAptNo(string);           //  ->  номер кімнати по імені мешканця
       void display();      // вивід і вставка в ФПО  Списку  Мешканців  (set)

};
```

```
class compareHabitants  // FO - порівняння імен Мешканців у списку
  {
    public:
       bool operator() (Habitant*, Habitant*) const;
   };
```

Прототип ПрОп виклик F

Клас "Меню Введення Мешканців" – HabitantMenu

```
class HabitantMenu  // клас  Меню введення мешканців
  {
     private:
         HabitantList* ptrHabitantList;   //  вказівник  на Список Мешканців   (setPtrHabs)
          string  tName;  // ім'я мешканця
          int  aptNo;       //  номер кімнати

     public:
          HabitantMenu(HabitantList* ptrHaL): ptrHabitantList(ptrHaL) // к-р 1 арг
             { }

          void getHabitant();   // ввод атрибутів Мешканців в set-Список Мешканців   setPtrsHabs
   };
```

Клас "Рядок Оплати Готельних Послуг" – HotelPaymentRow

```
class HotelPaymentRow                    //клас Рядок Оплати Готельних Послуг – ( одного номера за 12 місяців)
 {
   private:
     int aptNo;                          // номер кімнати
     float hotelPayment[12];             // масив помісячних оплат за aptNo-кімнату
   public:
     HotelPaymentRow(int);               // constr 1 арг - номер кімнати
     void setHotelPayment(int, float); // встановл. оплата за 1 міс
     float getSumOfRow();  // -> сума рядка оплати за кінмнату за 12 місяців
     bool operator < (const HotelPaymentRow&, const HotelPaymentRow&); // for store in set
     bool operator == (const HotelPaymentRow&, const HotelPaymentRow&);
     friend ostream& operator << (ostream&, const HotelPaymentRow&); // ПрОп встав
     friend istream& operator >> (istream&, const HotelPaymentRow&); //вилуч в ПО/ФПО
 };
```

```
class compareRows  // FO порівняння HotelPaymentRows
  {
    public:
      bool operator () (HotelPaymentRow*, HotelPaymentRow*) const;
  };
```

Клас "Таблиця оплат готельних послуг" – HotelPaymentTable

```
class HotelPaymentTable
  {                                  // містить атрибут - об'єкт-set вказівників на Рядки Опплат по усіх кімнатах
    private:
       set < HotelPaymentRow*, compareHotelPaymentRow >  setPtrsPrR;  // створ. set-*obj
       set < HotelPaymentRow*, compare HotelPaymentRow > :: iterator iter;
    public:
       ~HotelPaymentTable();
       void insertHotelPayment(int, int, float); //вставка HotelPaymentRow*-obj в Табл.оплат по всіх
                  // кімнатах, у т.ч.: шукає по aptNo кімнати вказівн. на Рядок Оплат HotelPaymentRow (для цього
                  // aptNo) і при наявності м-дом setHotelPayment(int mounth, float amount) і -> в нього плату
                  // amount за місяць month. Якщо вказівник на цей Рядок відсутній в set setPtrsPrR, то створ.
                  // новий (new HotelPaymentRow) для aptNo-кімнати, вводячи amount-плату за month-місяць
                  // м-дом setHotelPayment() Далі вставляє вказівник ptrRow на цей HotelPaymentRow в
                  // setPtrsPrR м-дом setPtrsPrR.insert(prtRow)

       void display();   // вивід і встака в ПО/ФПО setPtrsPrR (Табл. оплат по всіх кімнатах)

       float getSumOfHotelPayments();  // підсусумов оплат по всім 12 місяцям для HotelPaymentRow

};
```

Клас "Меню вводу оплати готельних послуг" – HotelPaymentMenu

```
class HotelPaymentMenu          // клас  Меню вводу HotelPayment- оплати готельн. послуг
 {
   private:
     HabitantList* ptrHabitantList;  // вказівник на set-Список Мешканців (SetPtrHabs)
     HotelPaymentTable* ptrHotelPaymentTable; //вказівник на  Табл.оплат STL-set (SetptrPrR)
     string  renterName;             // ім'я мешканця
     float hotelPaymentPaid;         // сума місячної HotelPayment-плати
     int month;                      // місяць
     int aptNo;                      // номер кімнати
   public:
     HotelPaymentMenu(HabitantList* ptrTL, HotelPaymentTable* ptrPrR) : // к-р 2 арг
                          ptrHabitantList(ptrTL), ptrHotelPaymentTable(ptrPrR)
                 { }

      void getHotelPayment();  // ввод HotelPayment –Рядків Олати в  Табл.оплат  STL-set SetptrPrR)
                               //  для одного мешканця і за 1 місяць :
                               // по renterName-імені мешканця  із списку мешканців  STL-set  SetptrPrR -
                    // ч/з вказівник на неї  ptrHabitantList  здійсн. Пошук № кімнати aptNo, в якій він проживає
                    // вводячи HotelPaymentPaid-плата за вказ. month- місяць(один з 12-ти), мдом
                    //  HotelPaymentTable::  InsertHotelPayment(aptNo,month,HotelPaymentPaid) здійсн.
                    // вставка вказівн.атрибутів у відпов. Існуюч. чи новоствор. HotelPaymentRow -Рядок Опл.
                    //  та  вказівника prtRow на ньогос в  set SetptrPrR  ч/з set-вказівник  prtHotelPaymentTable
 };
```

Клас "Рядок Витрат"

```
class ExpenseRow                  //  клас Рядок Витрат
 {
  public:
    int month, day;           // місяць, день
    string category, payee;   // категорія, одержувач
    float amount;             // сума
    ExpenseRow ()
       { }
    ExpenseRow (int m, int d, string c, string p, float a):       // к-р з 5 арг
                                  month(m), day(d), category(c), payee(p), amount(a)
       { }
     bool operator < (const ExpenseRow &, const ExpenseRow&); //  for sort.  in vector
     bool operator == (const ExpenseRow&, const ExpenseRow&);
     friend ostream& operator << (ostream&, const ExpenseRow &); // ПрОп вставки
     friend istream& operator >> (istream&, const ExpenseRow &); // вилуч Витрат в По/ФПО
 };
class compareDates  // FO  для сортув. алгор. sort()  ExpenseRow-objs  у vector по даті
 {
   public:
     bool operator() (ExpenseRow*, ExpenseRow*) const;  // викор.  Пр Оп <кл ExpenseRow
 };
class compareCategories  // FO  для сортув. алгор. sort()  ExpenseRow-objs  у vector по категоріях
 {
   public:
     bool operator() (ExpenseRow*, ExpenseRow *) const;  // викор. Для  ПрОп < кл ExpenseRow
 };
```

Клас "Таблиця витрат"

```
class ExpenseRowTable
 {
   private:     // vector of pointers to ExpenseRows
      typedef vector <ExpeneseRow*> VectExpTabl;
      VectExpTabl vecPrsExRT;  // створ.vector  -*obj ExpenseRow
      VectExpTabl :: iterator iter;

    public:
      ~ExpenseRowTable();
      void  insertExp(ExpenseRow*);  //вставл. Рядок витрат*  у STL-вектор vectPtrsExpenseRows
                                     // вказівник PtrEp на клас  ExpenseRow м-ом    push_back(PtrEp)
       void display();  // виід і вставка Табл.Витрат в ФПО, викор. STL –алгор. sort() з FO compareDates(),
                        // сортує об'єкти STL- вектора vectPtrsExpenseRows по дані (місяць, день)

       float displaySummary();  // обчисл. суми витрат окремо по кожній  категорії і загальну суму
                                // витрат з викор алг. accumulate(); вивід їх та вставка у ФПО

 };
```

Клас "Меню вводу витрат"

```
class ExpenseRowMenu      // клас Метю вводу витрат
 {
   private:
      ExpenseRowTable* ptrExpenseRowTable; // вказівн на Табл витрат (STL- вектор vectPtrsExpenseRows)
   public:
      ExpenseRowMenu(ExpenseRowTable*); // к-р з 1 арг
      void getExpenseRow();  // вводить атрибути витрат (month, day,  category,
                             // payee,   amount), створ.  новий запис  new ExpenseRow   та з доп. мду
                  // insertExp(prtExpenseRow) вставляє вказівник на цей запис  у STL-вектор  vectPtrsExpenseRows
 };
```

Клас "Річний Звіт"

```
class AnnualReport              // клас Річний Звіт
 {
   private:
      HotelPaymentTable* ptrPrR ;  // вказівник на  set-Табл Рядків Оплат   по усіх кімнатах (SetptrPrR)
      ExpenseRowTable* ptrER;      // вказівник на  vector-Табл Витрат  (STL-  vectPtrsExpenseRows )
      float ExpenseRows, HotelPayments;
  public:
      AnnualReport(HotelPaymentTable*, ExpenseRowTable*); //  k-r 2 arg  –obj Річний звіт
      void display(); //  вивід і вст в ФПО
 };
```

Клас "Інтерфейс Користувача"

```
class UserInterface            // клас Інтерфейс Користувача
 {
   private:          // атрибути-вказівники на об'єкти класів:
      HabitantList*          ptrHabitantList;          // Список Мешканців
      HabitantMenu*          ptrHabitantMenu;          // Меню Вводу Мешканців
      HotelPaymentTable*     ptrHotelPaymentTable;     // Список записів HotelPayment-оплат
      HotelPaymentMenu*      ptrHotelPaymentMenu;      // Меню Вводу Оплат
                                                       // HotelPayment-оплат
      ExpenseRowTable*       ptrExpenseRowTable;       // Вектор записів витрат
      ExpenseRowMenu*        ptrExpenseRowMenu;        // Меню Вводу Витрат
      AnnualReport*          ptrAnnualReport;          // Річний звіт
      char ch;

 public:
      UserInterface();    // к-р без арг
      ~UserInterface();
      void  interact();     // метод реалізації  сценаріїв ВВ системи
 };
```