

ЛАБОРАТОРНА РОБОТА

Тема: .NET Standard в розробці міжплатформного програмного забезпечення.

Мета: Ознайомитись та перевірити механізм .NET Standard для побудови міжплатформного програмного забезпечення.

УВАГА!

Повний код прикладів для даної лабораторної роботи можна знайти за посиланням:

<https://github.com/dmNetTutorials/.NET-Core-Samples/tree/main/.NET%20Standard>

Теоретичні відомості

Під час розробки додатків для різних платформ, наприклад, використовуючи технологію Xamarin, реалізуючи для кожної платформи однакові механізми, можна побачити що код дублюється. Для вирішення цієї проблеми в платформі .NET передбачено використання бібліотеки .NET Standard.

Платформи

Платформа описує набір компонентів. Ці компоненти дозволяють виконувати збірку та запуск додатків на визначених (target - цільових) операційних системах.

Зазвичай робота виконується з стандартними платформами .NET (Рис.3), які мають власні засоби розробки, середовище виконання, бібліотеки.

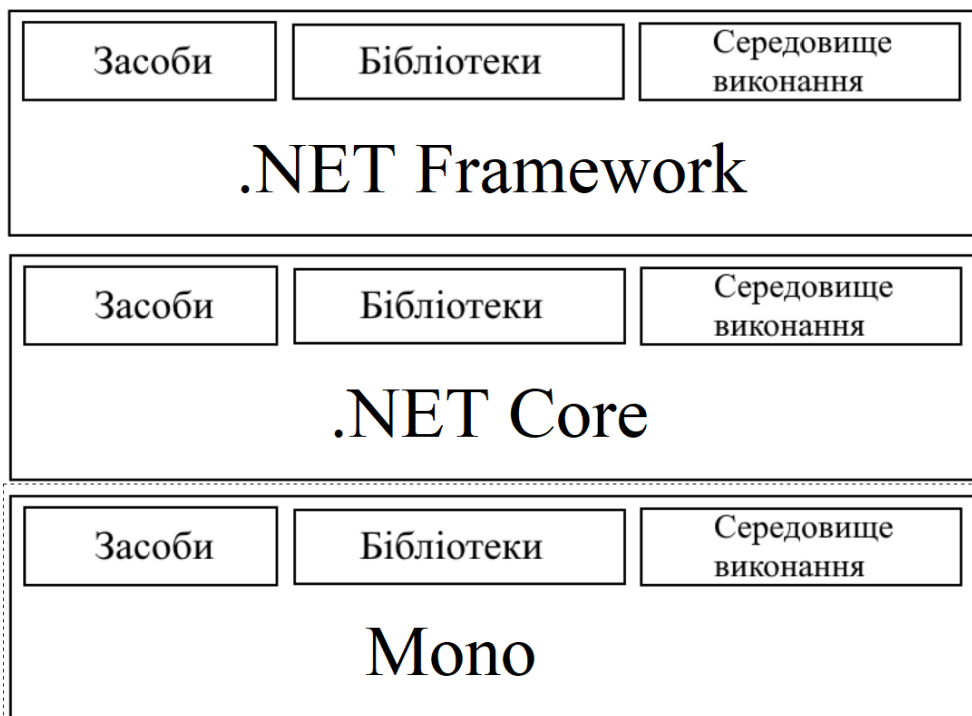


Рисунок 1 — Платформи .NET

Варіанти бібліотек

Бібліотеки для .NET платформи розробляються різними командами, тому доступні можливості програмування може відрізнятись, на кожній окремій ОС. Наприклад, використовується API, який в .NET Framework та .NET Core однаковий, але він може відрізнятись в Mono. Для виходу з цієї ситуації необхідно використовувати .NET Standard.

.NET Standard

.NET Standard використовується щоб узгодити API між платформами.

.NET Standard — специфікація, стандарт який повинна реалізовувати бібліотека платформи. Стандарт спрощує створення єдиної бібліотеки на всіх платформах. Але це не означає що стандарт буде реалізовано однаково на всіх платформах, для різних API реалізація може відрізнятись.

.NET Standard має декілька версій (актуальна 2.1), та продовжує розвиватись. Також передбачена повна зворотня сумісність зі стандартами попередніх версій.

Ранні версії .NET Standard переважно включали в себе інтерфейси API, які були доступні в стандартних профілях портативної бібліотеки класів (Portable Class Library). Починаючи з API, необхідних для основних бібліотек .NET були додані колекції, нові методи та класи для підвищення рівня сумісності.

На даний момент .NET Standard включає в себе всі основні бібліотеки від словників та мережевої взаємодії до керування потоками та шифрування.

В даному прикладі буде продемонстровано створення бібліотеки .NET Standard 2.1 в Visual Studio 2019 в ОС Windows 10 (x64), та її подальше використання в проєкті .NET Core 3.1 під ОС Kali Linux (x64).

ОС Windows

Спочатку необхідно створити відповідний проєкт в Visual Studio 2019, для цього необхідно натиснути кнопку “Create New Project” (Створити новий проєкт), та в пошуку написати “Class Library .NET Standard” після чого обрати тип проєкту **Class Library** (Рис.2).

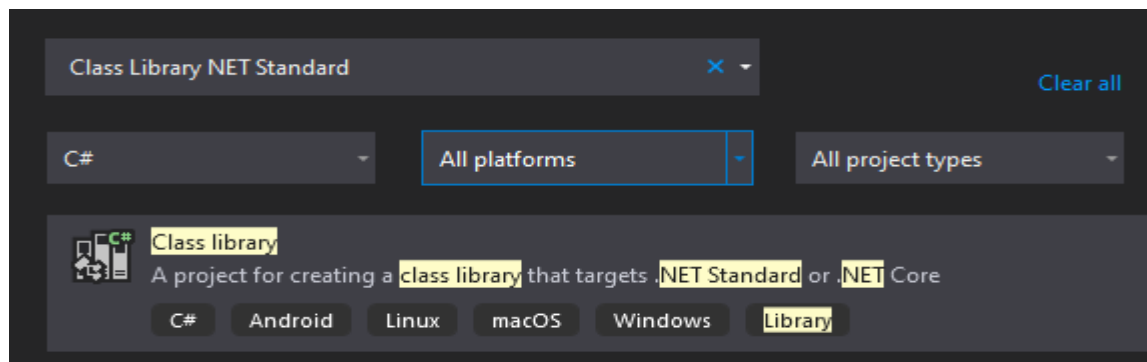
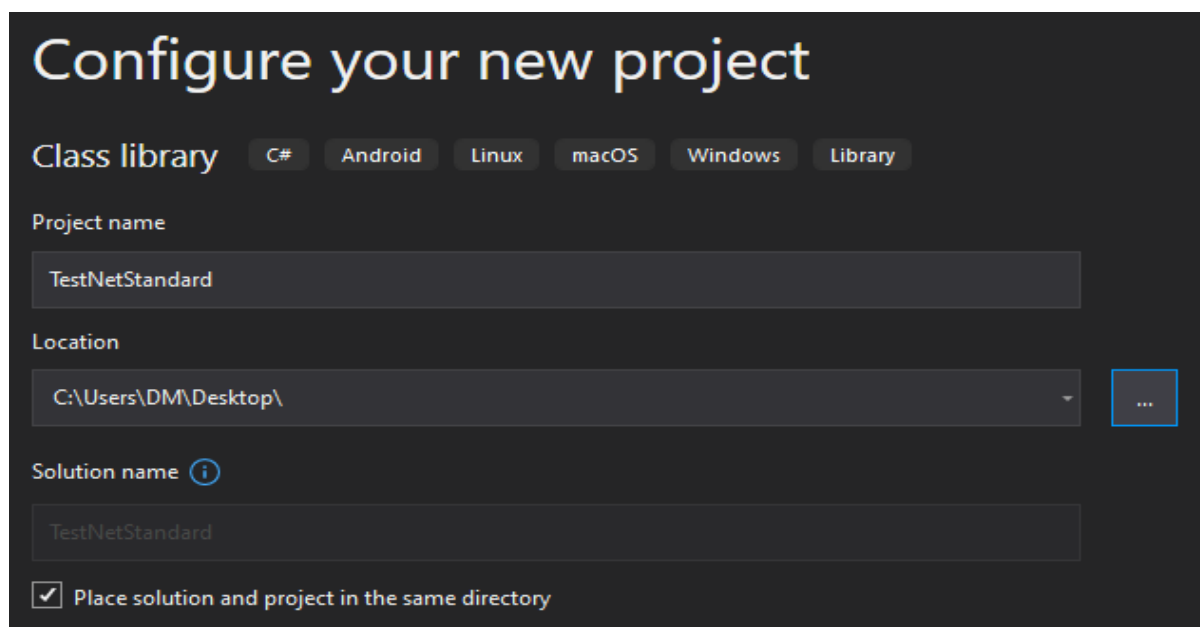


Рисунок 2 — Тип проєкту бібліотека класів .NET Standard

На наступному екрані необхідно вказати назву та місце збереження проєкту (Рис.3).



Configure your new project

Class library C# Android Linux macOS Windows Library

Project name

TestNetStandard

Location

C:\Users\DM\Desktop\

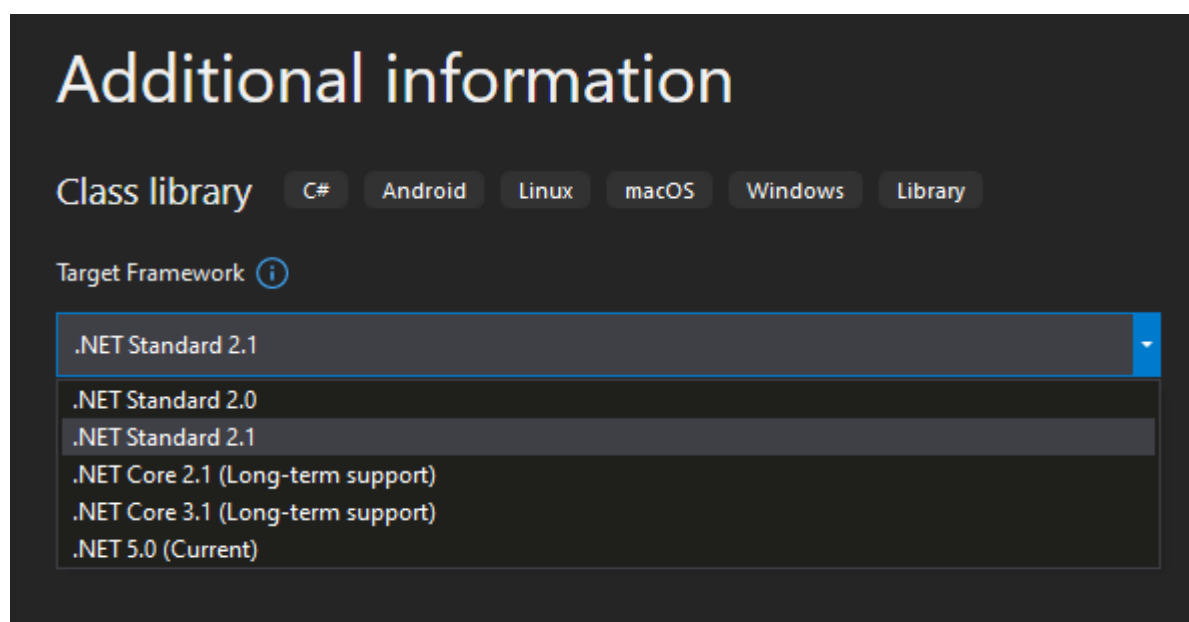
Solution name ⓘ

TestNetStandard

☒ Place solution and project in the same directory

Рисунок 3 — Параметри бібліотеки класів

На наступному етапі потрібно обрати цільовий фреймворк, для якого буде розробляться бібліотека класів, в даному випадку **.NET Standard 2.1** та натиснути кнопку **Create** (Створити) (Рис.4).



Additional information

Class library C# Android Linux macOS Windows Library

Target Framework ⓘ

.NET Standard 2.1

.NET Standard 2.0

.NET Standard 2.1

.NET Core 2.1 (Long-term support)

.NET Core 3.1 (Long-term support)

.NET 5.0 (Current)

Рисунок 4— Цільові фреймворки для бібліотеки класів

Після успішного створення проекту буде відкрито файл за замовчуванням з назвою **Class1.cs**. Даний файл можна перейменувати як-завгодно. В Solution Explorer («Оглядач рішення») можна додати будь-яку кількість файлів організованих в будь-яку структуру. В даному прикладі буде використовуватись 1 файл, код якого наведено нижче (Лістинг 1).

Лістинг 1. Код файлу Class1.cs

```
using System;
using System.Text.RegularExpressions;

namespace TestNetStandard
{
    public class Validation
    {
        public Validation()
        {
        }

        public static bool Validate(string email, Action Success, Action
Fail)
        {
            bool isValidated = false;
            string searchPattern
                = @"[a-zA-Z0-9_\.]+\@[a-zA-Z0-9]+\.[a-zA-Z]+";
            if (Regex.IsMatch(email, searchPattern, RegexOptions.Compiled |
                RegexOptions.IgnoreCase |
                RegexOptions.Singleline))
            {
                Success();
                isValidated = true;
            }
            else
            {
                Fail();
                isValidated = false;
            }

            return isValidated;
        }
    }
}
```

Основна функція тестової бібліотеки – виконання перевірки (validation) введеної адреси ел. пошти (змінна **email**). Для успішного виконання функції пошта повинна бути вказана в форматі :

<логін пошти> @ <домен пошти>

Приклад правильно вказаної пошти: test@localhost.com/

Також передбачено можливість виконання додаткової процедури (**Action Success**), яку розробники можуть використати, наприклад, для оновлення інтерфейсу. Результуючій змінній (**isValidated**) буде призначено значення **true**;

При невідповідності шаблону буде виконано процедуру яку може передати розробник (**Action Fail**) та результуючій змінній буде встановлено значення – **false**.

Перед виконанням збірки (Build) бібліотеки необхідно перевірити параметри бібліотеки, натиснувши правою кнопкою миші (ПКМ) в Solution Explorer на проекті **TestNetStandard** та обрати пункт Properties (Властивості). Буде відкрито вікно налаштувань параметрів бібліотеки (Рис.5). В полі Assembly name ("Ім'я збірки") має бути вказано **TestNetStandard**, так само і в Default

namespace («Простір імен за замовчуванням»). Target Framework («Цільовий фреймворк») має бути **.NET Standard 2.1**, та Output type («Результуючий тип») – **Class Library** («Бібліотека класів»).

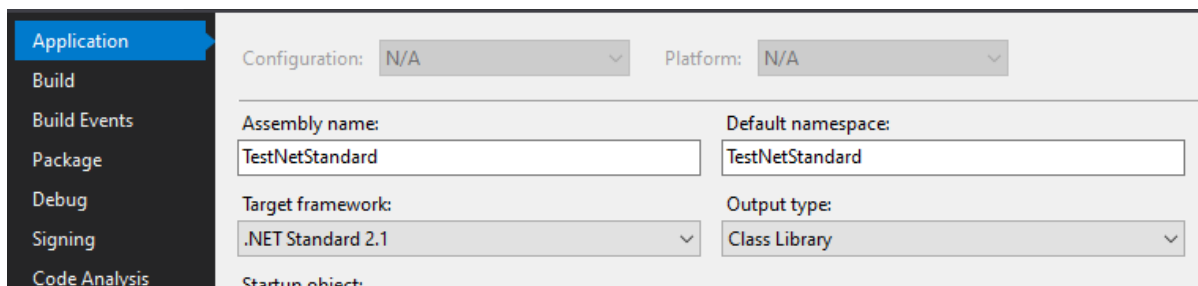


Рисунок 5 — Загальні налаштування бібліотеки класів

Наступним дуже важливим розділом є метадані пакету, які знаходяться в розділі **Package** («Пакет»). В цьому розділі необхідно щоб було встановлено для поля Package id значення **TestNetStandard**, та для Package Version («Версія пакету») значення **1.0.0**. Ці значення необхідно запам'ятати, для забезпечення доступу до бібліотеки з іншого проекту (Рис.).

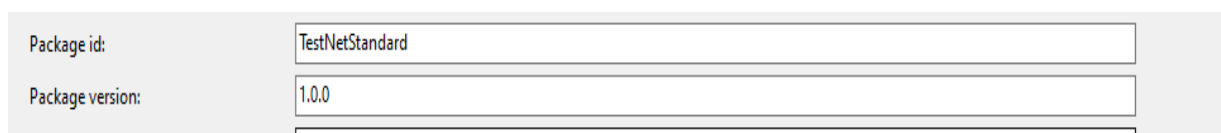


Рисунок 6 — Метадані бібліотеки класів

Також необхідно встановити конфігурацію побудови (Build Configuration), обравши пункт Release (Рис.7).

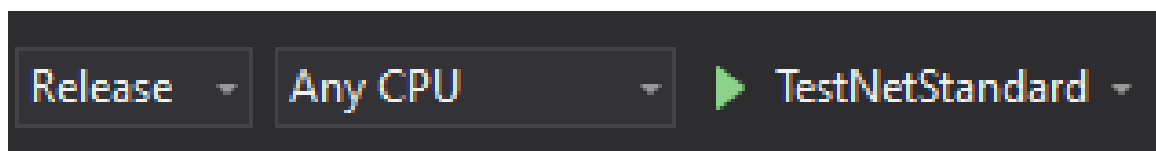


Рисунок 7 — Конфігурація побудови проекту

Коли всі налаштування зроблені необхідно виконати Build («Побудову проекту») натиснувши комбінацію клавіш **Ctrl + Shift + B** або відкрити пункт меню Build та обрати пункт **Build Solution** («Побудувати рішення»). Після успішної побудови бібліотеки, її можна знайти в каталозі **bin\Release\netstandard2.1** бібліотека матиме назву **TestNetStandard.dll**. Саме цей файл необхідно використовувати в інших проектах.

ОС Kali Linux

Відкривши термінал необхідно встановити .NET Core 3.1 та MS Visual Code виконавши команди (для дистрибутивів на основі Debian) [1]:

```
$\> sudo apt-get update; \  
$\> sudo apt-get install -y apt-transport-https && \  
$\> sudo apt-get update && \  
$\> sudo apt-get install -y dotnet-sdk-3.1  
$\> sudo apt-get install code
```

Для перевірки встановлених компонентів необхідно використати команду:

```
$\> sudo dotnet --info
```

Результат виконання команди (в розділі .NET SDK Installed та .NET Runtimes installed) зображено на Рис.8.

```
dm@dm:~/Desktop/Shared_Folder$ sudo dotnet --info
[sudo] password for dm:
.NET SDK (reflecting any global.json):
Version: 5.0.402
Commit: e9d3381880

Runtime Environment:
OS Name: kali
OS Version: 2021.3
OS Platform: Linux
RID: linux-x64
Base Path: /usr/share/dotnet/sdk/5.0.402/

Host (useful for support):
Version: 5.0.11
Commit: f431858f8b

.NET SDKs installed:
3.1.414 [/usr/share/dotnet/sdk]
5.0.402 [/usr/share/dotnet/sdk]

.NET runtimes installed:
Microsoft.AspNetCore.App 3.1.20 [/usr/share/dotnet/shared/Microsoft.AspNetCore.App]
Microsoft.AspNetCore.App 5.0.11 [/usr/share/dotnet/shared/Microsoft.AspNetCore.App]
Microsoft.NETCore.App 3.1.20 [/usr/share/dotnet/shared/Microsoft.NETCore.App]
Microsoft.NETCore.App 5.0.11 [/usr/share/dotnet/shared/Microsoft.NETCore.App]

To install additional .NET runtimes or SDKs:
https://aka.ms/dotnet-download
```

Рисунок 8 — Встановлений .NET Core 3.1 на ОС Kali Linux

Для перевірки встановлення MS Visual Code необхідно в терміналі виконати команду, внаслідок якої має відкритись вікно (Рис.9) :

\$\> code

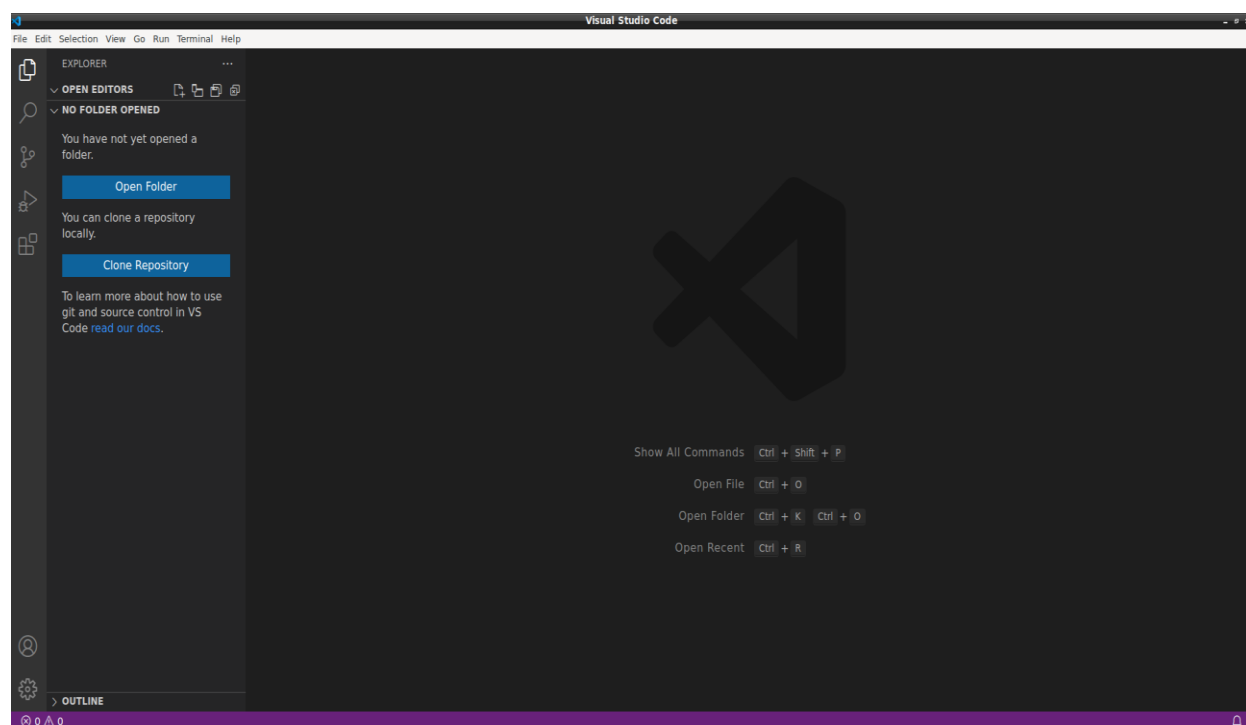


Рисунок 9 — Вікно MS Visual Code

Для зручної роботи з мовою програмування C# бажано встановити розширення мови C# від компанії Microsoft (Рис.10).

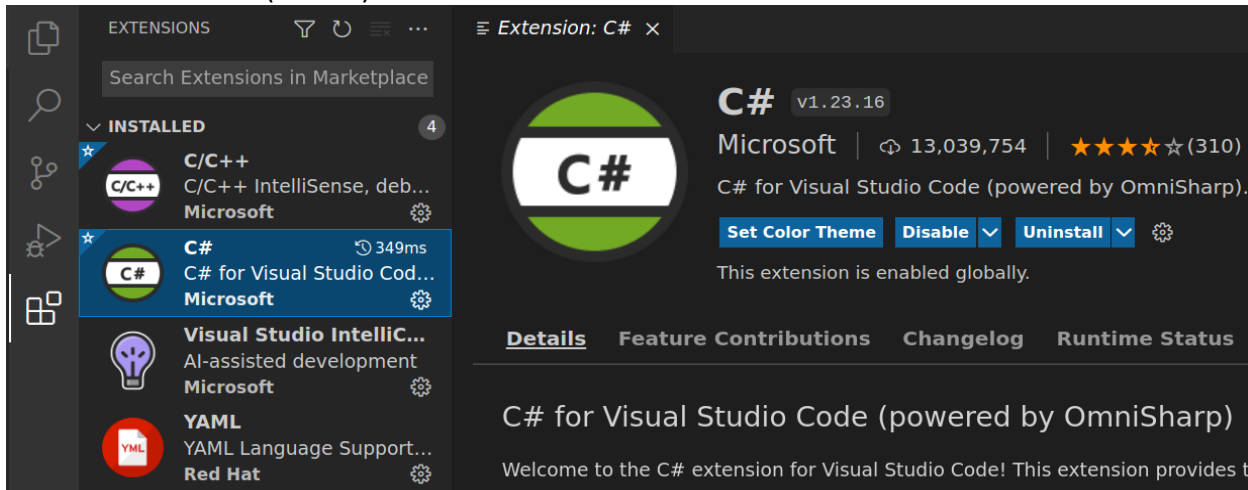


Рисунок 10 — Розширення C# для MS Visual Code

Створення та налаштування проекту .NET Core 3.1

Необхідно створити проект .NET Core 3.1, для чого необхідно обрати каталог в якому буде створено проект, та в терміналі виконати команду [2]:

```
$\> sudo dotnet new console -o testNet -f "netcoreapp3.1" --language "C#"
```

```
dm@dm:~/Desktop/Shared_Folder$ sudo dotnet new console -o testNet -f "netcoreapp3.1" --language "C#"
[sudo] password for dm:
The template "Console Application" was created successfully.

Processing post-creation actions...
Running 'dotnet restore' on testNet/testNet.csproj...
  Determining projects to restore...
  Restored /media/sf_SharedFolder/testNet/testNet.csproj (in 90 ms).
Restore succeeded.
```

Рисунок 11 — Повідомлення про успішне створення проекту .NET Core 3.1

Для подальшої роботи з проектом необхідно в MS Visual Code відкрити створений каталог, за допомогою послідовної комбінації клавіш Ctrl+K, Ctrl+O та потім обрати розташування каталогу. Також, в корінь каталогу зі створеним проектом, необхідно перенести створену бібліотеку класів TestNetStandard.dll. Для автоматизації побудови програми необхідно створити **makefile** (без розширення файлу) в каталозі проекту.

Загальна структура проекту зображена на Рис.12.

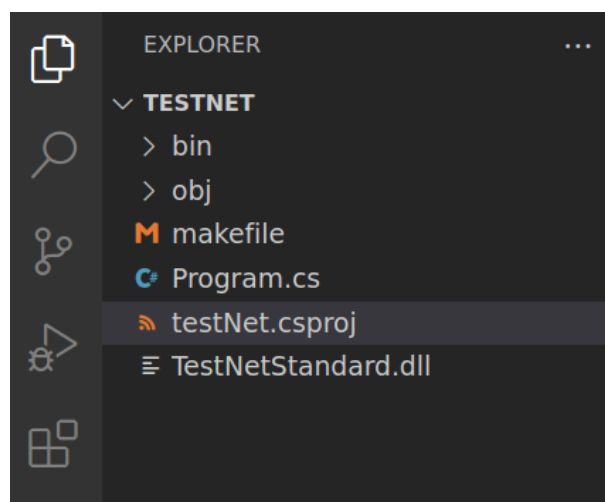


Рисунок 12 — Структура проекту .NET Core 3.1

В файлі з налаштуваннями проекту **testNet.csproj** необхідно підключити бібліотеку класів в проект дописавши код наведений в лістингу 2.

Лістинг 2. Підключення бібліотеки .NET Standard в проект .NET Core 3.1

```
<ItemGroup>
  <Reference Include="TestNetStandard.dll" Version="1.0.0">
    <HintPath>TestNetStandard.dll</HintPath>
  </Reference>
</ItemGroup>
```

Для підключення необхідно вказати назву бібліотеки в атрибуті **Include**, та вказати версію, яка була зазначена під час створення бібліотеки, тобто **1.0.0**.

Тег **HintPath** вказує на точне розташування бібліотеки, оскільки вона розташована на одному рівні з файлом **.csproj**, то достатньо вказати тільки назву.

Повний лістинг коду налаштувань проекту вказані в лістингу 3.

Лістинг 3. Повні налаштування проекту .NET Core 3.1

```
<Project Sdk="Microsoft.NET.Sdk">

  <PropertyGroup>
    <OutputType>exe</OutputType>
    <TargetFramework>netcoreapp3.1</TargetFramework>
    <RuntimeIdentifiers>linux-x64</RuntimeIdentifiers>
  </PropertyGroup>

  <ItemGroup>
    <Reference Include="TestNetStandard.dll" Version="1.0.0">
      <HintPath>TestNetStandard.dll</HintPath>
    </Reference>
  </ItemGroup>

</Project>
```

В якості результуючого типу (тег **OutputType**) вказано **exe**. Цей тип файлу можна отримати тільки після побудови самодостатнього (**self-contained**) додатку, та він означає що додаток буде виконуватись в консольному (текстовому) режимі. Для відлагоджувальних цілей буде створено файл з розширенням **dll**. Можливі також інші типи побудови проекту:

1. **winexe** — призначений для проектів з візуальною частиною (WPF, WinForms тощо);
2. **library** — проекти бібліотеки класів;
3. **module** — представлений PE-файлом з метаданими .NET, але без маніфесту збірки.

За замовчуванням мають розширення **.netmodule**. Модулі неможливо запустити, їх можна подати на вхід спеціальним утилітам.

Особливу увагу варто звернути на тег **RuntimeIdentifiers** [3], оскільки він вказує на використання середовища виконання для платформи, на якій буде виконуватись програма, в даному випадку це **linux-x64**.

Для того, щоб визначити поточну платформу (на основі архітектури системи) можна виконати одну з команд:

```
$\> uname -a
або
$\> lscpu | grep Architecture
```


Або

```
$\> file usr/share/dotnet/dotnet
```

Результати виконання команд зображено на Рис.13.

```
dm@dm:~$ uname -a
Linux dm 5.10.0-kali9-amd64 #1 SMP Debian 5.10.46-4kali1 (2021-08-09) x86_64 GNU/Linux
dm@dm:~$ file /usr/share/dotnet/dotnet
/usr/share/dotnet/dotnet: ELF 64-bit LSB pie executable, x86-64, version 1 (SYSV), dynamically linked, interpreter /lib64/ld-linux-x86-64.so.2
dm@dm:~$ lscpu | grep Architecture
Architecture:                x86_64
```

Рисунок 13 —Результати виконання різних команд

Серед найрозповсюдженіших середовищ виконання можна виділити [3]:

- 1) win-x64 – виконання в 64 розрядних ОС Windows;
- 2) win-x86 – виконання в 32 розрядних ОС Windows;
- 3) win-arm64 – виконання в 64 розрядних ОС Windows для пристроїв з архітектурою arm64. Наприклад, для процесора Microsoft SQ1 в планшеті Microsoft Surface Pro X;
- 4) win81-x64 - виконання в 64 розрядних ОС Windows 8.1 та Server 2012 R2;
- 5) linux-x64 - виконання в 64 розрядних дистрибутивах ОС Linux. Наприклад, CentOS, Debian, Fedora, Ubuntu;
- 6) linux-arm - виконання в дистрибутивах ОС Linux з різною розрядністю, оскільки зазвичай використовується в Embedded пристроях;
- 7) osx-x64 - виконання в Mac OS без прив'язки до версії, але починаючи з MacOS 10.12 Sierra.

Налаштування makefile

Щоб вручну не виконувати команди, в більшості дистрибутивів ОС Linux є засіб під назвою **make**, в який можна передати файл **makefile**, в якому вказані інструкції (в термінології make вони називаються **рецептами** (recipes)), які необхідно виконати [4]. В лістингу 4 відображено команди, які необхідно для виконання та очистки проекту.

Лістинг 4. Команди makefile

run:

```
sudo dotnet restore
sudo dotnet clean
sudo dotnet publish -c Release -r linux-x64
sudo dotnet publish -c Release -r linux-x64
sudo dotnet ${PWD}/bin/Release/netcoreapp3.1/linux-x64/testNet.dll
```

clear:

```
sudo dotnet restore
sudo dotnet clean
```

Тепер для виконання проекту достатньо знаходитись в каталозі с проектом, та виконати команду в терміналі:

```
$\> make run
```

Під час виконання можливе виникнення помилки зображеної на Рис.14. Для вирішення цієї проблеми необхідно ще раз виконати команду `make run`.

```
Copyright (C) Microsoft Corporation. All rights reserved.

Determining projects to restore...
All projects are up-to-date for restore.
/usr/share/dotnet/sdk/3.1.414/Sdks/Microsoft.NET.Sdk/targets/Microsoft.NET.Sdk.targets(424,5): error NETSDK1029: Unable to use '/usr/share/dotnet/packs/Microsoft.NETCore.App.Host.linux-x64/3.1.20/runtimes/linux-x64/native/apphost' as application host executable as it does not contain the expected placeholder byte sequence '63-33-61-62-38-66-66-31-33-37-32-30-65-38-61-64-39-30-34-37-64-64-33-39-34-36-36-62-33-63-38-39-37-34-65-35-39-32-63-32-66-61-33-38-33-64-34-61-33-39-36-30-37-31-34-63-61-65-66-30-63-34-66-32' that would mark where the application name would be written. [/media/sf_SharedFolder/testNet/testNet.csproj]
```

Рисунок 14 — Помилка під час побудови проекту

Для очищення проекту (очищення папки `bin`, видалення тимчасових файлів і т.д.) достатньо виконати команду:

```
$\> make clear
```

Налаштування `global.json`

Файл `global.json` необхідний коли встановлено більше однієї версії .NET. Наприклад, .NET 5 та .NET Core 3.1. За замовчуванням під час побудови та виконання проекту використовується найновіша версія. Для того, щоб вказати яку версію необхідно використовувати необхідно створити даний файл в корені каталогу проекту.

Щоб переглянути всі встановлені версії .NET SDK, необхідно виконати команду (Рис.15):

```
$\> sudo dotnet --list-sdks
```

```
dm@dm:~/Desktop/Shared_Folder/testNet$ sudo dotnet --list-sdks
[sudo] password for dm:
3.1.414 [/usr/share/dotnet/sdk]
5.0.402 [/usr/share/dotnet/sdk]
```

Рисунок 15 — Перегляд встановлених .v в системі NET SDK

Для того щоб використовувати версію 3.1.414 необхідно використати код наведений в лістингу 5. Властивість **`rollForward`** вказує, яка повинна обиратись версія, якщо не має доступу до версії вказаної в властивості **`version`** [5].

Лістинг 5. `global.json`

```
{
  "sdk": {
    "version": "3.1.414",
    "rollForward": "disable"
  }
}
```

Використання бібліотеки .NET Standard 2.1

Після підключення бібліотеки та налаштування проекту, необхідно відкрити файл коду **`Program.cs`**, та використовуючи директиву **`using`** підключити простір імен, який використовується в бібліотеці, тобто **`TestNetStandard`**. Після цього, можна використовувати клас **`Validation`**, який реалізований в бібліотеці. Повний код програми наведено в лістингу 6.

Лістинг 6. Основний код програми. Program.cs

```
/*
Email validation via netlibrary 12.10.2021
*/
using System;

// include .NET Standard library compiled in Windows
using TestNetStandard;

namespace testNet
{
    enum MenuActions : byte
    {
        VALIDATE = 1,
        EXIT = 2
    }

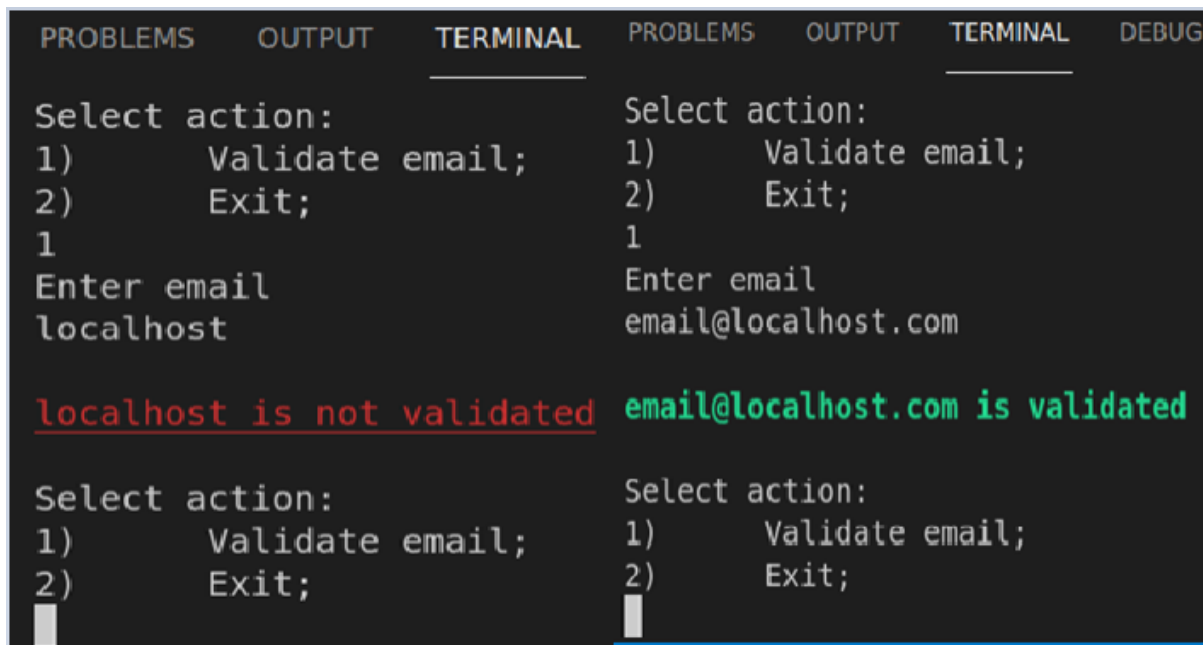
    class Program
    {
        static void Main(string[] args)
        {
            while (true)
            {
                Menu();
            }
        }

        private static void Menu()
        {
            Console.WriteLine("Select action:\n1)\tValidate email;\n2)\tExit;");
            var action = (MenuActions)Convert.ToByte(Console.ReadLine());
            switch (action)
            {
                case MenuActions.VALIDATE:
                    ExecValidation();
                    break;
                case MenuActions.EXIT:
                    Environment.Exit(0);
                    break;
            }
        }

        private static bool ExecValidation()
        {
            Console.WriteLine("Enter email");
            string email = Console.ReadLine();

            // Using Validation class from included library
            return Validation.Validate(email,
                () => { Console.WriteLine($"{"\n\x1b[32;1m{email} is validated\x1b[0m\n"}"); },
                () => { Console.WriteLine($"{"\n\x1b[31;4m{email} is not validated\x1b[0m\n"}"); });
        }
    }
}
```

Щоб виконати програму в MS Visual Code необхідно натиснути комбінацію клавіш **Ctrl+J**, перейти на вкладку **Terminal** та виконати команду **make run**. Результат успішного виконання команди можна побачити на Рис. 16.



```
PROBLEMS  OUTPUT  TERMINAL  PROBLEMS  OUTPUT  TERMINAL  DEBUG
Select action:
1) Validate email;
2) Exit;
1
Enter email
localhost

localhost is not validated
Select action:
1) Validate email;
2) Exit;
█

Select action:
1) Validate email;
2) Exit;
1
Enter email
email@localhost.com

email@localhost.com is validated
Select action:
1) Validate email;
2) Exit;
█
```

Рисунок 16 — Виконана програма з використанням .NET Standard

Загальні завдання

1. Ознайомитися з теоретичною частиною.
2. Створити новий репозиторій на GitHub під поточну лабораторну роботу;
3. Перейти в локальний репозиторій з лабораторними роботами;
4. Створити окрему гілку (git checkout -b) в системі контролю версій Git для поточної лабораторної роботи;
5. В локальному репозиторії створити новий проект з цільовою платформою .NET Standard 2.1;
6. Перенести код з попередньої лабораторної роботи в новий проект;
7. Зкомпілювати код;
8. Використовуючи іншу ОС (за допомогою віртуальної машини, чи іншому фізичному пристрої), створити консольний проект .NET Core;
9. Додати та підключити створену .NET Standard-бібліотеку в проект .NET Core;
10. Продемонструвати використання .NET Standard-бібліотеки в проекті .NET Core на іншій ОС;
11. Проект .NET Core створений під іншою ОС, перенести в локальний репозиторій;
12. Продемонструвати роботу програми;
13. Результат роботи програми зберегти у вигляді скріншоту (png або jpeg) в репозиторії з проектом;
14. Індексувати обидва проекти (.NET Standard та .NET Core) (git add);
15. Зафіксувати зміни (git commit);
16. Надіслати зміни у віддалений репозиторій (git push);
17. Створити та виконати запит на зміни (pull request);
18. Надіслати посилання на поточну лабораторну роботу у віддаленому репозиторії в GitHub.

Контрольні запитання

1. Що таке платформа?
2. .NET Standard. Призначення. Відмінність від інших цільових платформ.
3. Для чого потрібні метадані збірки?
4. Як створити новий консольний проект за допомогою dotnet?
5. Для чого призначені файли .csproj?
6. Призначення makefile
7. Призначення та роль файлу global.json

Перелік рекомендованих джерел

1. Install the .NET SDK or the .NET Runtime on Ubuntu. — 2021. — Mode of access: <https://docs.microsoft.com/en-us/dotnet/core/install/linux-ubuntu> Date of Access: 04.11.2021.
2. dotnet new. — 2021. — Mode of access: <https://docs.microsoft.com/en-us/dotnet/core/tools/dotnet-new> Date of Access: 04.11.2021.
3. .NET RID Catalog. — 2021. — Mode of access: <https://docs.microsoft.com/en-us/dotnet/core/rid-catalog> Date of Access: 04.11.2021.
4. GNU Make. — 2021. — Mode of access: <https://www.gnu.org/software/make/manual/make.pdf> Date of Access: 04.11.2021.
5. global.json overview. — 2021. — Mode of access: <https://docs.microsoft.com/en-us/dotnet/core/tools/global-json?tabs=netcore3x> Date of Access: 04.11.2021.
6. .NET. Free. Cross-platform. Open Source. A developer platform for building all your apps. — 2021. — Mode of access: <https://dotnet.microsoft.com/en-us/> Date of Access: 04.11.2021.
7. Введение в C#. Язык C# и платформа .NET. — 2021. — Mode of access: <https://metanit.com/sharp/tutorial/1.1.php> Date of Access: 04.11.2021.