МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Волинський національний університет імені Лесі Українки
Кафедра комп’ютерних наук та кібербезпеки

БАЗОВІ ПОНЯТТЯ .NET

Конспект лекцій

Луцьк-2023

УДК 004.4'2
Б 90

Рекомендовано до видання науково-методичною радою
Волинського національного університету імені Лесі Українки
(протокол № 4 від 19 грудня 2022 р.)

Рецензенти:

Собчук О. М. – кандидат пед. наук, доцент кафедри загальної математики та методики навчання інформатики Волинського національного університету імені Лесі Українки

Здолбіцька Н.В. – кандидат технічних наук, доцент кафедри комп'ютерних наук, ЛНТУ

Б 90 Базові поняття .Net [Електронний ресурс] : Конспект лекцій. / укладачі: В. В. Булатецький, Л. В. Булатецька; ВНУ ім. Лесі Українки. – Електронні текстові данні (1 файл: 0,99 Мбайт). Луцьк : ВНУ ім. Лесі Українки, 2023. 37 с.

Навчальне видання призначене для здобувачів вищої освіти, які вивчають сучасні методи програмування у вищих навчальних закладах. Рекомендовано здобувачам, спеціальності 122 Комп'ютерні науки, освітньо-професійної програми Комп'ютерні науки та інформаційні технології.

УДК 004.4'2

ЗМІСТ

ВСТУП

Вивчення .NET є актуальним на сьогоднішній час, так як .NET є найпродуктивнішою платформою для розробників. Додатки, розроблені на .NET можна писати, запускати та створювати на кількох платформах, включаючи Windows, Linux і macOS. За допомогою .NET можна створювати додатки для мобільних пристроїв, настільних комп'ютерів, хмари, тощо. .NET охоплює сучасні мовні конструкції, що постійно розвиваються – універсальні шаблони, Language Integrated Query (LINQ), асинхронне програмування, великі бібліотеки, API інтерфейси, багатомовну підтримку, та багато іншого. .NET – досить конкурентне рішення для сучасної веб-розробки, особливо серверної частини. Величезну кількість сучасних сайтів та застосунків написано з використанням платформи .NET. Використання менеджера пакетів NuGet для .NET дозволяє розробникам обмінюватися бібліотеками з відкритим кодом. NuGet.org, безкоштовна служба Microsoft для розміщення пакетів NuGet, є основним хостом для загальнодоступних пакетів NuGet, пропонує багато популярних пакетів від спільноти розробників.

Пропонований текст лекцій підготовлено відповідно до освітньо-професійної програми Комп'ютерні науки та інформаційні технології підготовки бакалавра галузі знань 12 Інформаційні технології, спеціальності 122 Комп'ютерні науки.

1. Історія .NET

У 1995 році компанія Sun Microsystems розробила високорівневу об’єктно-орієнтовану мову програмування Java. Особливістю цієї мови було те, що програми на Java транслюються в байт-код, що виконується віртуальною машиною java (Java Virtual Mashine. JVM) незалежно від комп’ютерної архітектури. Враховуючи те, що JVM відносно проста для реалізації, вона швидко стала доступною для великої кількості середовищ, що забезпечило кросплатформність розроблених додатків. Однією з проблем, яку не вирішили розробники Java це проблема багатомовного програмування (mixed-language programming), коли програми, написані на різних мовах, могли б успішно працювати одна з іншою. Така взаємодія необхідна для створення великих систем з розподіленим програмним забезпеченням, а також для програмування компонентів програмного забезпечення, оскільки найціннішим є компонент, який можна використовувати у широкому діапазоні комп’ютерних мов і операційних середовищ.

У 1999 році фірма Microsoft почала розробляти свою альтернативу Java, а в 2002 році було оголошено про створення нової платформи програмування, що отримала назву .NET Framework. Вона дозволяє розробляти проекти, частини яких можуть бути виконані у різних мовах програмування. Основний компонент .NET – це загальномовне середовище виконання CLR (Common Language Runtime), яке може працювати з різними мовами програмування, наприклад, C#, F#, J#, Ada, C++ тощо.

Подібно до технології Java, середовище розробки .NET, теж створює байт-код призначений для виконання віртуальною машиною, яка носить назву .NET Runtime. Так само, як і байт-код для віртуальної java-машини програма для середовища .NET Runtime може виконуватися будь-якою операційною системою, в якій NET Runtime встановлена. Насьогодні кількість операційних систем, де може виконуватись .NET Runtime, постійно зростає.

Реалізації .NET. Реалізації .NET включають .NET Framework, .NET 5+ (і .NET Core) і Mono.

Кожна реалізація .NET включає такі компоненти:

- одне або кілька середовищ виконання, наприклад .NET Framework CLR і .NET 5 CLR;
- бібліотека класів, наприклад, бібліотека базових класів .NET Framework і бібліотека базових класів .NET 5;
- до .NET Framework і .NET 5+ включено одну або кілька інфраструктур додатків, наприклад ASP.NET, Windows Forms і Windows Presentation Foundation (WPF);
- засоби розробки, які спільно використовуються між кількома реалізаціями.

Microsoft підтримує чотири реалізації .NET:

- .NET 5 (.NET Core) і новіші версії;
- .NET Framework;
- Mono;
- UWP.

.NET 5 і новіші версії. .NET 5+, який раніше називався .NET Core, – це кросплатформна реалізація .NET, розроблена для обробки серверних і хмарних робочих навантажень у масштабі. Також підтримує інші робочі навантаження, включаючи настільні програми. Працює на Windows, macOS і Linux. Реалізує .NET Standard, тому код, націлений на .NET Standard, може працювати на .NET 5+. ASP.NET Core , Windows Forms і Windows Presentation Foundation (WPF) працюють на .NET 5+. .NET 7 – остання версія цієї реалізації .NET.

.NET Framework – це реалізація .NET, яка існує з 2002 року. Версії 4.5 і пізніші реалізують .NET Standard, тому код, націлений на .NET Standard, може працювати на цих версіях .NET Framework. Містить додаткові API, специфічні для Windows, наприклад API для розробки настільних додатків Windows за

допомогою Windows Forms і WPF. .NET Framework оптимізовано для створення настільних додатків Windows.

Mono – це реалізація .NET, яка в основному використовується, коли потрібен невеликий час виконання. Це середовище виконання, яке підтримує додатки Xamarin на Android, macOS, iOS, tvOS і watchOS і орієнтоване насамперед на невеликому розмірі. Mono також підтримує ігри, створені за допомогою механізму Unity. Mono підтримує всі наразі опубліковані версії .NET Standard.

UWP (Універсальна платформа Windows) — це реалізація .NET, яка використовується для створення сучасних додатків Windows із сенсорним керуванням і програмного забезпечення для Інтернету речей (IoT). Його розроблено для об'єднання різних типів пристроїв, зокрема ПК, планшетів, телефонів і навіть Xbox. UWP надає багато служб, наприклад централізований магазин програм, середовище виконання (AppContainer) і набір Windows API для використання замість Win32 (WinRT). Програми можна писати на C++, C#, Visual Basic і JavaScript.

2. Базові поняття технології .NET Framework

.NET Framework – це технологія, яка підтримує створення та запуск веб-служб і додатків Windows. .NET Framework призначений для виконання наступних завдань:

- забезпечення узгодженого об'єктно-орієнтованого середовища програмування для локального збереження і виконання об'єктного коду, для локального виконання коду, розподіленого в мережі, або для віддаленого виконання;
- надання середовища виконання коду, яке мінімізує конфлікти в процесі розгортання програмного забезпечення та керування версіями, гарантує безпечне виконання коду, включаючи код, створений стороннім

виробником, усуває проблеми з продуктивністю середовищ виконання скриптів або інтерпретованого коду;

- забезпечення єдиного принципу розробки для різних типів програм, таких як додатки Windows та веб-додатки;
- забезпечення взаємодії на основі промислових стандартів, які гарантують інтеграцію коду платформи .NET Framework з будь-яким іншим кодом.

.NET Framework – сукупність множини служб і компонентів, які дозволяють розробленим на мові C# (а також будь-якій іншій .NET-підтримуваній мові високого рівня) додаткам працювати в середовищі Windows або іншої ОС.

.NET Framework складається з загальномовного середовища виконання (CLR) і бібліотеки класів .NET Framework. Спільне середовище виконання для багатьох мов є основою .NET Framework, яке виконує код і надає служби, які полегшують процес розробки. Середовище виконання керує кодом під час виконання, надаючи основні послуги, такі як керування пам'яттю, керування потоками. Поняття управління кодом є основним принципом середовища виконання. Код, націлений на середовище виконання, відомий як керований код, тоді як код, який не націлений на середовище виконання, не був розрахований на виконання під .NET, відомий як некерований код. Але ці поняття стосуються тільки самої .NET.

Отже, у **CLR** є взаємодія з операційною системою. **CLR** також має доступ до бібліотеки базових класів .NET Framework і користувацьких класів. Призначені для користувача класи можуть базуватися на базових класах.

Бібліотека класів – це комплексна, об'єктно-орієнтована колекція типів, які використовуються для розробки додатків, починаючи від традиційних додатків, які запускаються з командного рядка або додатків з графічним інтерфейсом (GUI) і закінчуючи додатками на основі останніх інновацій, наданих ASP.NET, таких як веб-форми та веб-служби XML.

.NET Framework може бути розміщений некерованими компонентами, які завантажують CLR в свої процеси та ініціюють виконання керованого коду, тим самим створюючи програмне середовище, яке використовує як керовані, так і некеровані функції. .NET Framework не тільки надає кілька хостів виконання, але й підтримує розробку сторонніх хостів виконання.

Наприклад, ASP.NET розміщує середовище виконання і забезпечує масштабоване середовище для керованого коду на стороні сервера. ASP.NET працює безпосередньо з середовищем виконання, щоб забезпечити виконання додатків ASP.NET та веб-служби XML.

Браузер Internet Explorer – це приклад некерованого додатку, у якому розміщено середовище виконання (у вигляді розширення типу MIME (Multipurpose Internet Mail Extensions)). Використання браузера Internet Explorer для розміщення середовища виконання дає змогу вбудовувати керовані компоненти або елементи керування Windows Forms у документи HTML.

На рис. 1 показано зв’язок середовища CLR та бібліотеки класів із програмами та загальною системою. На рис. 1 також показано, як керований код працює в рамках більшої архітектури.

3. Механізм роботи CLR.

CLR є деякою програмною оболонкою, яка керує виконанням коду програми. Це керування проявляється в пам’яті, в потоках додатків, у віддаленій взаємодії, а також в строгому контролі відповідності виконуваного коду переліку вимог. Тобто, CLR – це набір деяких служб, які і виконують наш код.

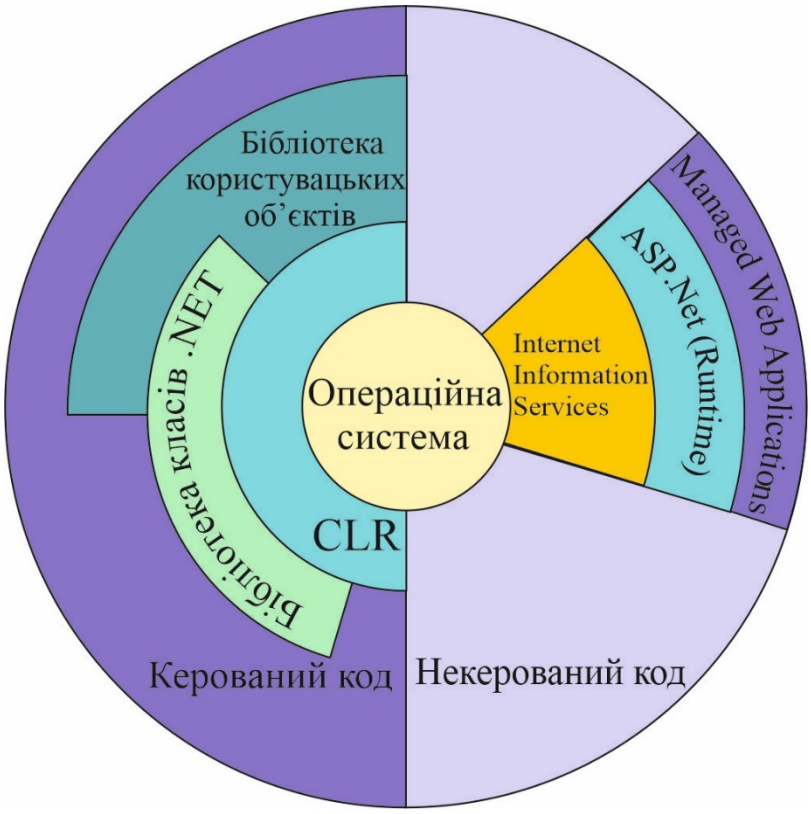

Рис. 1. Зв'язки між керованим та некерованим кодом, бібліотеками, CLR та операційною системою

Середовище розробки .NET, створює байт-код призначений для виконання віртуальною машиною .NET Runtime. Програма для середовища .NET Runtime може виконуватися будь-якою операційною системою, в якій ***NET Runtime*** встановлена.

Вхідна мова цієї машини в .NET називається MSIL (Microsoft Intermediate Language – проміжна мова Microsoft). Іноді використовується інша назва проміжної мови – CIL (Common Intermediate Language – загальноприйнята проміжна мова), або просто IL (проміжна мова).

Найбільш повну підтримку проміжної машинної мови CIL, реалізовано в операційних системах Microsoft починаючи з Windows XP. Цікавим є проект Mono. Це проект повноцінного використання системи .NET на базі вільного програмного забезпечення. Mono, платформа розробки з відкритим вихідним кодом, заснована на .NET Framework, дозволяє розробникам створювати

кросплатформні програми. Реалізація .NET Mono базується на стандартах ECMA для C# та Common Language Runtime.

Суттєвою позитивною відмінністю Microsoft .NET від існуючих аналогів на сучасному ринку програмного забезпечення є універсальна система типізації. В ході компіляції програма на .NET-сумісній мові програмування трансформується відповідно до заздалегідь заданої узагальненої специфікації мови Common Type System (CTS). CTS – стандарт, що задає правила визначення типів (рис. 2.). Система типів CTS повністю описує всі типи даних, підтримувані середовищем виконання, визначає їх взаємозв'язок. Система типізації Microsoft .NET являє собою частково впорядковану множину, яку на якісному рівні можна вважати як ISA-ієрархія (ISA походить від англійських слів "is a", які означають «є одним з»). Таким чином, система типів Microsoft .NET утворює ієрархію із зростанням множини знизу до верху, в якій явно виділяються дві великі групи типів, а саме, типи-посилання і типи-значення. Відмінність в них полягає в особливостях їх виклику в процедурах: по імені або за значенням (call-by-name, CBN) і за посиланням (call-by-reference, CBR).

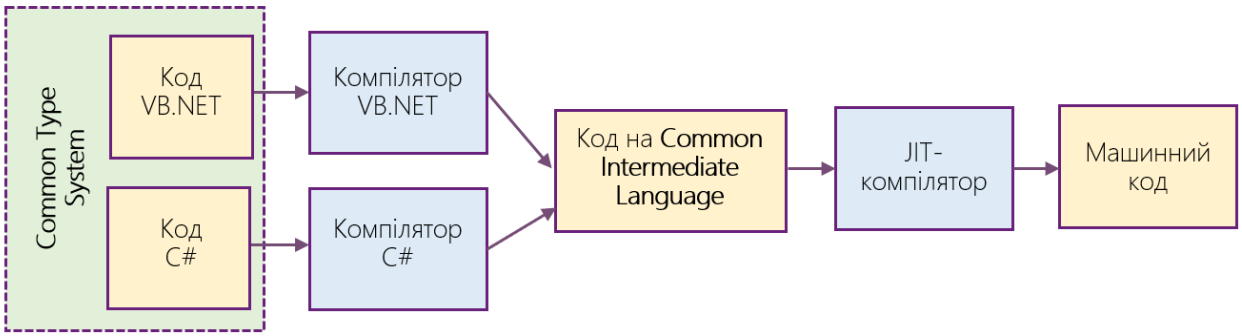

Рис. 2. Common Type System

Система типізації Microsoft .NET крім розвиненої ієрархії визначених типів дозволяє користувачеві створювати власні типи (як типи-посилання, так і типи-значення) на основі вже існуючих. .NET концепція потенційно

підтримує довільні мови програмування. Ці мови не можуть містити специфічних типів, що не підтримуються CTS.

Common Language Specification (CLS) – мінімальний набір правил, що визначають підмножину узагальнених типів даних, у відношенні яких гарантується, що вони безпечні при використанні у всіх мовах .NET. Цим правилам повинна задовольняти кожна мова (рис. 3.). Модуль на C# може викликати тільки ті методи модуля на VB.NET, які написані відповідно до CLS (рис. 3).

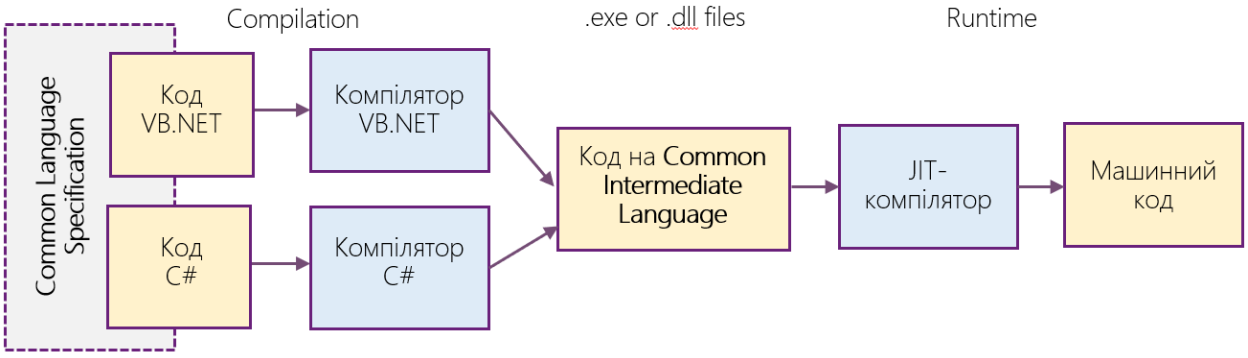

Рис. 3. Common Language Specification

Отже процес керованого виконання включає наступні кроки (рис. 4):

- **вибір компілятора** (щоб скористатися перевагами середовища CLR, необхідно використовувати один або декілька мовних компіляторів, які звертаються до середовища виконання);
- **компіляція коду в MSIL** (при компіляції вихідний код перетвориться в MSIL і створюються необхідні метадані);
- **компіляція інструкцій MSIL в машинний код** (під час виконання JIT-компілятор перетворює інструкції MSIL в машинний код, під час цієї компіляції виконується перевірка коду і метаданих MSIL з метою встановити, чи можна для них визначити, чи є код типобезпечним);
- **виконання коду** (середовище CLR надає інфраструктуру, яка забезпечує виконання коду, і ряд служб, які можна використовувати при виконанні).

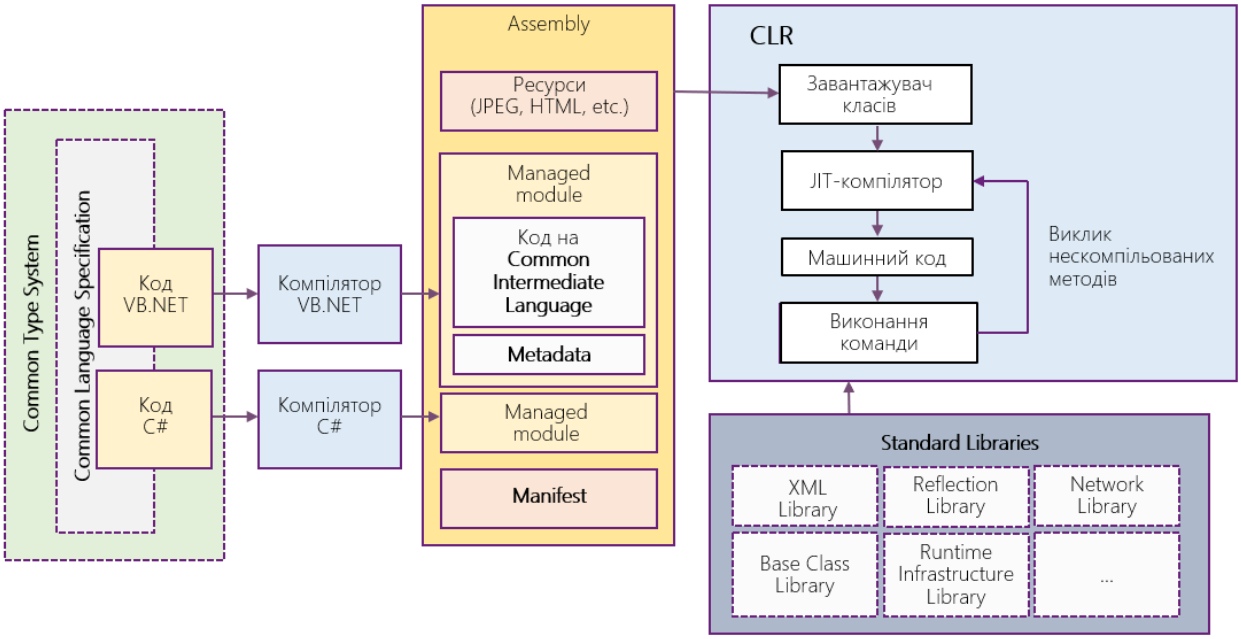

Рис. 4. Загальна схема перетворення мови програмування в машинний код

Вибір компілятора. Якщо ми маємо вихідний код, написаний на одній із мов які підтримуються .Net і проект необхідно передати на запуск, то перш за все, необхідно скомпілювати цей код в проміжний код. Щоб скористатися перевагами, наданими середовищем CLR, необхідно застосувати один або кілька мовних компіляторів, орієнтованих на середовище виконання, наприклад компілятор Visual Basic, C#, Visual C++, F#. Можна також використовувати компілятори від незалежних розробників, наприклад компілятор Eiffel, Perl або COBOL, але тільки той який відповідає тій .NET-сумісній мові, на якій писався вихідний код програми. Якщо було використано мову C#, то і компілювати потрібно C# -компілятором. Оскільки середовище виконання є багатомовним, воно підтримує широкий набір різноманітних типів даних і мовних засобів. Доступні засоби середовища виконання визначаються використовуваним мовним компілятором, і розробники створюють код з використанням цих засобів.

Використовуваний в коді синтаксис визначається компілятором, а не середовищем виконання**.** Щоб компонент повинен бути повністю доступний для компонент, написаних на інших мовах, то експортовані цим компонентом типи повинні надавати виключно мовні функції, включені до складу специфікації CLS (CLS). Атрибут CLSCompliantAttribute дозволяє гарантувати, що код є CLS-сумісним.

Компіляція в MSIL. При компіляції в керований код компілятор перетворює вихідний код в проміжну мову Microsoft (MSIL), що представляє собою незалежний від процесора набір інструкцій, який можна ефективно перетворити в машинний код. Мова MSIL включає інструкції для: завантаження, збереження, ініціалізації і виклику методів для об'єктів, а також інструкції для арифметичних і логічних операцій, потоків управління, прямого доступу до пам'яті, обробки виключень і інших операцій.

На даному етапі буде отримано проміжний код у вигляді виконуваних файлів (.exe), або файлів бібліотеки динамічних посилань (.dll). Ці файли називаються збірками. Збірка – це набір типів і ресурсів, створених для спільної роботи та формування логічної функціональної одиниці. Вони надають загальномовному середовищу виконання інформацію, необхідну для реалізації типів.

Проміжна компіляція вихідного коду необхідна завдяки підтримці платформою .NET вихідного коду безлічі різних мов програмування. Таким чином, незалежно від того, якою мовою було написано додаток, при його запуску CLR завжди отримує практично ідентичний MSIL-код (збірки). Це дає перевагу при виборі мови для розробки, а також використання декількох мов при створенні одного проекту. При цьому використовуються всі переваги середовища виконання, бібліотеки класів і компонентів, написаних іншими розробниками іншими мовами.

Коли компілятор створює код MSIL, одночасно створюються метадані. Метадані містять опис типів в коді, включаючи визначення кожного типу,

сигнатури кожного члена типу, члени, на які є посилання в коді, а також інші відомості, що використовуються середовищем виконання під час виконання. MSIL і метадані містяться в переносимому виконуваному (PE) файлі, який ґрунтується на форматах Microsoft PE і COFF, що раніше використовувалися для виконуваного контенту, але при цьому розширює їх можливості.

PE- Portable Executable – формат виконуваних файлів, об'єктного коду та динамічних бібліотек, використовуваний в 32- і 64-бітових версіях операційної системи Microsoft Windows. Формат PE є структурою даних, що містить всю інформацію, необхідну PE-завантажувачу для відображення файлу в пам'ять. Виконуваний код включає в себе посилання для зв'язування динамічно-завантажуваних бібліотек, таблиці експорту і імпорту API функцій, дані для управління ресурсами і дані локальної пам'яті потоку (TLS). В операційних системах сімейства Windows NT формат PE використовується для EXE, DLL, SYS (драйверів пристроїв) та інших типів виконуваних файлів.

Цей формат файлів, що дозволяє розміщувати код MSIL або машинний код, а також метадані, дозволяє операційній системі розпізнавати образи середовища CLR. Наявність у файлі метаданих поряд з MSIL дозволяє коду описувати себе. Це усуває потребу в бібліотеках типів і у мові IDL (Interactive Data Language – мова програмування для аналізу даних, використовується в астрономії та медичній візуалізації). Середовище виконання знаходить і видобуває метадані з файлу в міру необхідності при виконанні.

Компіляція MSIL в машинний код. Перед запуском MSIL його необхідно скомпілювати в машинний код в середовищі CLR для архітектури кінцевого комп'ютера. Платформа .NET Framework надає два способи такого перетворення:

- JIT-компілятор платформи .NET Framework;
- .NET Framework ***ngen.exe*** (генератор образів в машинному коді).

Компіляція з допомогою JIT-компілятора. При JIT-компіляції (рис. 4) код MSIL перетвориться в машинний код під час виконання додатка за вимогою, коли завантажується і виконується вміст збірки. При безпосередньому запуску на виконання файлів збірки, CLR аналізує проміжний код і завантажує в пам’ять необхідні класи.

Оскільки середовище CLR надає JIT-компілятор для кожної підтримуваної архітектури процесора, розробники можуть створювати набір збірок MSIL, які можуть компілюватися за допомогою JIT-компілятора і виконуватися на різних комп’ютерах з різною архітектурою. За допомогою JIT-компілятора проміжний код перетворюється в набір інструкцій для конкретний платформи. І тільки після цього отримані інструкції запускаються на виконання (рис. 4). Якщо *керований код* викликає специфічний для платформи машинний API або бібліотеку класів, то він буде виконуватися тільки у відповідній операційній системі.

При JIT-компіляції враховується можливість того, що певний код може ніколи не викликатися під час виконання. Щоб не витрачати час і пам’ять на перетворення всього, що міститься в PE-файл, при компіляції MSIL перетворюється в машинний код в міру необхідності під час виконання. Отриманий таким чином машинний код зберігається в пам’яті, що дозволяє використовувати його при подальших викликах в контексті цього процесу. Завантажувач створює і приєднує *заглушки* до кожного методу в типі, коли тип завантажується і ініціалізується. При першому виклику методу заглушка передає управління JIT-компілятору, який перетворює MSIL для цього методу в машинний код і замінює заглушку на створений машинний код. Тому наступні виклики методу, скомпільованого за допомогою JIT-компілятора, ведуть безпосередньо до машинного коду.

JIT-компілятор не компілює відразу весь код. Компіляція відбувається по частинам, в разі потреби. Тобто, компілюється та ділянку коду, яка безпосередньо в даний момент повинна піти на виконання. Одного разу

скомпілювавши деякий шматок коду, компілятор не видаляє його інструкції з пам’яті, а зберігає на випадок повторного виклику. Таким чином, одного разу скомпільований метод більше не компілюється і всі його наступні неодноразові виклики безпосередньо звертаються до вже машинного коду. JIT-компіляція відбувається кожного разу при запуску програми, адже вихідні виконувані файли програми зберігаються у вигляді збірок (.exe, .dll файлів). Тобто кожен раз, при запуску програми, CLR на льоту підключає необхідні класи і виконує JIT-компіляцію наданих йому збірок (рис. 4).

Створення коду під час установки за допомогою NGen.exe. Той факт, що JIT-компілятор перетворює MSIL-код збірки в машинний код при виклику окремих методів, визначених у цій збірці, негативно позначається на продуктивності під час виконання. У більшості випадків зниження продуктивності не є важливим. Що важливіше, код, створений JIT-компілятором, буде прив’язаний до процесу, який викликав компіляцію. Його не можна зробити спільним для декількох процесів. Щоб створений код можна було використовувати в декількох викликах додатків або в декількох процесах, які спільно використовують набір збірок, середовище CLR надає режим попередньої компіляції. У такому режимі компіляції для перетворення збірок MSIL в машинний код в стилі JIT-компілятора використовується програма **Ngen.exe** (генератор образів в машинному коді). Однак, робота Ngen.exe відрізняється від JIT-компілятора в трьох аспектах:

- Ngen.exe виконує перетворення з MSIL-коду в машинний код перед виконанням програми, а не під час його виконання;
- при цьому збірка компілюється повністю, а не по одному методу за раз;
- він зберігає створений код в кеші засобу виконання машинного коду у вигляді файлу на диску.

Перевірка коду. В процесі компіляції в машинний код MSIL-код повинен пройти перевірку, якщо тільки адміністратор не встановив політику безпеки, що дозволяє пропустити перевірку коду. MSIL-код і метадані

перевіряються на типобезопеку. Це означає, що код повинен звертатися тільки до тих адрес пам’яті, до яких йому дозволено доступ. Типобезпека допомагає ізолювати об’єкти один від одного і сприяє їх захисту від ненавмисного або зловмисного пошкодження. Вона також гарантує надійне застосування умов безпеки для коду.

Середовище виконання використовує істинність наступних тверджень для коду, що піддається типобезпеці:

- посилання на тип строго сумісна з типом, що адресується;
- для об’єкта викликаються тільки правильно визначені операції;
- посвідчення є справжніми.

У процесі перевірки коду MSIL робиться спроба підтвердити, що код може отримувати доступ до розміщення в пам’яті і викликати методи тільки через правильно визначені типи. Наприклад, код не повинен дозволяти доступ до полів об’єкта так, щоб можна було виходити за межі області розміщення його у пам’яті. Крім того, перевірка визначає, чи правильно був створений код MSIL, оскільки невірний код MSIL може призводити до порушення правил суворої типізації. У процесі перевірки передається правильно визначений типобезпечний код. Однак іноді типобезпечний код може не пройти перевірку через обмеження процесу перевірки, а деякі мови за своєю структурою не дозволяють створювати код, що піддається перевірці на типобезпеку. Якщо відповідно до політики безпеки використання типобезпечного коду є обов’язковим і код не проходить перевірку, то при виконанні коду можна створити виняток.

Виконання коду. Середовище CLR це інфраструктура, що забезпечує кероване виконання коду, і ряд служб, які можна використовувати при виконанні. Перед виконанням методу його необхідно скомпілювати в код для конкретного процесора. Кожен метод, для якого створено MSIL-код, компілюється за допомогою JIT-компілятора при першому виклику і потім запускається. При наступному виклику методу буде виконуватися існуючий

JIT-скомпільований код. Процес JIT-компіляції та подальшого виконання коду повторюється до завершення виконання.

Під час виконання для керованого коду доступні такі служби, як збирач сміття, забезпечення безпеки, взаємодія з некерованим кодом, підтримка налагодження на декількох мовах, а також підтримка розширеного розгортання і управління версіями.

У сучасних ОС Microsoft Windows завантажувач операційної системи виконує пошук керованих модулів шляхом аналізу біта в заголовку COFF (*Common Object File Format (COFF) – формат виконуваних файлів, файлів об'єктного коду та динамічних бібліотек*). Встановлений біт позначає керований модуль. При виявленні керованих модулів завантажується бібліотека Mscoree.dll, а підпрограми CorValidateImage і CorImageUnloading повідомляють завантажувач про завантаження та вивантаження образів керованих модулів. Підпрограма _CorValidateImage виконує наступні дії:

- перевіряє, чи є код допустимим керованим кодом;
- замінює точку входу в образі на точку входу в середовищі виконання;

У 64-розрядних системах Windows CorValidateImage змінює образ, що знаходиться в пам'яті, шляхом перетворення його з формату PE32 в формат PE32+.

4. Збірки в .NET

Збірки являють собою базові елементи розгортання, управління версіями, повторне використання, призначення областей активації та правовий доступ для програм на основі платформи .NET. Збірка являє собою колекцію типів і ресурсів, зібраних для спільної роботи та являє собою логічну функціональну одиницю. Збірки створюються у вигляді виконуваного файлу (exe) або файлу бібліотеки динамічної компоновки (dll) і є стандартними

блоками додатків .NET. Вони надають відомості для середовища CLR, які потрібні для розпізнавання реалізацій типів.

У .NET і .NET Framework збірку можна створити з одного або кількох файлів. Часто ці файли містять код, але до складу збірки також можуть входити графічні зображення, ресурси та інші бінарні дані, асоційовані з кодом. Збірки .NET Framework можуть містити один або кілька модулів. Завдяки цьому у великих проектах кілька розробників можуть працювати з окремими файлами або модулями вихідного коду, які разом утворюють єдину збірку.

Збірки мають наступні властивості:

- збірки реалізовані як файли .exe або .dll;
- для бібліотек, призначених для .NET Framework, збірки можна одночасно використовувати в кількох додатках, помістивши їх у глобальний кеш-збірок (GAC);
- збірки завантажуються в пам’ять тільки в тому випадку, якщо вони реально використовуються, інакше вони не завантажуються (завдяки цій властивості збірки можуть бути ефективним засобом для управління ресурсами у великих проектах);
- відомості про збірку можна отримати програмним шляхом за допомогою відображення;
- збірку можна завантажити лише для її перевірки, використовуючи клас MetadataLoadContext у .NET і .NET Framework (MetadataLoadContext змінює методи Assembly.ReflectionOnlyLoad).

Збірки в середовищі CLR. Збірки надають відомості для середовища CLR, які потрібні для розпізнавання реалізацій типів. Для середовища виконання тип не існує поза контекстом збірки.

Збірка визначає наступні відомості:

- код, що виконується середовищем CLR (кожна збірка може лише одну точку входу: DllMain, WinMain або Main);

- границя безпеки (збірка являє собою одиницю, для якої запитуються та надаються дозволи);
- границя типу (кожне визначення типу включає збірку імені, у якій розміщений даний тип, тобто тип з іменем MyType, завантажений у область дії однієї збірки, не входить до типу MyType, завантажений у область дії іншої збірки);
- границя області дії посилань (маніфест збірки містить метадані, які використовуються для дозволів типів і для виконання зв'язаних з ресурсами запитів, тобто вказує типи та ресурси, що надаються за межами збірки, а також перераховує інші збірки, від яких вона залежить; при відсутності зв'язаного маніфесту збірки код на проміжній мові MSIL, що знаходиться в переносимому виконуваному (PE) файлі, виконуватися не буде);
- границя версій (збірка є найменшою одиницею з підтримкою версії в середовищі CLR, де версія для всіх типів і ресурсів в одній збірці призначається як єдиному цілому; в маніфесті збірки описуються залежно від певної версії від інших збірок);
- одиниця розгортання (при запуску додатку можуть бути лише збірки, першочергово викликані додатком, а інші збірки, наприклад, що містять ресурси локалізації або допоміжні класи, можуть бути викликані за вимогою, що дозволяє додаткам зберегти просту структуру і малий розмір при скачуванні);
- одиниця паралельного виконання.

Створення збірки. Збірки можуть бути статичними або динамічними. Статичні збірки зберігаються на диску у вигляді переносимих виконуваних (PE) файлів. Статичні збірки можуть містити інтерфейси, класи та ресурси, такі як точні малюнки, файли JPEG та інші файли ресурсів (рис. 4). Крім того, можна створювати динамічні збірки, які запускаються безпосередньо з пам'яті

і не зберігаються на диску перед виконанням. Динамічні збірки можна зберегти на диску після виконання.

Існує кілька способів створення збірок. Можна використовувати засоби розробки, такі як Visual Studio, що дозволяють створювати файли .dll або .exe. Щоб створити збірку з використанням модулів інших засобів розробки, можна скористатися засобами Windows SDK. Для створення динамічних збірок також можна використовувати такі інтерфейси CLR, як System.Reflection.Emit.

Можна скомпілювати збірку, виконавши збірку в Visual Studio, за допомогою інтерфейсу командного рядка .NET Core або створивши збірки .NET Framework за допомогою компілятора командного рядка.

Маніфест збірки. Кожна збірка містить файл маніфесту збірки. Маніфест збірки виконує роль заголовку та містить наступне:

- ідентифікатор збірки (ім’я та версія);
- таблиця з описом усіх файлів, що входять у збірку, відносяться також інші збірки, від яких залежить файл .exe або .dll, та растрові зображення або файли відомостей.
- список посилань збірки, тобто список всіх залежностей, таких як бібліотеки dll або інші файли (посилання на збірки містять посилання на глобальні та приватні об’єкти; глобальні об’єкти доступні для всіх інших програм; в .NET Core глобальні об’єкти пов’язані з визначеним середовищем виконання; у .NET Framework глобальні об’єкти розташовані в глобальному кеші збірок (GAC); System.IO.dll є прикладом збірки в глобальному кеші збірок; закриті об’єкти повинні знаходитися в каталозі установок додатків або в одному з його підкаталогів).

Так як збірки містять відомості про вміст, версії та залежності, робота, пов’язаних з ними додатками не залежить від зовнішніх джерел, таких як реєстр у системах Windows. Збірки знижують ризик конфлікту бібліотек dll, а також підвищують надійність і простоту розгортання додатків. У багатьох

випадках для встановлення додатків на базі .NET достатньо просто скопіювати його файли на комп’ютер.

Додавання посилань на збірку. Щоб використовувати збірку в додатку, потрібно додати посилання на неї. Коли ви додаєте посилання на збірку, у вашому додатку стануть доступні всі надані в збірці типи, властивості, методи та інші елементи простору імен, так, ніби їх код є частиною файлу з вихідним кодом вашого додатку.

Посилання на більшість збірок з бібліотеки класів .NET створюються автоматично. Якщо на системну збірку не було створено посилання автоматично, то потрібно додати посилання наступним способом:

- Для .NET і .NET Core додайте посилання на пакет NuGet, що містить цю збірку. Використовуйте диспетчер пакетів NuGet у Visual Studio або додайте елемент <PackageReference> для для складання до проекту *.csproj* або *.vbproj* .
- Для .NET Framework додайте посилання на збірку за допомогою діалогового вікна «Додайте посилання» в Visual Studio або параметра reference командного рядка для компіляторів C# або Visual Basic.

У C# можна використовувати дві версії однієї і тої ж збірки в одній програмі.

5. Бібліотеки класів .NET

Інтерфейси API .NET містять класи, інтерфейси, делегати та типи значень, які полегшують та оптимізують процес розробки, а також забезпечують доступ до функцій системи. Для спрощення взаємодії між мовами більшість типів платформи .NET є сумісними CLS, і тому їх можна використовувати в будь-якій мові програмування, компілятор якої відповідає специфікації CLS.

Типи .NET є основою створення елементів управління, компонентів і додатків .NET. .NET містить типи, призначені для наступних завдань:

- подання базових типів даних та винятків;
- інкапсуляція структур даних;
- операції введення-виведення;
- доступ до даних про завантажені типи;
- виклик перевірок безпеки. Net;
- доступ до даних, надання графічного інтерфейсу користувача на стороні клієнта і керованого сервером графічного інтерфейсу користувача на стороні клієнта.

.NET пропонує розширений набір інтерфейсів, а також абстрактних та конкретних (неабстрактних) класів. Можна використовувати існуючі конкретні класи «як є». Крім того, у багатьох випадках на їх основі можна створювати власні класи. Щоб використовувати можливості інтерфейсу, можна створити клас, що реалізує інтерфейс, або створити похідний клас на основі одного з класів .NET, що реалізує інтерфейс.

Угоди про імена. Для типів платформи. NET використовується схема іменування через точку, що описує ієрархію. При такому підході пов'язані типи групуються у просторі імен, що спрощує їх пошук та створення посилань. Перша частина повного імені (аж до крайньої правої точки) це ім'я простору імен. Остання частина імені – це ім'я типу. Наприклад, System.Collections.Generic.List<T> представляє тип List<T>, який належить простору імен System.Collections.Generic. Типи System.Collections.Generic можна використовувати для роботи з універсальними колекціями.

Така схема іменування спрощує розробникам бібліотек завдання розширення .NET з метою створення ієрархічних груп типів та присвоєння їм узгоджених між собою та зрозумілих імен. Вона також дозволяє однозначно ідентифікувати типи за їх повними іменами (тобто простором імен та імені типу), що запобігає конфліктам імен типів. Очікується, що при іменуванні просторів імен розробники бібліотек керуватимуться наступною угодою:

CompanyName.TechnologyName

Наприклад, простір імен Microsoft.Word відповідає цьому правилу.

Використання шаблонів іменування для угруповання зв'язаних типів у просторі імен корисне при створенні та документуванні бібліотек класів. Однак така схема іменування не впливає на видимість, доступ до членів, наслідування, безпеку та прив'язки. Простір імен може бути розділений між кількома збірками, і одна збірка може містити типи з декількох просторів імен. Збірка являє собою формальну структуру для управління версіями, розгортання, забезпечення безпеки, завантаження та забезпечення видимості серед CLR.

Простір імен System. Простір імен System є кореневим простором імен для основних типів у .NET. Цей простір імен включає класи, що представляють базові типи даних, що використовуються всіма програмами, наприклад: Object (корінь ієрархії успадкування), Byte, Char, Array, Int32 та String. Багато з цих типів відповідають простим типам даних, що використовуються у мові програмування. При написанні коду із застосуванням типів .NET можна використовувати відповідне ключове слово мови для базового типу даних .NET.

Вони дозволяють поміщати корисні функції у модулі, які можуть використовуватися різними програмами. Вони також можуть використовуватися для підключення функцій, які не були потрібні або не відомі під час запуску програми. Бібліотеки класів описуються у форматі файлу збирання .NET.

Існує три доступні для використання типи бібліотек класів:

- бібліотеки класів, які залежать від платформи, мають доступ до всіх API на цій платформі (наприклад, .NET Framework у Windows, Xamarin iOS), але можуть використовуватися лише додатками та бібліотеками, призначеними для цієї платформи;

- переносимі бібліотеки класів мають доступ до підмножини API і можуть використовуватися додатками та бібліотеками, призначеними для декількох платформ;
- бібліотеки класів .NET Standard є об'єднанням специфічних для платформи та бібліотек, що переносяться, і поєднують кращі характеристики обох типів.

Специфічні для платформи бібліотеки прив'язані до однієї платформи .NET. Розробники, які створюють бібліотеки для певних платформ, можуть використовувати всі можливості базової платформи. При цьому такі бібліотеки працюватимуть лише на даній платформі, що робить необов'язковою перевірку платформи.

Бібліотеки, що переносяться, підтримуються в декількох реалізаціях .NET. Конфігурація платформи вибирається при створенні бібліотеки, що переноситься. Конфігурація платформи – це набір платформ, для яких потрібно забезпечити підтримку (наприклад, .NET Framework 4.5+ та Windows Phone 8.0+). Чим більше платформ ви вирішите підтримувати, тим менше буде доступно API, що є недоліком.

Бібліотеки .NET Standard замінюють концепції специфічних для платформи та бібліотек, що переносяться. Вони специфічні для платформи в тому сенсі, що вони надають всі функціональні можливості базової платформи (без штучних платформ або перетинів платформ). Вони переносяться в тому сенсі, що вони працюють на всіх підтримуваних платформах.

.NET Standard надає набір бібліотечних контрактів. Реалізації .NET повинні підтримувати кожен контракт повністю або взагалі не підтримувати. Таким чином, кожна реалізація підтримує набір стандартних .NET контрактів. Наслідком цього є те, що кожна бібліотека класів .NET Standard підтримується на платформах, які підтримують її контрактні залежності. Бібліотеки .NET Standard надають набагато більше API, ніж бібліотеки класів, що переносяться.

Наступні реалізації підтримують бібліотеки .NET Standard:

.NET Core

.NET Framework

Mono

Універсальна платформа Windows (UWP)

6. Огляд .NET Core

.NET Core – це нова версія .NET Framework, яка є безкоштовною платформою для розробки загального призначення з відкритим кодом, яку підтримує Microsoft. Це кросплатформна структура, яка працює в операційних системах Windows, macOS і Linux.

.NET Core можна використовувати для створення різних типів програм, таких як мобільні, настільні, веб-, хмарні, IoT, машинне навчання, мікросервіси, ігри тощо.

.NET Core створено з нуля, щоб зробити його модульним, легким, швидким і кросплатформним фреймворком. Він містить основні функції, необхідні для запуску базової програми .NET Core. Інші функції надаються як пакети NuGet, які можна додати у свою програму за потреби. Таким чином програма .NET Core підвищує продуктивність, зменшує обсяг пам’яті та стає легкою в обслуговуванні.

У .NET Framework є деякі обмеження. Наприклад, він працює тільки на платформі Windows. Крім того для різних пристроїв Windows, таких як Windows Desktop, Windows Store, Windows Phone і веб-програми, потрібно використовувати різні API .NET. На додаток до цього, .NET Framework є системою для всієї машини. Будь-які зміни, внесені до нього, впливають на всі програми, які залежать від нього.

Microsoft створила .NET Core з метою зробити .NET з відкритим вихідним кодом, сумісним із різними платформами, який можна

використовувати в різноманітних сферах, від центрів обробки даних до сенсорних пристроїв.

Характеристики .NET Core:

- кросплатформність (.NET Core працює в операційних системах Windows, macOS і Linux; існують різні середовища виконання для кожної операційної системи, яка виконує код і генерує однаковий результат);
- узгодженість між архітектурами (виконання коду з однаковою поведінкою в різних архітектурах наборів інструкцій, включаючи x64, x86 і ARM);
- широкий спектр додатків (різні типи додатків можна розробляти та запускати на платформі .NET Core: мобільні, настільні, веб-, хмарні, IoT, машинне навчання, мікросервіси, ігри тощо).
- підтримка кількох мов (можна використовувати мови програмування C#, F# і Visual Basic для розробки програм .NET Core та обирати IDE для розробки, зокрема Visual Studio 2017/2019, Visual Studio Code, Sublime Text, Vim тощо);
- модульна архітектура (.NET Core підтримує підхід модульної архітектури за допомогою пакетів NuGet; існують різні пакети NuGet для різних функцій, які за потреби можна додати до проекту .NET Core; бібліотека .NET Core теж надається як пакет NuGet; пакет NuGet для моделі програми .NET Core за замовчуванням – Microsoft.NETCore.App).

З .NET Core було включено ряд технологій. Платформа .NET Core орієнтована в першу чергу на розробку для серверних і хмарних рішень. Для настільних програм краще використовувати класичний .NET для Windows (з підтримкою WPF і Windows Forms) і Mono для Linux і macOS (з підтримкою Windows Forms). Мобільні проекти можна створювати, використовуючи

Xamarin. На рис. 5 показано, як технології розподілені всередині різних реалізацій .NET.

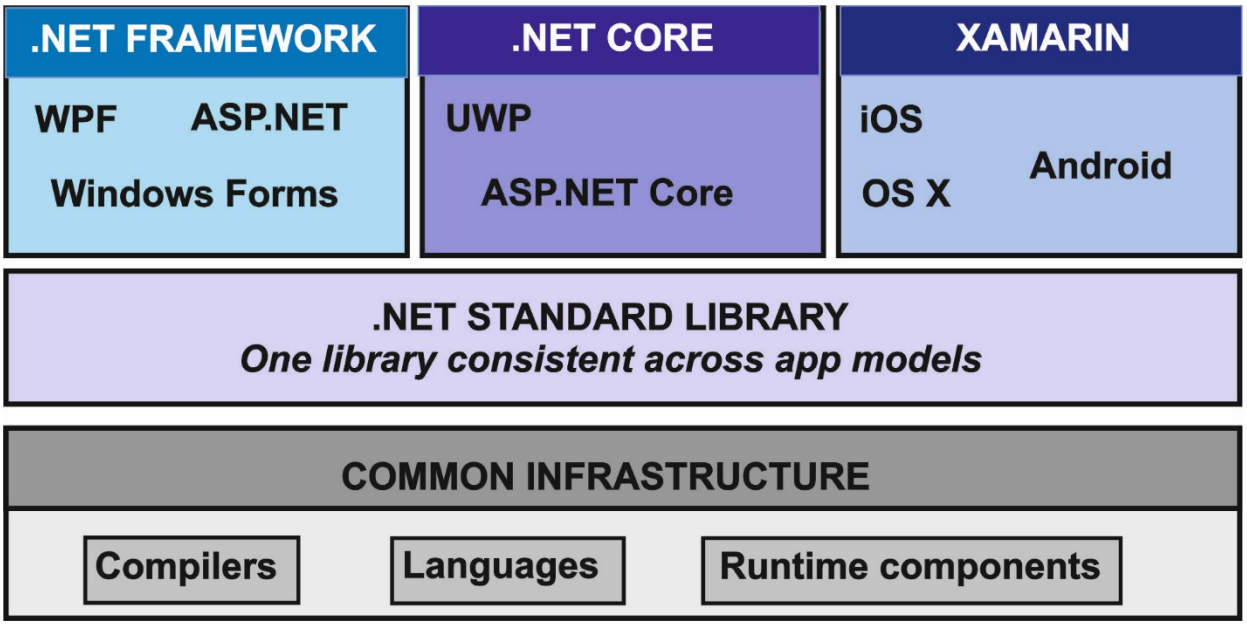

Рис.5. Технології .Net

Таким чином, у .NET Core були виключені:

- ASP.NET WebForms;
- WCF;
- WPF;
- Windows Forms.

При розробці більшість необхідних компонент додатків можна завантажити як окремі модулі через пакетний менеджер NuGet. Це дозволяє зменшити кількість надмірних залежностей і загальний розмір готового продукту.

На відміну від класичного .NET Framework, код якого переважно є закритим, код .NET Core повністю відкритий і поширюється під ліцензіями MIT і Apache2.

7. Розробка додатків на .NET

Консольні програми в .NET. Програми .NET можуть використовувати клас System.Console для виконання консольного вводу-виводу символів. Дані, що надходять від консолі, зчитуються зі стандартного потоку введення, дані, що виводяться на консоль, записуються в стандартний потік виведення, а відомості про помилки записуються в стандартний потік виведення помилок. Ці потоки автоматично зв'язуються з консоллю при запуску програми та представлені властивостями In, Out та Error відповідно.

Значенням властивості Console.In є об'єкт System.IO.TextReader, а значення властивостей Console.Out і Console.Error – об'єкти System.IO.TextWriter. Ці властивості можна пов'язувати з потоками, що не представляють консоль, що дозволяє задати для будь-якого потоку інше джерело або приймач даних. Наприклад, можна перенаправити вивід у файл, надавши властивості Console.Out об'єкт System.IO.StreamWriter, що інкапсулює System.IO.FileStream за допомогою методу Console.SetOut. Властивості Console.In та Console.Out не обов'язково повинні бути пов'язані з одним потоком.

Windows Forms .NET. Windows Forms – це платформа інтерфейсу користувача для створення класичних програм Windows. Вона забезпечує один з найефективніших способів створення класичних програм за допомогою візуального конструктора в Visual Studio. Такі функції, як розміщення візуальних елементів керування шляхом перетягування, полегшують створення класичних додатків.

У Windows Forms можна розробляти графічно складні програми, які просто розгортати, оновлювати і з якими зручно працювати як в автономному режимі, так і в мережі. Програми Windows Forms можуть отримувати доступ до локального обладнання та файлової системи комп'ютера, на якому працює програма.

Windows Presentation Foundation. WPF (*Windows Presentation Foundation*) – це платформа інтерфейсу користувача для створення клієнтських додатків. Інтерфейс програми пишеться за допомогою мови розмітки XAML (Extensible Application Markup), яка використовується для створення екземплярів об'єктів .NET. Хоча мова XAML – це технологія, яка може бути застосована до багатьох різних предметних областей, її головне призначення – конструювання інтерфейсів WPF. Іншими словами, документи XAML визначають розташування панелей, кнопок та інших елементів керування, що становлять вікна у додатку WPF. Для WPF XAML не є обов'язковим. Однак XAML відкриває можливості кооперації, оскільки інші інструменти проектування розуміють формат XAML.

ASP.NET – це популярна платформа веб-розробки для створення веб-програм на платформі .NET.

ASP.NET розширює платформу .NET інструментами та бібліотеками, спеціально призначеними для створення веб-програм.

ASP.NET додає до платформи .NET наступні можливості:

- базову структура для обробки веб-запитів у C# або F#;
- синтаксис шаблонів веб-сторінок, відомий як Razor, для створення динамічних веб-сторінок за допомогою C#;
- бібліотеки для загальних веб-шаблонів, таких як Model View Controller (MVC);
- систему автентифікації, яка включає бібліотеки, базу даних і сторінки шаблонів для обробки входів, включаючи багатофакторну автентифікацію та зовнішню автентифікацію за допомогою Google, Twitter тощо;
- розширення редактора для підсвічування синтаксису, доповнення коду та інших функцій спеціально для розробки веб-сторінок.

Під час використання ASP.NET внутрішній код, наприклад бізнес-логіка та доступ до даних, буде написаний за допомогою C#, F# або Visual Basic.

Оскільки ASP.NET розширює .NET, то можна використовувати велику екосистему пакетів і бібліотек, доступних усім розробникам .NET. Також є можливість створювати власні бібліотеки, які спільно використовуються між програмами, написаними на платформі .NET.

Razor надає синтаксис для створення динамічних веб-сторінок за допомогою HTML і C#. Код C# оцінюється на сервері, а отриманий вміст HTML надсилається користувачеві.

Код, який виконується на стороні клієнта, написаний на JavaScript. ASP.NET інтегрується з фреймворками JavaScript і містить попередньо налаштовані шаблони для фреймворків односторінкових програм (SPA), таких як React і Angular.

Програми ASP.NET можна розробляти та запускати в Windows, Linux, macOS і Docker.

Сімейство продуктів Visual Studio містить інструменти для створення програм .NET у будь-якій операційній системі. Існують також інструменти командного рядка та розширення для багатьох популярних редакторів.

ASP.NET Core – це версія ASP.NET із відкритим кодом, яка працює на macOS, Linux і Windows. ASP.NET Core був уперше випущений у 2016 році та є оновленим дизайном попередніх версій ASP.NET, які використовувалися лише для Windows.

UWP (Universal Windows Platform) – це один із багатьох способів створення клієнтських програм для Windows. Програми UWP використовують API WinRT для надання потужних інтерфейсів і розширених асинхронних функцій, які ідеально підходять для пристроїв, підключених до Інтернету. UWP – це один із варіантів створення програм, які працюють на пристроях Windows 10 та Windows 11 і можуть використовуватися на інших платформах. Додатки UWP можуть використовувати API Win32 та класи .NET.

В основі концепції додатків UWP лежить припущення про те, що користувачі хочуть мати можливість працювати з додатками на всіх своїх

пристроях, вибираючи для виконання конкретної задачі найбільш зручний або продуктивніший пристрій. Windows 10 полегшує розробку додатків для UWP, пропонуючи всього один набір API, один пакет додатку та один магазин для забезпечення роботи додатку на всіх пристроях під управлінням Windows 10 – комп'ютері, планшеті, телефоні та на інших пристроях. При розробці йде орієнтація на сімейство пристроїв, а не на конкретну ОС. Сімейство пристроїв визначає API-інтерфейси, характеристики системи та поведінку, очікувану на пристроях всередині сімейства. Воно також визначає набір пристроїв, на які може бути встановлено додаток UWP.

Всі додатки UWP розповсюджуються в вигляді пакетів AppX. Це гарантує надійність механізму встановлення та забезпечує безпроблемне розгортання і оновлення додатків.

Крім того, UWP не обмежується тільки виконанням прикладних програм, також вона на ядерному рівні підтримує роботу драйверів. Це дозволяє створювати драйвери, які функціонують на різних пристроях, за умови, що конкретний компонент, для якого призначений даний драйвер, один і той же.

UWP підтримує драйвери як рівня ядра, так і призначеного для користувацького рівня. Підсистема включає інтерфейси драйверів пристроїв (Device Driver Interface – DDI), з числа яких драйвер для UWP може використовувати.

Основний спосіб створення додатків для смартфонів з Windows 10 Mobile - це розробка **універсальних (UWP) додатків**

Xamarin – це платформа з відкритим вихідним кодом, призначена для побудови сучасних продуктивних програм для iOS, Android та Windows з .NET. Платформа Xamarin забезпечує управління взаємодією між загальним кодом і кодом базової платформи. Xamarin виконується в керованому середовищі, яке реалізує такі можливості, як виділення пам'яті та збирач сміття.

Завдяки Xamarin в середньому 90% коду програми можна використовувати без змін на різних платформах. За допомогою цього шаблону розробник може написати всю бізнес-логіку однією мовою (або використовувати існуючий код програми), але при цьому отримати характеристики продуктивності, оформлення та поведінку, характерні для кожної відповідної платформи.

Програми Xamarin можна писати на ПК або Mac і компілювати у власні пакети програм, наприклад файли з розширенням .apk для Android або .ipa для iOS.

Платформа Xamarin орієнтована на розробників, перед якими стоять такі завдання:

- спільне використання коду, тестів та бізнес-логіки на різних платформах;
- написання кросплатформних додатків мовою C# у Visual Studio.

За допомогою Xamarin можна створювати власний інтерфейс для кожної платформи і писати мовою C# загальну бізнес-логіку, яка буде використовуватися на різних платформах. У більшості випадків Xamarin дозволяє використовувати на різних платформах 80% коду програми.

Список використаних джерела

1. Введение в платформу .NET FrameWork. *C# Статьи, уроки, видео, обучение, видео курсы ASP.NET, ADO.NET, LINQ, Visual Studio*. URL: http://skillcoding.com/Default.aspx?id=79 (дата звернення: 06.10.2022).
2. Overview of .NET Framework - .NET Framework. *Microsoft Learn: Build skills that open doors in your career*. URL: https://learn.microsoft.com/en-us/dotnet/framework/get-started/overview (date of access: 06. 10.2022).
3. .NET implementations - .NET. *Microsoft Learn: Build skills that open doors in your career*. URL: https://learn.microsoft.com/en-us/dotnet/fundamentals/implementations (date of access: 06. 10.2022).
4. Настенко, Д. В., Нестерко А. Б. Об'єктно-орієнтоване програмування. Частина 1. Основи об'єктно-орієнтованого програмування на мові C# [Електронний ресурс] : навчальний посібник. Київ : НТУУ «КПІ», 2016. 76 с. URI: https://ela.kpi.ua/handle/123456789/16671 (date of access: 06. 10.2022).
5. What Is Windows Communication Foundation - WCF. *Microsoft Learn: Build skills that open doors in your career*. URL: https://learn.microsoft.com/en-us/dotnet/framework/wcf/whats-wcf (date of access: 06. 10.2022).
6. .NET class library overview. *Microsoft Learn: Build skills that open doors in your career*. URL: https://learn.microsoft.com/en-us/dotnet/standard/class-library-overview (date of access: 06.10.2022).
7. Warren M. Posts By Year. *Performance is a Feature!*. URL: https://mattwarren.org/postsByYear/ (date of access: 06.10.2023).
8. Assemblies in .NET. *Microsoft Learn: Build skills that open doors in your career*. URL: https://learn.microsoft.com/en-us/dotnet/standard/assembly/?source=recommendations (date of access: 06.10.2022).

9. What is Xamarin? - Xamarin. *Microsoft Learn: Build skills that open doors in your career*. URL: https://learn.microsoft.com/en-us/xamarin/get-started/what-is-xamarin (date of access: 06.10.2022).
10. Managed Execution Process. *Microsoft Learn: Build skills that open doors in your career*. URL: https://learn.microsoft.com/en-us/dotnet/standard/managed-execution-process (date of access: 06.10.2022).
11. .NET Core Overview. *TutorialsTeacher - Learn Technologies*. URL: https://www.tutorialsteacher.com/core/dotnet-core (date of access: 06.10.2022).

УДК УДК 004.655

Навчально-методичне видання

Базові поняття .NET.

укладачі
Булатецький Віталій Вікторович
Булатецька Леся Віталіївна

Конспект лекцій нормативної навчальної дисципліни
“Технології платформи .Net”

друкується в авторській редакції