МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ДОНЕЦЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ ІМЕНІ ВАСИЛЯ СТУСА

Зінченко Богдан Вікторович

Допускається до захисту:
завідувач кафедри
інформаційних технологій,
доктор технічних наук, доцент
_____Т. В. Нескородєва
«____» _____ 20__ р.

РОЗРОБКА СЕРВЕРНОЇ ЧАСТИНИ СИСТЕМИ АВТОМАТИЧНОЇ ПЕРЕВІРКИ ПРОГРАМНОГО КОДУ НА C#

Спеціальність 122 «Комп’ютерні науки»
Кваліфікаційна (бакалаврська) робота

Керівник:
Антонов Ю. С., доцент кафедри
інформаційних технологій,
к.т.н, доцент

Оцінка:___/___/_____
(бали за шкалою ЄКТС/за національною шкалою)
Голова ЕК:_____
(підпис)

Вінниця – 2022

АНОТАЦІЯ

Зінченко Б.В. Ця дипломна робота присвячена розробці сервісу для перевірки програмного коду на C#. Так, як на даний момент технології розвиваються дуже швидко, усі намагаються автоматизувати процеси, які донедавна робились виключно людьми.

Робота складається із 3 розділів, X літературних джерел, X рисунків. Загальний обсяг роботи складає X сторінки.

У вступі наведено актуальність розробки такого сервісу, як він може бути корисним та які механізми та інструменти було вивчено в ході роботи.

Другий розділ присвячено вибору інструментів для розробки та їх опису.

Третій розділ містить в собі розробку самого сервісу та його тестування. В ньому детально описано який функціонал притаманний цій системі та як розробляти подібні системи. Розписано, які технології використовувались та більше інформації безпосередньо про них. Також описано алгоритм за яким працює сервіс.

У висновку підбитий підсумок проведеної роботи та обгрунтована значимість цієї системи. Ключові слова: перевірка, C#, .NET, сервіс, сервер, автоматична.

Zinchenko B This thesis is dedicated to the development of a server for C# program code verification. Since technologies are developing very fast at the moment, everyone is trying to automate processes that were previously done exclusively by humans.

The work consists of 3 sections, X literature sources, X drawings. The total volume of the work consists of X pages.

In the introduction the relevance of developing such a service, how it can be useful and what mechanisms and tools have been studied in the course of the work is given.

The second section is devoted to the selection of design tools and their description.

The third section contains the design of the service itself and its testing. It describes in detail what functionality is needed for this system and how to develop such systems.

It is described which technologies were used and more information about them directly. Also described is the algorithm by which the service works. A summary of the work carried out and the significance of this system is substantiated. Key words: validation, C#, .NET, server, server, automatic.

ЗМІСТ

ВСТУП

Актуальність роботи:

В сучасному світі все більше набирає популярності тематика економії. Людина 21 – го століття повинна вміти економити все, починаючи від часу, закінчуючи ресурсами. Економія часу напряму залежить від якості, тому чим менше часу ми витрачаємо на перевірку – тим нижчою є якість нашої роботи. Цьому є чудовий приклад - професія тестувальник, робота якого напряму складається з перевірки: інтерфейсу, зручності використання, програмного коду і т.д.

В Україні викладачі університетів, котрі навчають майбутніх програмістів витрачають чимало часу на перевірку програмного коду, таким чином повністю відчуваючи на собі роботу тестувальника. Попри те, що це займає чималу частину часу, не завжди така перевірка є ефективною. Причина зрозуміла – людський фактор, який полягає у можливості помилитись, не зрозуміти, пропустити.

Люди навчилися автоматизовувати багато речей. У пилосмок додали трішки технологій і з’явився робот пилосмок, який без людського втручання, по вказаному маршруту за кілька хвилин прибирає квартиру, чи кухонний комбайн, який перемішує тісто швидко й ефективно, тим часом поки господиня займається іншими справами, посудомийна машина, розумний чайник та багато інших речей, котрі значно економлять час. Так і з’явилось автоматичне тестування програмного забезпечення, яке по написаному скрипту перевіряє код розробника.

Тому тема даного дипломного проєкту буде компілятор для покращеної перевірки програмного коду для тих, хто досі його не використовує по якихось причинам.

Реалізація проекту, спрямована на максимальне покращення автоматизації коду, його простота та швидкість.

Також таким чином майбутні програмісти отримують досвід, самостійно розвиваються в створенні продуктів та можуть покращувати й вдосконалювати свої навички.

Мета дослідження: Розробка серверної частини системи автоматичної перевірки програмного коду на C#

Завдання дослідження:

- Огляд існуючих аналогів
- Ознайомлення з мовою програмування C#
- Ознайомлення з JSON
- Розробка серверної частини системи автоматичної перевірки програмного коду на C#
- Тестування сервісу

Об’єкт дослідження: складнощі, що виникають у перевірці програмного коду молодих розробників

Предмет дослідження: система перевірки коду та його аналіз

Практичне значення одержаних результатів: створення незалежного додатку та взаємодія з користувачем для вирішення проблем.

Структура кваліфікаційної роботи: Бакалаврська робота складається із вступу, трьох розділів та висновків до них та списку використаних джерел.

РОЗДІЛ 1. ІСТОРІЯ ЕВОЛЮЦІЇ МОВ ПРОГРАМУВАННЯ, КОМПІЛЯТОРІВ ТА ІНТЕРПРЕТАТОРІВ

У даному розділі описана історія створення обчислювальної техніки, також розглянуто створення перших мов програмування з коротким аналізом їх шляху до актуальності та сучасності на сьогодення, Наведений аналіз та досвід використання мов програмування в Україні і в європейських державах.

1.1 Історія створення обчислювальної техніки та перші мови програмування

Досліджуючи історію створення обчислювальної техніки, варто згадати про найперше - винайдення комп’ютера. Неможливо точно відповісти на питання, хто саме його винайшов. Справа у тому, що комп'ютер не є винаходом однієї людини. Він увібрав у собі ідеї та технічні рішення багатьох вчених та інженерів. Розвиток обчислювальної техніки стимулювався потребою у швидких та точних обчисленнях і тривав сотні років. У процесі розвитку обчислювальна техніка ставала дедалі більш досконалою. Цей процес триває і в наш час.

Є кілька сторін, які сприяли розвитку сучасної обчислювальної техніки. Перш за все - це розвиток пристосувань для рахунку, методів обчислень, розвиток систем числення, математичної логіки, що визначило логічну схему комп’ютера, з іншої сторони - це розвиток електронної теорії, науки та техніки у галузі електрики, що визначило елементну базу сучасних комп’ютерів.

Первісні люди не знали чисел і використовували для запам'ятовування певної кількості предметів наочне уявлення – різні підручні засоби: камінці, мушлі тощо. Розвиток рахунку пішов значно швидше, коли людина вирішила звернутися до своїх пальців, як до самого природного рахункового апарату. Від пальцевого рахунку бере початок п'ятіркова система числення (одна рука), десяткова (дві руки), двадцяткова (пальці рук і ніг).

Далі настала епоха, так званого, абаку (рахівниця) - це рахувальна дошка родом з Древньої Греції. Грецький абак являв собою дошку, на якій паралельні лінії позначали розряди одиниць, десятків, сотень і т.д. На лініях вміщували відповідне число жетонів (камінців, кісточок).

Щодо створення та розвитку комп'ютерної техніки, слід зазначити, що перші ЕОМ почали з'являтись після Другої світової війни. Вони були були дуже громіздкими та працювали від електронних ламп. Прикладом таких машин першого покоління є ENIAC, який міг виконувати п'ять тисяч операцій за секунду.[2]

Машини другого покоління, що з'явилися у 1955 році, були меншими за розмірами, адже використовували не лампи, а транзистори. Крім меншого розміру, такі ЕОМ були більш продуктивними та споживали менше електроенергії.

Транзистори призвели до появи компактніших і надійніших комп'ютерів, які витіснили дорожчі комп'ютери загального призначення. Зниження вартості та зменшення займаної площі призвели до зростання попиту на комп'ютери. Однак транзисторні комп'ютери залишалися недоступними, оскільки були надто дорогими та великими для звичайних домогосподарств, тому попит, як і раніше, в основному визначався фірмами.

Оренда першого транзисторного комп'ютера - IBM 608 - коштувала 1760 доларів на місяць. Для порівняння, оренда першого "персонального" комп'ютера з вакуумною трубкою - IBM 650 - коштувала 3200 доларів на місяць.

Таким чином, 1950-60-ті роки започаткували сучасну комп'ютерну індустрію, яка зрештою консолідувалась навколо IBM.

У 1960-х роках IBM виробляла 70% комп'ютерів у світі та 80% комп'ютерів, що використовуються у США.

Мікропроцесор

Компанія Intel представила перший комерційний мікропроцесор – Intel

4004 – у 1971 році. Однак саме Intel 8008 (представлений в 1972) став основою для наступного покоління комп'ютерів - IBM PC.

Мікропроцесори радикально знизили вартість виробництва комп'ютерів, забезпечивши масове виробництво систем із процесорами, виготовленими на замовлення. Мікропроцесор дозволив створити міні-комп'ютер, персональний комп'ютер, ноутбук і, зрештою, мобільний телефон, який кинув виклик великим транзисторним комп'ютерам IBM.

1970-х роках вартість залишалася у рамках виробництва апаратного забезпечення, але почала мігрувати у бік виробників мікропроцесорів. Завдяки перевазі, отриманій завдяки випуску Intel 4004 і 8008, компанія Intel контролювала 100% сегменту ринку мікропроцесорів на початку ери. Зрештою, Motorola представила 6800 у 1974 році, що започаткувало "війну" мікропроцесорів між Intel і Motorola.

ПК

Поява мікропроцесорів спричинило посилення конкуренції в апаратній частині комп'ютерів. Цінність більше не створювалася (здебільшого) за рахунок удосконалення апаратного забезпечення (оскільки більшість основ апаратного забезпечення було закладено), а скоріше створювалася за рахунок збільшення маржі та підвищення продуктивності існуючих архітектур. Найбільшими виробниками апаратного забезпечення в цю епоху були Intel, Zilog, Motorola та MOS Tech. Конкуренція стала ще жорсткішою, коли на ринку з'явилися японські чіпи від Hitachi, NEC, Fujitsu та Toshiba.

Масове виробництво комп'ютерів викликало підвищений попит користувачів на загальну операційну систему (ОС), де працювало б стандартне програмне забезпечення. Microsoft визначила цю можливість і скористалася нею, розробивши власну операційну систему та отримавши права на поширення у виробників комп'ютерів. Microsoft зосередилася на залученні розробників, об'єднавши усі додатки в одну операційну систему. Клієнтів приваблювала платформа

Windows, оскільки вона мала найбільшу кількість сумісних програм. В результаті до кінця 90-х операційна система Microsoft використовувалася на 97% всіх комп'ютерних пристроїв. Таким чином, формування вартості перемістилося вгору технологічним стеком на програмний рівень, де відкрився новий, практично необмежений простір для проектування.

Незважаючи на зростання конкуренції в галузі апаратного забезпечення, Intel продовжувала домінувати на ринку мікропроцесорів. Чіпи Pentium від Intel у поєднанні з операційною системою Windows 3.x від Microsoft швидко розширили ринок ПК. Windows почала підтримувати неінтелеві процесори тільки з Windows NT 3.51. Незважаючи на підтримку неінтелових процесорів, альянс Intel/Windows, який отримав назву Wintel, продовжував визначати ринок ПК. Ця замкнутість екосистеми була основною для успіху двох організацій, що заслуговує на окреме обговорення.

2000-2010: веб

Поява Linux дозволило створити безкоштовну операційну систему з відкритим вихідним кодом, а Web (HTTP) – мережу вільного розповсюдження. І Linux, і HTTP призвели до двох ключових змін на ринку обчислювальної техніки:

1) Інтернет-браузери (через HTTP та інші протоколи з відкритим вихідним кодом) забезпечили крос-ос доступ для користувачів, тим самим покінчивши з перевагою Windows у вигляді блокування програм.

2) Споживання комп'ютерів почало зміщуватися у бік мобільних пристроїв - явище, яке значною мірою було втрачено Microsoft.

1.2 Перші та сучасні мови програмування

У сучасному світі комп'ютерне програмування необхідне для забезпечення безперебійної роботи систем і пристроїв, якими ми користуємося щодня. Мови

програмування дозволяють людям взаємодіяти з машинами та змушувати їх виконувати необхідні операції. Люди та машини обробляють інформацію по-різному, і мови програмування – це ключ до подолання розриву між людьми та комп'ютерами.

Прийнято вважати, що Алгоритм для аналітичного двигуна Ади Лавлейс є першою в історії комп'ютерною мовою. Його метою було допомогти Чарльзу Беббіджу у обчисленнях чисел Бернуллі, і Ада розробила його у 1883 році. За словами Лавлейса, її машина відрізняється від попередніх обчислювальних машин тим, що її можна запрограмувати на вирішення завдань будь-якої складності. Її внесок у світ комп'ютерного програмування важливий, оскільки вона продемонструвала можливості обчислювальних пристроїв майже за 100 років до того, як виникла думка про сучасний комп'ютер, що програмується.

Мова асемблера з'явилася в 1949 році і незабаром набула широкого застосування в автоматичних калькуляторах з електронним накопичувачем затримки. Асемблер була низькорівневою комп'ютерною мовою, яка спрощувала мову машинного коду, тобто конкретні інструкції, необхідні для роботи комп'ютера.

Ранні комп'ютерні мови існували у багатьох випадках, і всі вони були об'єднані загальним терміном: Автокод. Autocode з'явився в 1952 році і, будучи першою мовою програмування, що компілюється, міг бути переведений безпосередньо в машинний код за допомогою програми, так званим компілятором.

Які старі комп'ютерні мови використовуються сьогодні?

Створений в 1957 році Джоном Бекус, Fortran (скорочення від Formula Translation), можливо, є найстарішою мовою програмування, яка використовується досі. Він призначений для виконання складної статистичної, математичної та наукової роботи.

До інших найважливіших мов цього періоду відносяться:

Algol (1958), що означає "алгоритмічний мову", був розроблений комітетом для наукового використання і став відправною точкою для розвитку Java, C,

C++ і Pascal.

COBOL (1959), тобто. Common Business Oriented Language, був створений Грейс Мюррей Хоппер як мова, яка могла б працювати на комп'ютерах всіх типів та марок. Сьогодні ця популярна мова програмування використовується в обробці кредитних карток, банкоматах, комп'ютерах державних установ та лікарень, телефонних системах, світлофорах та автомобільних системах.

LISP (1959) був вперше створений для допомоги у дослідженнях у галузі штучного інтелекту. LISP є другою найстарішою мовою програмування високого рівня і може використовуватися і досі у ситуаціях, коли використовуються Python або Ruby.

BASIC, розроблений у 1964 році, був модифікований Полом Алленом та Біллом Гейтсом і незабаром став найпершим продуктом компанії Microsoft. Розробники Apple, з іншого боку, використовували Pascal (1970) у перші роки своєї роботи через те, що він був потужним та простим у вивченні.

Крім того, у 1970-х роках було розроблено безліч важливих мов:
Smalltalk (1972), який дозволив комп'ютерникам вносити зміни до коду "на льоту", і ввів речі, які сьогодні присутні у таких життєво важливих мовах, як Java, Ruby та Python.
C (1972) був першою мовою високого рівня. Мова програмування C уможливила використання Unix на широкому спектрі різних комп'ютерів. Його вплив простежується в багатьох популярних мовах кодування.
SQL (1972) зробив революцію в базах даних і уможливив додавання, перегляд або видалення даних за допомогою запитів.
MATLAB (1978) залишається однією з найкращих мов кодування для написання математичних програм. Він переважно використовується у наукових дослідженнях, математиці та освіті. [15]

Сьогодні є кілька лідерів мов програмування, які будуть розвиватись і використовуватись наступні кілька років в Україні та Європі, серед яких Python,

JavaScript, Java, C++, C#. На практиці, доведеться поєднувати, або навіть замінювати одну мову іншою. Причиною зльоту Python є стрімкий розвиток Data Science.

1.3 Еволюція інтерпретаторів та компіляторів

Слід зазначити що швидкість оновлення та знаходження нових ідей для покращення роботи компіляторів та інтерпретаторів значно знизилась. Сучасні мови програмування намагаються як можна оптимальніше використовувати ресурси комп'ютерів. Саме ресурси є найбільшою проблемою. Для написання коду не потрібно багато оперативної пам'яті та ядер процесору, але саме ці речі впливають на швидкість компіляції чи інтерпретації.[3]

Існує декілька видів реалізації програмного коду, основними з них є за допомогою програми інтерпретатора – це програма чи технічні засоби, необхідні для виконання інших програм, вид транслятора, який здійснює по операторну (по командну, по строкову) обробку, перетворення у машинний код та виконання програми або запиту, та програма компілятор– це комп'ютерна програма, що перетворює вихідний код, написаний певною мовою програмування , на семантично еквівалентний код в іншій мові програмування і виконує його.

Все починалося з спеціальних карт, на яких потрібно було виокремлювати спеціальні символи та слова, потім вставлялося у спеціальний отвір та комп'ютер перетворював все що було зроблено на тій карті у код програми та виконував її. Це було щось спільне між інтерпретатором та компілятором.

Далі розробниками архітектури та систем було вибрано методику за допомогою якої було максимально швидко перевести код мови програмування у зрозумілий машині та реалізувати його. За допомогою спеціального програмного забезпечення, перевірявся написаний код розробником, аналізувався, та реалізовувався. Саме така і методика використовується зараз. Поки що не було вигадано чогось що б працювало швидше та не потребувало великої кількості ресурсів. Зараз компанія Google веде розробку компілятора який розташований у хмарі, та

використовує ресурси сервісів, але чи буде ця розробка доступна іншим користувачам і чи буде вона на стільки ефективна як про неї говорить розробник невідомо.

З часом мови програмування почали переходити на вищій рівень, але найоптимальніший метод реалізації коду залишався той самий. На даний момент різні мови програмування, різні сервіси використовують різну реалізацію. Компілятор зараз використовують Сі-подібні мови програмування такі як C, C++, C#, Scala і тд. На іншому інтерпретатори використають мови Java, PHP, Ruby, Python.

Компілятор і інтерпретатор обидва призначені для виконання тієї самої роботи, але відрізняються в операційній процедурі, компілятор приймає вихідний код у агрегованому вигляді, тоді як інтерпретатор приймає складові частини вихідного коду, тобто оператор за твердженням.

Хоча і компілятор, і інтерпретатор мають певні переваги і недоліки, такі як інтерпретовані мови вважаються крос-платформними, тобто код є портативним. Також не потрібно компілювати інструкцію раніше, на відміну від компілятора, що економить час. Складені мови швидше відносяться до процесу компіляції.

1.4 Огляд аналогів

У вільному доступі є безліч сервісів, основною задачею яких є навчити молодих і починаючих програмістів писати примітивні алгоритми та вдосконалювати свої навички, чи створити челендж для вже досвідчених спеціалістів.

Яскравим представником одного з таких сервісів є Codewars (див. рис.1.1).

Тут розробники практикуються у різних навичках та виконують завдання різної складності. Цей додаток є яскравим прикладом як можна об’єднувати

ком'юніті девелоперів та вдосконалювати навички. Даним сервісом користуються мільйони спеціалістів різних рівнів.

Codewars надає змогу взяти участь у різних змагання серед програмістів а також сприяє обміну досвідом між ними.[4]

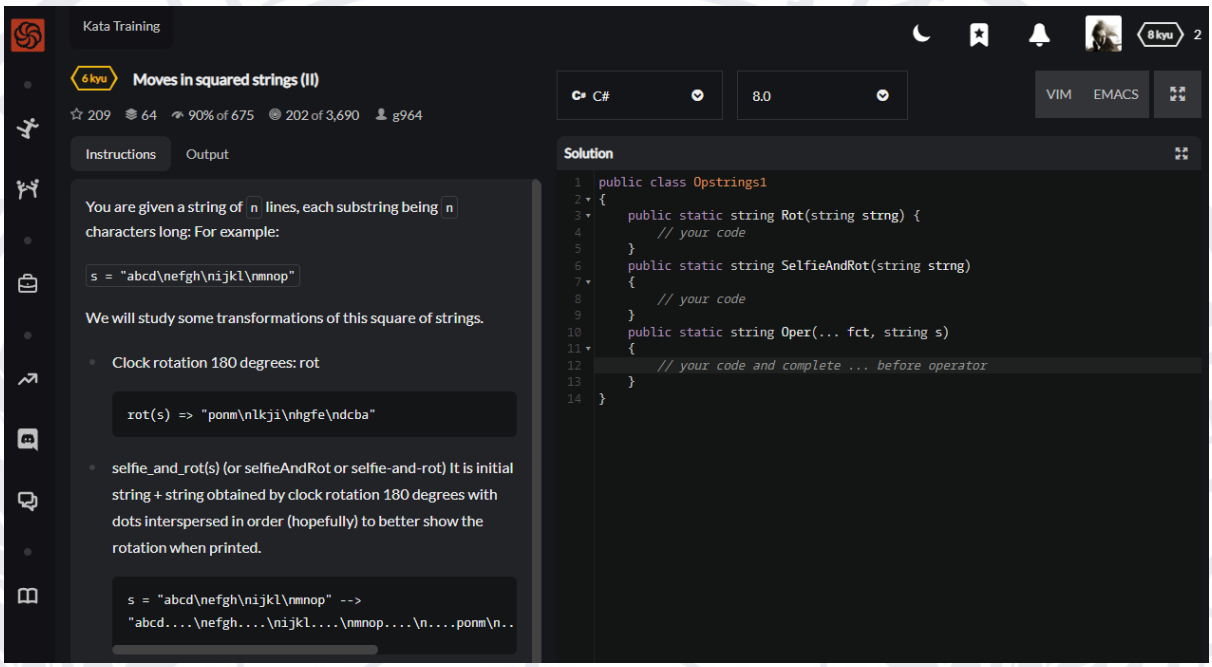

Рисунок 1.1 - Сервіс Codewars

Також існують різні веб рішення подібного характеру які використовуються в університетах та під час олімпіад. Як правило кожен організатор олімпіад має свій подібний сервіс з закритим кодом та своєю екосистемою, щоб звичайні люди не мали доступу до цих сервісів і не могли обійти їх захист.

Codeforces, напевно, один з найбільш популярних ресурсів для компіляції і перевірки коду. За допомогою цього можна тренуватися в написанні якісного коду. Цим ресурсом користується ICPC, міжнародна
організація, яка організовує олімпіади з програмування у всьому світі. На цьому сервісі ведуться прямі ефіри, де розробники з досвідом на власному прикладі і в онлайні показують як вирішувати задачи так як правильно мислити задля їх вирішення.[5]

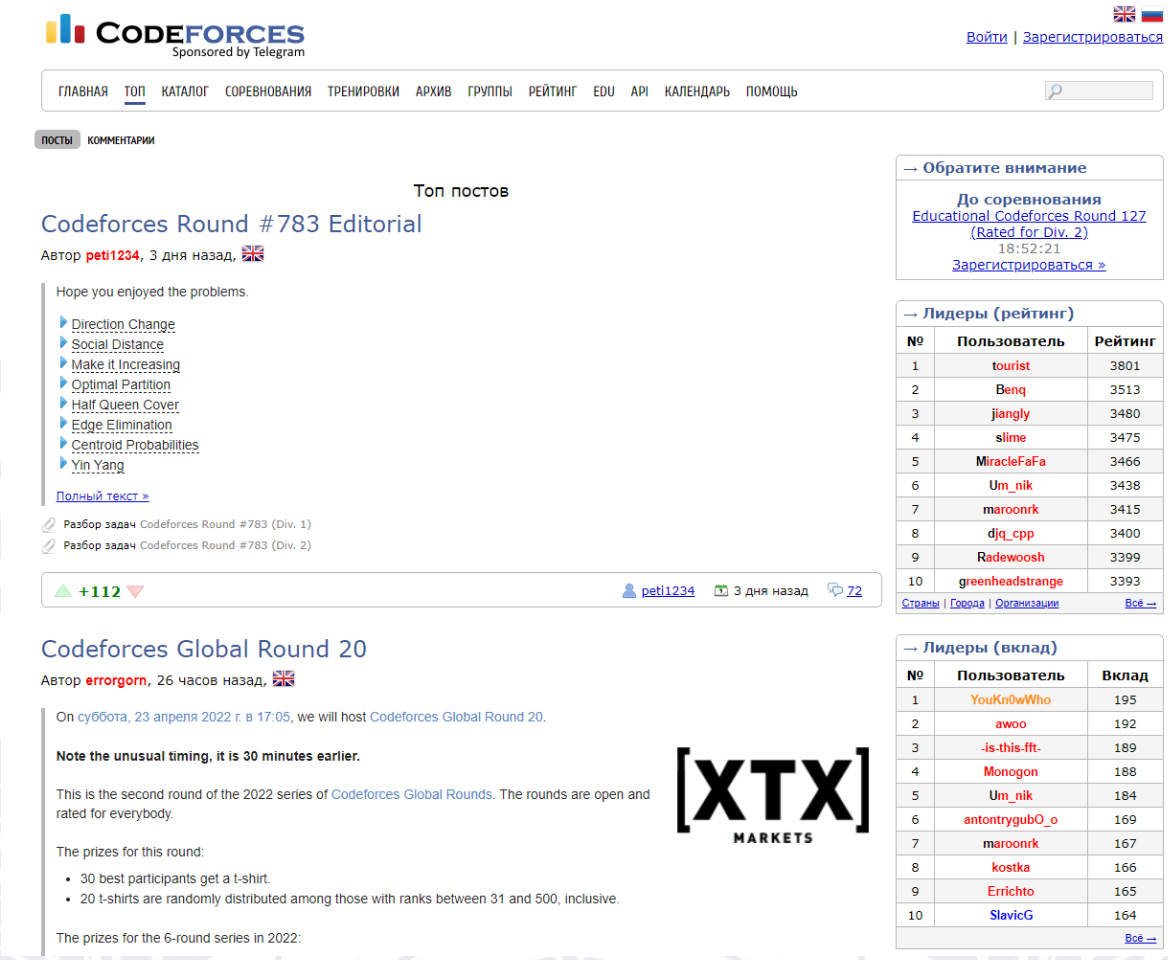

Рисунок 1.2 - Сервіс Codeforces

Щорічно Google проводить хакатон на якому дає змогу проявити себе великій кількості спеціалістів, а головним призом для переможця може стати стажування в компанії. Щоб потрапити на такий хакатон потрібно пройти відбірковий етап, на якому використовуються подібні сервіси для компіляції і перевірки коду. Вони допомагають автоматично, без втручання розробника перевірити чи правильно відпрацьовує код і видати результат його роботи. Такий сервіс не просто компілює чужий код чи цілу програму, але і аналізує саме рішення, його швидкодію та актуальність.

Висновки до розділу 1

У даному розділі було розглянуто історію створення обчислювальної техніки, мов програмування, їхній розвиток та осучаснення. Також описано аналоги компіляторів і сервісів для перевірки знань спеціалістів.

РОЗДІЛ 2. ПОСТАНОВКА ЗАДАЧІ ТА ПРОЕКТУВАННЯ СЕРВІСУ

Даний розділ містить в собі постановку задачі, визначення потрібного функціоналу, вимог системи та безпосереднє проектування сервісу.

2.1 Постановка задачі

Необхідно розробити серверну частину для додвання, аналізу, компіляції та перевірки результату виконання програмного коду на мові C#. Призначенням автоматизованого сервісу є перевірка програмного коду, в переважній більшості, студентів спеціальності яких так чи інакше стикаються та вивчають програмування, а також спеціалісти, які хочуть покращити свої навички.
Перевагами майбутнього сервісу мають стати:

- Аналіз програмного коду
- Швидка компіляція програмного коду
- Легке налаштування завдання
- Безпека виконання, та захист від загроз по типу Sql ін'єкцій
- Легка взаємодія з front-end частиною
- Підтримка сервісу на різних платформах

Основним завданням є створити досконалий та надійно працюючий компілятор. Він повинен відповідно до завдання компілювати програмний код який повинен бути написаний відповідно до всіх дотримань SOLID. Компілятор повинен підтримувати увесь namespace system, та всіх його наслідників. Сервіс повинен бути реалізований за архітектурою REST API. Розробити інтерфейсну частину за допомогою якої можна буде тестувати та перевіряти сервіс. Так як це back-end частина сервісу, він повинен легко інтегруватися з різними мовами програмування для розробки front-end частини.

Компілятор мусить швидко та ефективно обробляти код та реагувати на

помилки як в самому коді так і в результаті відповіді.

Завдання мають бути приведені до уніфіцированого типу, щоб не було потреби при створенні якогось додаткового завдання, створювати додаткові класи. Обов'язковим є дотримання принципів ООП.

Сервіс повинен бути захищеним від різних спроб отримати до нього доступ. Повинен бути захист від DDOS атак чи SQL ін'єкцій. Сервіс мусить підтримувати режим високих нагрузок та не втрачати своєї ефективності і не уповільнювати.

Можливість розгорнути сервіс має бути реалізована на різних платформах. Також потрібно розгорнути сервіс в хмарі, на операційній системі Linux чи Window.

2.2 Проектування сервісу

Для розробки сервісу була вибрана мова програмування C# та технологія ASP. Net Core 6.

Вони призначені для створення веб-програм різних типів та користується великим попитом. ASP.Net Core надає низку переваг:

- Підтримка кросплатформності
- Використання контейнерів Docker
- Високі вимоги до продуктивності та масштабованості
- Паралельне встановлення версій .Net за допомогою програми на тому самому сервері.

Це сучасний фреймворк який дає змогу швидко та надійно розробляти великі та малі системи, сервіси для різних потреб з використанням різних сучасних технологій.

В першу чергу потрібно спроектувати REST Api. Для роботи з сервісом було достатньо спроектувати декілька запитів. Це повинні бути post запити для того щоб на сервер можна було передати якусь інформацію. У випадку з сервісом компілятором, це повинен бути чи файл, чи string змінна.

Проектування компілятора, одна з найважливіших частин сервісу. Головною задачею було створити сервіс, який зможе обробляти файли та компілювати їх у багато-потоковому процесі. Для цього була використані правила асинхронного програмування за допомогою async await. Сам сервіс повинен бути піднятий в scoped режимі, щоб при кожному запиті компілятор створювався окремо для кожного користувача. Саме таким чином буде забезпечена швидкодія сервісу та потоко-безпека сервісу.

Основною небезпекою є можливість зловмисників загрузити файл з шкідливим кодом, наприклад Sql ін'єкцій. Для таких ситуацій в архітектурі компілятора закладено, що код який виконується у ньому, виконується в окремому контейнері, який ніяк не може вплинути безпосередньо на сервіс. Це розроблено для забезпечення роботи сервісу та для його захисту.

Додано swagger для наявної демонстрації методів Api, та для швидкого створення документації. Створення контрактів для об'єднання сервісу з front-end, та забезпечення їх взаємодії.

Також було спроектовано внутрішні зв'язки у сервісі. Усі внутрішні компоненти зв'язані за допомогою Dependency Injection. Це головний єднальний компонент у Asp Net Core. Було обрано для кожних компонентів їх життєвий цикл об'єктів.

Висновки до розділу 2

У даному розділі було розглянуто постановку задачі та покрокове проектування сервісу.

РОЗДІЛ 3. ОПИС ТА ПІДБІР ІНСТРУМЕНТІВ ДЛЯ РОЗРОБКИ. ПРОЦЕС РОЗРОБКИ

Підбір інструментів та методів розробки сервісу займає важливу частину підготовки, оскільки важливо розуміти, як ефективно використовувати обрані технології, задля зменшення помилок та кількості витраченого часу. Тому у даному розділі ознайомлено з використаними під час розробки інструментами, безпосередній опис розробки сервісу, та не менш важлива частина - тестування програми. Також варто звернути уваги на вимоги до апаратного забезпечення та розгортання сервісу. Посилання на github: https://github.com/BohdanZinchenko/WebCodeRunner

3.1 .Net та C#

.NET – це платформа від Microsoft, яка дозволяє створювати програмні додатки. Перший випуск .NET Framework відбувся в 2002 році. Вважається, що .NET Framework було створено в якості альтернативи платформі Java від компанії Sun. Головна відмінність полягає в тому, що .NET Framework офіційно розрахована на роботу саме з операційними системами родини Microsoft Windows.

В 2016 році додатково до .NET Framework було випущено модульну платформу .NET Core, сумісну з різноманітними операційними системами. Іншими словами, вона є кросплатформною. Кросплатформність .NET Core відкрила безліч нових сценаріїв і можливостей її застосування. Це зіграло суттєву роль в просуванні .NET серед розробників і представників бізнесу.

Мови й платформи створені для вирішення конкретних завдань і розробки відповідних програмних продуктів. .NET також має свою специфіку. При цьому ді-

апазон продуктів, над створенням яких працюють .NET-розробники, дуже широкий. . Загалом, все різноманіття програмних продуктів, які створюються під .NET, можна згрупувати наступним чином.

- Web Development
- Client Application
- Game Development
- Internet of Things (IoT)
- Enterprise

У розробці було використано платформу для створення веб-сервісу та серверної частини.Мабуть, це найбільш розповсюджена група програм, які пишуть під .NET. Особливість web-додатків полягає в тому, що вони працюють через браузер і вимагають, як правило, стабільного інтернет-підключення.

Web-додатки можуть бути різної складності. Для створення комплексного web-додатку потрібно багато зусиль. Яскравим прикладом додатку, простого з точки зору зовнішнього вигляду і складного з точки зору навантаження на серверну частину, є сайт Stack Overflow, відомий кожному розробнику.

Щоб написати web-додаток під .NET, необхідно знати C# і володіти фреймворком ASP.NET MVC. Також потрібно розуміти, що таке клієнт/сервер, як влаштований протокол HTTP, REST, JavaScript, розрізняти Frontend і Backend. І якщо мова йде про сучасну розробку, то важливо мати уявлення про домени, хостинги, плани, а ще – про хмарні технології (MS Azure, Amazon, Yandex Cloud).

Привабливість .NET для молодих спеціалістів можна пояснити не лише різноманіттям продуктових напрямів. Знання основ .NET дає вам змогу бути гнучкими у виборі спеціалізації та сфери програмування.

Можна піти в Backend, стати професіоналом у WCF та ASP.NET Core і писати сервіси для додатків. А використання Razor/Blazor дозволяє створювати під .NET

повноцінні web-додатки як з клієнтською, так і з серверною частинами. Якщо потрібен перехід до якого-небудь сучасного рушія (React/NG/Vue), то навчання для backend-розробника стане більш зрозумілим із застосуванням TypeScript – мови-обгортки над JavaScript, розробленої Microsoft спеціально для C#-програмістів.

Модульність ASP.NET Core забезпечується завдяки пакетному менеджеру NuGet. З його допомогою можна розширяти функціонал додатку за необхідності. Для цього необхідно лише відкрити менеджер пакеті, в пошуковому рядку знайти необхідну для Вас бібліотеку класів, обрати версію та встановити. [8]
Існує ще один аргумент на користь вибору .NET – це її дружність до початкового етапу навчання. У більшості на домашньому комп'ютері встановлено ОС Windows. Набагато простіше починати кар'єру саме з .NET-платформи, тому що вона працюватиме без встановлення та конфігурування додаткового програмного забезпечення. Інші платформи потребують спеціального налаштування Windows. Скажімо, якщо ви збираєтесь опановувати Java, то спочатку на комп'ютер потрібно встановити віртуальну машину Java. А для iOS взагалі придбати MacBook і редактор XCode.

Додаткова перевага .NET для початківців – розвинене .NET-ком'юніті. На таких популярних ресурсах, як GitHub або Stack Overflow, можна знайти відповіді на питання, які неминуче виникають у спеціалістів-початківців, і навіть подивитися приклади коду.

C# - це сучасна, об'єктно-орієнтована і статично типізована мова програмування. C# дає змогу розробникам будувати безліч безпечних і надійних програм що запускаються за допомогою .NET. Мова яка відноситься до C-подібних мов програмування і схожа на C, C++, Java та інші.

C# надає мовні конструкції для безпосередньої підтримки концепцій об'єктно-орієнтованої розробки, роблячи C# природною мовою для створення та використання програмних компонентів. З моменту свого виникнення в C# були

додані функції для підтримки нових робочих навантажень і нових методів проектування програмного забезпечення. За своєю суттю C# є об'єктно-орієнтованою мовою. Ви визначаєте типи та їх поведінку.

Кілька функцій C# допомагають створювати надійні та довговічні програми. Garbage Collector автоматично відновлює пам’ять, зайняту недоступними невикористаними об’єктами. Nullable types, захищають від змінних, які не посилаються на виділені об’єкти. Exception handling забезпечує структурований і розширений підхід до виявлення та відновлення помилок. Лямбда-вирази підтримують методи функціонального програмування. Language Integrated Query (LINQ) створює загальний шаблон для роботи з даними з будь-якого джерела.

Підтримка мови для асинхронних операцій забезпечує синтаксис для побудови розподілених систем. C# має уніфіковану систему типів. Усі типи C#, включаючи примітивні типи, такі як int і double, успадковуються від одного кореневого типу object. Крім того, C# підтримує як визначені користувачем reference types, так і value types. C# дозволяє динамічно розподіляти об'єкти та вбудовано зберігати полегшені структури. C# підтримує загальні методи та типи, які забезпечують підвищену безпеку та продуктивність типів. C# надає ітератори, які дають змогу реалізаторам класів колекції визначати користувацьку поведінку для клієнтського коду.

C# робить акцент на версійності, щоб програми та бібліотеки могли розвиватися з часом сумісним чином. Аспекти дизайну C#, на які безпосередньо вплинули міркування щодо версій, включають окремі віртуальні модифікатори та модифікатори перевизначення, правила вирішення перевантаження методів і підтримку явного оголошення членів інтерфейсу.

3.2 Visual Studio

Visual Studio - інтегроване середовище розробки (IDE) від Microsoft.

Його застосовують при розробці комп'ютерних програм, веб-програм, сайтів та додатків. Дане середовище дозволяє використовувати різні мови програмування при створенні додатків за допомогою нього, а також дозволяє розробляти додатки під платформи Android та iOS, а не лише під Windows.

Visual Studio містить у собі редактор коду і рефакторинг коду.

Підтримка мов, які не є вбудованими у Visual Studio відбувається завдяки плагінам. Крім цього , Visual Studio містить інструмент NuGet, який призначений для завантаження додатків та плагінів різного роду.

При створенні програм за допомогою Visual Studio можна використовувати два підходи. Перший із них передбачає для кожної задачі створення окремого об'єкту. У результаті по кожній задачі буде створено папку task ,в якій будуть міститися файли проекту.

Другий підхід є більш універсальним. В результаті його застосування проекти об'єднуються в Рішення. Одне рішення містить у собі одразу декілька проектів, що робить даний підхід кращим та більш раціональним.

3.3 Серверна частина

Серверна частина або back-end, по своїй суті увесь сервіс і є серверною частиною. Це back-end сервіс який виконує тяжкі операції на сервері до яких користувач не має доступу. Це використовується для того щоб життєво важливі частини сервісу були під захистом і звичайні користувачі не мали до нього доступу і ніяк не могли вплинути на виконання програми.

Наприклад, якби такий сервіс розроблявся, як front-end додаток, кожен користувач, який розуміється на мові програмування JavaScript і хоч трохи розуміє як створюється і працюють подібні сервіси, мав би змогу у результаті підмінити

значення і обвинуватити сервіс у некоректній роботі.

Кожен сучасний сервіс потребує свого серверу, чи машини яка вічно буде давати доступ до своїх ресурсів та підтримувати сервіс і даний проєкт не виняток. Він має свій сервер, який розташований в хмарі, місце на якій було куплено у Azure. Усі дії, що виконуються на сервері записуються в логи і розробник має можливість у будь-який момент зайти й переглянути їх. У логах зберігаються усі помилки і усі звернення користувачів до сервісу. Завдяки цьому можна в подальшому удосконалювати та налаштовувати сервіс, а також знаходити у ньому дефекти.

Наразі розробка серверів та їх підтримка являється найбільш ресурсо-затратною частиною розробки у вебі, адже ціна помилки дуже велика.

На серверах можна піднімати не лише образи сервісів за допомогою Docker контейнерів, чи звичайних розгортань под на сервер, який підтримує контеризацію самостійно, але й піднімати повноцінні бази даних. Для налаштування, серверу потрібно розуміти скільки пам'яті і яких затрат потребуватиме сервіс, адже якщо підняти сервер з недостатньою кількістю оперативної пам'яті, він перестане коректно працювати й буде видавати OutOfMemoryException, а якщо взяти занадто багато пам'яті, то кошти на підтримку будуть втрачатись в нікуди. На даний момент у кожно користувача є вільна можливість підняти свій власний сервер за допомогою таких сервісів як AWS, Azure, Google Cloud, Digital Ocean та інші.

3.4 Swagger

Swagger це набір інструментів, які допомагають описувати API. Завдяки йому користувачі краще розуміють можливості REST API без доступу до коду. За допомогою Swagger можна швидко створити документацію та надіслати її іншим розробникам чи клієнтам.

Основною функцією swagger є можливість протестувати запити REST API без зовнішньої UI частини чи іншого Http-клієнту. Він надає можливість відразу надсилати запити та отримувати відповіді від серверу у різних форматах, такі як json чи xml.

Swagger пропонує два основних підходи до генерування документації:

1) Автогенерація з урахуванням коду.
2) Самостійна розмітка-напис.

Перший підхід простіший. Додаються залежності в проект, конфігуруються налаштування та отримуємо документацію. Сам код через це може стати менш читабельний, документація теж не буде ідеальною. Але завдання вирішене — код задокументований.

Щоб користуватися другим підходом, необхідно знати синтаксис Swagger. Визначення Swagger може бути записано в JSON або YAML. (див. Рис. 3.1)[12]

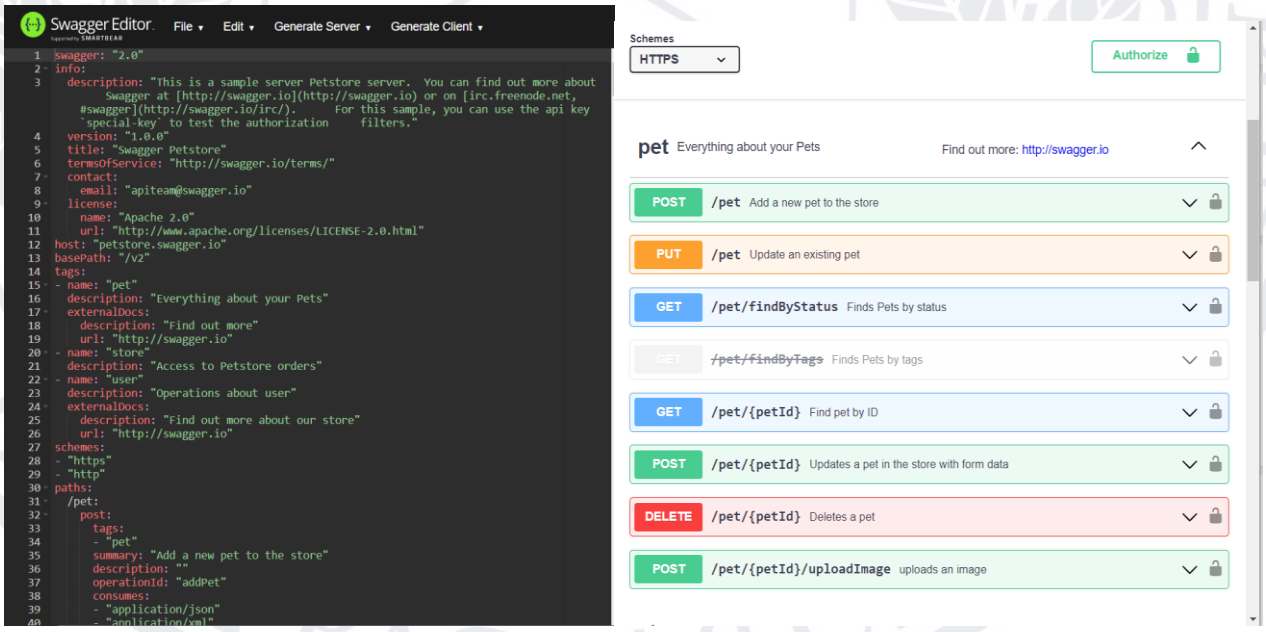

Рисунок 3.1.

Щоб додати Swagger до Web API, просто можна встановити проект із відкритим вихідним кодом під назвою Swashbuckle через NuGet. Після встановлення можна побачити swaggerconfig. cs у папці app_start в відповідного проекту.

Swashbuckle - це проект з відкритим вихідним кодом для створення документів Swagger для веб-інтерфейсів API, створених за допомогою ASP.NET Core. Є три основні компоненти: AspNetCore. SwaggerGen – надає функціональність для генерації JSON. Swashbuckle – це скоріше пакет (або бібліотека), який можна використовувати у своїх проектах .NET Web API. Його мета полягає у генерації специфікації Swagger для конкретного проекту. Крім того, інтерфейс користувача Swagger міститься в Swashbuckle, тому при розробці API в .NET, це хороший універсальний пакет. Він майже повністю написаний на C#. [7]

3.5 Git

Особливістю Git, яка дійсно виділяє його серед багатьох інших SCM, є його модель розгалуження.

Git дозволяє і заохочує створення кількох локальних гілок, які можуть бути повністю незалежними один від одного. Створення, об'єднання та видалення цих гілок займає секунди.

Це означає, що можна робити такі речі, як:

- Безперешкодне перемикання контексту. Можна створити гілку, щоб перевірити ідею, закомітити кілька разів, перейти назад на те місце, звідки була зроблена гілка, далі виправити помилки, перейти назад на те місце, де перевіряли, і об'єднати його.
- Кодові рядки на основі ролей. Означає мати гілку, яка завжди містить тільки те, що йде в продакшн; іншу, в яку зливається робота для тестування, та кілька маленьких для повсякденної роботи.
- Робочий процес з урахуванням функцій. Можливість створювати нові гілки для кожної нової функції, над якою працюєте, щоб можна було легко перемикатися між ними, а потім видаляти кожну гілку, коли ця функція буде об'єднана в основну лінію.

- Одноразові експерименти. Створення гілки для експериментів, після переконання, що вона не працює, легко видалити її, покинувши роботу, при цьому ніхто ніколи не побачить її (навіть якщо за цей час запушені інші гілки).

Варто зазначити, що під час передачі у віддалений репозиторій не обов'язково передавати усі свої гілки. Можна поділитися тільки однією зі своїх гілок, декількома чи всіма. Як правило, це звільняє людей від необхідності пробувати нові ідеї, не переймаючись тим, що їм доведеться планувати, як і коли вони збираються об'єднати їх або поділитися ними з іншими.

Існують способи реалізувати деякі з цих завдань за допомогою інших систем, але робота, пов'язана з цим, набагато складніша і часто супроводжується помилками. Git робить цей процес неймовірно простим та змінює підхід до роботи більшості розробників.

Git – це швидко. У Git майже всі операції виконуються локально, що дає йому величезну перевагу у швидкості в порівнянні з централізованими системами, яким постійно доводиться спілкуватися з сервером.

Git був створений для роботи на ядрі Linux, що означає, що він повинен був ефективно працювати з великими репозиторіями з першого дня. Git написано мовою C, що знижує накладні витрати на час виконання, пов'язані з мовами вищого рівня. Швидкість та продуктивність були основною метою розробки Git із самого початку.

Модель даних, яку використовує Git, забезпечує криптографічну цілісність кожного біта проекту. Кожен файл та кожен коміт перевіряється за контрольною сумою та витягується за контрольною сумою при зворотній перевірці. З Git неможливо отримати нічого, крім тих бітів, які туди були внесені.

Перед створенням нової гілки необхідно зафіксувати зміни в поточній гілці або видалити їх з робочого каталогу та області встановлення.

Щоб створити нову гілку та перейти до роботи над нею, виконується така команда:

git checkout -b <ім'я нової гілки>.
Батьківською гілкою буде та гілка, над якою працюєте на даний момент.
Є кілька базових команд, які потрібні для роботи з Git. Деякі з них:

- git add - додавання файлів в область зберігання

- git commit -m "message" - фіксування файлів у локальному репозиторії

- git reset - зняти зміни

- git status - перевірити статус файлів проекту

- git log - показати журнал фіксацій

Робота з віддаленим репозиторієм

- git clone url folder - створює копію віддаленого git-репозиторію з url у локальну папку

- також привласнює віддаленому репозиторію ім'я "origin" і встановлює його як віддалений за замовчуванням

- git remote add name url - створює зв'язок між поточним репозиторієм та названим віддаленим репозиторієм

3.6 Linux та Azure

Все більше і більше організацій вирішують розміщувати свої веб-програми в хмарі за допомогою сервісів як Microsoft Azure. Для розміщення сервісу було використано саме варіант з Azure, який має наступні переваги:

- Зниження інвестицій у витрати на центр обробки даних;
- Гнучкі ціни;
- Надзвичайна надійність;
- Покращена мобільність додатків;
- легко змінювати, де і як розгортається ваш додаток.
- Гнучка ємність;
- масштабувати вгору або вниз на основі реальних потреб.

Створення веб-додатків за допомогою ASP.NET Core, що розміщується в Azure, пропонує багато конкурентних переваг над традиційними альтернативами**.** ASP.NET Core оптимізовано для сучасної розробки веб-додатків та хмарних сценаріїв хостингу. Його модульна конструкція дозволяє додаткам залежати лише від тих функцій, які використовуються, удосконалюючи безпеку та продуктивність додатків при одночасному зниженні вимог до ресурсів хостингу. [11]

Також дана технологія повністю підтримує введення залежностей як внутрішньо, так і на рівні програми. Інтерфейси можуть мати декілька реалізацій, які при необхідності можуть бути замінені. Встановлення залежностей дозволяє програмам вільно з’єднуватися з цими інтерфейсами, що полегшує їх розширення, обслуговування та тестування.

Оскільки сервіс розгорнутий на операційній системі Linux, варто зазначити, що Linux - це Unix-подібна операційна система (ОС) з відкритим вихідним

кодом, розроблена спільнотою для комп'ютерів, серверів, мейнфреймів, мобільних пристроїв та пристроїв, що вбудовуються. Вона підтримується практично на всіх основних комп'ютерних платформах, включаючи x86, ARM і SPARC, що робить її однією з операційних систем, що найбільш широко підтримуються.

Кожна версія ОС Linux управляє апаратними ресурсами, запускає та обробляє програми, а також надає ту чи іншу форму інтерфейсу користувача. Величезне співтовариство розробників і широкий спектр дистрибутивів означає, що версія Linux доступна практично для будь-якого завдання, і Linux проникла в багато сфер обчислювальної техніки.

Наприклад, Linux стала популярною ОС для веб-серверів, таких як Apache, а також для мережевих операцій, наукових обчислень, що вимагають величезних обчислювальних кластерів, роботи з базами даних, настільних та кінцевих комп'ютерів та мобільних пристроїв із такими версіями ОС, як Android. [9],[13]

Отож Linux має високу конфігурованість і залежить від модульної конструкції, яка дозволяє користувачам налаштовувати власні версії Linux. Тому, залежно від програми, Linux може бути оптимізований для різних цілей.

3.7 Json

JSON (JavaScript Object Notation) – це легкий формат обміну даними. Його легко читати та писати людям. Машини легко розбирають та генерують його. Він заснований на підмножині стандарту мови програмування JavaScript ECMA-262 3rd Edition - December 1999. JSON - це текстовий формат, який повністю незалежний від мови, але використовує угоди, знайомі програмістам сімейства мов C, включаючи C, C++, C#, Java, JavaScript, Perl, Python та багато інших. Ці властивості роблять JSON ідеальною мовою обміну даними. [16]

Json – це головний транспортний об’єкт, за допомогою якого можна об’єднувати back-end та front-end частини. У більшості випадків, коли front-end надсилає дані на back-end, вони будуть відформатовані в JSON, і коли back-end отримає їх, вони будуть розшифровані з JSON назад до об'єкту. Також у зворотному напрямку, коли back-end надсилає деякі дані на front-end, вони будуть відформатовані в JSON, і коли ваш front-end отримає ці дані, вони будуть розібрані з JSON назад до об'єкту.

Оскільки більшість back-end програмістів створюють API, використовувані різними клієнтами, це можуть бути різні веб-сайти, побудовані на Java, .Net, PHP чи мобільні клієнти. Можна легко знайти парсер для об'єктного формату JSON у всіх мовах, які використовуються для створення програмного забезпечення. Таким чином, використання JSON допомагає створювати більш досконалі back-end системи.

В останні кілька років спостерігається зростання баз даних NoSQL, багато з яких використовують формат JSON для зберігання даних.

JSON побудований на двох структурах:

Колекція пар ім'я/значення. У різних мовах це реалізується як об'єкт, запис, структура, словник, хеш-таблиця, список із ключами або асоціативний масив.

Упорядкований перелік значень. У більшості мов він реалізується як масив, вектор, список чи послідовність.

Це є універсальні структури даних. Майже всі сучасні мови програмування підтримують в тій чи іншій формі. Логічно, що формат даних, що взаємозамінний з мовами програмування, також має бути заснований на цих структурах.

У JSON вони мають такі форми:

Об'єкт - це невпорядкований набір пар ім'я/значення. Об'єкт починається з {ліва дужка і закінчується} правою дужкою. За кожним ім'ям слід: двокрапка, а пари ім'я/значення розділяються комою.

Масив – це впорядкована колекція значень. Масив починається з [лівої дужки і закінчується] правою дужкою. Значення розділяються комою.

Значенням може бути рядок у подвійних лапках, чи число, чи true, false чи null, чи об'єкт, чи масив. Ці структури може бути вкладеними.

Рядок - це послідовність з нуля або більше символів Unicode, укладених у подвійні лапки з використанням зворотного слешу. Символ представлений як рядок з одного символу. Рядок дуже схожий на рядок мови C або Java.

Число дуже схоже на число мовою C або Java, за винятком того, що вісімковий та шістнадцятковий формати не використовуються. [6]

JSON легко використовувати на передньому плані (в браузері). Розвиток front-end фреймворків, таких як Angular, призвело до того, що багато веб-сайтів/веб-додатків перейшли від серверів, що обслуговують html-сторінки, до простого обслуговування, достатнього для завантаження веб-додатку, а потім передачі всіх даних через API за допомогою JSON. У майбутньому, у міру того, як веб-сайти будуть все більш складними і "схожими на програми", JSON, ймовірно, стане ще більш поширеним.

3.8 Розробка сервісу

Перед початком розробки сервісу, було створення проекту та гіт репозиторія. Для проекту було обрана Default Web Api в конструкторі Api у visual studio. Одразу було створено папку з контролером, прості настройки сервісу з доданими залежностями у dependency injections. Наступним кроком було підключення swagger, та push першого коміту на гіт, для заповнення репозиторію.

Так як розробкою сервісу займалася одна людина, не було потреби в створенні

інших допоміжних гілок, достатньо було вести розробку в одній вітці.

Далі був сетап контролера. Тут відбулося його початкове налаштування, написання відносних шляхів. Для цього були використані практики Microsoft та їх “синтаксичний цукор”. Додавання усіх post запитів, та початкова їх реалізація.

Для роботи з сервісами було обрано паттерн CQRS, який розшифровується як Command and Query Responsibility Segregation, шаблон, який розділяє операції читання та оновлення для сховища даних. Реалізація CQRS у програмі може максимально підвищити її продуктивність, масштабованість та безпеку. Гнучкість, створена в результаті переходу на CQRS, дозволяє системі краще розвиватися з часом і запобігає тому, щоб команди оновлення викликали конфлікти злиття на рівні домену. Щоб реалізувати даний патерн було використано бібліотеку Mediatr.

Mediatr — це поведінковий шаблон дизайну, який допомагає зменшити хаотичні залежності між об’єктами. Шаблон обмежує прямий зв'язок між об'єктами і змушує їх співпрацювати тільки через об'єкт-посередник. Посередник використовується для зменшення складності зв'язку між кількома об'єктами або класами. Цей шаблон надає клас-посередник, який зазвичай обробляє всі комунікації між різними класами і підтримує легке обслуговування коду за рахунок слабкого зв’язку. [10]

Наступною була розробка компілятора. Основною проблемою було створити компілятор всередині іншої програми, по складності це схоже зі створенням власної мови програмування. Тут також є активні компоненти, які мають вірно транслюватися в машинний код. Сервіс повинен був перевести текст з файлу, у машинний код та виконати його так, наче це створена і працююча програма. Було

розроблено алгоритм, за яким код, що надсилався до сервісу, спочатку перетворювався у бібліотеку з розширенням .dll, а далі зберігався серед файлів. Після чого викликався спеціальною функцією в асинхронному методі.

```
public object? Compilation(string code, List<object> testInfo,string methodName)
{
    var tree = SyntaxFactory.ParseSyntaxTree(code);
    string fileName=$"{methodName}{Guid.NewGuid().ToString()}-.dll";

    var systemRefLocation :string =typeof(object).GetTypeInfo().Assembly.Location;

    var systemReference = MetadataReference.CreateFromFile(systemRefLocation);

    var compilation = CSharpCompilation.Create(fileName)
        .WithOptions(
            new CSharpCompilationOptions(OutputKind.DynamicallyLinkedLibrary))
        .AddReferences(systemReference)
        .AddSyntaxTrees(tree);

    string path = Path.Combine(Directory.GetCurrentDirectory(), fileName);
    EmitResult compilationResult = compilation.Emit(path);

    if (!compilationResult.Success) return null;

    Assembly asm =
        AssemblyLoadContext.Default.LoadFromAssemblyPath(path);

    object? result =
        asm.GetType( name: "MyProgram.Program") // Type?
            ?.GetMethod(methodName)?. // MethodInfo?
            Invoke( obj: null, parameters: testInfo.ToArray());
    return result;
}
```

Рисунок 3.2 (Поверхнева частина сервісу компіляції)

Наступною задачею стало надати можливість сервісу приймати файли та розпізнавати текст у них. Для цього використані бібліотеки самого Asp Net Core, завдяки яким легко вдалося перетворити файл в байтовий потік та прочитати його. Теж саме було зроблено і з звичайними string змінними, але вони вже од-

разу вважались як програмний код і починали проходити весь алгоритм компіляції.

Крайньою задачею, була розробка уніфікированого сервісу перевірки, який міг перевірити різні типи задач, на їх правильність виконання. Для цього було створено клас, у якому зберігалися змінні, та результат їх виконнаня. Після компіляції коду, саме ці змінні надсилалися у робочу бібліотеку, яка була створення на основі користувацького коду, і порівнювалися з результатом.

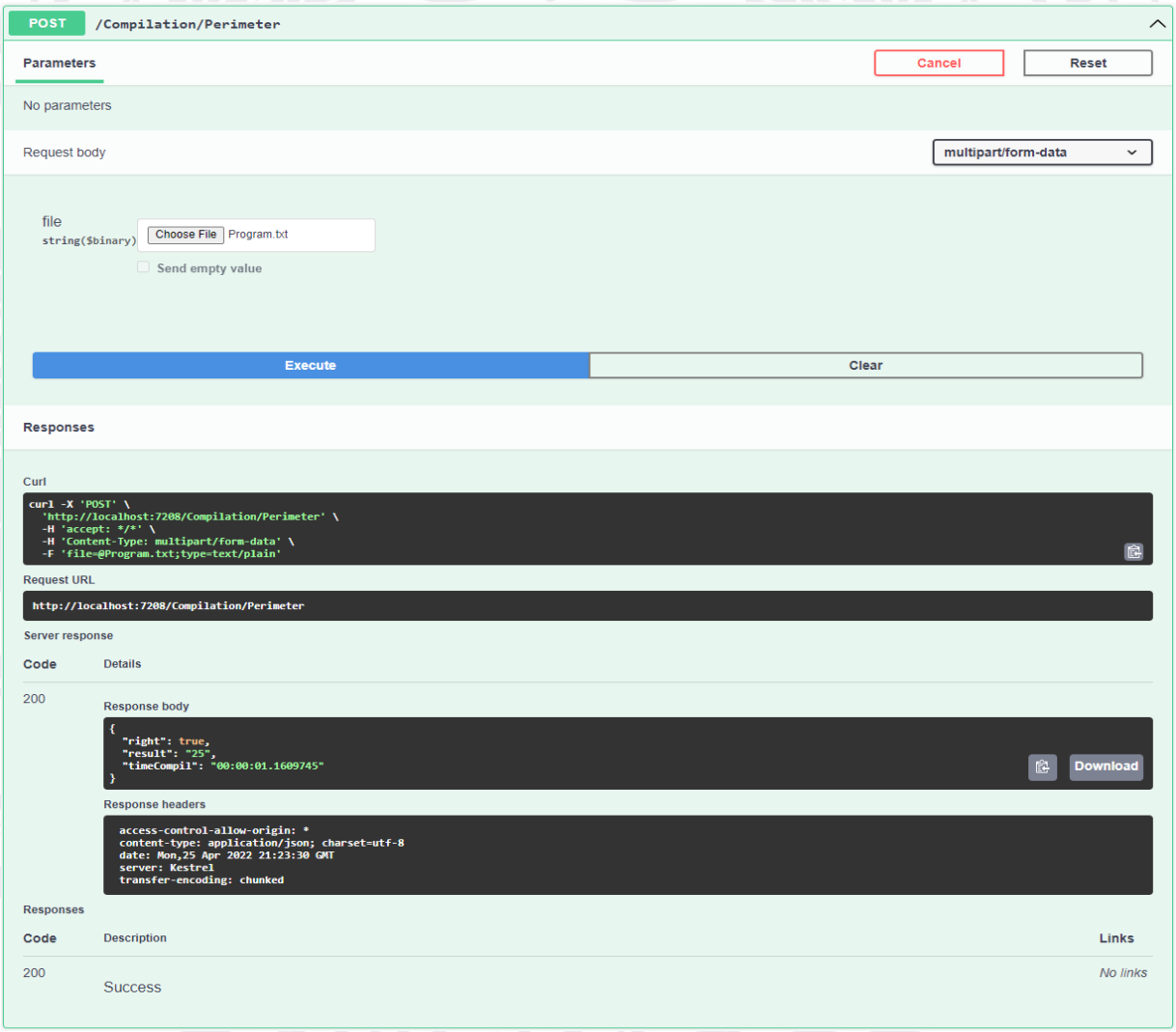

Рисунок 3.3 (Працюючий сервіс)

У наступному кроці підключено усі бібліотеки та заведенно сервіси у єдиний реєстр сервісів для користування. Саме після цього можна було повноцінно

використовувати сервіс, усі залежності були об'єднані у dependency injection і викликалися за потреби.

Останнім кроком стало розгорнення сервісу на хмарі Azure. За допомогою влаштованого у visual studio, розгортувача, який міг підхопити і реалізувати Docker контейнер на хмарі. Було створено докер файл, який збирав та загортав проект у поду, і після цього розгортувач отримував цю поду, та розміщував її.

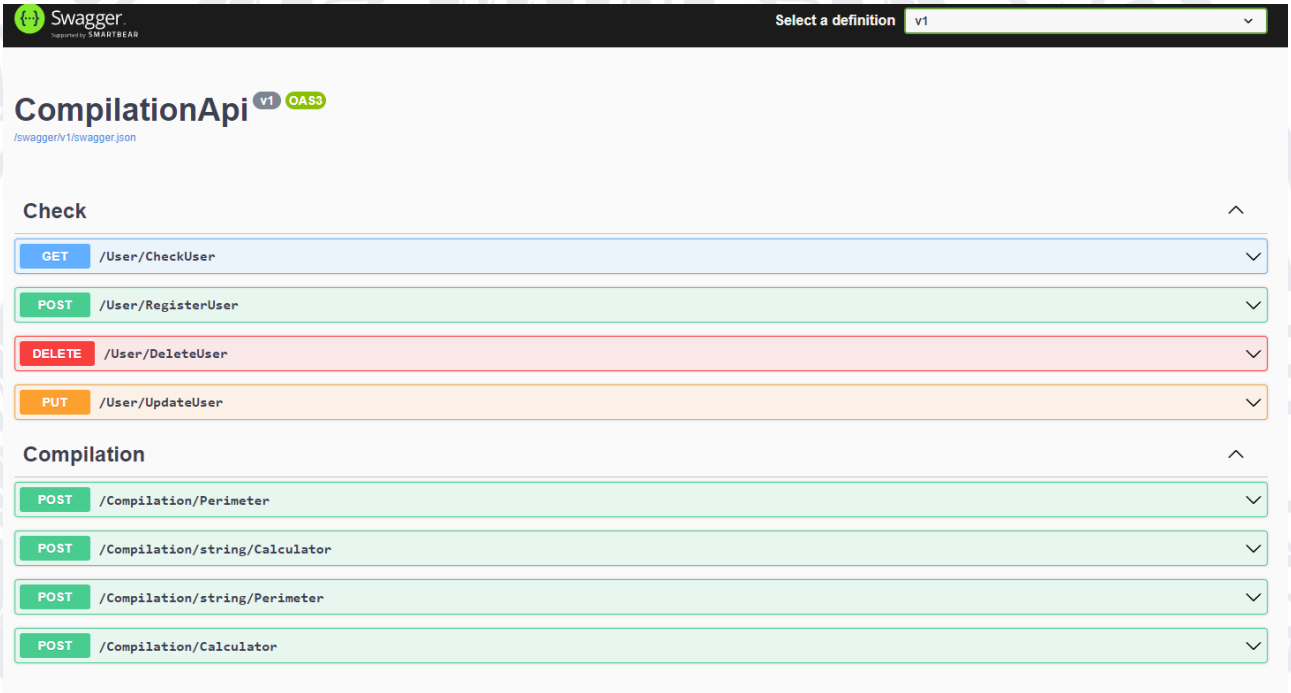

Рисунок 3.4 Swagger розгорнутого сервісу

3.9 Тестування сервісу

Будь-яке програмне забезпечення може вийти з ладу несподіваними способами у відповідь на зміни. Щоб такого не сталося потрібно проводити тести для всіх внесених змін. Таке тестування потрібно проводити для всіх додатків.

Ручне тестування – це дуже повільний та дорогий спосіб тестування програмного забезпечення. Окрім цього надійність таких тестів також викликає сум-

ніви. Такий спосіб тестування використовують лише тоді, коли відсутні спеціальні додатки для тестування, а ручне тестування, в такому випадку, стає єдиним можливим варіантом.

Додатки, розроблені за допомогою ASP.NET Core, підтримують автоматизоване інтегрування функціональних тестів, так званих Unit Tests.

Існує багато видів автоматизованих тестів для програмних додатків, а найпростішим вважається одиничний тест. Окрім цього існують також інші види тестів:

- Інтеграційні тести;
- Функціональні тести;
- Unit тести;
- Піраміда тестів.

Інтеграційні тести використовують для перевірки правильної взаємодії між шарами коду. Такі тести, як правило, повільніші і складніші у налаштуванні, на відміну від одиничних тестів, адже вони часто залежать від зовнішніх зв'язків та інфраструктури. Через це, якщо не виникає гострої необхідності для використання інтеграційних тестів, то краще використовувати модульні або одиничні тести.

Функціональні тести – тести написані з точки зору користувача та покликані для того, щоб перевірити правильність системи на основі її вимог. Функціональні тести працюють на системному рівні і можуть вимагати певного ступеня автоматизації інтерфейсу.

Unit тести використовуються для перевірки однієї частини логіки додатку. Модульний тест не перевіряє взаємодію коду з іншими частинами програми,

адже для цього призначені інтеграційні тести. Модульний тест не перевіряє правильність фреймворку, який був використаний для написання коду. Модульний тест працює повністю в пам’яті і в процесі та не взаємодіє з файловою системою чи базою даних. Такі тести перевіряють лише написаний програмістом код.

В силу того, що модульні тести перевіряють лише написаний код, то виконуються вони дуже швидко. Таким чином є можливість запускати багато одиничних тестів за кілька секунд.

Тестування піраміди. В такому випадку програма представлена шарами, де на вершині проводиться порівняно менша кількість функціональних тестів. Посередині піраміди знаходяться інтеграційні тести, а в основі піраміди лежать unit тести.

3.10 Вимоги до апаратного забезпечення та розгортання системи

Розгортання додатків, написаних з допомогою ASP.NET Core, можна легко автоматизувати з допомогою безперервного інтегрування та безперервно розгортання – так зване CI/CD. Microsoft Azure має інтегровану підтримку сховищ Git і може автоматично розгортати оновлення до вказаної вітки чи тегу. Azure DevOps забезпечує повнофункціональну CI/CD збірку та конвеєр розгортання, а GitHub Action надає ще один варіант для розміщених в ньому проектів.

Для написання додатку, використовується технологія ASP.NET Core. Вона є кросплатформною, веб орієнтованою, має велику пропускну здатність і займає мало пам’яті за рахунок того, що технологія є модульною. Дана програма запускається вона на сервері під управлінням відомих операційних систем.

Таблиця 1

Операційна система: · Windows; · Linux; · MacOS.
Процесор: Будь-які серверні рішення від AMD чи Intel, не старіші 6-го покоління.
ОЗУ: 2GB і більше.

Першим кроком для розгортання подібної системи є її публікація за допомогою команди публікації dotnet CLI. Це компілює програму та розмістить всі необхідні файли в спеціальну директорію. Під час розгортання Visual Studio цей крок виконується автоматично.

Програми ASP.NET Core - це консольні програми, які потрібно запускати під час завантаження сервера та перезапустити, якщо програма (або сервер) аварійно завершує роботу. Для автоматизації цього можна використовувати менеджер процесів. Найпоширенішими менеджерами процесів для ASP.NET Core є Nginx або Apache на Linux та IIS або Windows Service у Windows.

Висновки розділу 3

У даному розділі було детально розглянуто технології та реалізовані сучасні патерни програмування. Було розроблено сервіс компіляції та перевірки програмного коду та його розгортання у хмарі.

ВИСНОВКИ

З усього вище сказаного можна зробити невеликий підсумок по декільком пунктам, так як у даній роботі було розглянуто можливість розробки сервісу компіляції коду, то:

1) Даний сервіс можна використовувати у різних галузях і різними спеціалістами від початкового рівню до професіоналів справи.

2) Розглянуто сучасні технології та їх розвиток для розуміння як краще на сьогодні створювати сервіси.

3) Проаналізовано різноманітність систем, та варіантів розгортання одного сервісу на безлічі з них.

4) Досліджено роботу з технологію .Net та аналіз її сильних й сторін. Розглянуто сучасні методи проектування важких систем, їх створення та подальшу підтримку.

5) Важливість даної наукової роботи в її актуальності та потребі тих можливостей, які сервіс надає, а саме: а) допомога в розвитку програмістів, б) спрощення перевірки програмного коду викладачам, тим самим вирішуємо проблематику роботи – економія часу і ресурсів, в) швидкість та якість перевірки програмного коду.

Дана робота не являється плагіатом.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Referatu.net.ua [Електронний ресурс]: «Комп'ютерна техніка та інформатика». - Режим доступу: http://referatu.net.ua/referats/94/40175

2.Bestreferat.ru [Електронний ресурс]: «Використання комп'ютерів в суспільстві, роль програмного забезпечення». - Режим доступу: https://www.bestreferat.ru/referat-115494.html

3.Studcon.org [Електронний ресурс]: «Роль комп'ютерної техніки у сучасному суспільстві». - Режим доступу: http://studcon.org/rol-kompyuternoyi-tehniky-u-suchasnomu-suspilstvi

4.Codewars.com [Електронний ресурс]: «Codewars» - Режим доступу: https://www.codewars.com

5.Codeforces.com [Електронний ресурс]: «Codeforces» - Режим доступу: http://codeforces.com

6. Json.org [Електронний ресурс]: «Introducing JSON» - Режим доступу: https://www.json.org/json-en.html

7.Docs.microsoft.com [Електронний ресурс]: «ASP.NET Core» - Режим доступу: https://docs.microsoft.com/en-us/aspnet/core/tutorials/getting-started-with-swashbuckle?view=aspnetcore-6.0&tabs=visual-studio

8. Metanit.com [Електронний ресурс]: «ASP.NET Core». – Режим доступу: https://metanit.com/sharp/aspnet5/1.1.php

9. Linux.com [Електронний ресурс]: «What is Linux?». – Режим доступу: https://www.linux.com/what-is-linux/

10. Medium.com [Електронний ресурс]: «Use MediatR in Asp.Net core» - Режим доступу: https://medium.com/dotnet-hub/use-mediatr-in-asp-net-or-asp-net-core-cqrs-and-mediator-in-dotnet-how-to-use-mediatr-cqrs-aspnetcore-5076e2f2880c

11. Azure.microsoft.com [Електронний ресурс]: «Azure» – режим доступу: https://azure.microsoft.com/

13. Habr.com [Електронний ресурс]: «Swagger and Asp net core» – режим доступу: https://habr.com/

14. Linux.org [Електронний ресурс]: «Linux» – Режим доступу : https://www.linux.org/

15. Studcon.org [Електронний ресурс]: «Роль комп'ютерної техніки у сучасному суспільстві». - Режим доступу: http://studcon.org/rol-kompyuternoyi-tehniky-u-suchasnomu-suspilstvi

16. Wiki [Електронний ресурс]: «JSON». – Режим доступу: https://uk.wikipedia.org/wiki/JSON

17. Intel [Електроний ресурс]: «Intel» – Режим доступу: https://www.intel.com/content/www/us/en/homepage.html

18. AMD [Електроний ресурс]: «AMD» – Режим доступу: https://www.amd.com/en

19. Amazon Web Services [Електроний ресурс]: «AWD» - Режим доступу: https://aws.amazon.com

20. Investopedia [Електроний ресурс]: «Original Equipment Manufacturer» - Режим доступу: https://www.investopedia.com/terms/o/oem.asp

21. Visual Studio Code [Електроний ресурс]: «Visual Studio Code» - Режим доступу: https://code.visualstudio.com/

22. JetBrains [Електроний ресурс]: «JetBrains» - Режим доступу: https://www.jetbrains.com/

23. Myservername [Електроний ресурс]: «Найкращі 22 Інструменти компілятора c++» - Режим доступу: https://uk.myservername.com/top-22-online-c-compiler-tools-best-c-ide

24. Arm [Електроний ресурс]: «Arm» - Режим доступу: https://www.arm.com/

25. Apple [Електроний ресурс]: «MacOs Monterey» - Режим доступу: https://www.apple.com/macos/monterey/

26. RedHat [Електроний ресурс]: «What is serverless» Режим доступу: - https://www.redhat.com/en/topics/cloud-native-apps/what-is-serverless

27. Global Logic [Електроний ресурс]: «Мікросервісна архітектура» Режим доступу: - https://www.globallogic.com/ua/insights/blogs/microservices-architecture-

for-beginners-part-one/

28. Java Logic [Електроний ресурс]: «Java» Режим доступу: https://www.java.com/

29. Dev.ua [Електроний ресурс]: «Компілятори» Режим доступу: https://dev.ua/news

30. Swagger [Електроний ресурс]: «Swagger» Режим доступу: https://swagger.io/