МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ДОНЕЦЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ ІМЕНІ ВАСИЛЯ СТУСА

ДУДНИК МАКСИМ ВАЛЕРІЙОВИЧ

Допускається до захисту:
в.о. завідувача кафедри
інформаційних технологій
канд. техн. наук, доцент
_____ О.В.Зелінська
«__» _____2024р.

РОЗРОБКА СЕРВІСУ ДЛЯ МЕНЕДЖМЕНТУ ІНФРАСТРУКТУРИ

Спеціальність 122 Комп’ютерні науки

Кваліфікаційна (бакалаврська) робота

Керівник:
О.В.Зелінська , доцент кафедри
інформаційних технологій,
к. т. н., доцент
_____

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

Вінниця - 2024

АНОТАЦІЯ

Дудник М.В. Розробка сервісу для менеджменту інфраструктури. Спеціальність 122 «Комп’ютерні науки». Донецький національний університет імені Василя Стуса, Вінниця, 2024.

У кваліфікаційній (бакалаврській) роботі було розглянуто та структуровано проблеми, які виникають під час підтримки інфраструктури, проаналізовано цифрові інструменти для їх вирішення. За допомогою технологій C#, .NET та .JSON було розроблено сервіс для вирішення визначених завдань.

Ключові слова: інфраструктура, облік, C#, .Net, багатопотоковість, мережа.

57 с., 3 табл., 27 рис., 41 джерело

ABSTRACT

Dudnyk M.V. Development of a Service for Infrastructure Management. Specialty 122 “Computer Science”. Vasyl' Stus Donetsk National University, Vinnytsia, 2024.

In the qualification (bachelor's) thesis, the problems that arise during infrastructure support were considered and structured, and digital tools for solving them were analyzed. Using C#, .NET and .JSON technologies, a service was developed to solve the identified problems.

Keywords: infrastructure, accounting, C#, .Net, multithreading, network.

57 p., 3 tables, 27 figures, 41 sources

ЗМІСТ

ВСТУП

Актуальність. В сучасному світі промисловість, економіка та соціум є основними факторами глобального людського добробуту і представляють з себе певний набір активів та складних взаємозв'язків між ними. З цього формується певна достатньо складна інфраструктура, оптимізація якої є складним і довгим процесом, в якому майже завжди своє місце знаходить людський фактор.

Мета кваліфікаційної (бакалаврської) роботи полягає в аналізі проблем сучасної інфраструктури, та створенні технічного рішення, яке оптимізує процес її менеджменту та контролю процесів, цим самим забезпечивши ще більш стрімкий розвиток впливових сфер людської діяльності.

Для досягнення мети, автор вирішив такі **завдання**:

- Структуризація проблем при підтримці інфраструктури
- Аналіз існуючих технічних засобів їх вирішення, їх переваг та недоліків
- Вибір інструментів для реалізації програмного рішення.
- Створення сервісу, для вирішення завдань. Можливості та потенціалу для його розвитку

Об'єктом дослідження є процеси створення сервісу для менеджменту інфраструктури.

Предмет дослідження – методи та процеси створення сервісів. Дослідження технічних рішень для оптимізації процесів та мінімізація можливості помилки.

РОЗДІЛ 1
АНАЛІЗ ТА СТРУКТУРИЗАЦІЯ ПОТРЕБ ІНФРАСТРУКТУРИ

У даному розділі більш детально сформовано проблеми та складності в підтримці інфраструктури та аналітика існуючих засобів для їх вирішення.

1.1 Аналіз потреб у підтримці інфраструктури

В сучасному світі щодня стрімко розвивається промисловість та економіка, щодня з’являються нові підприємства, які значно впливають на добробут та якість життя, збільшуючи перелік послуг, які можуть бути надані населенню та бізнесу. Кожнне підприємство також збільшує кількість робочих місць, що значно впливає на економічні показники держав. В Україні, в стані війни не можна недооцінювати важливість існування підприємств та бізнесу, адже кожне підприємство робить значний вклад в економіку, що в свою чергу впливає на обороноздатність країни[1].

Інфраструктурою підприємства можна вважати сукупність ресурсів підприємства так як кожне підприємство має свою внутрішню структуру, яка складається з працівників та ресурсів, які використовуються для генерації доданої вартості. Цей процес може бути представлений в різному вигляді, наприклад в кав’ярнях працівники надають клієнтам послгуи у вигляді приготування кави та наданні емоцій при відвідуванні кав’ярні. Собівартість інгредієнтів кави є незначною, проте вона збільшується при приготуванні кави, яке виконується працівником.

Якщо від прибутку відняти оплату працівникам та вартість ресурсів, то можна отримати додану вартість, що і буде показником згенерованого прибутку.

Для аналізу можливостей підтримки та збільшення ефективності роботи варто дослідити кілька прикладів інфраструктури, на їх основі варто побудувати основні характеристики, принципи роботи та проблеми її підтримки:

- Рекламна агенція (промислова інфраструктура)

- Завод літаків (промислова інфраструктура)
- Організація концерту (соціальна інфраструктура)

Принцип роботи рекламної агенції полягає в розміщенні фізичної реклами в визначених місцях. Кожне визначене для розміщення реклами місце має свої вимоги до рекламного банеру. При великих обсягах роботи з'являється проблема з менеджментом банерів. Так як кожен банер має свої фізичні характеристики та фізичний розмір, власникам компанії потрібно буде шукати вирішення проблеми з обліком та фізичним пошуком ресурсів на складі.

Також при великих обсягах роботи, при використанні технічних рішень, одного пристрою буде недостатньо для обліку, так як ним потрібно буде користуватись великій кількості працівників, що значно сповільнить роботу з пристроєм обліку.

На даному етапі можна виділити основні проблеми такої інфраструктури:

- Необхідність обліку ресусів
- Централізованість
- Фізичний пошук ресурсів

В прикладі з заводом літаків існує велика кількість деталей, необхідних для будівництва літаків. Зазвичай одночасно будується не одна модель літака, а запчастини між ними дуже часто одні й ті самі. В такому випадку також потрібно вести облік ресурсів, який в свою чергу буде використовуватися не однією людиною, так як виготовлення літальних апаратів це складний і довгий процес, в якому залучена велика кількість людей. В цьому прикладі, так само як і з агенством реклами можна знайти спільні проблеми в підтримці інфраструктури [9].

Організація концерту відрізняється від попередніх прикладів, адже це приклад соціальної інфраструктури, в якій немає фізичних ресурсів[10]. Для організації концерту потрібно знайти музичні гурти, місце для проведення концерту, також організувати охорону, касирів та технічний персонал. В цьому

прикладі ресурсами є персонал та місця проведення заходу. Для кожного місця необхідна різна кількість персоналу.

Хоча це і є прикладом складнішої інфраструктури, проте її підтримка зводиться до однієї проблеми – обліку. Так як при великій кількості можливих місць проведення концертів, організаторам потрібно знайти те місце, яке буде відповідати їх очікуванням щодо кількості місць та залученого персоналу.

Отже, на прикладах можна побачити, що для підтримки будь-якого виду інфраструктури необхідно технічний інструмент, який допоможе її підтримувати. Хоча приклади різні, проблеми для вирішення в них одні й ті самі – облік ресурсів, централізованість та необхідність фізичного пошуку ресурсів.

1.2 Аналіз існуючих сервісів для підтримки інфраструктури

Підтримка інфраструктури та оптимізація роботи з нею є поширеною проблемою [2] у сучасному світі, тому на сьогодні існує багато програмних рішень [3, 11]. Варто розглянути найпопулярніші з них та виділити їх переваги та недоліки. Це допоможе покращити якість сервісу створеного в результаті цього дослідження.

Mirosoft Excel [4] – один з найпоширеніших засобів для вирішення технічних проблем та розрахунків великих обсягів складних даних[5]. Свою популярність здобув завдяки простоті у використанні, функціональності та універсальності у використанні. Цей інструмент можливо використовувати при менеджменті інфраструктури, проте це призведе і до ряду проблем.
Серед проблем варто виділити:

- Централізованість.
- Наростаюча складність при великих обсягах даних.
- Повна візуалізація даних.
- Дороговизна використання

Проблема централізованості полягає у тому, що всі дані зберігаюься в одному файлі і на одному пристрої. Синхронізований доступ до цих даних з кількох пристроїв неможливий. Це створює складності для роботи з даними, коли доступ до їх змін необіхдно мати двом і більше працівникам, так як для змін може бути використаний лише один пристрій.

Однією з критичних проблем використання цього сервісу є наростаюча складність при роботі з великими обсягами даних. Ця проблема полягає в тому, що усі дані в Excel візуалізуються у вигляді таблиць. При великих обсягах даних, вони не можуть бути візуалізовані повністю, тому користувачам потрібно витрачати час на пошук потрібних їм даних. Крім цього, при великій кількості даних інтерфейс для роботи з ними виглядає перевантажено, користувач замість потрібних йому даних також бачить інформацію, яка йому не потрібна для обробки. Це подовжує пошук необхідних даних та робить використання цього сервісу неефективним.

Не зважаючи на проблеми, Excel також має ряд переваг:

- Швидкість та недороговартість навчання
- Універсальність у використанні
- Простота використання при малих обсягах даних

Однією з переваг Microsoft Excel є його швидкість та недороговартість навчання. Завдяки зручному інтерфейсу та великій кількості навчальних матеріалів користувачі можуть швидко освоїти основні функції та можливості цієї програми. Існує велика кількість відеоуроків, довідкових матеріалів та форумів, які дозволяють користувачам у відносно короткі терміни отримати необхідні навички, які необхідні для роботи з цим сервісом[12]. Такий легкий доступ до навчальних ресурсів робить Excel доступним для широкого кола користувачів, що особливо важливо для малих та середніх підприємств, які прагнуть оптимізувати свої процеси за мінімальні витрати.

Іншою перевагою Excel є його універсальність у використанні, що робить його зручним інструментом у багатьох галузях та для різних завдань. Його можна застосовувати для фінансового аналізу, управління проектами,

обробки статистичних даних, створення графіків та діаграм, а також для автоматизації рутинних задач за допомогою макросів. Завдяки широкому спектру можливостей, Excel легко може бути використаним до потреб будь-якого бізнесу або особистого використання. Це робить його універсальним інструментом, який може ефективно використовуватись як в офісних умовах, так і для домашніх завдань.

Простота використання Excel проявляється при роботі з малими обсягами даних. Його інтерфейс дозволяє швидко редагувати дані, виконувати обчислення, а також створювати прості таблиці та графіки. При невеликій кількості інформації Excel надає високу швидкість та зручність обробки, що дозволяє користувачам швидко отримувати потрібні результати.

Отже, хоча цей сервіс і вирішує проблему обліку та фізичного пошуку ресурсів, він створює нові проблеми у вигляді складносі їх обробки при великих обсягах даних. Також він не вирішує проблему централізованості, що критично важливо для підтримки складної інфраструктури.

IBM Maximo[13] це комплексна інструментальна система для менеджменту інфраструктури, яка розрахована на підтримку великої кількості даних та виконання складних розрахунків, особливо в контексті технічних об’єктів.

Цей сервіс варто розглядати як комплексну систему управління активами (Enterprise Asset Management system, EAM), що використовується для планування, виконання та моніторингу активів підприємства. Вона базується на принципах системного підходу до управління активами, використовуючи інтегровані методи та інструменти для оптимізації ресурсів, зниження ризиків, збільшення ефективності підприємства.

Проте, як і з попереднім прикладом, IBM Maximo має свої недоліки:

- Висока складність навчання
- Висока ціна використання
- Орієнтовність на технічну інфраструктуру

Однією з проблем IBM Maximo можна виділити високу складність навчання, вона проявляється в тому, що для ефективного використання цього сервісу необхідно пройти спеціалізоване навчання, яке може зайняти значний час і потребувати додаткових витрат. Складність системи може стати перешкодою для впровадження, особливо для малих та середніх підприємств, які не мають можливості виділити достатні ресурси для навчання персоналу.

Іншою проблемою є висока ціна використання. IBM Maximo є дорогим інструментом, це може бути непосильним для багатьох підприємств. Високі витрати на ліцензування та підтримку можуть перевищувати бюджет, що робить цю систему доступною лише для достатньо великих компаній.

Ще одним з недоліків можна вважати орієнтованість цього сервісу на технічну інфраструктуру. Хоча IBM Maximo має потужні інструменти для управління технічними активами, її можливості можуть бути надмірними для інших типів інфраструктури, що не потребують такого високого рівня технічної деталізації. Це обмежує універсальність використання системи у менш технічно насичених інфраструктурах.

Також варто виділити переваги цього сервісу:

- Велика кількість інтегрованих інструментів обліку
- Зручний інтерфейс
- Децентралізація

Однією з переваг IBM Maximo є значна кількість інтегрованих інструментів обліку. Система дозволяє комплексно керувати активами, здійснювати їх облік та моніторинг, що забезпечує високу точність та ефективність використання.

Також перевагою є зручний інтерфейс. Незважаючи на складність системи, можна відзначити інтуїтивно зрозумілий інтерфейс, який полегшує використання функцій та інструментів. Це сприяє підвищенню продуктивності роботи та зменшує час на виконання завдань.

Децентралізація також є важливою перевагою IBM Maximo. Система дозволяє користувачам мати доступ до даних з різних пристроїв, що забезпечує

гнучкість у роботі та підвищує ефективність управління ресурами. Це особливо важливо для великих підприємств з розподіленою інфраструктурою, де необхідно оперативно реагувати на зміни та координувати дії з різних пристроїв.

Підсумовуючи, IBM Maximo має багато переваг, проте він не універсальний, дороговартісний та складний у використанні та навчанні.

Наступним прикладом сервісу для менеджметну інфраструктури варто взяти Zoho Inventory. Це інструмент для обліку ресурсів та підтримки інфраструктури підприємства, який здобув популярність завдяки своїй простоті у використанні, широким можливостям інтеграції інших сервісів та адаптивності під потреби інфраструктури. Цей сервіс дозволяє підприємствам ефективно керувати своїми ресурсами, відстежувати продажі та замовлення, а також оптимізувати процеси постачання. Однак, незважаючи на переваги, Zoho Inventory також має свої недоліки:

- Неефективність використання для великих підприємств.
- Висока вартість при використанні розширених функцій.

Проблема неефективності використання для великих підприємств полягає в тому, що цей сервіс більше орієнтований на малі та середні підприємства. Для великих компаній з складними процесами управління запасами цей інструмент може не мати всіх необхідних функцій та можливостей для масштабованості. Це обмежує його використання у великих інфраструктурах, які потребують більш розширених можливостей для управління своїми ресурсами.

Висока вартість використання розширених функцій також є перешкодою для багатьох підприємств. Zoho Inventory пропонує кілька рівнів тарифних планів, і деякі з більш розширених функцій доступні лише у дорожчих планах. Це збільшить загальні витрати на використання інструменту, особливо для компаній, які потребують доступу до цих розширених можливостей.

Враховуючи проблеми, Zoho Inventory також має ряд переваг:

- Простота використання.

- Широкі можливості інтеграції з іншими платформами.
- Гнучкість та адаптивність.

Однією з головних переваг Zoho Inventory є його зручність у використанні. Інтерфейс користувача інтуїтивно зрозумілий, що дозволяє швидко навчитися використовувати основні можливості цього інструменту. Навчальні матеріали, такі як відеоуроки та документація, доступні в достатній кількості, що дозволяє користувачам легко отримати знання, необхідні для ефективної роботи з програмою.

Іншою вагомою перевагою є здатність Zoho Inventory інтегруватися з іншими платформами. Інструмент підтримує інтеграцію з популярними сервісами, такими як Shopify, Amazon. Це забезпечує безперебійний обмін даними між різними системами, що сприяє підвищенню ефективності роботи та оптимізації процесів.

Крім того, цей сервіс відзначається своєю гнучкістю та адаптивністю. Система дозволяє налаштовувати функціонал відповідно до конкретних потреб інфраструктури, що забезпечує її адаптацію до змінюваних умов і вимог. Це робить Zoho Inventory вдалим рішенням для підприємств, які потребують гнучкого і масштабованого інструменту для підтримки інфраструктури.

Таким чином, Zoho Inventory є ефективним інструментом для управління інфраструктурою, який відзначається високою продуктивністю та простотою використання. Незважаючи переваги, цей сервіс має кілька недоліків, які впливають на його універсальність та доступність

Висновки до першого розділу

У першому розділі було детально проаналізовано проблеми пов’язані з підтримкою інфраструктури підприємств, а також розглянуто існуючі сервіси для їх вирішення. Встановлено, що у сучасному світі промисловість та економіка розвиваються надзвичайно швидко, що сприяє появі нових підприємств, які впливають на добробут та якість життя населення. Особливо

важливою є роль підприємств в Україні, з огляду на їх внесок у підтримку економіки та обороноздатності країни.

Аналіз різних типів інфраструктури, таких як рекламна агенція, завод літаків та організація концертів, показав спільні проблеми, з якими стикаються підприємства: необхідність обліку ресурсів, централізованість даних та фізичний пошук ресурсів. Ці проблеми ускладнюють ефективне управління інфраструктурою, особливо при великих обсягах даних та багатокористувацькому доступі.

Було також проведено аналіз існуючих сервісів для підтримки інфраструктури, таких як Microsoft Excel, IBM Maximo та Zoho Inventory. Кожен з цих інструментів має свої переваги та недоліки. Microsoft Excel є простим у використанні та доступним, але має обмежену функціональність при великих обсягах даних та централізованість. IBM Maximo пропонує потужні інструменти для управління технічними активами, але є дорогим та складним у використанні. Zoho Inventory відзначається простотою використання, гнучкістю та широкими можливостями інтеграції, проте має обмежену функціональність для великих підприємств та високу вартість при використанні розширених функцій.

Таким чином, на основі проведеного аналізу можна зробити висновок, що в сучасному світі існує запит на новий сервіс для ефективної підтримки інфраструктури підприємств, який зможе забезпечити гнучке управління ресурсами, знижувати ризики та підвищувати ефективність роботи.

РОЗДІЛ 2
ІНСТРУМЕНТИ ТА ТЕХНОЛОГІЇ ДЛЯ РОЗРОБКИ СЕРВІСУ ДЛЯ МЕНЕДЖМЕНТУ ІНФРАСТРУКТУРИ

В даному розділі на основі визначених для вирішення проблем буде проведений аналіз технічного завдання для сервісу, який потрібно розробити. Також буде розглянуто технологію, яка використана для вирішення розробленого технічного завдання.

2.1 Визначення технічного завдання

В попередньому розділі було розглянуто кілька прикладів інфраструктури, з підтримкою якої виникають логістичні та технічні проблеми. Так як оптимального рішення цих проблем немає, варто створити нове програмне рішення для вирішення визначених проблем.

Серед визначених проблем варто виділити основні:

- Централізованість
- Зручний облік та поповнення ресурсів
- Фізичний пошук ресурсів

Централізованість полягає у наявності лише одного пристрою, з якого можна вести облік ресурсів та аналізувати його. Ця проблема загострюється при великій кількості ресурсів у підприємства та великій кількості користувачів, яким цей облік потрібно вести[6]. Вирішенням цієї проблеми буде наявність у створюваному сервісі клієнт-серверної архітектури[14,15]. Вона дозволить вести облік одразу з кількох пристроїв в реальному часі та дозволить найбільш оптимально організувати облік наявних ресурсів. Крім цього, дані варто зберігати на сервері в одному екземплярі, для найбільшої точності обліку[16, 17].

Зручний облік та поповнення ресурісів є одним з основних факторів в якості програмного забезпечення. Завдяки комфорту у використанні сервісу, користувачі зможуть швидко та ефективно його використовувати. Варто

виділити основні фактори зручності, це правильна візуалізація даних та зрозумілий набір вбудованих інструментів для взаємодії з цими даними. Правильної візуалізації даних можна досягнути завдяки приховуванні другорядної інформації про ресурс. Набір вбудованих інструментів варто створювати не перевантажуючи його непортібними функціями та зробити його універсальним[18].

Проблема фізичного пошуку ресурсів полягає у необхідності користувачам вручну шукати потрібний ресурс. Ця проблема, як і потреба централізованості, загостюється при великій кількості ресурсів та користувачів. Найбільш вдалим рішенням для цієї проблеми можна вважати фіксацію фізичного розташування кожного ресурсу, яку можна впровадити за потреби. Варто брати до уваги що ця проблема хоча і виникає в більшості випадків підтримки інфраструктури, є окремі випадки коли проблема фізичного пошуку ресурсів не виникає. Так як одним з основних пріоритетів розробки є універсальність програмного рішення, то варто надати користувачу вибір щодо вирішення цієї проблеми.

Отже, варто підсумувати(табл. 2.1) список проблем, їхній вплив на роботу інфраструктури та спосіб вирішення, для подальшої структуризації технічних завдань.

Таблиця 2.1 – Список проблем підтримки інфраструктури, їх вплив та вирішення.

Проблема	Вплив	Вирішення
1	2	3
Централізація	Неможливість вести облік та впливати на інфраструктуру з різних пристроїв	Впровадження клієнт-серверної архітектури

Продовження таблиці 2.1

1	2	3
Зручність обліку	Збільшення часу, необхідного на виконання обліку	Зважене налаштування інтерфейсу
Необхідність фізичного пошуку ресурсів	Значні витрати часу на роботу з ресурсами	Впровадження характеристик кожного ресурсу

Враховуючи проблематику вказану в табл. 2.1 варто зупинитися на основних принципах побудови сервісу[7]. Ключовий принцип – клієнт-серверна архітектура застосунку[19, 20]. Таке рішення вирішить проблему централізованості системи, що збільшить максимальне можливе навантаження застосунку та зручність роботи з ресурсами. Впровадження такої архітектури також збільшить потенціал для розвитку програми в майбутньому[21]. При такому підході до архітектури варто визначити ролі для серверу і користувачів. Сервер – зберігає інформацію про ресурси, обробляє запити про їх зміни та відслідковує їх[22]. Також при потребі необхідно щоб була можливість вносити зміни в перелік з серверу та блокувати запити зі сторони клієнтів для випадків порушення безпеки. Клієнт – програма, яка під’єднується до серверу з будь-якого пристрою, має можливість вести облік ресурсів та вносити в нього зміни. Підключення зі сторони клієнта здійснюється по локальні мережі за допомогою протоколу TCP/IP [23]. Для підключення з цією технологією користувачу потрібно знати лише IP-Адресу та порт для підключення.

Клієнт і сервер повинні мати можливість обмінюватися пакетами даних. Клієнт приєднується до серверу і при потребі надсилає пакети даних, які зберігають детальну інформацію про запит. Сервер обробляє кожен запит та надає клієнту відповідь у вигляді пакету даних, який зберігає інформацію про

відповідь. Після цього пакет з відповіддю розшифровується програмою-клієнтом та візуалізується.

Облік ресурсів полягає у формуванні ресурсу як набору певних характеристик. В більшості ресурсів набір характеристик буде збігатися, що дозволить вести їх сортування та підбір. Список характеристик кожного з ресурсів може бути змінений в реальному часі будь-яким користувачем, якщо це будуть дозволяти параметри безпеки серверу.

Враховуючи усе вищесказане, можна виділити основні пріоритети розробки, які варто брати до уваги під час створення архітектури[24] додатку та під час прийняття рішень щодо розробки, цими пріоритетами є:

- Децентралізованість
- Універсальність
- Зручність роботи з даними
- Швидкість роботи з даними

2.2 Обґрунтування вибору технологій і мови програмування для розробки додатку

Для виконання поставлених завдань варто виважено підходити до вибору мови програмування та набору технологій, які будуть використані в створеному сервісі, адже цей вибір повпливає на швидкість реалізації сервісу, ефективність використання ресурсів пристрою та можливості подальшого розвитку проекту. Враховуючи знайдені недоліки і переаги існуючих сервісів та пріоритети розробки, для цієї роботи було обрано такий набір технологій:

- C#
- .NET
- JSON
- TCP/IP

Завдяки такому набору технологій сервіс матиме широкий спектр інструментів для реалізації поставлених завдань та високу швидкість роботи. Так як жодна

з цих технологій не є інструментом для реалізації конкретного завдання, то розроблений сервіс матиме потенціал для розвитку та збільшення кількості виконуваних завдань, а його інтергація з додатковими технологіями не буде потребувати великих часових та фінансових витрат.

C#[25] - Одна з найпопулярніших мов програмування на момент дослідження[26]. Основними її особливостями є орієнтованість на Об’єктно-орієнтоване програмування(Object-oriented programming, OOP)[8], підтримка поліморфізму[27], строга статична типізація та відсутність множинного наслідування. Ця мова з’явилася 2001 року, розроблена Андерсом Гейлсбергом, Скотом Вілтамутом та Пітером Гольде під егідою Microsoft Research (належить Microsoft).

C# є дуже близьким родичем мови програмування Java[28]. Java була створена компанією Sun Microsystems для вирішення завдань розподілених обчислень, викликаних розвитком інтернету. Вона базувалася на популярній мові C++, але виключила потенційно небезпечні елементи, такі як неконтрольовані вказівники. Для розподілених обчислень була розроблена концепція віртуальної машини та машинно-незалежного байт-коду, який служить посередником між вихідним кодом програм і апаратними інструкціями комп'ютерів та інших пристроїв.

Java стала дуже популярною, і Microsoft також отримала ліцензію на її використання. Однак Sun звинуватила Microsoft у створенні клону Java, сумісного лише з платформою Windows, що суперечило концепції машинно-незалежного середовища і порушувало ліцензійну угоду. Після судового процесу Microsoft була зобов'язана припинити некоректне використання Java.

У відповідь Microsoft вирішила створити власну мову програмування, аналогічну Java, яку вона повністю контролюватиме. Так з'явилася мова C#, яка успадкувала концепції віртуальної машини (.NET), байт-коду (MSIL) та підвищеної безпеки коду програм, а також врахувала досвід використання Java.

Нововведенням C# стала полегшена взаємодія з кодом, написаним на інших мовах, що є важливим при створенні великих проектів. Якщо програми

на різних мовах виконуються на платформі .NET, .NET забезпечує їх сумісність, зокрема типів даних.

Сьогодні C# є основною мовою програмування корпорації Microsoft, оскільки вона максимально використовує нові можливості .NET. Інші мови також підтримуються, але вважаються менш ефективними для роботи з .NET через певні обмеження.

Мова C# підійде для розробки сервісу, адже під час розробки буде створено складну об’єктно орієнтовану архітектуру [29, 30], в якій будуть виконуватися складні взаємодії між об’єктами та обробка даних. Швидкість виконання інструкцій цією мовою дозволить створити сервіс, який буде ефективно розподіляти ресурси комп’ютера та швидко виконувати запити та розрахунки. Створення такої програми також займе менше часу та ресурсів, порівняно з іншими мовами, адже мова C# є коспромісом між швидкістю виконання та складністю написання коду.

.NET [31] - це програмна платформа, розроблена компанією Microsoft, яка надає середовище для створення і виконання широкого спектра додатків, включаючи комп’ютерні, веб-додатки, мобільні додатки і хмарні сервіси[32]. Ця платформа складається з багатьох компонентів, які формують систему для ефективної розробки програмних рішень.

Основним компонентом .NET є Common Language Runtime (CLR), середовище виконання, яке керує виконанням коду, включаючи збір сміття, обробку виключень і управління пам'яттю та безпекою. CLR дозволяє програмам, написаним на різних мовах програмування, працювати разом.

Іншим компонентом є .NET Framework Class Library (FCL), набір багаторазово використовуваних класів, інтерфейсів і типів значень, які забезпечують широкий набір функцій, таких як робота з файлами, введення/виведення, доступ до баз даних, маніпуляція XML і робота з графічним інтерфейсом користувача. FCL є спільним для всіх мов програмування, що працюють на .NET.

.NET підтримує декілька мов програмування, включаючи C#, VB.NET, F# та багато інших. Можливість використання різних мов дозволяє виконати вибір мови, використання якої буде найефективнішим для конкретного завдання.

.NET Core, запущений як крос-платформенна версія .NET, підтримує Windows, macOS і Linux. У 2020 році Microsoft об'єднала .NET Framework і .NET Core в єдину платформу під назвою .NET 5, яка продовжує розвиватися у версіях .NET 6, .NET 7 і так далі.

Основні характеристики .NET включають крос-платформенність, яка дозволяє створювати додатки, які працюють на різних операційних системах, високу продуктивність, вбудовані механізми безпеки, включаючи управління користувачами, авторизацію та аутентифікацію, криптографію, та безпечне управління пам'яттю. Крім того, .NET забезпечує розширюваність та модульність завдяки NuGet, системі управління пакетами для .NET. Дослідивши список користувачів .NET[33], можна виділити кілька основних напрямків програмних продуктів, побудованих на цій платформі:

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

Таке широке охоплення сфер, які використовують цю платформу пояснюється широким набором якіснинх інструментів, які надає .NET.

JSON(JavaScript Object Notation)[34] - Формат, за допомогою якого можливо реалізувати комп’ютерні дані у тексті та текст у вигляді комп’ютерних даних. За допомогою JSON можливо переформатувати програмний об’єкт у текст і навпаки - текст у програмний об’єкт. JSON має простий синтаксис, що полегшує роботу з даними. Легкість аналізу та генерації JSON-даних робить його універсальним для передачі даних між серверами і клієнтами. Хоча JSON виник з JavaScript, він може бути

використаний з багатьма іншими мовами програмування, такими як Python, Ruby, PHP, Java та обраною для дослідження мовою - C#. Більшість мов мають вбудовані або сторонні бібліотеки для роботи з JSON.

JSON є основним форматом обміну даними між веб-серверами і клієнтами[35]. Наприклад, при розробці веб-додатків на основі AJAX, JSON часто використовується для передачі даних між сервером і браузером. Багато веб-сервісів і API використовують JSON для передачі даних, що дозволяє легко інтегрувати різні системи і забезпечує зручний формат для передачі складних структур даних. JSON також використовується для зберігання конфігураційних файлів і інших типів даних. Завдяки своїй простоті і зрозумілості, JSON є популярним вибором для зберігання налаштувань і даних додатків.

Формат був розроблений Дугласом Крокфордом у 2002 році та швидко набув популярності за рахунок своєї зручності та ефективності. Хоча цей формат в більшості випадків використовуєьться для реалізації веб-сторінок, його використання для розробки програмного сервісу для підтримки інфраструктури буде вдалим рішенням.

Для взаємодії цього технічного рішення з C# та .NET буде використано бібліотеку Newtonsoft.Json[36], яка дозволить ефективно реалізувати програмне рішення для підтримки інфраструктури.

Важливою складовою створюваного сервісу є формат мережевого з’єднання. Мережеве з’єднання буде реалізовано за протоколом TCP/IP (Transmission Control Protocol/Internet Protocol) - це набір комунікаційних протоколів, які забезпечують обмін даними між пристроями в мережі. Цей набір протоколів є основоположним для функціонування мережі, забезпечуючи правила та стандарти для передачі даних, що дозволяють різним пристроям взаємодіяти незалежно від їх апаратного або програмного забезпечення.

TCP/IP складається з декількох шарів. Першим є Internet Protocol (IP), який забезпечує адресацію пакетів даних між пристроями. Кожен пристрій у мережі має унікальну IP-адресу. Існують дві основні версії IP - IPv4, яка

використовує 32-бітні адреси, що дозволяє мати близько 4,3 мільярда унікальних адрес, і IPv6, яка використовує 128-бітні адреси, забезпечуючи практично необмежену кількість можливих адрес.

Інший шар - Transmission Control Protocol (TCP) відповідає за надійність передачі даних. TCP розбиває дані на менші частини, звані пакетами, і гарантує, що всі пакети будуть доставлені цілісними та у правильному порядку. Він здійснює контроль за відновленням втрачених пакетів і забезпечує механізми контролю потоку даних, що дозволяє уникнути перевантаження мережі. Для розробки сервісу правильним рішенням буде використання саме протоколу TCP, так як мережеве з’єднання буде використано для передачі точних даних. Існує також інший протокол передачі даних - User Datagram Protocol (UDP), він не забезпечує відстеження та відновлення пакетів даних, тому його використання в розроблюваному сервісі буде недоречним, адже в програмному рішенні критично важлива цілісність інформації.

Функціонування TCP/IP базується на передачі даних у вигляді пакетів. Під час відправки даних по мережі, з них формуються пакети, кожен з яких містить заголовок з інформацією про відправника, отримувача та порядок збирання пакетів назад у вихідні дані. IP забезпечує маршрутизацію цих пакетів до кінцевого пункту призначення. Маршрутизатори на кожному вузлі мережі визначають найкращий шлях для пакета на основі IP-адреси. TCP створює з'єднання між двома пристроями, відправляє пакети, підтверджує їх отримання та повторно відправляє втрачені пакети, забезпечуючи цілісність та правильний порядок даних. Завдяки своїй гнучкості та надійності, TCP/IP став стандартом для мережевих комунікацій, що лежить в основі Інтернету та більшості локальних мереж.

Висновок до другого розділу

У цьому розділі проведено аналіз технічного завдання для розробки сервісу, який має вирішувати основні проблеми, пов'язані з менеджментом інфраструктури. Визначено, що серед ключових проблем є централізованість,

необхідність зручності обліку ресурсів та проблема їх фізичного пошуку. Для кожної з цих проблем визначено технічні рішення, такі як впровадження клієнт-серверної архітектури, оптимізація інтерфейсу користувача та фіксація фізичного розташування ресурсів у характеристиках.

Обґрунтовано вибір технологій і мови програмування для розробки сервісу. Обрано такі технології, як C#, .NET, JSON та TCP/IP, що забезпечують ефективну реалізацію технічного завдання. Використання C# та .NET дозволить створити гнучкий сервіс з високою продуктивністю і можливістю масштабування. JSON забезпечить зручний обмін даними між сервером і клієнтом, а протокол TCP/IP гарантує надійне та безпечне мережеве з'єднання.

Враховуючи всі аспекти, визначені в цьому розділі, можна зробити висновок, що вибрані інструменти та технології забезпечать надійний набір технологій для реалізації сервісу для менеджменту інфраструктури. Обрані рішення дозволять створити систему, яка дозволить користувачам ефективно керувати ресурсами, забезпечувати зручний облік та підтримку інфраструктури, а також легко адаптуватися до потреб підприємств.

РОЗДІЛ 3
ПРОГРАМНА РЕАЛІЗАЦІЯ СЕРВІСУ

В даному розділі буде розглянуто технічні рішення для побудови сервісу, який вирішуватиме проблеми підтримки інфраструктури, визначені у другому розділі.

3.1 Серверна частина

Основою програмного рішення для підтримки роботи інфраструктури є серверна частина. Сервер представляє з себе програму, яка може бути виконана на будь-якому пристрої з списку тих, які підтримує платформа .NET. Основною вимогою до запуску серверу буде підключення пристрою до локальної мережі для можливості підключення клієнтів.

Сервер має широкий функціонал та є вагомою частиною усього сервісу. Функціонал серверу передбачає:

- Обробка під’єднання нових клієнтів
- Обробка запитів клієнтів
- Взаємодія з базою даних

Враховуючи передбачений функціонал, на сервері буде присутнім ряд класів. Відштовхуючись від приницпів Об’єктно-орієнтованого програмування та сучасних стандартів та підходів до архітектури програмного забезпечення[37,38], список класів буде виглядати так:

- Program, основний клас, виконується при запуску серверу та зберігає інформацію, необхідну для роботи усієї системи.
- MenuWindows, клас який відповідає за візуалізацію даних та інтерфейс користувача.
- Data, клас який зберігає дані серверу, інструменти для роботи з ними та алгоритми для обробки запитів.
- Server, клас який зберігає інструменти для роботи серверу в мережі.

- Operation, клас який зберігає шаблон, за яким програма може зафіксувати одну операцію на сервері.
- Request, шаблон за яким зі сторони клієнта формуються запити. На сервері цей клас потрібен для правильного розшифрування та структуризації отриманого від клієнта запиту.
- Answer, шаблон за яким формується відповідь сервера на запити клієнтів

Обробку під'єднання нових клієнтів можна характеризувати кількома етапами, кожен з яких виконується у класі Server. Перший етап проводиться лише один раз - це визначення мережевої адреси серверу та надання її клієнтам. Завдяки цьому клієнти будуть мати можливість підєднатися до серверу напряму. Адреса серверу визначається автоматично, користувач може дізнатися її в будь-який момент роботи програми. Адреса складається з IP адреси серверу в локальні мережі та порту з'єднання, який виділяється на роботу програми. Порт визначаєтсья автоматично та обирається серед вільних для створення з'єднання. Визначення вільного порту виконується за допомогою методу GetPort класу Server, який повертає вільний порт, структуру цього методу зображено на рис. 3.1.

```
private int GetPort()
{
   TcpListener listener = new TcpListener(IPAddress.Loopback, 0);
   listener.Start();
   port = ((IPEndPoint)listener.LocalEndpoint).Port;
   listener.Stop();
   Console.WriteLine($"Port {port} found!");
   return port;
     }
```

Рисунок 3.1 – Структура методу GetPort

Визначення IP адреси виконується за допомогою методу GetIPAdress, який знаходиться у класі Server(рис 3.2).

```
        private IPAddress GetIPAddress()
    {
      var host = Dns.GetHostEntry(Dns.GetHostName());
      foreach (var ips in host.AddressList)
      {
        if (ips.AddressFamily == AddressFamily.InterNetwork)
        {

          Console.WriteLine("IP found!");
          return ips;
        }
      }
      return IPAddress.None;
    }
```

Рисунок 3.2- Структура методу GetAdress

Другий етап – обробка запиту на під'єднання, виконується для кожного клієнта. Для цього створено окремий сокет, який прослуховується на запити про підключення клієнтів. Так як сервер передбачає паралельне виконання широкого функціоналу, то прослуховування нових з'єднань виконується на окремому потоці[39,40]. Створення нового потоку для очікування з'єднань виконується під час створення серверу в методі Run() класу Server. Окрім цього, у випадку, коли пристрій на якому було запущено сервер не підєднаний до локальної мережі, користувачу виводиться на екран помилка створення з'єднання.

Третій етап – додавання нового користувача. При правильному під'єднанні, сервер налаштовує з'єднання з користувачем. Зі сторони серверу цей процес полягає у створенні нового потоку для обробки запитів користувачів та додавання інформації про з'єднання кожного окремого користувача у список з'єднань у форматі Dictionary<TcpClient, NetworkStream>. Необхідність використання багатопотоковості пояснюється розрахунком на значну кількість користувачів, які одночасно взаємодіють з сервером та виконують запити.

Другий і третій етапи виконуються у методі ListeningNewClients класу Server(рис. 3.3).

```
        public void ListeningNewClients()
{


    while (true)
    {
        TcpClient client = tcpListener.AcceptTcpClient();
        NetworkStream stream = client.GetStream();
        Program.clients.Add(client, stream);
        Program.clientThreads.Add(new Thread(delegate () {
        Listening(stream); }));
    }
        }
```

Рисунок 3.3 – Структура методу ListeningNewClients

Цей метод запускає циклічне нескінченне прослуховування на запити про під'єднання до адреси серверу.

Після під'єднання клієнт отримує доступ до надсилання запитів та виділений потік на їх обробку незалежно від запитів інших клієнтів. Так як база даних стандартизована та зберігається в єдиному місці, шанс колізії запитів, при якій кілька з запитів будуть одночасно змінювати дані про один і той самий елемент, мінімальний.

На цьому етапі можна виділити кілька основних потоків серверу. Кожен з них відповідає за свій функціонал, який повинен виконуватися асинхронно від функціоналу інших потоків(рис 3.4). Основний потік відповідає за взаємодію оператора серверу з налашутваннями серверу та ресурсами. Основному потоку належить весь функціонал, який може бути використаний для роботи з сервером на пристрої, на якому програма-сервер виконується.

Другий потік відповідає за під'єднання нових клієнтів. Він створюється і запускається з основного потоку під час налашутвання серверу. Цей потік виконує важливу для роботи серверу функцію, яка полягає у підєднанні нових клієнтів, налаштуванні з'єднання з новим клієнтом та форматування з'єднання

для правильної роботи з ним. Також з нього створюються потоки обробки запитів.

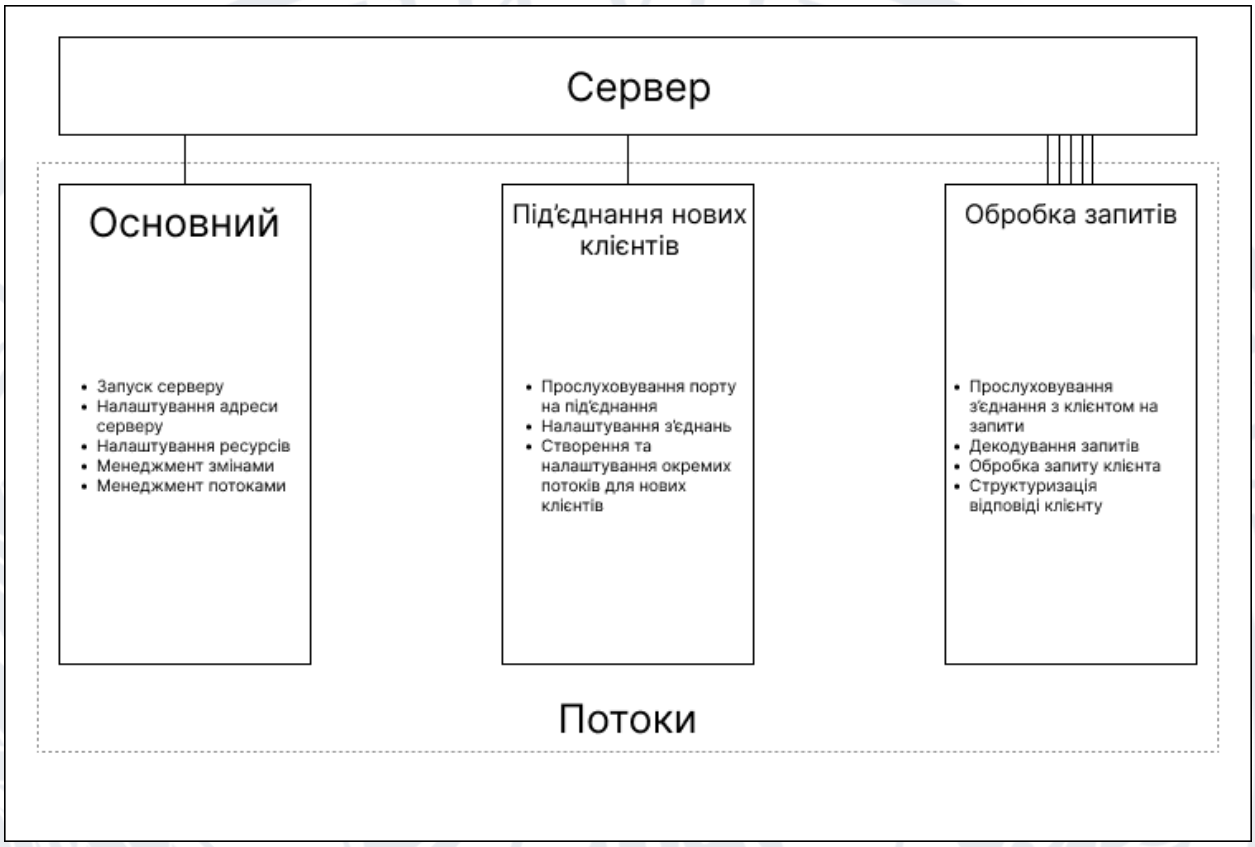

Рисунок 3.4 – Структура потоків серверу

Потоки обробки запитів відрізняються від двох попередніх. Основна відмінність – їх кількість нефіксована та обмежена лише обчислювальними можливостями серверу. Завданням кожного з таких потоків є очікування, обробка запитів користувачів та повернення відповіді клієнту. Кожен такий потік створюється окремо для кожного з клієнтів. Це дозволяє виконувати та обробляти запити клієнтів паралельно. Завдяки такому технічному рішенню при великій кількості запитів клієнтам не потрібно очікувати на обробку запитів інших користувачів в черзі.

Завдяки використанню багатопотоковості та сучасних технологій паралельних обчислень програмне рішення може виконувати широкий функціонал з незначними витратами часу та комфортом для користувачів.

Структура даних

Сервер володіє широким функціоналом в напрямку обробки та збереження даних. Основними функціями можна виділити:

- Збереження даних на пристрої
- Завантаження даних з пристрою
- Обробка даних

Під час запуску серверу користувачу пропонується вибір – завантажити дані чи створити нові(рис. 3.5).

Рисунок 3.5 – Можливі варіанти роботи з файлами даних

При створенні нових даних користувач вводить ідентифікатор даних. Після корректного введення та перевірки на наявнсіть файлу з введеним ідентифікатором, на сервері створюється новий об’єкт класу Data, на пристрої створюється файл збереження у форматі JSON. Ім’я файлу збереження формується згідно ідентифікатору(Рис 3.6)

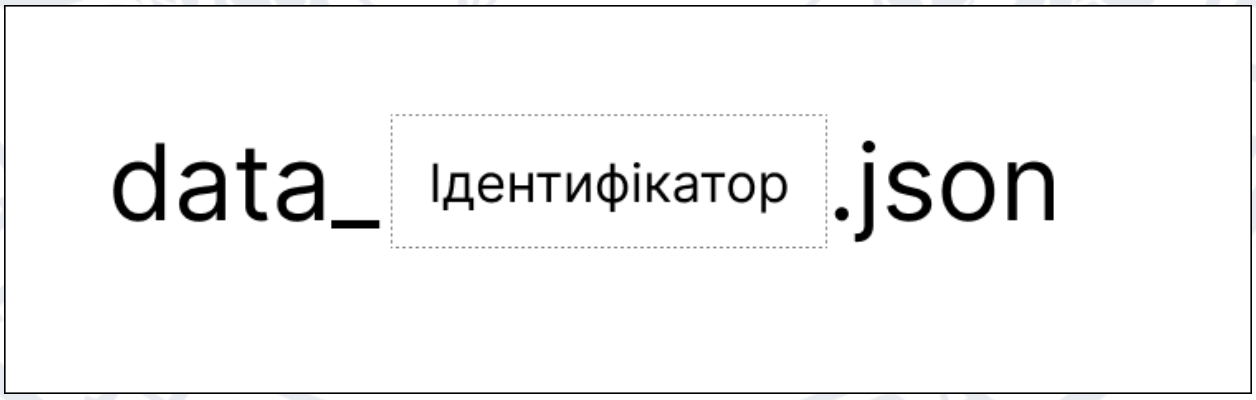

Рисунок 3.6 – Форматування назви файлу збереження

Таким чином один сервер може зберігати кілька наборів ресурсів та підтримувати одночасно кілька різних інфраструктур.

При завантаенні даних сервер запитує користувача ідентифікатор даних, якщо ідентифікатор введено правильно, сервер імпортує дані з файлу збережнння з відповідним ідентфікатором.

На сервері дані реалізовані у вигляді об’єкту data типу даних Data(рис. 3.7), який зберігається у класі Program. Дані серверу складаються з ідентифікатору, списку ресурсів, списку операцій та методів роботи з ними. Така архітектура даних дозволяє ефективно використовувати весь технічний потенціал пристрою, на якому знаходиться сервер та в перспективі легко доповнювати набір інструментів для роботи з даними.

```
public class Data
{
    public List<Operation> operationPool = new List<Operation>();
    public List<Resource> resources = new List<Resource>();
    public string title = "DefaultData";
```

Рисунок 3.7– Структура класу Data

Набір методів для роботи з даними складається з методів:

- SaveObjectsToFile – зберігає інформацію в файл, універсальний метод
- Save – швидке збереження, зберігає лише файл з даними
- LoadObjectsFromFile – завантажує дані з пристрою на сервер
- ProcessRequest – обробляє запит клієнтів
- AddResource – Додає ресурси
- RemoveResource – Прибирає ресурси
- ChangeResource – Змінює ресурси

SaveObjectsToFile приймає тип даних об’єкту, який потрібно зберегти на пристрої та назву файлу збереження. Цей універсальний метод надає потенціал для подального розвитку проекту та відповідає принципу поліморфізму[41]. Метод Save – його аналог, створений для швидкого збереження файлу даних. Він використовує метод SaveObjectsToFile, але автоматично заповнює всі необхідні дані збереження, відштовхуючись від ідентифікатора, який зберігається в об’єкті data. LoadObjectsFromFile приймає тип даних, в який

потрібно форматувати файл збереження та назву файлу збереження. Повертає об’єкт того ж типу даних, який і був прийнятий при виклику цього методу.

ProcessRequest – Метод, який обробляє запити користувачів. Хоча він і знаходиться в класі Data, його виконання відбувається на потоках виділених для користувачів. Цей метод ідентифікує тип запиту, обробляє його, робить зміни в базі даних при коректному запиті, додає операцію до списку, формує відповідь та повертає її на місце виклику цього методу. Обробка запиту здійснюється викликом одного з методів обробки даних. Визначення потрібного методу відбувається за рахунок аналізу запиту. Основні можливі варіанти роботи з даними:

- Перегляд всіх ресурсів
- Додавання нових ресурсів
- Видалення визначених ресурсів
- Зміна певних характеристик конкретного ресурсу

Запит на перегляд усіх ресурсів не має свого методу, адже алгоритм формування відповіді на нього не передбачає обчислень, лише формування відповіді, яка повертає набір усіх ресурсів з бази даних.

Процес додавання нових ресурсів полягає в додаванні до бази даних набору ресурсів з запиту від користувача. Цей метод передбачає попередній пошук хоча б одного повністю ідентичного запиту ресурсу в базі даних. При наявності такого запит не виконується належним чином та повертає помилку.

Видалення ресурсів відбувається лише при повній наявності в базі даних ресурсів з запиту. Під час виконання запиту сервер перевіряє кожен ресурс з запиту на наявність у базі даних та відмічає його відсутність. Якщо хоча б один ресурс відсутній, то відповіддю на запит буде помилка і жоден з запропонованих до видалення ресурсів не буде видаленим. В іншому випадку з бази даних буде видалено кожен з ресурсів зі списку, який подавався в запиті. Так як форматування даних передбачає однакові назви в ресурсів, то усі перевірки на наявність або відсутність ресусру відбуваються на основі ідентифікаторів, які в кожного ресурсу унікальні.

Кожен з запитів представлений у вигляді об’єкту класу Request(рис 3.8), який складається з типу запиту та списку ресурсів. Відповідно інформації про тип запиту, сервер ідентифікує вид роботи, який потрібно провести зі списком ресурсів. Цей формат запиту передбачає правильне формування запиту зі сторони клієнта, в іншому випадку запит буде виконано некоректно.

```
public class Request
{
    public enum requestType
    {
        AddResource,
        RemoveResource,
        getAllResources,
        changeResource
    }
    public requestType type;
    public List<Resource> resources = new List<Resource>();
}
```

Рисунок 3.8 – структура класу Request

Кожен ресурс представлений у вигляді об’єкту класу Resource(рис 3.9), який складаєтсья з словника характеристик, ідентифікатору, назви та інструментів для роботи з ними. Словник характеристик складається з ключа та значення які є назвою характеристики та її значенням. Це дозволяє використовувати весь потенціал програми та вносити будь-який формат значень у характеристики.

Ідентифікатор ресурсу є числовим значенням, яке складається з шести цифр. Ідентифікатор є унікальним для кожного ресурсу та надається під час його створення за допомогою конструктору. Під час створення ідентифікатору кожен ресурс з бази даних перевіряється на наявність випадкової цифри з 6 символів. Якщо однакові ідентифікатори знайдено, генерується новий та знову виконується перевірка.

Через особливості програми для найбільш ефективного використання серверу варто використовувати менш ніж 400000 ресурсів, адже після

перебільшення цього числа програмі потрібно більше часу для обробки запитів на створення нових ресурсів.

```
public class Resource
{
    public string name = "Unnamed";
    public int id = 0;
    public Dictionary<string,string> values = new Dictionary<string,string>();

    public Resource() ...
    public void AddValue(string name, string value)...
}
```

Рисунок 3.9 – структура класу Resource

Інтерфейс користувача

Інтерфейс користувача є критично важливою функцією для роботи серверу адже він дозволяє оператору серверу взаємодіяти з даними напряму та працювати з функціоналом сервісу. Інтерферс користувача реалізований у класі MenuWindows, який складається з ряду методів, кожен з яких реалізує візуалізацію інтерфейсу для конкретних дій. Основним методом який виконується одразу після запуску сервісу є метод Startup(рис 3.10), який надає оператору серверу вибір – створити нові дані чи завантажити існуючі. Також користувач має можливість переглянути інформацію про програму.

Рисунок 3.10- Інтерфейс користувача на початку роботи з сервером

Інтерфейс реалізований у вигляді списку можливих дій, між якими користувач може рухатись за допомогою клавіатури. Обрана дія підсвічується

жовтим. При натисканні на клавішу «Стрілка донизу» обраною стає дія, яка знаходиться від обраної попередньо дії. Якщо наступної дії не існує, обрана користувачем дія не змінюється. При натисканні клавіши «Стрілка доверху» виконується такий самий алгоритм. Для того щоб виконати обрану дію користувач має натиснути клавішу «Enter». В такому разі сервер визначить обрану дію та виконає її.

Цей алгоритм дій для списку можливих варіантів наявний в більшості існуючих програмних вікон. Для його реалізації необхідна змінна яка зберігає обрану користувачем дію в форматі int, змінна типу bool, яка відповідає за те, чи зробив користувач свій вибір. По стандарту значення цієї змінної false, при цьому значенні виконується нескінченний цикл, який спочатку очищує консоль, потім виводить на екран всі дані та очікує натискання клавіші. При натисканні стрілок змінна, яка зберігає обране вікно збільшує або зменшує своє значення попередньо перевіривши чи таке можливо. При натисканні клавіші «Enter» значення змінної, яка зберігає наявність вибору користувача змінюється на true та зупиняє цикл, після чого в залежності від ораного варіанту виконує певну дію.

Список можливих виборів зберігається у форматі масиву типу string, кожен елемент якого є заголовком можливого вибору корситувача. У методі Startup цей список має такий вигляд:

- Create Data
- Import Data
- Info

Кожен можливий вибір супроводжується виконанням методу з таблиці 3.1

Таблиця 3.1 – Можливі варіанти роботи з даними під час запуску серверу

Назва вибору	Функціонал методу	Назва методу, який виконається
Create Data	Створення та налашутвання нового файлу з даними серверу	CreateData
Import Data	Імпортування існуючого файлу з даними та його подальше використання в роботі серверу	ImportData
Info	Візуалізація інформації про сервер	InfoStartup

При виконанні методу InfoStartup на екран виводиться інформація про сервер та очікується натискання будь-якої клавіші для нового виконання методу Startup.

Метод CreateData виконує функцію створення нових даних для серверу. Для його правильної роботи необхіда змінна, яка буде зберігати ідентифікатор для створених даних. Під час запуску цього методу запускається нескінченний цикл, який спочатку виводить на екран повідомлення про потребу у введенні ідентифікатору даних(рис. 3.11).

Рисунок 3.11– Інтерфейс користувача під час введення нового ідентифікатору

Після введення і натискання клавіші «Enter» програма перевіряє наявність файлу даних на пристрої, на якому знаходиться сервіс. При наявності файлу даних з введеним ідентифікатором програма повідомляє про наявність такого файлу даних(рис. 3.12) та цикл виконується знову.

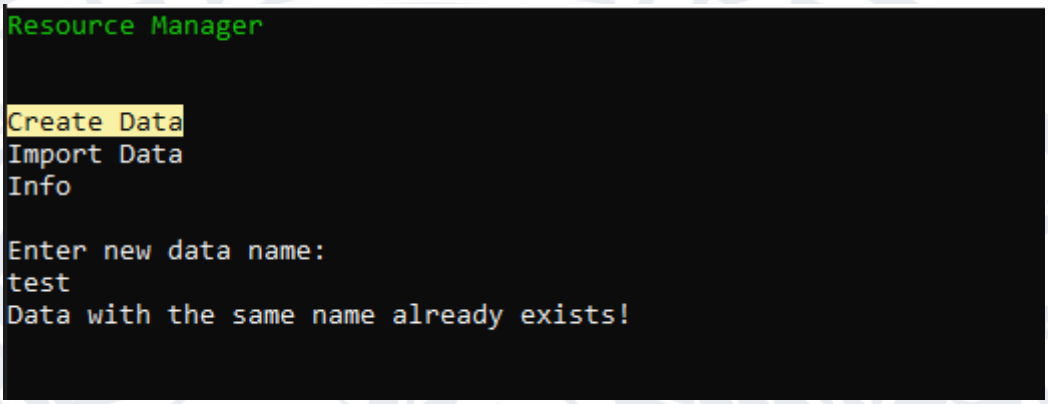

Рисунок – 3.12 Інтерфейс користувача під час створення даних з існуючим ідентифікатором

При введенні ідентифікатору, файлу з яким не існує на пристрої, такий файл створюється та налаштовується для правильної роботи, після чого користувачу повідомляється про успішне створення файлу даних. Після цього етапу викликається метод ServerCreation, який налаштовує мережеве з'єднання серверу та створює потік для прослуховування на під'єднання нових клієнтів. На цьому етапі сервер починає свою роботу. Після цього сервер повідомляє користувача про налаштування серверу, виводить на екран інформацію яка зандобиться для під'єднаня до серверу та очікує на натискання будь-якої клавіші для подальшої роботи з сервером(рис. 3.13). Пілся натискання клавіші виконуєтсья метод MainMenu, який виконує функцію головного меню серверу.

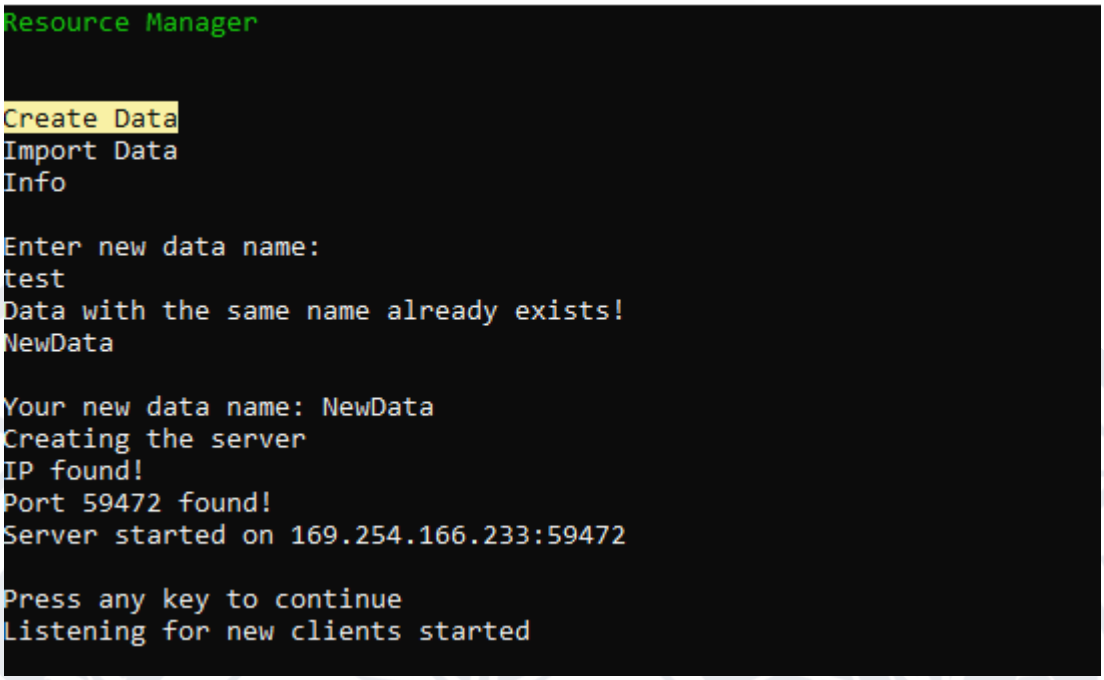

Рисунок 3.13 – Інтерфейс користувача під час створення нових даних

Метод ImportData створений для пошуку даних на пристрої серверу за допомогою ідентифікатору. Під час його виконання запускається нескінченний цикл, на екран виводиться запит ідентифікатору у користувача(рис 3.14). Після введення ідентифікатору та натискання клавіші «Enter» програма шукає файл з введеним ідентифікатором. При його відсутності користувач повідомляється про це, цикл переходить на нову ітерацію та знову очікує введення ідентифікатору даних. При введенні ідентифікатору існуючих даних, сервер імпортує дані з файлу з введеним ідентифікатором та працює з ними. Після виконання цих дій, як і у методі CreateData, виконується метод ServerCreation та MainMenu.

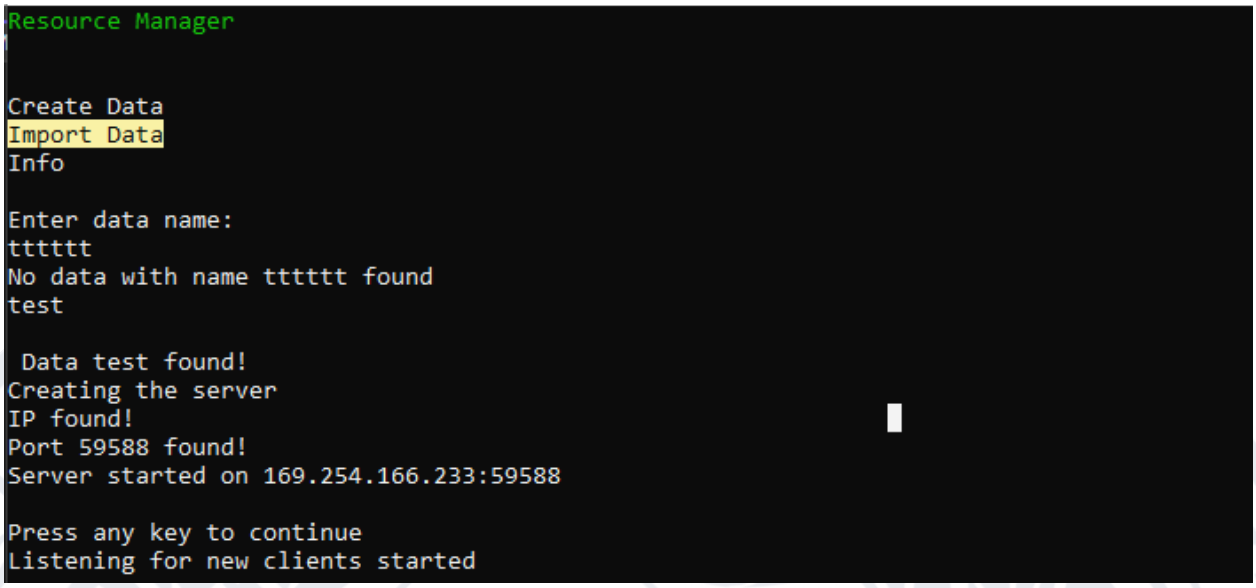

Рисунок 3.14 – Інтерфейс користувача під час імпортування даних

Метод MainMenu виконує функцію головного меню та дозволяє оператору серверу виконувати весь можливий функціонал серверу. Головне меню відкривається для користувача лише у випадку, коли під час завантаження даних та створення з'єднання не виникло помилок.

Виконання цього методу реалізоване за допомогою списку можливих дій користувача, які він може обирати за допомогою клавіатури. Цей список дій реалізує основний функціонал серверу та надає оператору серверу більше можливих варіантів дій ніж має клієнт. Список можливих варіантів та методів які виконуються при їх виборі знаходиться в таблиці 3.2

Метод InfoConnected Виконує такий самий функціонал, який виконує метод Info. Єдиною відмінністю є наявність інформації про мережеву адресу серверу для того щоб надати оператору можливість дізнаватися її в будь який момент для передачі клієнтам. По цій мережеві адресі серверу клієнти під'єднуютьтся до серверу.

Таблиця 3.2 – Доступні для вибору варіанти роботи з даними з головного меню серверу

Назва вибору	Функціонал методу	Назва методу, який виконається
Operations	Візуалізація списку виконаних з сервером операцій	ShowOperations
Clients	Візуалізація інформації про під'єднаних клієнтів	ShowClients
Resources	Взаємодія з списком ресурсів	ShowResources
Info	Інформація про сервер	InfoConnected

Метод ShowOperations дозволяє оператору серверу переглянути історію дій з даними та загальний список виконаних дій(рис 3.15). Завдяки цій функції сервер має більш обширні можливості для моніторингу та аналізу роботи сервісу. Цей метод виводить на екран список операцій з списку operationsPool обраного файлу даних та очікує від користувача натискання будь-якої клавіші для повернення в головне меню.

Рисунок 3.15 – Інтерфейс користувача під час перегляду виконаних операцій

Метод ShowClients візуалізує список під'єднаних до серверу клієнтів. Візуалізація реалізується виконанням циклу, який проходиться по кожному з'єднанню в словнику clients. Для кожного з'єднання визначається мережева

адреса клієнта та виводиться на екран(рис 3.16). При натисканні будь-якої клавіші програма повернеться до головного меню.

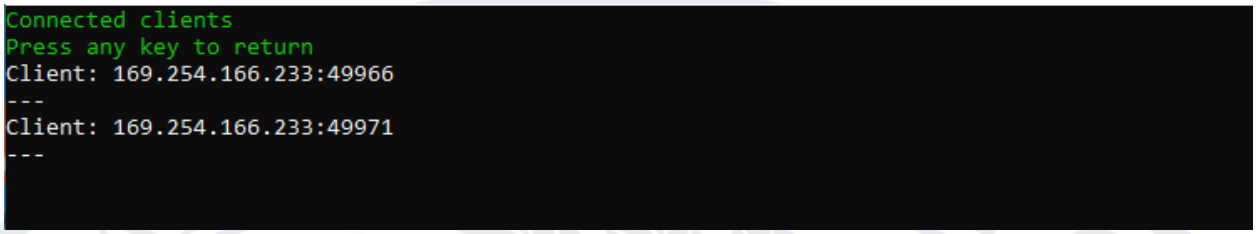

Рисунок 3.16– Інтерфейс користувача під час перегляду списку під’єднаних клієнтів

Метод ShowResources має найширший функціонал серед методів можливих для виконання з головного меню. Цей метод реалізує повний список інстурментів для роботи з списком ресурсів. Під час його виконання програма виводить на екран список можливих дій з обраним ресурсом та список ресурсів з під’єднаної бази даних(рис 3.17).

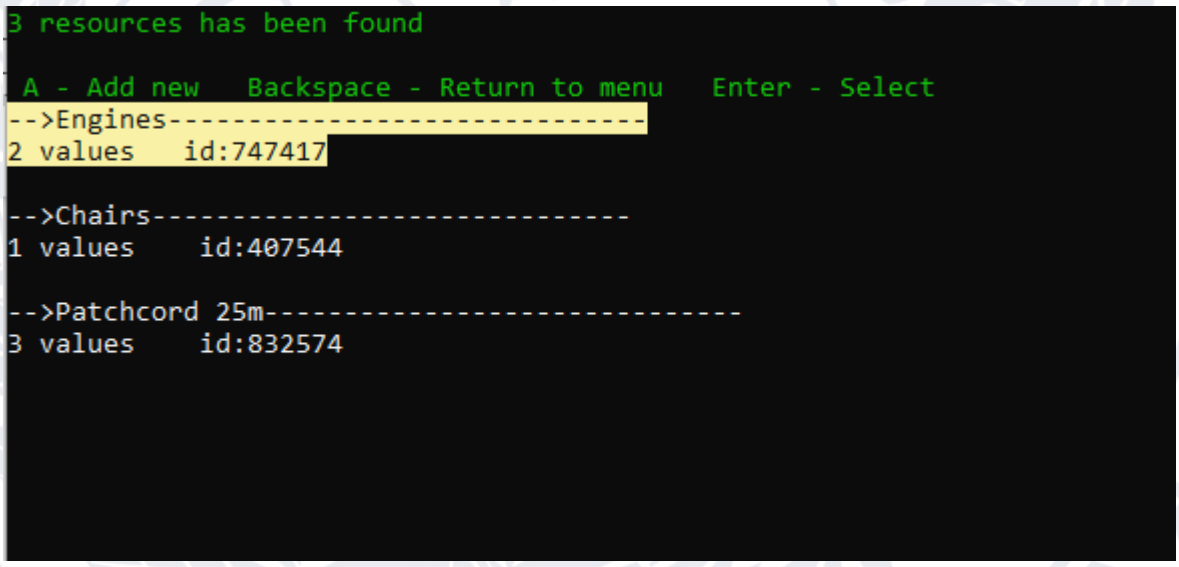

Рисунок 3.17 – Реалізація методу ShowResources

Список ресурсів виводиться циклом, який проходить по кожному елементу в списку resources об’єкту data. На екран виводиться назва ресурсу, кількість його характеристик та ідентифікатор. Для додаткового комфорту при роботі з ресурсами максимальна кількість ресурсів, яка візуалізується на екрані – 8. Для реалізації цієї функції потрібно 2 додаткових змінних – from і to. Вони зберігають порядковий номер ресурсу з якого починається візуалізація

та номер ресурсу до якого виконується візуалізація включно. Якщо кількість наявних в базі даних ресурсів менше восьми, візуалізуються всі ресурси.

В роботі цього методу реалізована змінна selection, яка зберігає номер обраного ресурсу. При натисканні клавіші «Enter» виконується метод ShowInfoResource, який виконує візуалізацію додаткової інформації про обраний ресурс. Після натискання клавіші «A» виконується метод ShowAddResource, який відкриває інтерфейс додавання нового ресурсу до бази даних. При натисканні клавіші «Backspace» користувач повертається до головного меню, адже в такому випадку виконується метод MainMenu.

Метод ShowInfoResource виводить на екран користувача інформацію про обраний в попередньому вікні ресурс. Ця інформація включає в себе назву ресурсу, список характеристик та значень цих характеристик(рис 3.18). Також користувачу повідомляється набір дій які він може виконати з ресурсом.

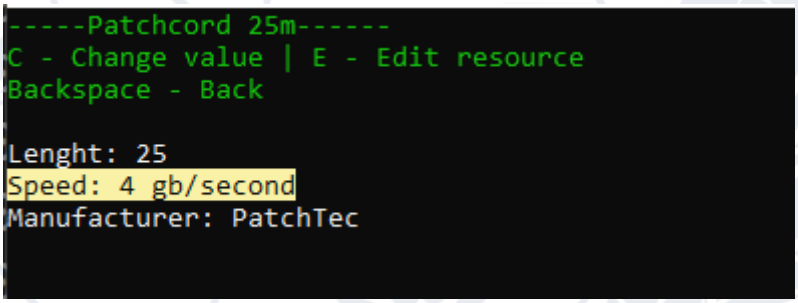

Рисунок 3.18 – Реалізація методу ShowInfoResource

У цьому вікні користувач має можливість змінити лише значення характеристик ресурсу. Така можливість може бути використаною при виборі характеристики, занчення якої потрібно змінити та натисканні клавіші «C». В такому випадку викликаєтсья метод ChangeValue для обраної характеристики. Цей метод приймає нове значення та заміняє попереднє у базі даних(рис 3.20).

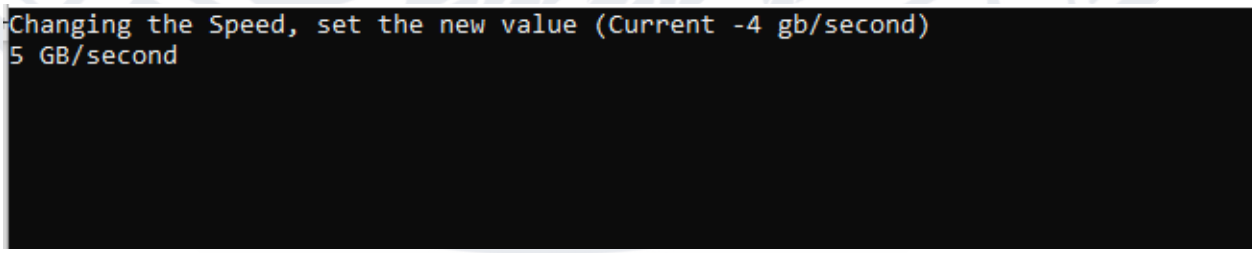

Рисунок 3.20 – Реалізація методу ChangeValue

При необхідності у зміні структури інформації про ресурс, користувачу потрібно натиснути клавішу «E». В такому випадку буде викликано метод ChangeResource.

Метод ChangeResource реалізує можливість зміни струкутри ресурсу користувачем. Цей метод виводить на екран всю наявну інформацію про ресурс, не враховуючи ідентифікатор, та список можливих дій(рис. 3.21). Користувач має можливість обрати одну з характеристик та змінити назву цієї ж характеристики(клавіша «C») або видалити її(клавіша «R»). Також користувач має можливість додати нову характеристику до вже існуючого ресурсу(клавіша «A»).

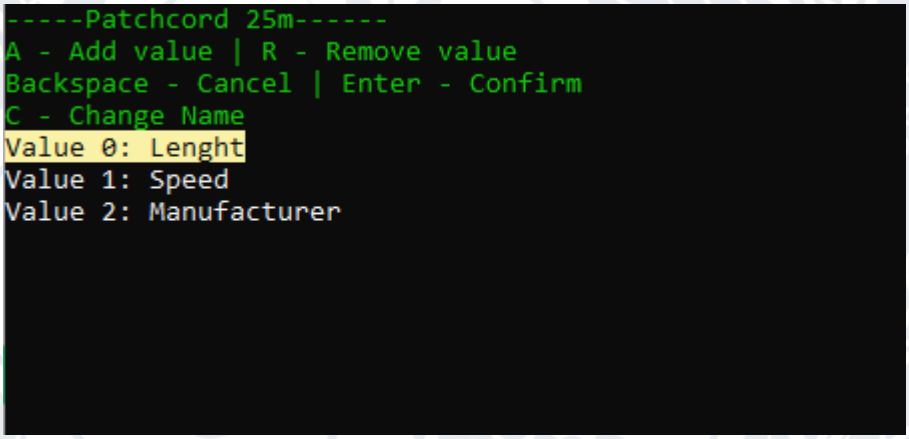

Рисунок 3.21 – Зміна структури ресурсу

Можливість зміни та додавання нової характеристик супроводжуєтсья одним методом – ChangeName(рис 3.22), різниця лише в реалізації цього методу.

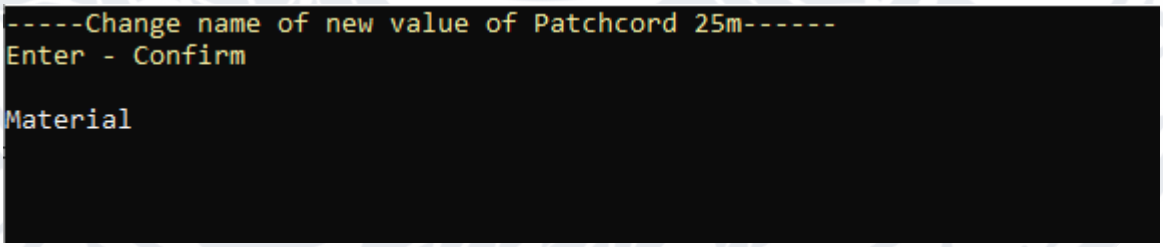

Рисунок 3.22 – Додавання нової характеристики до ресурсу

Під час додавання нової характеристики вона створюється автоматично і змінюється за допомогою цього ж методу. У випадку з зміною характеристики цей метод виконується до вже існуючої характеристики. Також однією з відмінностей в реалізації цього методу є заголовок вікна, яке бачить користувач при створенні нової або зміні існуючої характеристики, крім назви

воно також показує користувачу дію, яку він виконує. Цей приклад наочно реалізує принцип поліморфізму, який є одним з основних в Обєктно-орієнтованому програмуванні.

При натисканні клавіші «Backspace» в активному вікні зміни хаарктеристик ресурсу користувач повертається до списку ресурсів ен зберігаючи зміни в базі даних. Для збереження змін користувач має натиснути клавішу «Enter».

Виконання методу ShowAddResource(рис. 3.23) дуже схоже на виконання методу ChangeResource, адже виконує той самий функціонал, але для нового ресурсу. В методі ShowAddResource виконується редагування нового ресурсу, який буде доданий до бази даних. Також однією з відмінностей цього вікна є те, що навза ресурсу показана у вигляді характеристики, для того щоб була можливість його змінити. Єдиною відмінністю назви ресурсу від характеристик є неможливість видалення імені.

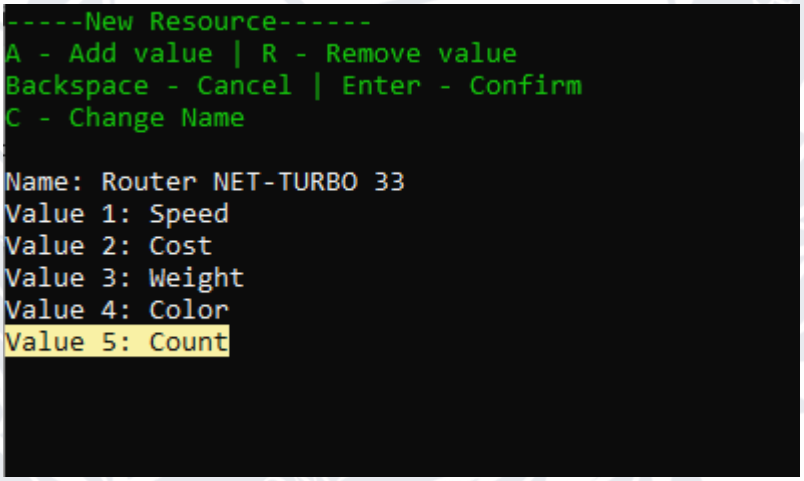

Рисунок 3.23 – Виконання методу ShowAddResource

Враховуючи все вищенаписане, можна прийти до висновку що сервер має широкий функціонал та складний інтерфейс, який дозволяє використовувати цей функціонал у повному обсязі.

3.2 Клієнтський сервіс

Клієнт є важливою частиною сервісу, адже завдяки наявності можливості роботи з даними з боку клієнта існує можливість працювати з базою даних з інших пристроїв, під’єднаних до локальної мережі, в якій існує пристрій з сервером.

Наявність клієнта вирішує проблему централізації сервісу, яка є критичною для інфраструкутри, яка оперує великою кількістю даних. В випадку програмного рішення, яке реалізується в цьому дослідженні клієнт є програмою, яка може бути виконана на будь-якому пристрої, який підтримує виконання програм створених на платформі .NET.

Сервіс розрахований на під’єднання одразу кількох клієнтів до серверу. Без під’єднання до серверу жоден клієнт не може виконувати свій функціонал. Функціонал клієнта полягає у можливості роботи з базою даних, яка знаходиться на сервері. Клієнт має менший функціонал ніж сервер, за допомогою нього неможливо переглянути операції виконані на сервері та переглянути список під’єднаних клієнтів.

Клієнт має повний функціонал для роботи з ресурсами, який включає:

- Перегляд наявних
- Додавання нових
- Зміна характеристик наявних ресурсів
- Зміна значень характеристик наявних ресурсів
- Видалення ресурсів

Цей функціонал може бути використаний в програмі-клієнті завдяки його інтерфейсу. Так як функціонал клієнта обмежений лише роботою з даними, то робота з клієнтом виконується простіше ніж з сервером. Алгоритм роботи клієнта можна зобразити схемою зображеною на рисунку 3.24

Рисунок 3.24 – Алгоритм роботи клієнта

Алгоритм роботи клієнта починається з введення інформації про сервер, до якого клієнт буде під'єднаний(рис. 3.25). Доступ до цієї інформації можна отримати у оператора серверу, вона складається з мережевої адреси серверу та порту з'єднання, які вводяться почергово. Також важливим етапом є надання користувачу варіанту виведення інформації про програму. Такий вибір реалізовано так само, як і у серверу.

```
Resource Manager Client


Connect to server
Info
Enter IP:
169.254.166.233
Enter port:
50489
Connected!
Press any key to continue
```

Рисунок 3.25 – Інтерфейс клієнта на етапі під'єднання до серверу

Після правильного введення інформації про з'єднання клієнт переходить до наступного етапу – створення запиту. Цей етап виконується одразу. Адже одразу після під'єднання клієнт надсилає на сервер запит про отримання усього списку наявних ресурсів. Це важливо для правильної подальшої роботи з сервером. Після обробки сервером запиту, клієнт отримує відповідь та візуалізує працівнику список усіх ресурсів, ідентично до того як це виконується на сервері.

Після виведення на екран списку усіх ресурсів, клієнт має можливість обрати дію, яку він хоче виконати. Цей етап повністю ідентичний до того, як це відбувається на сервері, відмінність лише у тому, як вибір працівника буде оброблений програмою. Коли працівник, який працює з клієнтом, обрав дію

яку він хоче виконати, програма клієнт формує запит, який в подальшому надсилається на сервер. Після надсилання запиту клієнт очікує на відповідь сервера, після чого відповідь серверу візуалізується на екран. Ця відповідь складається того, чи успішний результат, списку задіяних ресурсів та заголовку, який коротко інформує про результат виконання запиту.

Наступним етапом є формування нового запиту на список усіх ресурсів, який дозволить візуалізувати оновлений список ресурсів. Після обробки цього запиту клієнт переходить на нову ітерацію цикла алгоритма, починаючи з очікування запиту.

3.3 Взаємодія серверу та клієнта

Взаємодія серверу та клієнта є важливою складовою архітектури сервісу, адже завдяки правильно налаштовані взаємодії клієнт та сервер мають можливість швидко та ефективно обімнюватись інформацією. Правильно налаштована взаємодія критично необхідна для роботи сервісу у мережі з децентралізованим доступом до даних.

Взаємодія полягає у структуруванні інформації на конкретинх етапах роботи серверу та клієнта. Ця інформація структурована у вигляді запиту та відповіді, які реалізовані у вигляді класів Request та Answer. Клас Request представляє інформацію, яка необхідна для повної обробки запиту сервером. Ця інформація складається з типу запиту – отримати всі ресурси, видалити ресурс, додати ресурс. Завдяки цьому набору можливих запитів, можна реалізувати будь-яку дію з ресурсом, правильно реалізувавши порядок їх виконання. Наприклад, щоб змінити налаштування ресурсу, потрібно спочатку видалити цей ресурс, потім додати новий ресурс з певними змінами. Також важливою складовою запиту є бібліотека ресурсів. Завдяки цьому відкриваються перспективи для подальшого розвитку програми без оновленян архітектури запитів. Список ресурсів необхідний для того, щоб зберігати інофрмацію про ресурс, з яким необхідно провести зміни. Якщо є необхідність

у видаленні ресурсу, сервер перевіряє ідентифікатор ресурсу в запиті та видаляє з бази даних ресурс з відповідним ідентифікатором. Це допоможе вирішити проблему видалення ресурсу при зміні його налаштувань та пошук конкретного ресурсу серед ресурсів з однаковими назвами.

Інший тип запиту – додавання нового ресурсу, потребує наявності усієї інформації про необхідний для додавання ресурс. Під час обробки такого формату запиту до бази даних додається новий ресурс з вказаними у запиту характеристиками.

Найважливіший тип запиту – Отримати всі ресурси. Завдяки цьому запиту клієнт може переглянути повний список ресурсів, які зберігаються на севері. Завдяки обробці такого запиту, клієнт може в подальшому виконувати інші можливі запити.

Після кожного обробленого запиту сервер повертає клієнту відповідь, яка складається з результату обробки, це може бути позитивний чи негативний результат, позитивний вказує на те, що запит успішно виконаний, негативний – що запит не оброблено по певним причинам. Для того, щоб дізнатися причини невиконання запиту у відповіді зберігається рядок символів, який може зберігати будь-який текст. В цей рядок сервер записує причину невиконання запиту. Якщо запит виконаний успішно, сервер записує в цей рядок додаткову інформацію, яка може знадобитися клієнту.

Останньою складовою відповіді є список ресурсів, який може вміщати в себе як один ресурс, так і усі наявні. Такий формат дозволяє повернути всю необхідну інформацію у відповідь на запит клієнта.

Виконання запитів і їх обробка сервером має виконуватися на певних етапах роботи обох програм. Завдяки використанню технології багатопотоковості сервер має можливість паралельно обробляти запит оператора серверу, обробляти запит одного з клієнтів та очікувати на запит іншого клієнта, схема передачі інформації знаходиться на рисунку 3.26.

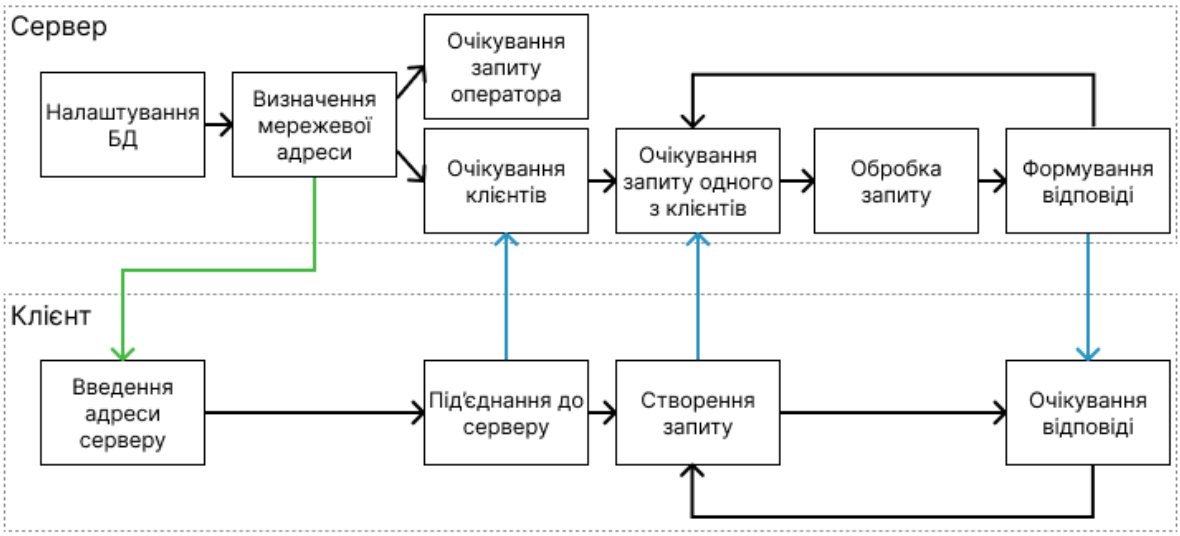

Рисунок 3.26 – Схема передачі даних між клієнтом та сервером

Робота з даними зі сторони серверу полягає в кількох складових. Перші дані, які серверу необхідно надіслати клієнту – адресу для під’єднання. Її неможливо передати через мережу, тому вони передаються вживу. Саме тому доступ до цієї інформації оператор серверу може отримати в будь-який момент, коли ці дані існують. Наступним етапом є очікування на підключення нових клієнтів. Коли з’єднання створено, сервер пасивно очікує нових клієнтів для їх подальшого під’єднання. Цей процес винесено на окремий потік, тому він не заважає іншим процесам. Далі, коли з’єднання з клієнтом створено сервер на окремому потоці очікує запит окремого клієнта у вигляді рядку бітів. Після отримання запиту сервер форматує його у вигляд програмного об’єкту. Цей запит можливий лише у вигляді об’єкту типу Request, що дозволяє швидко його структурувати та визначити тип запиту. Після форматування запит обробляється та формується відповідь. Весь цей процес відбувається на відокремленому потоці, єдиним обмеженням є неможливість одночасної обробки кількох запитів від одного і того ж клієнта зі сторони серверу. Стуркутрована та готова відповідь переформатовується у вигляд бінарного рядка та передається клієнту. Пісял цього етапу сервер повертається до очікування нового запиту.

Зі сторони клієнта поцес передачі даних починається з отримання адреси під’єднання від працівника. На цьому етапі дані вводяться вручні під час роботи програми. Наступним етапом після введення адреси є з’єднання з пристроєм, який знаходиться за цією адресою та під’єднання до порту який використовує сервер. Так як сервер в цей час очікує на під’єднання, то цей процес відбувається дуже швидко. Після під’єднання до серверу програма клієнт готова до використання. Під час роботи з клієнтом працівник формує запит та надсилає його на сервер, зі сторони якого запит очікується. Далі клієнт одразу очікує на відповідь. Відповідь отримується у вигляді бінарного рядку та завжди може бути представлена у вигляді програмного об’єкту типу Answer. Після отримання відповіді, вона форматується у вигляд програмного об’єкту та візуалізується на екрані. Після візуалізації програма автоматично надсилає запит на сервер про усі наявні ресурси, для візуалізації релевантних даних про наявність ресурсів на сервері. Після отримання відповіді вона візуалізується як інтерфейс для подальшого формування запитів.

Під час роботи сервісу дані набувають різної форми. Це зумовлено різними цілями використання цих даних, адже один і той самий тип даних може не відповідати потребам технологій для їх обробки. До прикладу, текстовий формат не підійде для збереження та обробкиданих під час роботи серверу, адже це сповільнить обробку запитів та підтримку сервісу в майбутньому. Бінарний формат даних не підійде для візуалізації, а формат даних в стурктурі програмних об’єктів не можна використати для передачі по мережі. Саме тому під час роботи сервісу дані постійно змінюють свій формат в реальному часі, щоб програмне рішення могло найбільш ефективно їх використовувати.

Під час роботи сервісу використовуються 4 основних формати даних:

- Дані в структурі об’єктів
- Бінарний вигляд
- JSON формат
- Текстовий документ

Під час роботи серверу дані зберігаються в структурі об’єктів, які знаходяться в оперативні пам’яті, це дозволяє найбільш ефективно проводити обчислення в реальному часі та використовувати всю потужність пристрою, на якому знаходиться сервер.

Для передачі даних по мережі сервіс використовує форматування JSON, яке дозволяє форматувати дані в стуркутрі об’єктів в текстовий формат. Після форматування даних в формат JSON, який представелний у вигляді набору символів, дані форматуються в двійковий код для подальшої передачі по мережі. Це дозволяє ефективно зберігати та передавати дані між сервером та клієнтом. Після передачі даних у двійковому вигляді, вони переформатовуються назад до вигляду програмних об’єктів по тому самому алгоритму(рис. 3.27).

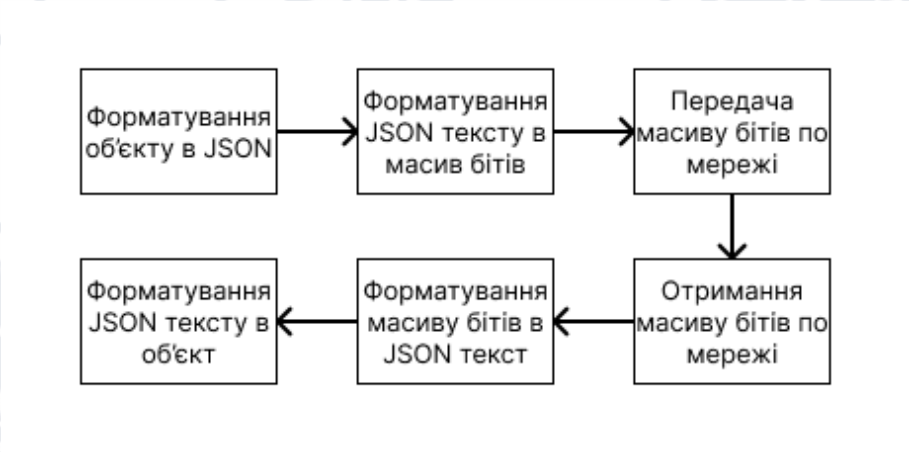

Рисунок 3.27 – Алгоритм форматування даних для передачі по мережі

Для збереження даних на пристрої-сервері використовується формат текстового документу. Зміни в нього вносяться кожного разу як змінюються дані у форматі об’єкту, які знаходяться в програмному середовищі. Процес кожної зміни супроводжується стрічкою коду, яка зберігає об’єкт типу Data в текстовий документ з відповідним ідентифікатором. Для зручного збереження використовується попереднє форматування програмного об’єкту в текстовий JSON формат.

Висновок до третього розділу

У цьому розділі було докладно розглянуто технічні рішення для побудови сервісу, який вирішує визначені раніше проблеми підтримки інфраструктури. Проведено аналіз та опис серверної та клієнтської частин сервісу, а також взаємодії між ними

Серверна частина є основою програмного рішення і забезпечує обробку підключень нових клієнтів, обробку їх запитів та взаємодію з базою даних. Для реалізації цих функцій використано сучасні підходи до об’єктно-орієнтованого програмування, що полягають у створенні низки класів, які відповідають за різні аспекти роботи серверу. Сервер забезпечує багатопотоковість для одночасної обробки запитів від великої кількості клієнтів, що дозволяє підвищити ефективність і швидкість роботи сервісу.

Клієнтська частина сервісу забезпечує можливість роботи з даними з інших пристроїв, підключених до локальної мережі. Клієнт має повний функціонал для роботи з ресурсами, включаючи перегляд, додавання, зміну характеристик та видалення ресурсів. Основним завданням клієнта є формування запитів до сервера та обробка відповідей для подальшої роботи з даними.

Взаємодія між сервером та клієнтом реалізована через структуру запитів та відповідей, що забезпечує ефективний обмін даними. Запити формуються у вигляді об’єктів класу Request, що містять тип запиту та список ресурсів, а відповіді сервера оформлюються у вигляді об’єктів класу Answer, які повертають результат обробки запиту, додаткову інформацію та список ресурсів.

Для передачі даних між сервером і клієнтом використовується формат JSON, який забезпечує зручне форматування даних у текстовий вигляд, що потім перетворюється у двійковий код для передачі по мережі. Після отримання даних вони знову перетворюються у формат об’єктів. Такий підхід дозволяє ефективно зберігати та передавати дані.

Отже, розглянуті технічні рішення забезпечують високу ефективність, гнучкість, масштабованість та надійність створеного сервісу для менеджменту інфраструктури.

ВИСНОВКИ

У цій роботі було досліджено проблематику менеджменту інфраструктури та розроблено сервіс, який вирішує актуальні проблеми сучасної інфраструктури. Процес розробки складався з кількох етапів, кожен з яких був детально описаний у відповідних розділах.

В першому розділі було проведено аналіз потреб у підтримці інфраструктури, на прикладах визначено основні проблеми, з якими стикаються підприємства, розглянуто існуючі технічні рішення для їх вирішення. Було виявлено, що сучасні сервіси мають низку недоліків, таких як централізованість, низька універсальність та висока вартість використання.

В другому розділі було визначено технічне завдання для розробки нового сервісу, яке полягає у вирішенні проблем централізації, забезпечення зручного обліку ресурсів та оптимізацію процесу фізичного пошуку ресурсів. В розділі було складено та обгрунтовано вибір мови програмування та технологій, таких як C#, .NET, JSON та TCP/IP, які забезпечать ефективність та масштабованість сервісу.

В третьому розділі було розглянуто та обгрунтовано технічні рішення для побудови окремих частин сервісу. Серверна частина реалізує обробку запитів клієнтів, взаємодію з базою даних та забезпечує багатопотоковість для підвищення продуктивності. Клієнтська частина забезпечує зручний інтерфейс для роботи з ресурсами, надаючи можливість додавання, зміни та видалення ресурсів у режимі реального часу.

Таким чином, розроблений сервіс для менеджменту інфраструктури успішно вирішує визначені проблеми, забезпечуючи децентралізоване управління ресурсами, зручний облік та швидкий доступ до необхідної інформації. Це робить його ефективним інструментом для підтримки інфраструктури різного масштабу.

СПИСОК ВИКОРИСТАНИХ ПОСИЛАНЬ

1. Овсянікова В. І. Економіка України під час війни. 2022.
2. Потапова Н., Зелінська О.В. Передумови виникнення та основні категорії інформаційної логістики. Цифрова економіка як фактор інновацій та сталого розвитку суспільства. Тези доповідей III Міжнародної науково-практичної конференції учених та студентів, 2022. С. 15-16.
3. Левченко М.Р., Зелінська О.В. Ретроперспектива та аналіз інструментів бізнес-аналітики. Прикладні аспекти сучасних міждисциплінарних досліджень. 2024, С. 221-222.
4. Microsoft – офіційна домашня сторінка [електронний ресурс]. microsoft.com – Режим доступу до ресурсу: https://www.microsoft.com/uk-ua/ (Дата звернення: 14.03.2024)
5. Спільнота програмістів | DOU [електронний ресурс]. dou.ua . Режим доступу до ресурсу: https://dou.ua /(Дата звернення: 30.05.2024)
6. Посвістак, В. С., Демківська, Т. І.. Клієнт-серверна архітектура та її використання при розробці програмного забезпечення. Інформаційні технології в науці, виробництві та підприємництві. 2020.
7. Сеспедес Гарсія, Н., Сеспедес Гарсія, П. Моделі життєвого циклу розробки програмного забезпечення. Молодий вчений, С. 17-20
8. Рефакторинг і патерни програмування [електронний ресурс]. refactoring.guru – Режим доступу до ресурсу: https://refactoring.guru/uk (Дата звернення: 01.06.2024)
9. Li R., Rao J., Wan L..The digital economy, enterprise digital transformation, and enterprise innovation. Managerial and Decision Economics, 2875-2886. 2022

10. Grewal R., Sridhar S. Commentary: Toward formalizing social influence structures in business-to-business customer journeys. Journal of Marketing, 85(1), 98-102. 2021.
11. Shree D., Singh R. K., Paul J., Hao A., Xu S. Digital platforms for business-to-business markets: A systematic review and future research agenda. Journal of Business Research, 137, 354-365.2021.
12. Wikibooks [Електронний ресурс]. uk.wikibooks.org – Режим доступу до ресурсу: https://uk.wikibooks.org/wiki/C_Sharp (Дата звернення: 20.04.2024)
13. Enterprice asset management software – Maximo [електронний ресурс]. ibm.com - Режим доступу до ресурсу: https://www.ibm.com/products/maximo/asset-management (Дата звернення: 27.05.2024)
14. Oluwatosin H. S.. Client-server model. IOSR Journal of Computer Engineering. 2014. P. 67-71.
15. What is network Architecture? [Електронний ресурс]. cisco.com - Режим доступу: https://www.cisco.com/c/en/us/solutions/enterprise-networks/what-is-network-architecture.html (Дата звернення: 7.05.2024)
16. Akella R., Tamirisa A. K., Kunani S. K., Muthiyalu B. G.. Enterprise Application Development with C# 9 and. NET 5: Enhance your C# and. NET skills by mastering the process of developing professional-grade web applications. Packt Publishing Ltd. 2021
17. Software Architecture Articles [Електронний ресурс]. Martin Fowler - Режим доступу: https://martinfowler.com (Дата звернення: 26.04.2024)
18. Software Development Articles [Електронний ресурс]. Ycombinator - Режим доступу: https://news.ycombinator.com (Дата звернення: 23.05.2024)
19. Client-Server Programming Articles [Електронний ресурс]. CodeProject - Режим доступу: https://www.codeproject.com/Tags/Client-Server (Дата звернення: 9.05.2024)

20. Developer Community [Електронний ресурс]. Stack Overflow - Режим доступу: https://stackoverflow.com (Дата звернення: 01.06.2024)
21. Software Architecture Magazine [Електронний ресурс]. IEEE Software - Режим доступу: https://www.computer.org/csdl/magazine/so (Дата звернення: 22.05.2024)
22. Architecture & Design [Електронний ресурс]. InfoQ - Режим доступу: https://www.infoq.com/architecture-design (Дата звернення: 23.05.2024)
23. Software Development Articles [Електронний ресурс]. Medium - Режим доступу: https://medium.com/topic/software-development (Дата звернення: 18.05.2024)
24. Martin R. C. Clean architecture. 2017.
25. Albahari J. C# 10 in a Nutshell. " O'Reilly Media, Inc.". 2022
26. What is C#? [Електронний ресурс]. Microsoft Docs - Режим доступу: https://docs.microsoft.com/en-us/dotnet/csharp/ (Дата звернення: 8.05.2024)
27. C# Code Examples [Електронний ресурс]. DotNetPerls - Режим доступу: https://www.dotnetperls.com (Дата звернення: 9.05.2024)
28. Developer Community Articles [Електронний ресурс]. Dev.to - Режим доступу: https://dev.to (Дата звернення: 17.05.2024)
29. Albahari J., Albahari, B. C# 10 Pocket Reference. O'Reilly Media, Inc".2022.
30. What is C# used for? [Електронний ресурс]. stackify.com - Режим доступу: https://stackify.com/what-is-c-used-for/ (Дата звернення: 3.05.2024)
31. What is .NET? [Електронний ресурс]. dotnet.microsoft.com - Режим доступу: https://dotnet.microsoft.com/en-us/learn/dotnet/what-is-dotnet (Дата звернення: 7.05.2024)
32. Kurkela H.. Application Development With. NET Technologies. 2022.
33. Microsoft Costumer Stories Search [електронний ресурс]. customers.microsoft.com – Режим доступу до ресурсу: https://customers.microsoft.com/enUS/search?sq=.NET&ff=&p=4&so=story_publish_date%20desc (Дата звернення: 19.05.2024)
34. JSON [електронний ресурс]. json.org – Режим доступу до ресурсу: https://www.json.org/json-en.html (Дата звернення: 27.05.2024)

35. Peng D., Cao L., Xu W. Using JSON for data exchanging in web service applications. Journal of Computational Information Systems. 2014. P. 5883-5890.
36. Json.NET Newtonsoft [електронний ресурс]. newtonsoft.com - Режим доступу до ресурсу: https://www.newtonsoft.com/json/ (Дата звернення: 22.05.2024)
37. Clean Architecture [Електронний ресурс]. Clean Coder Blog - Режим доступу: https://blog.cleancoder.com (Дата звернення:14.05.2024)
38. Architectural Patterns [Електронний ресурс]. Microservices.io - Режим доступу: https://microservices.io/patterns/index.html (Дата звернення: 23.05.2024)
39. Nemirovsky M., Tullsen D.. Multithreading architecture. Springer Nature. 2022.
40. What is multithreading? [Електронний ресурс]. Techtarget - Режим доступу: https://www.techtarget.com/whatis/definition/multithreading (Дата звернення: 15.05.2024)
41. Domain-Driven Design Community [Електронний ресурс]. DDD Community - Режим доступу: https://dddcommunity.org (Дата звернення: 28.04.2024)