

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний університет «Запорізька політехніка»

Факультет комп'ютерних наук і технологій

(повне найменування факультету)

Кафедра програмних засобів

(повне найменування кафедри)

Пояснювальна записка

до дипломного проекту (роботи)

бакалавра

(ступінь вищої освіти)

на тему РОЗРОБКА ВЕБЗАСТОСУНКУ ДЛЯ ОБРОБКИ ЗАПИТІВ НА САЙТІ
ФІЛЬМІВ ТА СЕРІАЛІВ. DEVELOPMENT OF THE WEB APPLICATION TO
PROCESS REQUESTS AT A MOVIE AND SERIES WEBSITE

Виконав(ла): студент(ка) 4 курсу, групи КНТз-119

Спеціальності 121 Інженерія програмного

(код і найменування спеціальності)

забезпечення

Освітня програма (спеціалізація)

Інженерія програмного забезпечення

ДОСАЄВА Г.А.

(ПРИЗВИЩЕ та ініціали)

Керівник КАЗУРОВА А.Є.

(ПРИЗВИЩЕ та ініціали)

Рецензент ТЕРЕЩЕНКО Е.В.

(ПРИЗВИЩЕ та ініціали)

2023__

(назва освітньої програми (спеціалізації))

«_____» 2023 року

Слайди презентації

6. Консультанти розділів проєкту (роботи)

Розділ	ПРИЗВИЩЕ, ініціали та посада консультанта	Підпис, дата	
		завдання видав	прийняв виконане завдання
1-4 Основна частина	КАЗУРОВА А.Є., доцент		
Нормоконтроль	КАМІНСЬКА Ж.К., асистент		

7. Дата видачі завдання « 7 » квітня 2023 року.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів дипломного проєкту (роботи)	Строк виконання етапів проєкту (роботи)	Примітка
1	Постановка завдання роботи.	1 тиждень	Завдання, ТЗ
2	Аналіз предметної області.	1 тиждень	Розділ 1
3	Вибір мови програмування та інших технологій розробки.	2 тиждень	Розділ 2
4	Розробка архітектури програми.	2 тиждень	Розділ 3
5	Розробка програми.	3-4 тижні	Розділ 3, 4
6	Тестування та експериментальне дослідження програмного забезпечення.	5 тиждень	Розділ 4
7	Оформлення пояснювальної записки та документів до неї.	6 тиждень	Додатки
8	Нормоконтроль та рецензування.	7 тиждень	
9	Захист роботи.	8 тиждень	

Студент(ка)

_____ Галина ДОСАЄВА
(підпис) (Ім'я ПРИЗВИЩЕ)

Керівник проєкту (роботи)

_____ Аліна КАЗУРОВА
(підпис) (Ім'я ПРИЗВИЩЕ)

РЕФЕРАТ

Пояснювальна записка до дипломної кваліфікаційної роботи бакалавра: 66 с., 11 рис., 1 табл., 3 дод., 27 джерел.

ВЕБСЕРВІС, АРІ, ЗАПИТ, ВІДПОВІДЬ, МАРШРУТИЗАЦІЯ, КОНТРОЛЛЕР, .NET

Об'єкт дослідження – програмні засоби для пошуку і обробки необхідних даних через доступ до них через АРІ.

Предмет дослідження – Web АРІ для взаємодії зі сторонніми джерелами даних.

Мета роботи – розробка Web АРІ застосунку для взаємодії з 3rd party АРІ для отримання і обробки даних з бази даних розважального медіаконтенту.

Матеріали, методи та технічні засоби: об'єктно-орієнтоване програмування, мови програмування С#, персональний комп'ютер з процесором Intel I5 під управлінням операційної системи Microsoft Windows 10/11.

Результати. Створено вебсервіс який допомагає знайти необхідні дані або отримати звіт по унікальному запиту.

Висновки. Розроблено вебсервіс для пошуку персонажів, епізодів та локацій з мультсеріалу “Рік та Морті” та подальшу обробку цих даних або отримання звіту по конкретному запиту.

Галузь використання – для створення розважального контенту, дослідження сучасних трендів, аналізу даних(в маркетингових або інших цілях)

ABSTRACT

Explanatory note to the diploma qualifying work of the bachelor: 66 pages, 11 figures, 1 tables, 3 appendixes, 27 sources.

WEBSERVICE, API, REQUEST, RESPONSE, ROUTING, CONTROLLER, .NET

The object of the research is software tools for searching and processing the necessary data through access to them via API.

The subject of research is a Web API for interaction with third-party data sources.

The purpose of the work is to develop a Web API application for interaction with 3rd party APIs for receiving and processing data from the entertainment media content database.

Materials, methods, and technical means: object-oriented programming, C# programming languages, a personal computer with an Intel I5 processor running the Microsoft Windows 10/11 operating system.

The results. A web service has been created that helps to find the necessary data or receive a report on a unique request.

Conclusions. A web service has been developed for searching for characters, episodes, and locations from the cartoon series "Rick and Morty" and further processing this data or receiving a report on a specific request.

Field of use – for creating entertaining content, researching current trends, data analysis (for marketing or other purposes)

ЗМІСТ

	С.
ПЕРЕЛІК СКОРОЧЕНЬ ТА УМОВНИХ ПОЗНАК	7
ВСТУП	8
1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ	9
1.1 Основні поняття	9
1.2 Опис предметної області	13
1.3 Визначення вимог до інформаційної системи	14
1.4 Висновок	15
2 ОПИС ПРОГРАМИ	16
2.1 Технології створення вебдодатків	16
2.2 ASP.NET. Його переваги та можливості	21
2.3 Висновок	24
3 РОЗРОБКА ПРОГРАМИ	25
3.1 Загальний опис системи	25
3.2 Архітектура системи	25
3.3 Висновок	27
4 ЕКСПЛУАТАЦІЯ, ТЕСТУВАННЯ ТА ЕКСПЕРИМЕНТАЛЬНЕ ДОСЛІДЖЕННЯ ПРОГРАМИ	28
4.1 Опис застосування програми	28
4.2 Керівництво програміста	28
4.2.1 Звернення до програмного продукту (файли проекту)	29
4.2.2 Алгоритм роботи застосунку	32
4.3 Керівництво користувача	33
4.4 Висновок	37
ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАНЬ	39
ДОДАТОК А Технічне завдання	42
ДОДАТОК Б Текст файлів програм	47
ДОДАТОК В Слайди презентації	61

ПЕРЕЛІК СКОРОЧЕНЬ ТА УМОВНИХ ПОЗНАК

ЛОР	– (англ. Lore) - сукупність основної інформації про світ того чи іншого художнього твору.
ООП	– об’єктно-орієнтоване програмування;
ПП	– програмний продукт;
API	– Application Programming Interface;
IDE	– Integrated Development Environment;
JSON	– JavaScript Object Notation;
REST	– Representational State Transfer;
SOLID	– Single responsibility, open–closed, Liskov substitution, interface segregation и dependency inversion;
TDD	– Test-driven development;
XML	– eXtensible Markup Language;

ВСТУП

В сучасному світі концепція програмного продукту поділяє архітектуру на frontend, backend, бази даних. Це три незалежні складові, які можуть виконуватись різними командами розробників під різні задачі, з різними правами і доступами або взагалі мати по 2 і більше “варіантів” інших рівнів. Тому актуальним і важливим є вміння працювати з API, або ж з прикладним програмним інтерфейсом, для того, щоб брати дані, обробляти їх і посилати назад.

Часто буває, що ми не маємо доступу до бази даних напряму, або змушені звертатись до чужих програмних продуктів. Саме тому, в даній дипломній роботі нам треба розробити вебсервіс для обробки запитів на сайті фільмів та серіалів. З допомогою технологій ASP.NET необхідно створити Web API з контролером, який прийматиме GET та POST запити та сервісами, які організують зв'язок зі сторонніми API та обробку отриманих даних в ту форму, яка дана за вимогою.

В даному прикладі використовуватиметься стороння API, яка зробить нам зручний доступ до бази даних медіаконтенту популярного мультсеріалу. Через те, що стороння API має докладну документацію, відкриваються широкі можливості для “спілкування” програмних продуктів між собою, тому коли виникає бізнес-вимога на створення унікального звіту, ми можемо використовувати логіку оформлення звіту в своєму продукті, не втручаючись в реалізацію сторонньої програми.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Основні поняття

Вебсайт – це єдина чи декілька логічно пов'язаних вебсторінок; і розташування вмісту сервера. Як правило, сайт в Інтернеті – це група пов'язаних даних, які мають унікальну адресу та сприймаються користувачами як єдине ціле. Вебсайти називаються так, тому що доступ до них здійснюється через протокол HTTP [1].

Клієнт — апаратний або програмний компонент комп'ютерної системи, який надсилає запити на сервер.

Сервер — апаратний або програмний компонент комп'ютерної системи, який отримує запити від клієнтів і обчислює їх [2].

.NET – це платформа від Microsoft, яка дозволяє створювати програмні додатки. Перший випуск .NET Framework відбувся в 2002 році [3]. 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. Візуальне середовище можна переглянути на рис 1.1.

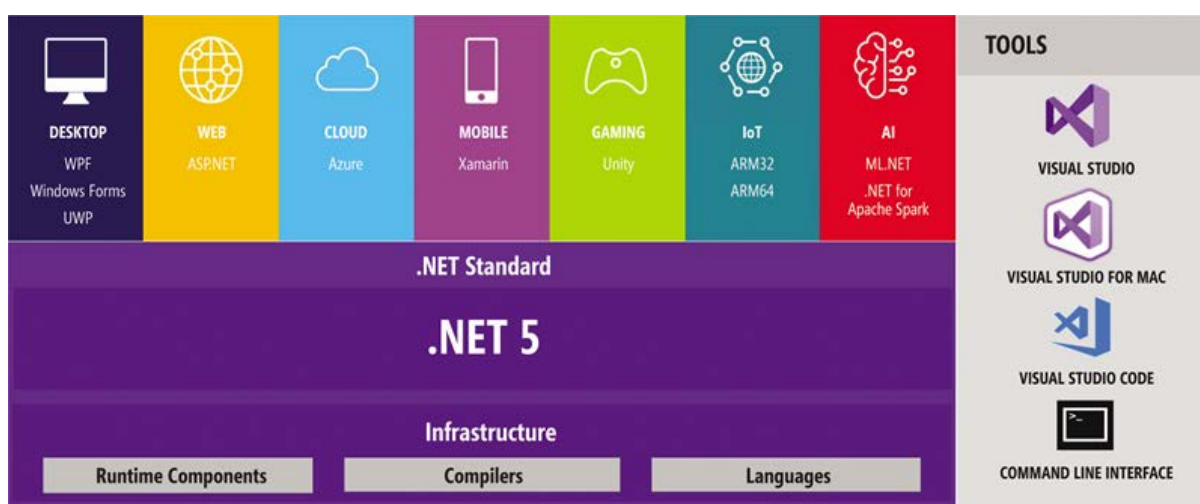


Рисунок 1.1 – .NET платформа

C# – є об'єктно-орієнтованою мовою програмування. Читати її назву треба як «сі шарп». C# – це оновлення та вдосконалення мови C [4].

В 70-х роках виникла така мова програмування, як С. Ця мова швидко стала базою для інших мов та альтернативою мові асемблер. Наступною мовою після С стала мова С++ (с-плюс-плюс), який підтримує парадигми програмування загального призначення та об'єктно-орієнтованого. С++ залишається популярною мовою програмування по сей день [5].

HTTP – це протокол передачі даних. На базі цього протоколу функціонує всесвітня павутина. Цей протокол дозволяє нам відвідувати та взаємодіяти з веб-сайтами в наших браузерях: здійснювати завантаження файлів, з однієї сторінки переходити на іншу, відправляти повідомлення, переглядати зображення та здійснювати покупки. На рис 1.2 показано приклад обміну даними від користувача до сервера за протоколом HTTP [6].

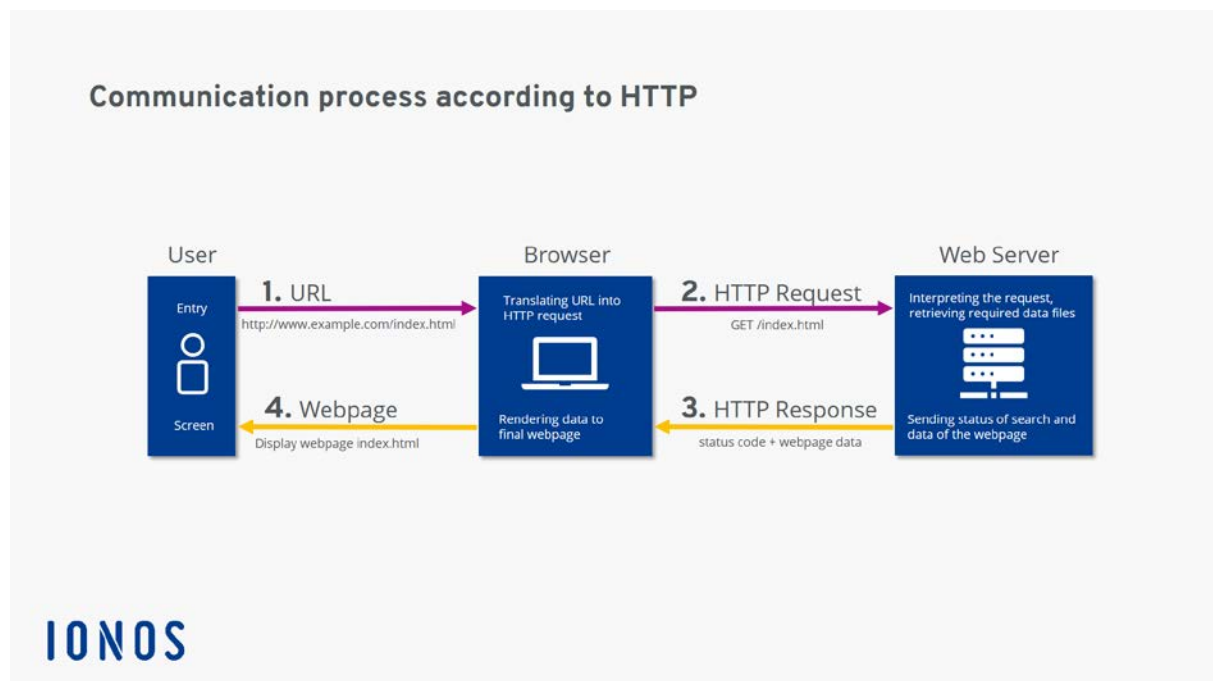


Рисунок 1.2 – HTTP Workflow

Абревіатура HTTP розшифровується як HyperText Transfer Protocol – протокол передачі гіпертексту (тобто текстових документів, які містять посилання інші документи). Зараз з його допомогою передається будь-який формат даних.

Об'єктно-орієнтоване програмування (ООП) – це парадигма

програмування, яка розглядає програму як набір «об'єктів», які взаємодіють один з одним. Він використовує кілька методів із попередніх парадигм, включаючи успадкування, модульність, поліморфізм та інкапсуляцію. Хоча ця парадигма виникла в 1960-х роках, вона не була широко використана до 1990-х років [7].

Об'єктно-орієнтовані мови програмування:

- класичні: Simula 67, Smalltalk, Actor, Eiffel, Objective C;
- сучасні: C++, Object Pascal, Java, C#.

Коріння об'єктно-орієнтованого програмування можна простежити до створення мови програмування Simula 67 у 1960-х роках, у той час, коли загострилися дискусії про кризу програмного забезпечення. Оскільки обладнання та програмне забезпечення ставали складнішими, якість програм було важко підтримувати. Об'єктно-орієнтоване програмування частково вирішує цю проблему, підкреслюючи модульність програми.

Всупереч традиційному погляду, коли програма розглядається як набір підпрограм або список комп'ютерних інструкцій, ООП програми можна розглядати як набір об'єктів. Згідно з парадигмою об'єктно-орієнтованого програмування, кожен об'єкт здатен отримувати повідомлення, обробляти дані та надсилати повідомлення іншим об'єктам. Кожен об'єкт є незалежним автоматом з різними призначеннями та відповідальностями.

Об'єктно-орієнтоване програмування — це підхід до програмування, заснований на представленні програми як набору взаємодіючих об'єктів, причому кожен об'єкт є екземпляром деякого класу, який є членом певної ієрархії успадкування.

Вебзапит – це звернення комп'ютера-клієнта до комп'ютера-сервера, з метою отримати, відредагувати, передати, видалити дані. В рамках цієї роботи будуть використовуватись запити за протоколом HTTP. Тобто запит може містити header та body [8].

Response – це відповідь сервера на запит від клієнта. Кожен response містить в собі header та body. В верхньому блоці може міститись мета-

інформація для маршрутизації, обробки виключень, код response та інше. В залежності від обставин, response може містити коди 1xx (службові коди для продовження вебсесії), 2xx (успішне з'єднання/виконання запиту), 3xx (перенаправлення або повідомлення про зміну домену), 4xx (помилки на стороні клієнта: на сервері не знайдено те, що шукав користувач; не авторизовано; проблеми з підключенням; доступ заборонений тощо), 5xx (проблеми на стороні серверу: перевантаження, відключення серверу, неможливість штатно працювати, помилки додатку на стороні серверу).

ASP.NET – це кросплатформена платформа з відкритим кодом для створення вебсервісів та застосунків. Для цієї платформи типовим є побудова застосунку на технології MVC (Model-View-Controller), втім для поставленої задачі це буде надлишковим. Саме тому для виконання завдання був обраний варіант з Web API [9].

Маршрутизація – це процес визначення обробки запиту, згідно до сценаріїв використання сервісу. В даному випадку було продемонстроване вміння працювати з маршрутизацією для вказування зручної і логічної адреси. Будь-якої, яку забажає власник продукту. Маршрутизація даного проекту була побудована з допомогою атрибутів [10].

Controller – це елемент застосунку, з яким в першу чергу взаємодіє користувач. Можна сказати, що це перша ланка маршрутизації, після якої йде Action [11].

Бібліотека – це код, який був написаний і оформлений для того, щоб спростити роботу для інших розробників. В даному дипломному проекті будуть використовуватись System.Text.Json, System.Net.Http, Microsoft.AspNetCore.Routing.Template, System.Runtime.InteropServices. Перша з них дозволяє працювати з JSON файлами, перетворюючи дані з файлів в такі, з якими можна працювати на програмному рівні і навпаки, формувати з даних на рівні класів і структур класичні JSON файли. Друга бібліотека дозволяє спілкуватись з іншими сервісами чи API: надсилати запити і приймати запити. Дана бібліотека дуже важлива, так як виконує дуже

багато роботи по встановленню зв'язку з іншими серверами.

Git – це система контролю версій проекту і інструмент для розробки програмного забезпечення в команді розробників, дозволяючи синхронізувати всі зміни в одному проекті. Також гіт виконує безпекову функцію. Наприклад, коли нові зміни призвели до критичних проблем, завжди можна повернутись на попередній commit і змусити проект працювати штатно [12].

Репозиторій (англ. repository) – це такий сервер, де різне програмне забезпечення може як зберігатися, так і завантажуватись з нього [11]. Архів програмних продуктів, які можна завантажувати, зберігаються на цьому сервері. Забезпечується збереження даних будь-якого типу та їх обслуговування. Збереження цих даних в репозиторії частіше за все реалізується у вигляді файлів, які можна надалі поширювати в мережі. В якості приклада такого репозиторія можна навести репозиторій вільного програмного забезпечення Sisyphus [14], який розробляється ALT Linux Team. Широко використання репозиторії знайшли в системах контролю версій. Документи, їх історія змін і інша інформація службового характеру – все разом зберігається там. Поняття репозиторій перекладається як сховище.

1.2 Опис предметної області

Існує потреба обробки великих масивів даних у маркетологів, блогерів, письменників і інших людей задіяних в медіасфері або творчості. Маркетологам потрібно орієнтуватись в сучасних трендах, оцінювати популярність, будувати кластери з даних та зв'язки між поняттями, подіями, персонажами, локаціями. Блогери та інші творчі люди створюють свій унікальний контент, надихаються та будують теорії/гіпотези, спираючись на ЛОР. Таким людям треба зручний спосіб для отримання розгорнутих звітів по персонажам, епізодам, локаціям та засоби перевірки наявності персонажа в тому чи іншому епізоді, дослідження його першої згадки, рідного виміру.

1.3 Визначення вимог до інформаційної системи

Для здійснення подальшого проєктування вебзастосунку та програмної реалізації необхідно визначити функціональні вимоги:

- вміння отримувати POST та GET запитів;
- пошук має бути швидшим за одну секунду;
- перший запуск і пошук у користувача має займати не більше двадцяти секунд;
- вебзастосунок має працювати без збоїв.

Метою цієї роботи є створення вебзастосунку, яким будь-який користувач зможе користуватися за допомогою мобільного телефону чи комп'ютера з встановленим Інтернет-браузером без необхідності встановлення додаткового програмного забезпечення.

Необхідно розробити API з ендпоінтами `api/v1/person/[name]` для GET запиту та `api/v1/check-person/` для POST запиту з тілом запиту:

```
{
  personName: [PersonName],
  episodeName: [EpisodeName],
}
```

Для POST запиту результатом має бути перевірка чи був вказаний персонаж в цій серії. Якщо персонаж був в серії, вивести `true`, якщо персонажа не було в даній серії – `false`. Якщо ж такої серії чи такого персонажа не існує, необхідно вивести код 404 Not Found.

Для GET запиту необхідно отримувати звіт по персонажу формату:

```
{
  name: [PersonName],
  species: [Species],
```

```
status: [Status],  
gender: [Gender],  
type: [Type],  
origin: [Origin name], [Origin type], [Origin dimension]  
}
```

Якщо ж персонажа з таким ім'ям не існує, вивести 404 Not Found.

Для демонстрації взаємодії з іншим API слід використовувати сайт наведений в посиланні [15].

Вебзастосунок має отримувати ці дані з відкритих даних і оновлювати їх.

1.4 Висновок

Цей розділ досліджує предметну область, тобто основні поняття та терміни, що стосуються предметної області та об'єкта розробки, та характеристики комплексу задач по створенню вебзастосунку для швидкої обробки запитів з пошуку на сайті фільмів та серіалів.

Сформульовано та конкретизовано робоче завдання.

2 ОПИС ПРОГРАМИ

Розвиток мережі Інтернет призвело до появи нової категорії програм – вебдодатків. До вебдодатків відносять набір вебсторінок, сценаріїв і інших програмних засобів, які розміщуються на одному або декількох комп'ютерах (клієнтських та серверних), які об'єднуються для виконання завдання програми. У той же час вебдодатки, що публікують бази даних в Інтернет, являють собою окремий клас вебдодатків. Сучасні інформаційні системи, засновані на основі вебдодатків, що використовують бази даних, базуються на багаторівневій клієнт-серверній архітектурі та принципах функціонування Інтернету [16].

Вебдодатки – це особливий тип програм, які побудовані за архітектурою «клієнт-сервер». Особливість їх полягає в тому, що саме вебдодаток знаходиться і виконується на сервері, а клієнт при цьому отримує тільки результати роботи. Робота вебдодатку ґрунтується на отриманні запитів від користувача (клієнта), їх обробці та видачі результату. Передача запитів від клієнта до сервера та отримання результатів їх обробки відбувається через мережу інтернет. [17]

2.1 Технології створення вебдодатків

Сучасні вебдодатки – це унікальні розробки, спрямовані на вирішення певних завдань, які стають все більш складними. Це користується все більшим попитом, оскільки дозволяє ефективніше керувати бізнес-процесами.

Подібні додатки відображаються на екрані монітора або смартфона у вигляді вебсторінки. На цій вебсторінці містяться не тільки певна інформація, але й передбачені функціональні кнопки та вікна. Усе це дозволяє виконувати конкретні дії для досягнення бажаного результату.

Позитивна особливість розробки полягає в тому, що клієнтові не потрібно встановлювати на свої пристрої програмне забезпечення. Платформа

працює безпосередньо у браузері [18].

Для розробки серверної частини використовуються різні мови програмування:

- Ruby;
- Java;
- Php;
- Python;
- C#;
- JavaScript.

Розробка клієнтської частини передбачає використання:

- JavaScript;
- HTML;
- CSS.

Мови низького рівня програмування.

Серед характеристик часто зустрічаються: обмеження на абстракції даних, сильна статична типізація, відсутність проміжного середовища виконання, прямий доступ до пам'яті.

Приклади: C, C ++, Assembler.

Позитивне – повний контроль практично над усім; ви використовуєте тільки те, що вам потрібно.

Негативне – додатковий контроль означає додаткові складнощі, які можуть зробити начебто прості завдання більш важкими в реалізації.

Позитивне – більший контроль над пам'яттю; ви можете зробити те, що практично неможливо в інших мовах.

Негативне – управляти пам'яттю може швидко стати дуже складно.

Позитивне – дозволить вам краще зрозуміти, що відбувається за лаштунками в високорівневих мовах і навчить цінувати абстракції.

Негативне – легко закопатися в синтаксисі і дрібних деталях замість того, щоб зрозуміти концепцію і загальну картину.

Позитивне – мотивує думати про ефективність.

Негативне – потрібна попередня оптимізація.

Позитивне – мотивує думати про архітектуру наперед.

Негативне – зміни в поганій архітектурі можуть бути болісними, а гарну архітектуру важко придумати; відносно бідна стандартна бібліотека означає, що ви повинні часто покладатися на третіх осіб або винаходити колесо; необхідно часто вставляти допоміжні шматки коду (boilerplate), що збільшує час на розробку.

Мови середнього рівня програмування.

Серед характеристик часто зустрічаються: фокус на абстракціях, сильна статична типізація, середовище виконання, обмеження на прямий доступ до пам'яті.

Приклади: Java, C #.

Позитивне – управляти пам'яттю не обов'язково, але при бажанні ви можете це робити самостійно.

Негативне – досі потрібно розуміти, як влаштована пам'ять і як працює збірка сміття, але мова цьому навчанню не сприяє.

Позитивне – багаті стандартні бібліотеки.

Негативне – багато абстракцій заважають новачкові в освоєнні концепцій, так як незрозуміло, чому вони створені саме таким чином.

Позитивне – компілюється в байт-код, що спрощує взаємодію з іншими мовами.

Негативне – байт-код вимагає встановленого середовища виконання; досі потрібно часто вставляти стандартні шматки коду (boilerplate), незважаючи на наявність абстракцій.

Мови високого рівня програмування.

Серед характеристик часто зустрічаються: сильне абстрагування, динамічна та / або слабка типізація, повністю незалежне управління пам'яттю і / або наявність середовища виконання.

Приклади: Python, Ruby, JavaScript, Common Lisp.

Позитивне – абстракції роблять складні завдання простими.

Негативне – надбудови для реалізації абстракцій знижують продуктивність.

Позитивне – в цілому все просто і інтуїтивно, навіть при внесенні змін.

Негативне – архітектура може страждати, тому досить просто вносити зміни майже в будь-якому місці замість того, щоб вносити їх там, де дійсно потрібно.

Позитивне – порівняно великі стандартні бібліотеки означають, що те, що ви хочете зробити, швидше за все вже реалізовано і доступно.

Негативне – через приховані деталі складно з'ясовувати причини виникнення проблем, коли вони з'являються.

Позитивне – менше вставок стандартного коду (boilerplate) – синтаксис значно простіше.

Негативне – динамічна типізація ускладнює пошук помилок без запуску коду.

Ruby – динамічна, рефлексивна, інтерпретована високорівнева мова програмування. Мова володіє незалежною від операційної системи реалізацією багатопоточності, сильною динамічною типізацією, збирачем сміття та багатьма іншими можливостями. Підходить для використання в невеликих проектах, в яких немає складної логіки, немає високих вимог до безпеки. [19]

Java – суворо типізована об'єктно-орієнтована мова програмування загального призначення, розроблена компанією Sun Microsystems. Розробка ведеться спільноту, організованою через Java Community Process; мова та основні технології, що реалізують її, поширюються за ліцензією GPL. Дана мова підходить до великих і високонавантажених проектів, які планують масштабуватись або збільшуватись. Велика кількість бібліотек та рішень дозволяють відносно швидко будувати застосунки з різними вимогами і побажаннями [20].

PHP – C-подібна скриптова мова загального призначення, що інтенсивно використовується для розробки вебдодатків. В даний час

підтримується переважна більшість хостинг-провайдерів і є одним з лідерів серед мов, що застосовуються для створення динамічних вебсайтів. Сучасні проекти намагаються не обирати дану мову для розробки сайтів через відносну складність в адмініструванні сайту, до того ж, досить непросто реалізувати безпечність проекту та стійкість до високого навантаження. Нині ця технологія не є актуальною і не задовольняє більшість потреб сучасних клієнтів [21].

Python – високорівнева мова програмування загального призначення з динамічною строгою типізацією та автоматичним управлінням пам'яттю, орієнтована на підвищення продуктивності розробника, читання коду та його якості, а також на забезпечення переносимості написаних на ньому програм. Ця мова програмування – дуже вдале рішення для написання невеликих скриптів, ембедед проектів або для використання в «домашніх умовах». Через особливості компіляції коду, таку мову не можна назвати найефективнішою, сам код непросто читається в силу свого синтаксису. Тому для написання великого проекту з інтеграцією сторонніх сервісів/API – це не найкращий варіант [22].

JavaScript – мультипарадигменна мова програмування. Підтримує об'єктно-орієнтований, імперативний та функціональний стилі. Є реалізацією специфікації ECMAScript. JavaScript зазвичай використовується як вбудована мова для програмного доступу до об'єктів додатків. Попри популярність цієї мови, її можливості, використовувати JavaScript для backend дуже незручно. Відсутність суворої типізації може створювати конфліктні ситуації, помилки в логіці або в коді. Цю проблему частково вирішує сучасна мова програмування TypeScript, але для створення серверної частини додатку – це не найефективніший, не найшвидший варіант. До того ж, для розробки проекту було обрано ООП стиль, а JavaScript та TypeScript підтримують ООП досить номінально. [23]

HTML – стандартизована мова гіпертекстової розмітки документів для перегляду вебсторінок у браузері. Веббраузери отримують HTML документ від

сервера за протоколами HTTP/HTTPS або відкривають з локального диска, далі інтерпретують код в інтерфейс, який відображатиметься на екрані монітора. Важливо розуміти, що HTML не є мовою програмування, тому це не є засіб для розробки програмного продукту. Це лише контракт, зобов'язання і визначення порядку вигляду документу, який буде відрендерений і показаний користувачу [24].

CSS – формальна мова для опису зовнішнього вигляду документа, написаного за допомогою мови розмітки. Також може застосовуватися до будь-яких XML-документів, наприклад SVG або XUL. Відповідає за стиль і зовнішній вигляд документу, спосіб його відображення, колір елементів, фонів, тощо. Актуальним також є Sass. Sass – модуль, включений до Haml. Sass - це метамова на основі CSS, призначена для збільшення рівня абстракції CSS-коду та спрощення файлів каскадних таблиць стилів. Мова Sass має два синтаксиси: Sass (Syntactically Awesome Style Sheets) – спрощений синтаксис CSS. А також SCSS (Sassy CSS) – заснований на стандартному CSS синтаксисі [25].

Для розробки вебзастосунку для пошуку даних була вибрана мова програмування C# на технологіях ASP.NET, а також середовище Visual Studio Code.

C# – це високорівнева мова програмування, яка є дуже зручною для розробки вебдодатків. Ця мова має дуже велику кількість базових бібліотек, в ній передбачена робота з великою кількістю різних структур даних з великою кількістю перевантажень. Дуже гнучка мова, в якій в повній мірі реалізовані інструменти розробки програмного забезпечення в ООП (об'єктно орієнтоване програмування) стилі. Саме ця мова була обрана для розробки Web API.

2.2 ASP.NET. Його переваги та можливості

ASP.NET є фреймворком розробки вебдодатків, розробленим компанією Microsoft. Він використовується для створення потужних та масштабованих вебдодатків з високою продуктивністю. Однією з головних переваг ASP.NET є

його інтеграція з платформою .NET, що надає розробникам широкий набір інструментів та можливостей для ефективного програмування. ASP.NET також підтримує багато мов програмування, таких як C#, Visual Basic та F#, що дає змогу розробникам вибирати найбільш підходящу для них мову.

Одним з ключових компонентів ASP.NET є Web API, що дозволяє розробникам створювати RESTful вебсервіси. Web API надає можливість обробки HTTP-запитів та відповідей, а також передачі даних у форматі JSON або XML. Це дозволяє розробникам побудувати потужні та гнучкі додатки, які можуть взаємодіяти з різними клієнтськими пристроями та платформами. [26]

Структура проекту ASP.NET може бути організована згідно шаблону MVC (Model-View-Controller) або шаблону MVVM (Model-View-ViewModel). У шаблоні MVC, модель відповідає за управління даними, представлення (вигляд) займається відображенням інтерфейсу користувача, а контролер обробляє запити користувача та координує взаємодію між моделлю та представленням. У шаблоні MVVM, на додаток до моделі, представлення та контролера, доданий ще один компонент – ViewModel, яка відповідає за збереження та управління даними, що використовуються в представленні. Схема роботи алгоритму запитів до Web API наведений на рис. 2.5.

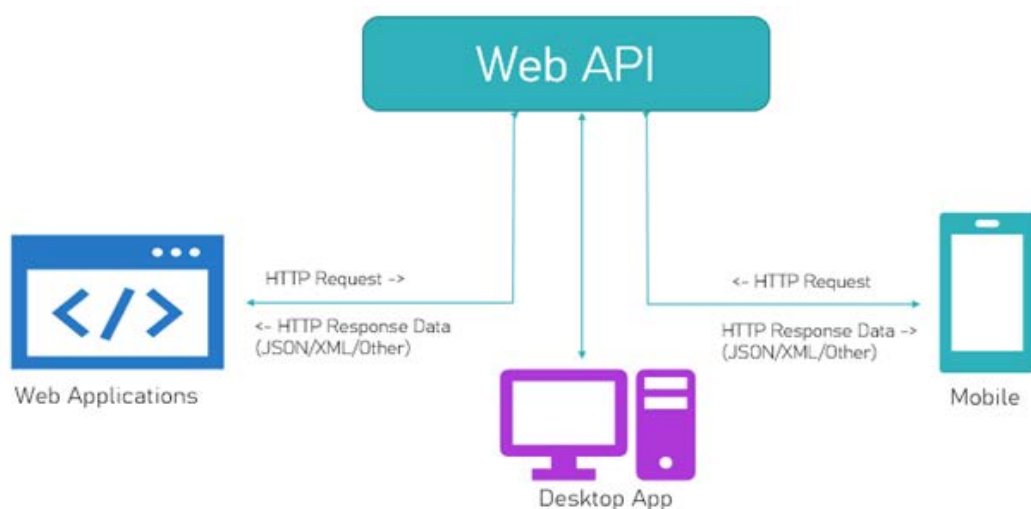


Рисунок 2.5 – Схема запитів до Web API

Загалом, ASP.NET – це потужний фреймворк, який дозволяє розробникам ефективно створювати вебдодатки. Він надає переваги завдяки

інтеграції з платформою .NET, широкому вибору мов програмування та можливості створення Web API для реалізації вебсервісів. Крім того, зручна структура проекту, які MVC або MVVM, допомагає розробникам організувати код та забезпечити чистоту та підтримку додатків на рівні розробки.

Відповідно до вимог, поставлених до програмного забезпечення проаналізуємо мову C# (табл. 2.1).

Таблиця 2.1 – Порівняння мов програмування

Параметр	PHP	Node.js	C#
Сфера застосування	Веброзробка (сервер)	Веброзробка (сервер)	Веброзробка (сервер), мобільні та desktop додатки
Наявність CMS	Багато	Небагато	Небагато
Швидкодія	Висока	Середня	Висока
Вартість	Безкоштовний	Безкоштовний	Безкоштовний
Паралельна робота	Багатопоточність	Асинхронність одного потоку	Багатопоточність
SSR	+	+	+

Мова програмування C# вважається сьогодні однією з самих потужних мов. Вона розвивається досить швидко та користується сильним попитом в ІТ-галузі. Цією мовою пишуться різноманітні програми: починаючи від маленьких десктопних і закінчуючи більш масштабними вебсервісами та вебпорталами. Цими програмами користуються мільйони людей щодня.

Мову C# сьогодні вже не можна вважати молодого мовою, так само як і вся .NET платформа, вона пройшла суттєвий шлях. В лютому 2002 була випущена перша версія цієї мови, тоді ж був випуск Microsoft Visual Studio .NET. На зараз використовується версія мови C# 11, яка була випущена в листопаді 2022.

Синтаксис мови C# подібний до синтаксису мови Cі та деякою мірою подібний до C++ та Java. Отже, якщо ви володієте якоюсь з зазначених мов, неважко буде опанувати й мову C#.

Мова C# є об'єктно-орієнтованою мовою і багато чого в цьому сенсі перейняла у Java та C++. В якості прикладу можна навести те, що C# підтримує такі явища, як успадкування, статичну типізацію, поліморфізм та навантаження операторів. Цей підхід робить можливим вирішення таких завдань, як створення великих і, водночас гнучких й масштабованих додатків з можливістю розширюватись.

2.3 Висновок

В даному розділі було проведено аналіз та вибір інструментарію розробки програмного продукту. Проведено порівняльну характеристику мов програмування та встановлено, що найбільші переваги для розробки має мова програмування C#. А саме будуть використані можливості платформи .NET та технологія ASP.NET.

3 РОЗРОБКА ПРОГРАМИ

3.1 Загальний опис системи

Програмне забезпечення у вигляді вебзастосунку, яке було наведено у цій роботі, призначене для пошуку персонажів та перевірки валідності зв'язок персонаж-епізод. Це програмне забезпечення повинно забезпечити отримання правдивої інформації без збоїв та будь-якої затримки. Вебзастосунок завантажує дані з сайту відкритих даних та поновлює їх у своєму сервісі та надає змогу обробляти згідно з бізнес-логікою.

Швидкість роботи досягається відсутністю frontend складової. Використані кращі практики побудови застосунка для можливості масштабування, модифікації і комфортної підтримки продукту. Разом з тим, використання API дозволяє іншим продуктам взаємодіяти з даним застосунком.

3.2 Архітектура системи

Архітектура визначає як саме працюватиме застосунок, розподіл ролей, порядок обробки даних. Це спосіб структурувати елементи застосунку задля ефективної підтримки продукту. На сьогоднішній день існує два основних підходи до побудови проєктів: монолітний застосунок та застосунок побудований на мікросервісній архітектурі. Перевага мікросервісної архітектури в тому, що застосунок складається з абсолютно різних незалежних проєктів, що безпечно в разі відмови чи падіння одного з сервісів, але це накладає додаткові вимоги до розробки і менеджменту процесів, робить розробку залежною від певної інфраструктури (наприклад, Azure Fabric Service). Така інфраструктура доречна для великих проєктів. [27] Алгоритм роботи даної архітектури можна побачити на схемі 3.1.

Монолітні застосунки діляться на ще декілька стилів архітектури: multilevel, multilayer та onion. Різниця між цими стилями в деталях. Для нашого проєкту доречним буде скористатись multilevel-архітектурою, де ми зможемо

мати різні рівні абстракцій для обробки запитів та даних. Детальніше схему роботи multilevel-архітектури на прикладі теми дипломної роботи можна побачити на рис. 3.2.

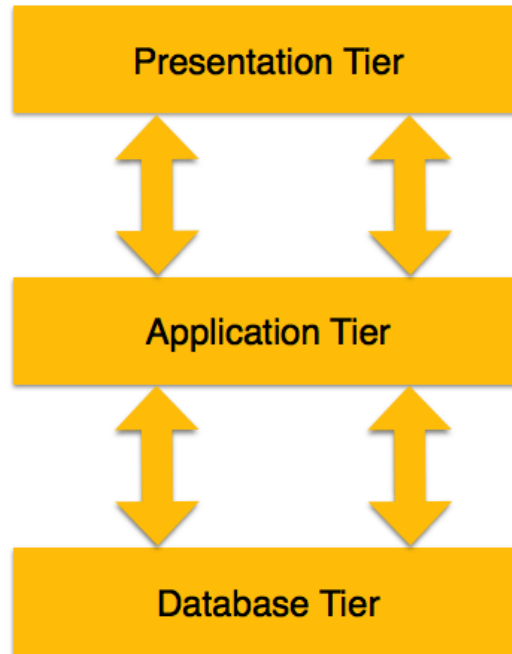


Рисунок 3.1 – Схема багаторівневої архітектури

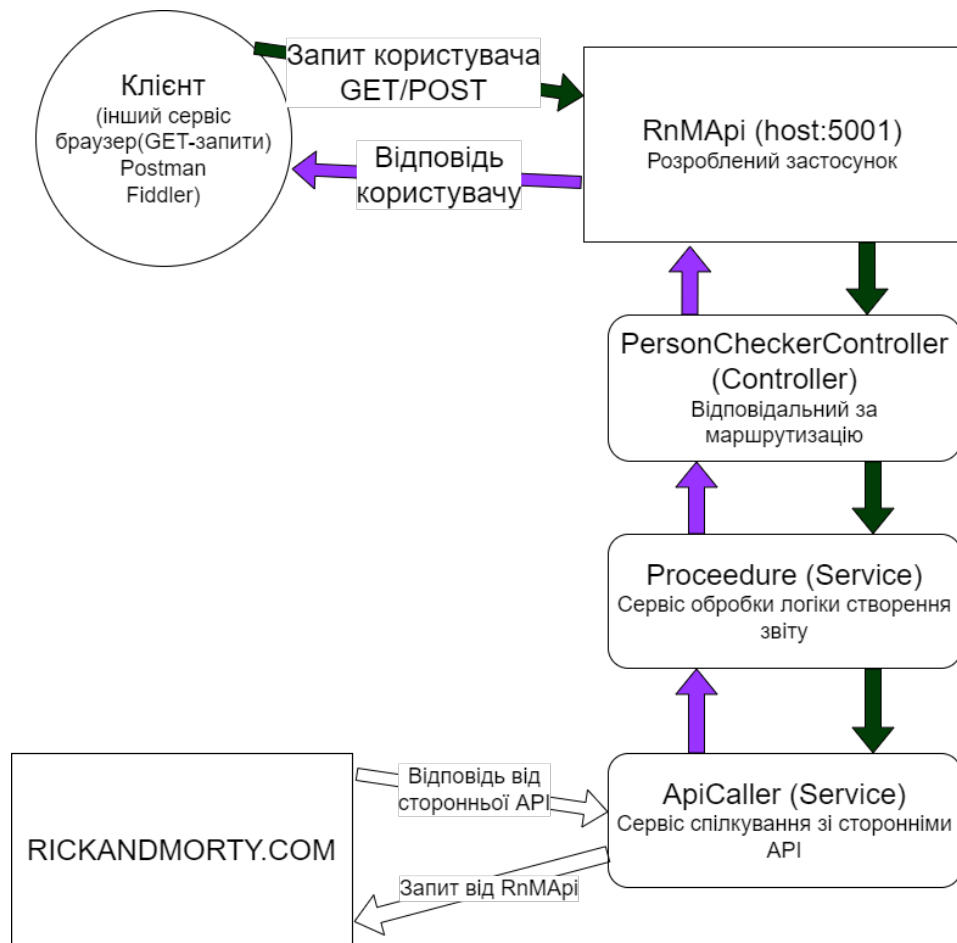


Рисунок 3.2 – Схема архітектури застосунка RnMApi

3.3 Висновок

В даному розділі було обрано та проаналізовано інструментарій розробки програмного продукту та спроектовано архітектуру майбутнього програмного продукту.

4 ЕКСПЛУАТАЦІЯ, ТЕСТУВАННЯ ТА ЕКСПЕРИМЕНТАЛЬНЕ ДОСЛІДЖЕННЯ ПРОГРАМИ

4.1 Опис застосування програми

Вебзастосунок призначено для отримання інформації з мультсеріалу:

- пошук конкретного персонажа;
- пошук епізоду;
- пошук локації, де персонаж з'явився вперше;
- співставлення персонажа з епізодом (перевірка наявності).

4.2 Керівництво програміста

Мінімальні та необхідні системні вимоги до апаратної платформи ПП:

- мінімум 2 ГБ оперативної пам'яті, рекомендується 4 ГБ оперативної пам'яті;
- вільне місце на диску 5 ГБ мінімум;
- Windows, Linux, macOS.

Також необхідно встановити такі програми/пакети:

- .NET SDK;
- середовище розробки (Visual Studio Code, JetBrains WebStorm, або інше але з підтримкою веброзробки);
- ASP.NET;
- Postman;
- веббраузер (Google Chrome, Mozilla Firefox, Safari, Opera, Chromium, або інший).

Для старту проекту необхідно встановити пакети, для цього виконати команду `npm i`. Потім виконати команду `npm run dev`. А потім перейти на посилання.

4.2.1 Звернення до програмного продукту (файли проекту)

Структура проекту базована на структурі класичного Web API застосунку, який реалізовано на платформі .NET. Додатково в директорію проекту були додані такі папки: Interfaces, Models, Utils, Services, та була створена папка Controllers для кращої організації контролерів (для покращення масштабування застосунку).

В папці Models містяться моделі: Episode.cs, Location.cs, Origin.cs, Person.cs, PersonResponse.cs. Episode.cs складається здебільшого з string за винятком id та масиву персонажів, які присутні в епізоді. Структура цієї моделі містить назву епізоду(знадобиться для формування репорту), дати, номеру епізоду, url для прямого доступу з API(що також буде використано) і інформація про дату створення серії. Аналогічно в Location – здебільшого це string за винятком id, масиву “мешканців локації”. На відміну від Episode, тут є поля “вимір” та “тип”. Origin містить три поля string, де є ім’я, тип і вимір. Два окремі класи характеризують персонажа. Один потрібен для сереалізації JSON файлу, який прийде у відповідь на запит, а інший потрібен для формування custom звіту, який вимагається в ТЗ. Все це згідно до документації.

В папці Utils містяться файли RequestBodyCheckPersonEpisode.cs та RnMAppSettings.cs. Перший клас необхідний для формування і обробки запиту для перевірки чи був персонаж в серії. Використовується для сереалізації/десереалізації та в використанні логіки обробки цього запиту. Другий клас існує для можливості швидко і зручно змінювати адресу сторонньої API, на випадок заміни API або у випадку змін на самій API в її контролерах. Цей клас дає можливість вказати ендпоінти для запитів в якості властивостей appSettings.json.

В папці Interfaces знаходяться файли: IApiCaller.cs, IProceeder.cs та IRnMApiCaller.cs. Дані інтерфейси необхідні для реалізації принципів розробки SOLID, зокрема для реалізації механізму Dependency injection. Даний принцип дозволяє гнучко налаштовувати застосунок, змінюючи класи

під конкретні потреби. Тому для роботи зі сторонніми API був створений інтерфейс `IApiCaller.cs` з методом `GetDataFromApi`, з параметром (`string url`), який повертає `string` з результатом запиту. Цей результат – це по суті є документ в форматі JSON, який додатково треба обробити. Даний інтерфейс є базовим і містить лише один метод щоб задовольняти вимогу принципів `solid`, де `I` – `interface segregation`. Це означає, що інтерфейс має містити в собі мінімальну кількість вимог до контракту. Це чудово працює тому що мова `C#` дозволяє реалізувати безліч інтерфейсів, але не дозволяє успадкувати більше, ніж один клас. `IRnMApiCaller.cs` – це більш конкретний інтерфейс, який базується на вищезгаданому інтерфейсі і додає ще три методи для кожного контролеру конкретної API, яка була обрана для демонстрації. Кожен з методів `GetPersons`, `GetEpisodes`, `GetLocations` приймає аргументом `string`. У випадку з першим методом, це ім'я персонажа, з другим – назва серії і для третього – назва локація. Так як на такий запит до сторонньої API, `response` – це список, для зберігання і обробки цієї інформації використовується структура даних `List<T>`, де `T` – в залежності від методу, це модель з папки `Models`: `Person`, `Episode`, `Location`.

Останній же інтерфейс – це інтерфейс для обробки запитів і відповідей. Так як в ТЗ до цієї задачі нам необхідно перевіряти чи є персонаж в серії та отримувати інформацію про персонажа, було передбачено два методи: `CheckPersonInEpisode`, який приймає ім'я персонажа, назву епізоду та параметр `found` та `GetPersonData`, що приймає тільки ім'я персонажа. Тепер ми можемо розробити декілька класів для служб, які можуть відрізнитись навіть логікою, але завдяки контракту, який вони будуть виконувати, ми завжди будемо отримувати те, що передбачено.

В папці `Services` знаходяться файли: `ReqResProceeder.cs` та `RickAndMortyApiCaller.cs`. Перший файл містить ін'єкцію об'єкту `IRnMApiCaller` для гнучкого налаштування даного сервісу. Це необхідно для того, щоб будь-який інший клас, написаний для взаємодії зі сторонніми API міг працювати. Всередині цього сервісу описана логіка обробки запитів в

методах `CheckPersonInEpisode` та `GetPersonData`. Ці два методи отримали тут реалізацію через вимогу інтерфейса `IProseeder`, який реалізовується даним сервісом. Логіка `CheckPersonInEpisode` полягає в тому, щоб викликати методи об'єкту `apiCaller` зі списком епізодів та списком персонажів. Якщо ж серія і персонаж існують і такі записи були знайдені, відбувається перевірка на наявність. Алгоритм перевірки полягає в тому, щоб перевірити в масиві `episodes` об'єкту `Person` співпадіння по `id` епізоду, саме для отримання `id` і потрібні детальні дані по епізодах, тому що в полі `url` зберігається повна адреса, де номер в кінці – це `id` серії. Обробивши `string` і діставши з нього необхідні нам дані, ми можемо порівнювати `id` серій. Оптимальним є порівняння методом `Contains()` з простору імен `LINQ`. Це зручно і тим, що не потрібно виділяти додаткову пам'ять для нового масиву чи для змінних циклу `for`, і тим, що метод поверне `bool`, що якраз і треба за умовами ТЗ. З методом `GetPersonData` завдання полягає в тому, щоб отримати репорт не такого виду, як нам дає `response` від `API`, а більш розширений. Для цього була створена додаткова модель даних, яка необхідна для “збирання” об'єкту з різних деталей. Тут використовуються вже відомі методи `apiCaller`, зокрема `GetPersons()`. Отримавши інформацію про персонажа, з'являється можливість шукати конкретну `Origin`. Якщо ж персонажа не було знайдено, відправляється `NotFound`.

В папці контролерів міститься єдиний контролер `PersonCheckerController.cs`. Тут так само передбачена ін'єкція об'єкту, який реалізує інтерфейс `IProseeder`. А використовується він всередині двох `Action`, у яких реалізована маршрутизація через атрибути. Всередині контролерів описано мінімум логіки, лише та, яка необхідна для перевірки імен на коректність, приведення параметрів до одного виду (`lowercase`) і виклики методів `GetPersonData(string name)` та `CheckPersonInEpisode(string personName, string episodeName, out bool found)`.

4.2.2 Алгоритм роботи застосунку

Робота застосунку починається з запуску вебсерверу. В залежності від вказаного в налаштуваннях, веб сервер почне прослуховувати якийсь з портів. На цей порт можна надсилати запити в формі, яка вказана в ТЗ. В залежності від аргументу, буде задіяний один з контролерів. Важливо, що передбачена робота з одного роду GET запитом і одним POST запитом. Саме контролер відповідальний за прочитання параметрів запиту. В GET запиті параметри вказуються в адресному рядку, в той час, коли в POST запиті, параметри передані в тілі запиту. Тіло вчитується за допомогою атрибуту [FromBody]. Доступ до властивостей тіла запиту реалізовується по іменам властивостей. В випадку даного проекту, ім'я персонажу можна отримати так:

```
var name = body.personName;
```

Після вчитки аргументів запиту, вони перевіряються на коректність (чи не пусті ці поля) і приводяться до одного виду завдяки методу ToLower(). Це важливо для порівняння і відправлення запитів на сторонню API, для уникнення конфліктів з документацією. Після перевірок і приведення даних, викликається метод обробника, В процес включаються сервіси, один з яких відповідальний за “спілкування” з API для отримання необхідних даних. В середині другого сервісу виконується обробка даних і в залежності від результату, контролер отримує коротку відповідь для виведення її клієнту. Варто виокремити метод, який перевіряє чи був персонаж в епізоді. Один з параметрів методу, що перевіряє цю інформацію, є out параметр. Він потрібен тому, що без нього false означатиме або те, що персонаж не був в цій серії, або те, що чи то персонажа, чи то епізода не існує. Для внесення ясності в цю логіку, було додано out параметр, по якому видно, чи персонаж просто не був знайдений, чи його взагалі не існує. Результат роботи застосунку отримується такий же, як вказано в ТЗ. Для уникнення помилок чи виключень, на кожному етапі роботи застосунку передбачені перевірки на null. Сервіси і контролер спроектовані так, що навіть отримавши null це лише означає лише сценарій, коли якісь з даних не були знайдені на стороні API. Ну а перевірка на null

вбудована і на етапі контролеру, і на етапі прийняття даних з API.

4.3 Керівництво користувача

Для взаємодії з застосунком знадобиться використовувати програми типу Fiddler або Postman. Зокрема для відправлення POST запитів. Для GET запитів буде достатнім відкрити веббраузер і перейти за адресою, включаючи контролер, екшн та параметр.

На зображенні (рис.4.1) тестується контролер Person для GET запиту з параметром name=unity. Дане значення обране за замовчуванням і саме цей запит виконується при запуску API як вебзастосунка. Як результат, ми бачимо всю доступну інформацію за цим персонажем у форматі, який був вказаний в ТЗ. Подібний запит можна зробити і з веббраузера. Відповідь буде показана замість HTML-сторінки

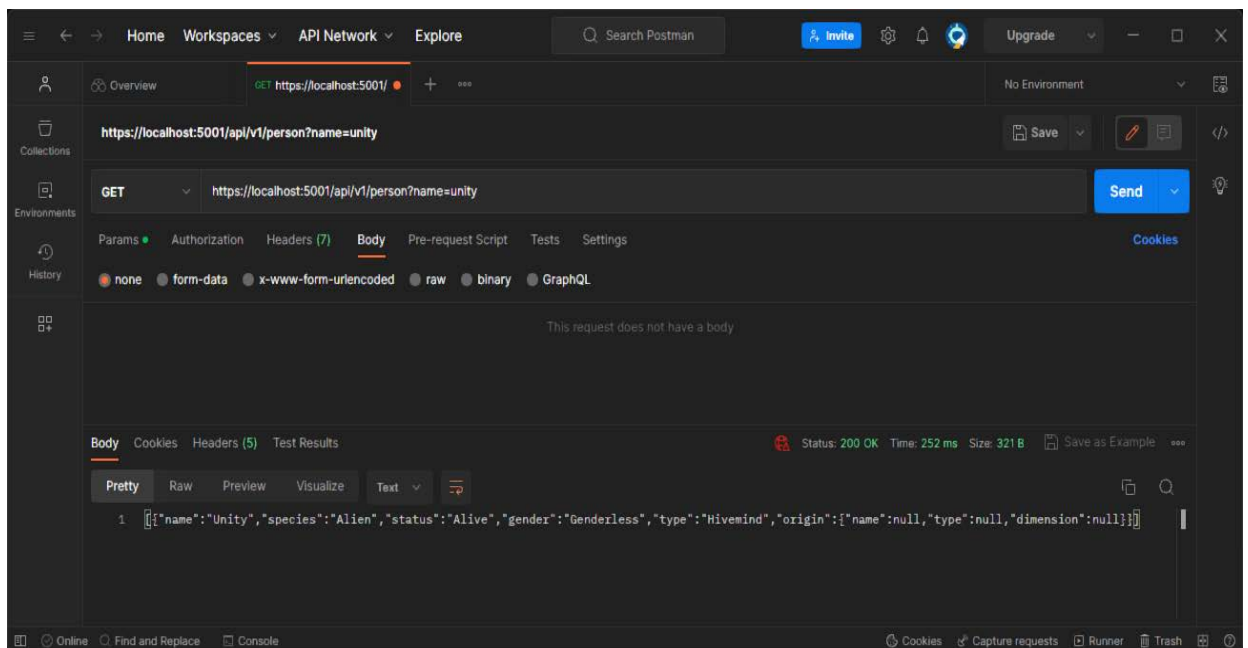


Рисунок 4.1. Тестування контролеру Person для GET запиту з параметром name=unity

Наступне зображення (рис.4.2) демонструє роботу API при запиті персонажа з ім'ям Rick Sanchez. При обробці цього запиту виникає дві

проблеми: текст надрукований з використанням різних регістрів і даний параметр складається з двох слів. Для цього випадку все спрацьовує штатно і так, як вказано в ТЗ, але пошук по цьому імені повертає масив персонажів, тому що в даному мультсеріалі існує декілька персонажів з іменем Rick Sanchez. Вихід з цієї ситуації можна знайти двома шляхами. В представленій версії, вибирається перший з колекції запис, але демонстрація усього масиву також передбачена. Ця можливість реалізована в іншій гілці репозиторію. За потребою можна додавати аргумент (зі значенням за замовчуванням), який означатиме чи показувати одного персонажа, або масив.

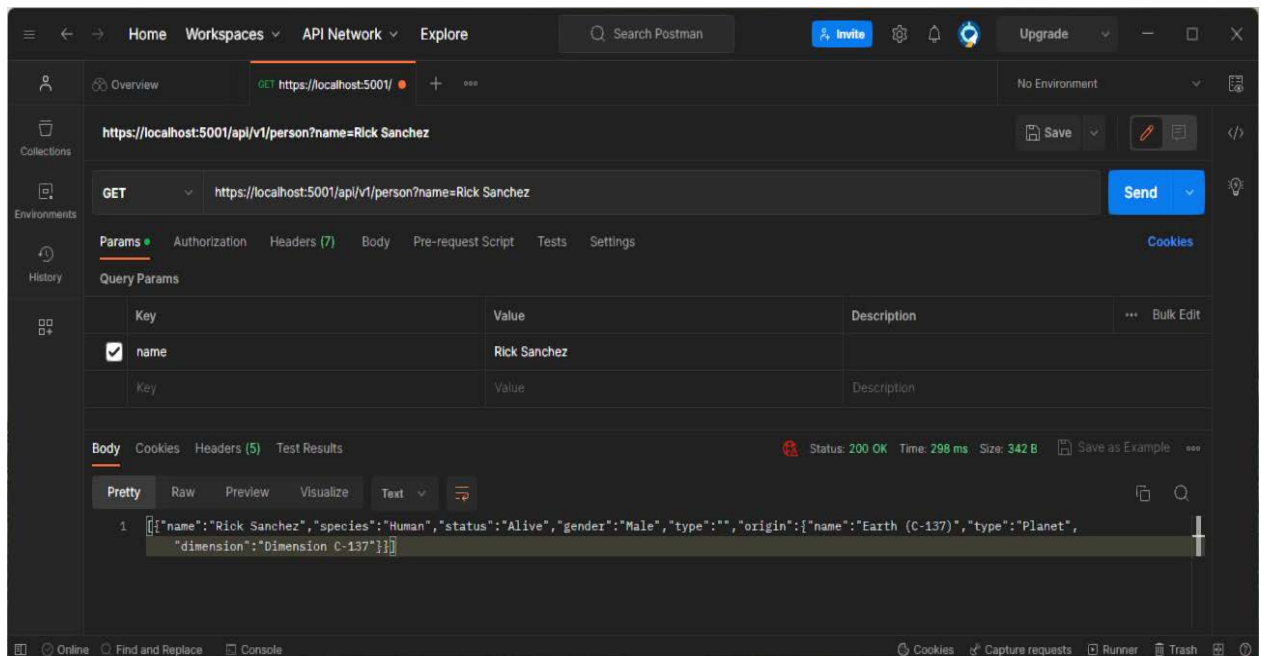


Рисунок 4.2. Демонстрація роботи API при запиті персонажа з ім'ям Rick Sanchez

Останній тест (рис. 4.3) для цього контролеру демонструє що буде, якщо даного персонажа не існує. В такому випадку має повертатись response з кодом 404 і повідомленням Not Found. Контроллер відпрацьовує як і задумано.

Перевірка наявності персонажа в епізоді відбувається через POST метод, з тілом запиту. Приклад оформлення запиту сформовано згідно до ТЗ. Перший запит має повернути true, оскільки в кожній серії мультсеріалу персонаж Rick Sanchez є. Важливо, що з веббраузера подібний запит зробити

не можна. Знадобляться додаткові утиліти.

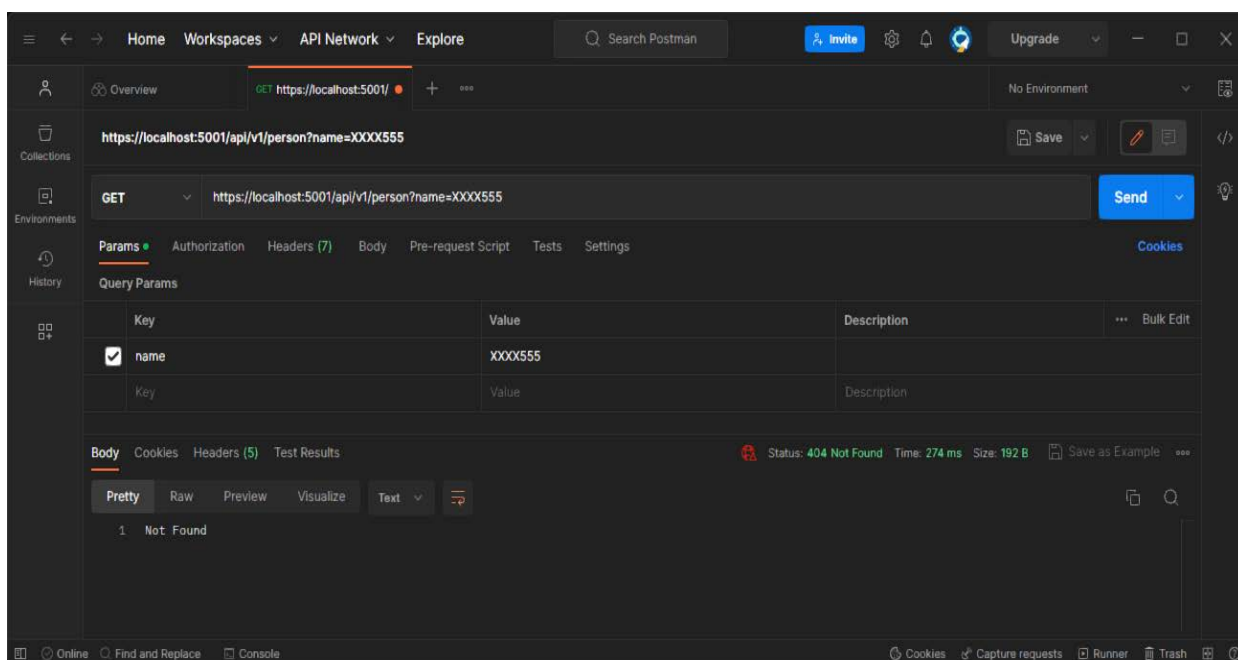


Рисунок 4.3. Демонстрація, що буде, якщо даного персонажа не існує

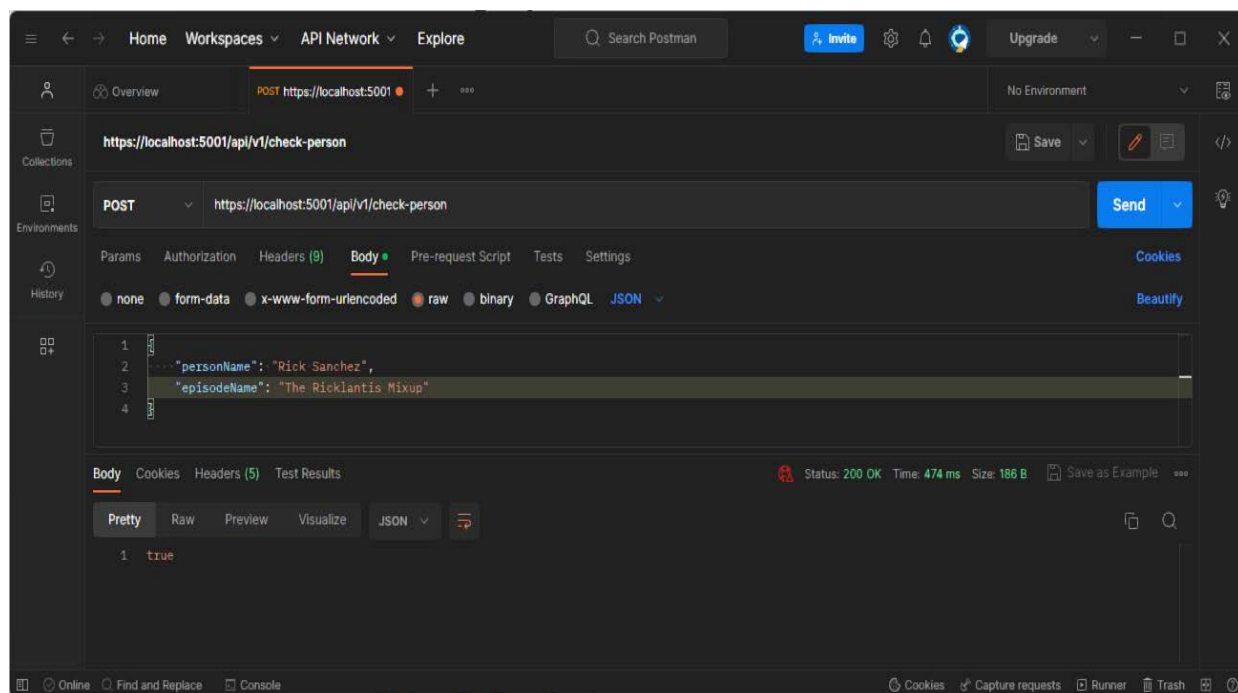


Рисунок 4.4. Коректна робота застосунку при валідному запиті

В другому варіанті (рис. 4.5), персонажа Unity в серії немає, тому API має повернути нам `false`. Що ми і бачимо у відповіді.

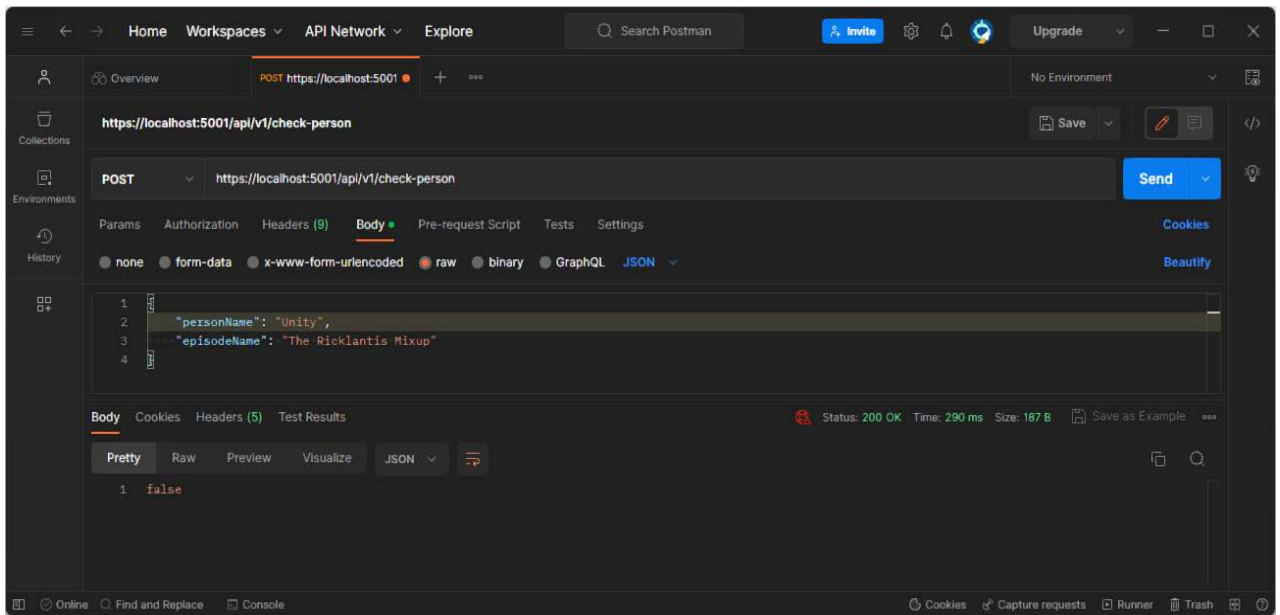


Рисунок 4.5. Демонстрація тесту, якщо персонажа в серії немає

Останній запит (рис. 4.6) – це приклад, коли ні серії, ні персонажа не існує. Але даний запит не ламає застосунок, не викликає виключень, а штатно повідомляє, що таких даних не було знайдено (код 404, Not Found). Це зроблено задля безпеки використання і зручності користувача, тому що може бути допущена помилка друку або злоумисники захочуть скористатись вразливостями для блокування роботи застосунку.

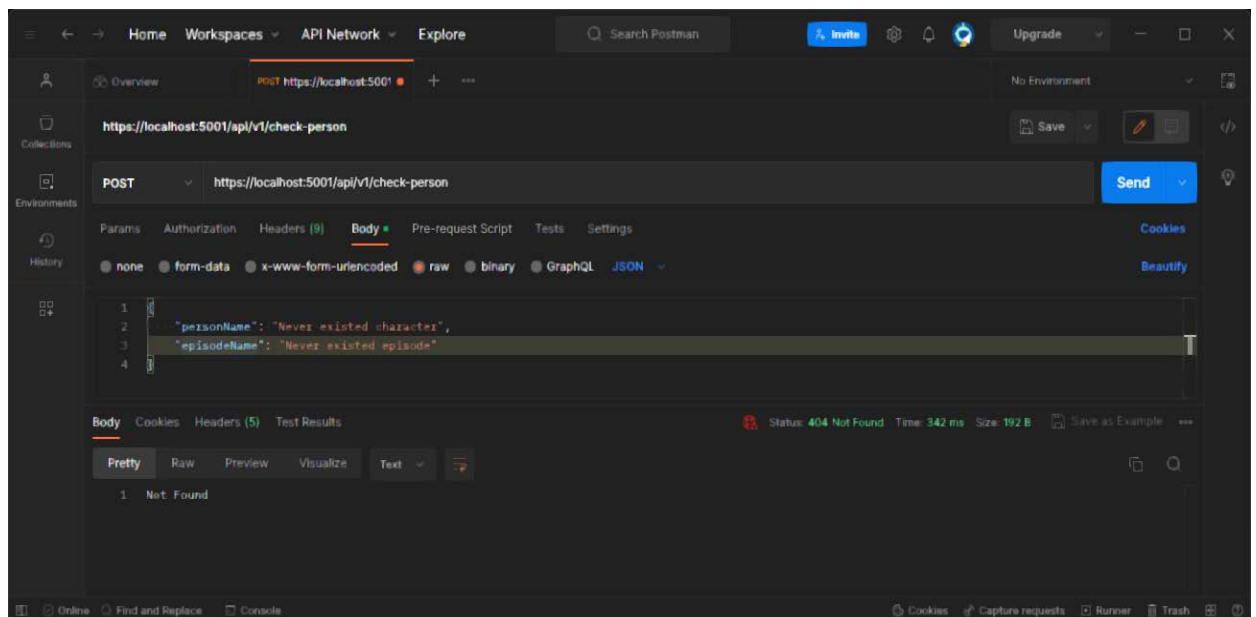


Рисунок 4.6. Демонстрація тесту, коли ні серії, ні персонажа не існує

Таким чином, даний застосунок має затребуваний функціонал, захист від помилки оператора чи втручання зловмисників, а також були передбачені різні варіанти взаємодії.

4.4 Висновок

В даному розділі було відображено призначення розробленого ПП та мінімально необхідна апаратна платформа для встановлення та використання цього ПП. Також була наведена інструкція по роботі оператора з ПП. Інтерфейс користувача розробленого ПП є простим та інтуїтивно зрозумілим і не потребує додаткового навчання оператора.

ВИСНОВКИ

Дана робота виконана за концепцією ООП та принципами SOLID з побудовою ієрархії класів та інтерфейсів, що дозволяє гнучко розширювати можливості продукту. Реалізовані класи для спілкування з іншими API з використанням бібліотеки HttpClient, клас-обробник запитів, контролер для взаємодії користувача з програмним продуктом.

У користувача є можливості формувати необхідні звіти по персонажам, отримувати інформацію про епізоди та локації, в яких був вперше помічений персонаж, є перевірка чи був той чи інший персонаж наявний в конкретному епізоді.

Для отримання даних про мультсеріал була використана публічна API, в якості архітектурного скелету була використана мультирівнева архітектура з побудовою за принципами SOLID.

Програмний продукт був розроблений за допомогою візуального середовища Visual Studio Code, з використанням технології GIT, користуючись платформою Github для розміщення репозиторію.

Продукт може працювати на будь-якій платформі, яка має доступ до інтернету і встановлені пакети для роботи застосунків на технологіях .NET. Публічний доступ можливий з будь-якого пристрою, який має доступ в інтернет, веббраузер та утиліти подібні до Postman або Fiddler.

Програмний засіб був протестований та виявився працездатним, має велику швидкість роботи (~350мс).

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАНЬ

1. Анатомія web-сторінки [Електронний ресурс]. – Режим доступу: <https://training.qatestlab.com/blog/technical-articles/web-page-anatomy/>
2. Сервер [Електронний ресурс]. – Режим доступу: <https://uk.wikipedia.org/wiki/%D0%A1%D0%B5%D1%80%D0%B2%D0%B5%D1%80>
3. Що таке .NET? [Електронний ресурс]. – Режим доступу: <https://training.epam.ua/News/Items/301?lang=ua>
4. С#: що це за мова [Електронний ресурс]. – Режим доступу: <https://robotdreams.cc/uk/blog/284-s-cto-eto-za-yazyk-i-gde-ego-ispolzuyut>
5. С# [Електронний ресурс]. – Режим доступу: https://ru.wikipedia.org/wiki/C_Sharp.
6. Різниця між HTTP та HTTPS [Електронний ресурс]. – Режим доступу: <https://hostiq.ua/wiki/ukr/http-https/>
7. OOP [Електронний ресурс]. – Режим доступу: <https://disted.edu.vn.ua/courses/learn/4461>
8. Комп'ютерні мережі [Електронний ресурс]. – Режим доступу: <https://docplayer.net/68814675-Komp-yuterni-merezhi-internet.html>
9. ASP.NET чи PHP? [Електронний ресурс]. – Режим доступу: <https://wezom.com.ua/ua/blog/chto-vybrat-dlja-razrabotki-aspnet-ili-php>
10. Програмний засіб налаштування маршрутизаторів через веб-інтерфейс [Електронний ресурс]. – Режим доступу: http://dspace.wunu.edu.ua/bitstream/316497/39797/1/%D0%BA%D0%B0%D1%80%D0%BF%D0%B5%D1%86_%D0%94%D0%9F_2018.pdf
11. Контроллер [Електронний ресурс]. – Режим доступу: <https://www.yiiframework.com/doc/guide/1.1/uk/basics.controller>
12. Системи контролю версій на прикладі GIT [Електронний ресурс]. – Режим доступу: https://dut.edu.ua/ua/news-1-626-9170-sistemi-kontrolyu-versiy-na-prikladi-git_kafedra-kompyuternih-nauk-ta-informaciynih

tehnologiy

13. Репозиторій програмного забезпечення [Електронний ресурс]. – Режим доступу:

https://www.wikiwand.com/uk/%D0%A0%D0%B5%D0%BF%D0%BE%D0%B7%D0%B8%D1%82%D0%BE%D1%80%D1%96%D0%B9_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BD%D0%BE%D0%B3%D0%BE_%D0%B7%D0%B0%D0%B1%D0%B5%D0%B7%D0%BF%D0%B5%D1%87%D0%B5%D0%BD%D0%BD%D1%8F

14. Sisyphus [Електронний ресурс]. – Режим доступу: <https://docs.altlinux.org/ru-RU/archive/2.4/html-single/master/alt-docs-devel/ch01s03.html>

15. Сайт Рік і Морті [Електронний ресурс]. – Режим доступу: <https://rickandmortyapi.com/documentation/>

16. Вебдодаток і його характеристики [Електронний ресурс]. – Режим доступу: <https://www.centum-d.com/veb-dodatok-yogo-harakteristiki>.

17. Web-технології [Електронний ресурс]. – Режим доступу: <https://quizlet.com/738090601/%D0%A2%D0%B5%D0%BC%D0%B0-13-web-%D1%82%D0%B5%D1%85%D0%BD%D0%BE%D0%BB%D0%BE%D0%B3%D1%96%D1%97-%D0%9A%D0%BB%D1%96%D1%94%D0%BD%D1%82-%D1%81%D0%B5%D1%80%D0%B2%D0%B5%D1%80%D0%BD%D0%B0-%D0%B0%D1%80%D1%85%D1%96%D1%82%D0%B5%D0%BA%D1%82%D1%83%D1%80%D0%B0-flash-cards/>

18. Веб-додаток і його характеристика [Електронний ресурс]. – Режим доступу: <https://www.centum-d.com/veb-dodatok-yogo-harakteristiki/>

19. Що таке “Ruby”? [Електронний ресурс]. – Режим доступу: <https://virtuapc.ru/technology/chto-takoe-rubi-ruby-yazyk-programmirovaniya-rubi-yazyk-programmirovaniya-ruby-rubi-programmirovan/>

20. Java [Електронний ресурс]. – Режим доступу: <https://uk.wikipedia.org/wiki/Java>
21. Php [Електронний ресурс]. – Режим доступу: <https://uk.wikipedia.org/wiki/PHP>
22. Python [Електронний ресурс]. – Режим доступу: <https://uk.wikipedia.org/wiki/Python>
23. Чому JavaScript - перспективна мова програмування? [Електронний ресурс]. – Режим доступу: <https://dou.ua/forums/topic/35184/>
24. HTML [Електронний ресурс]. – Режим доступу: <https://ru.wikipedia.org/wiki/HTML>
25. Вибір інструментальних засобів розробки [Електронний ресурс]. – Режим доступу: http://ni.biz.ua/12/12_9/12_92943_vibor-instrumentalnih-sredstv-razrabotki.html
26. Документація ASP.NET [Електронний ресурс]. – Режим доступу: <https://learn.microsoft.com/ru-ru/aspnet/core/?view=aspnetcore-7.0>.
27. Архітектура програмного забезпечення [Електронний ресурс]. – Режим доступу: <https://wezom.com.ua/ua/blog/arhitektura-programmnogo-obespecheniya>

ДОДАТОК А
Технічне завдання

Вступ

Розроблюваний програмний продукт використовується для вирішення проблеми пошуку і перевірки коректності даних.

ПП забезпечує процес пошуку і актуалізації даних для медіаконтенту.

A.1 Мета розробки і її призначення

Об'єкт дослідження – програмні засоби для пошуку і обробки необхідних даних через доступ до них через і.

Предмет дослідження – Web API для взаємодії зі сторонніми джерелами даних.

Мета роботи – розробка Web API застосунку для взаємодії з 3rd party API для отримання і обробки даних з бази даних розважального медіаконтенту.

Актуальність роботи полягає в тому, що використання медіаконтенту постійно зростає і для роботи з великою їх кількістю необхідні засоби для швидкого і зручного доступу до конкретної інформації.

A.2 Підстави для розробки

Підставою для виконання розробки є завдання на дипломний проєкт на тему “Розробка вебзастосунку для обробки запитів на сайті фільмів та серіалів” затверджене наказом за Національним університетом «Запорізька політехніка» № 87 від 4 квітня 2023 року.

A.3 Основні вимоги до програми, що розробляється, початковим даним і результату

A.3.1 Вимоги до функціональних характеристик

Вимоги до функціональних характеристик:

- вміння отримувати POST та GET запитів;
- зручний та привабливий інтерфейс вебзасобу для користувачів;
- пошук повинен працювати швидше 1 секунди;
- час, який займе у користувача на перший запуск та пошук, не повинен перевищувати 20 секунд;
- вебзастосунок повинен працювати без збоїв.

A.3.2 Вимоги до надійності

ПП повинна виконувати наступні вимоги до надійності:

- захищати ПП від помилкових дій користувача (наприклад користувач ввів невалідні дані);
- обробляти виняткові ситуації (наприклад, відсутність підключення до бази даних).

ПП повинен нормально функціонувати при безперервній роботі комп'ютера. У разі виникнення збоїв в роботі апаратури, відновлення нормальної роботи ПП повинно проводитися після перезавантаження системи.

A.3.3 Умови експлуатації

Умови експлуатації:

- програмне забезпечення не потребує обслуговуючого персоналу;
- для користування програмним забезпеченням не потрібна додаткова кваліфікація;
- необхідна утиліта типу Fiddler або Postman для отримання повного функціоналу.

A.3.4 Вимоги до складу та параметрів технічних засобів

Вимоги до складу та параметрів технічних засобів:

- мінімум 2 ГБ оперативної пам'яті, рекомендується 4 ГБ оперативної пам'яті;
- вільне місце на диску 5 ГБ мінімум;
- необхідна утиліта типу Fiddler або Postman для отримання повного функціоналу;
- Windows, Linux, macOS.

A.3.5 Вимоги до маркування та упакування

Програмне забезпечення не потребує додаткових дій при маркуванні та упакуванні.

A.3.6 Вимоги до транспортування та збереження

Вимоги до транспортування й зберігання аналогічні вимогам, які пропонуються до носія, на якому записана програма.

A.4 Вимоги до програмної документації

Склад програмної документації повинен включати в себе:

- технічне завдання;
- керівництво програміста;
- керівництво оператора;
- опис програми.

A.5 Основні обмеження на установку і використання програмного продукту

Розроблюваний програмний продукт не потребує інсталяції. Програмний продукт повинен використовуватись на комп'ютерах, які

задовольняють вимогам та параметрам технічних засобів. Програмний продукт може використовуватись користувачем у будь-який момент часу, не конфліктує з іншими програмами.

ДОДАТОК Б
Текст файлів програм

Б.1 Файл PersonCheckerController.cs

```

using Microsoft.AspNetCore.Http;
using Microsoft.AspNetCore.Mvc;
using System.Threading.Tasks;
using System.Net.Http;
using System;
using System.Web;
using apiRnM.Interfaces;
using apiRnM.Utills;
using System.Xml.Linq;
using System.Net;

namespace apiRnM.Controllers
{
    [ApiController]
    [Route("api/v1")]
    public class PersonCheckerController : Controller
    {
        private IProceeeder _proceeeder;

        public PersonCheckerController(IProceeeder proceeeder)
        {
            _proceeeder = proceeeder;
        }

        [HttpGet]
        [Route("person")]
        public ActionResult<string> Person(string name)
        {
            if (name is null)
                return NotFound();

            JsonResult result = _proceeeder.GetPersonData(name.ToLower());

            return result;
        }

        [HttpPost]
        [Route("check-person")]
        public ActionResult<bool> CheckPerson([FromBody] RequestBodyCheckPersonEpisode body)

```

```

{
    if (body == null)
        return NotFound();

    bool found;
    var name = body.personName.ToLower();
    var episode = body.episodeName.ToLower();
    var result = _proceeder.CheckPersonInEpisode(name, episode, out found);

    if (!found)
        return NotFound();

    return result;
}
}

```

Б.2 Файл IApiCaller.cs

```

namespace apiRnM.Interfaces
{
    public interface IApiCaller
    {
        public string GetDataFromApi(string url);
    }
}

```

Б.3 Файл IProceeder.cs

```

using Microsoft.AspNetCore.Mvc;

namespace apiRnM.Interfaces
{
    public interface IProceeder
    {
        public bool CheckPersonInEpisode(string PersonName, string EpisodeName, out bool found);
        public JsonResult GetPersonData(string PersonName);
    }
}

```

Б.4 Файл IRnMApiCaller.cs

```
using Microsoft.Extensions.Options;
using System.Threading.Tasks;
using apiRnM.Models;
using System.Collections.Generic;

namespace apiRnM.Interfaces
{
    public interface IRnMApiCaller : IApiCaller
    {
        public List<Person> GetPersons(string name);
        public List<Episode> GetEpisodes(string episode);
        public List<Location> GetLocations(string location);
    }
}
```

Б.5 Файл Episode.cs

```
namespace apiRnM.Models
{
    public class Episode
    {
        public int id { get; set; }
        public string name { get; set; }
        public string air_date { get; set; }
        public string episode { get; set; }
        public string[] characters { get; set; }
        public string url { get; set; }
        public string created { get; set; }
    }
}
```

Б.6 Файл Location.cs

```
namespace apiRnM.Models
{
    public class Location
    {
        public int id;
        public string name;
```

```

        public string type;
        public string dimension;
        public string[] residents;
        public string url;
        public string created;
    }
}

```

Б.7 Файл [carNumber].js

```

namespace apiRnM.Models
{
    public class Origin
    {
        public string name { get; set; }
        public string type { get; set; }
        public string dimension { get; set; }
    }
}

```

Б.8 Файл Person.cs

```

namespace apiRnM.Models
{
    public class Person
    {
        public int id { get; set; }
        public string name { get; set; }
        public string status { get; set; }
        public string species { get; set; }
        public string type { get; set; }
        public string gender { get; set; }
        public object origin { get; set; }
        public object location { get; set; }
        public string image { get; set; }
        public string[] episode { get; set; }
        public string url { get; set; }
        public string created { get; set; }
    }
}

```

Б.9 Файл PersonResponse.cs

```

namespace apiRnM.Models

```

```
{
  public class PersonResponse
  {
    public string name { get; set; }
    public string species { get; set; }
    public string status { get; set; }
    public string gender { get; set; }
    public string type { get; set; }
    public Origin origin { get; set; }
  }
}
```

Б.10 Файл launchSettings.json

```
{
  "$schema": "http://json.schemastore.org/launchsettings.json",
  "iisSettings": {
    "windowsAuthentication": false,
    "anonymousAuthentication": true,
    "iisExpress": {
      "applicationUrl": "http://localhost:54220",
      "sslPort": 44381
    }
  },
  "profiles": {
    "IIS Express": {
      "commandName": "IISExpress",
      "launchBrowser": true,
      "launchUrl": "api/v1/person",
      "environmentVariables": {
        "ASPNETCORE_ENVIRONMENT": "Development"
      }
    },
    "apiRnM": {
      "commandName": "Project",
      "launchBrowser": true,
      "launchUrl": "api/v1/person?name=unity",
      "applicationUrl": "https://localhost:5001;http://localhost:5000",
      "environmentVariables": {
        "ASPNETCORE_ENVIRONMENT": "Development"
      }
    }
  }
}
```

```

    }
}
}

```

Б.11 Файл ReqResProceeder.cs

```

using apiRnM.Interfaces;
using apiRnM.Models;
using Microsoft.AspNetCore.Mvc;
using System.Collections.Generic;
using System.Linq;
using System.Text.Json;

namespace apiRnM.Services
{
    public class ReqResProceeder : IProceeder
    {
        private IRnMApiCaller _apiCaller;

        public ReqResProceeder(IRnMApiCaller apiCaller)
        {
            _apiCaller = apiCaller;
        }

        public bool CheckPersonInEpisode(string personName, string episodeName, out bool found)
        {
            var personList = _apiCaller.GetPersons(personName);
            var episode = _apiCaller.GetEpisodes(episodeName).First();
            found = true;

            if(!personList.Any() || episode == null)
            {
                found = false;
                return false;
            }
            else
            {
                var allEpisodesQuery = personList.Select(
                    person => person.episode.Select(
                        episodeUrl => episodeUrl.Split('/').Last()))
                    .Distinct();
            }
        }
    }
}

```

```

        foreach(var ep in allEpisodesQuery)
        {
            if (ep.Contains(episode.id.ToString()))
                return true;
        }
    }
    return false;
}

public JsonResult GetPersonData(string PersonName)
{
    var persons = _apiCaller.GetPersons(PersonName);
    List<Location> locations = null;
    List<PersonResponse> personResponse = null;

    foreach (var person in persons)
    {
        var or = person.origin.ToString();
        var jdoc = JsonDocument.Parse(or);

        var jel = jdoc.RootElement;

        var location = _apiCaller.GetDataFromApi(jel.GetProperty("url").ToString());

        if (location != null)
            locations.Add(JsonSerializer.Deserialize<List<Location>>(location).First());
    }

    for(int counter = 0; counter < persons.Count; counter++)
    {
        var origin = persons[counter].origin.ToString();
        var jelement = JsonDocument.Parse(origin).RootElement;
        var
    }

    var response = new JsonResult(persons);
    return response;
}
}
}

```

Б.12 Файл RichAndMortyApiCaller.cs

```

using apiRnM.Interfaces;

using apiRnM.Models;
using Microsoft.Extensions.Options;
using apiRnM.Utills;
using System.Linq;
using System.Net.Http;
using System.Text.Json;
using System.Xml.Linq;
using System;
using System.Net;
using System.Threading.Tasks;
using System.Collections.Generic;
using System.IO;
using System.Text;
using Microsoft.AspNetCore.Routing.Template;
using System.Runtime.InteropServices;

namespace apiRnM.Services
{
    public class RickAndMortyApiCaller : IRnMApiCaller
    {
        public RnMAppSettings RnMSettings;
        private HttpClient client = new HttpClient();

        string queryParamsName = "/?name=";

        public RickAndMortyApiCaller(IOptions<RnMAppSettings> RnMSettings)
        {
            this.RnMSettings = RnMSettings.Value;
        }

        public List<Person> GetPersons(string name)
        {
            var url = string.Concat(RnMSettings.characters, queryParamsName, name.ToLower());
            var personResult = new List<Person>();

            var characters = GetDataFromApi(url);

```

```

        if(characters != null)
            personResult = JsonSerializer.Deserialize<List<Person>>(characters);

        return personResult;
    }

    public List<Episode> GetEpisodes(string episode)
    {
        var url = string.Concat(RnMSettings.episodes, queryParamsName, episode.ToLower());
        var episodeResult = new List<Episode>();

        var episodes = GetDataFromApi(url);

        if (episodes != null)
            episodeResult = JsonSerializer.Deserialize<List<Episode>>(episodes);

        return episodeResult;
    }

    public List<Location> GetLocations(string location)
    {
        var url = string.Concat(RnMSettings.locations, queryParamsName, location.ToLower());
        var locationResult = new List<Location>();

        var locations = GetDataFromApi(url);

        if (locations != null)
            locationResult = JsonSerializer.Deserialize<List<Location>>(locations);

        return locationResult;
    }

    public string GetDataFromApi(string url)
    {
        try
        {
            var response = client.GetStringAsync(url).Result;

            var jdoc = JsonDocument.Parse(response);

            var jel = jdoc.RootElement;

```

```

        return jel.GetProperty("results").ToString();
    }
    catch
    {
        return null;
    }
}
}
}

```

Б.13 Файл RequestBodyCheckPersonEpisode.cs

```

namespace apiRnM.Utls
{
    public class RequestBodyCheckPersonEpisode
    {
        public string personName { get; set; }
        public string episodeName { get; set; }
    }
}

```

Б.14 Файл RnMAppSettings.cs

```

namespace apiRnM.Utls
{
    public class RnMAppSettings
    {
        public string characters { get; set; }
        public string episodes { get; set; }
        public string locations { get; set; }
    }
}

```

Б.15 Program.cs

```

using Microsoft.AspNetCore.Hosting;
using Microsoft.Extensions.Configuration;

```

```

using Microsoft.Extensions.Hosting;
using Microsoft.Extensions.Logging;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;

namespace apiRnM
{
    public class Program
    {
        public static void Main(string[] args)
        {
            CreateHostBuilder(args).Build().Run();
        }

        public static IHostBuilder CreateHostBuilder(string[] args) =>
            Host.CreateDefaultBuilder(args)
                .ConfigureWebHostDefaults(webBuilder =>
                {
                    webBuilder.UseStartup<Startup>();
                })
        }
    }
}

```

B.16 Startup.cs

```

using Microsoft.AspNetCore.Builder;
using Microsoft.AspNetCore.Hosting;
using Microsoft.AspNetCore.HttpsPolicy;
using Microsoft.AspNetCore.Mvc;
using Microsoft.Extensions.Configuration;
using Microsoft.Extensions.DependencyInjection;
using Microsoft.Extensions.Hosting;
using Microsoft.Extensions.Logging;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;
using apiRnM.Utills;
using apiRnM.Interfaces;

```

```

using apiRnM.Services;

namespace apiRnM
{
    public class Startup
    {
        public Startup(IConfiguration configuration)
        {
            Configuration = configuration;
        }

        public IConfiguration Configuration { get; }

        // This method gets called by the runtime. Use this method to add services to the container.
        public void ConfigureServices(IServiceCollection services)
        {
            services.AddControllers();

            var RnMSources = Configuration.GetSection("RnM_sources");
            services.Configure<RnMAppSettings>(RnMSources);

            services.AddScoped<IRnMApiCaller, RickAndMortyApiCaller>();

            services.AddScoped<IProceeder, ReqResProceeder>();
        }

        // This method gets called by the runtime. Use this method to configure the HTTP request pipeline.
        public void Configure(IApplicationBuilder app, IWebHostEnvironment env)
        {
            if (env.IsDevelopment())
            {
                app.UseDeveloperExceptionPage();
            }

            app.UseHttpsRedirection();

            app.UseRouting();

            app.UseAuthorization();

            app.UseEndpoints(endpoints =>
            {

```

```
        endpoints.MapControllers();
    });
}
}
}
```

ДОДАТОК В
Слайди презентації

Міністерство освіти і науки України
Національний університет "Запорізька політехніка"
Кафедра програмних засобів

Дипломна робота на тему "РОЗРОБКА
ВЕБЗАСТОСУНКУ ДЛЯ ОБРОБКИ ЗАПИТІВ НА
САЙТІ ФІЛЬМІВ ТА СЕРІАЛІВ. DEVELOPMENT OF
THE WEB APPLICATION TO PROCESS REQUEST
AT A MOVIE AND SERIES WEBSITE"

Група: КНТз-119
Студентка: Досаєва Галина Андріївна
Керівник: к.т.н. Казурова Аліна Євгеніївна

2023

Рисунок В.1 – Слайд № 1

Мета, об'єкт, предмет і завдання дослідження

Об'єкт дослідження – програмні засоби для пошуку і обробки необхідних даних через доступ до них через API.

Предмет дослідження – WEB API для взаємодії зі сторонніми джерелами даних.

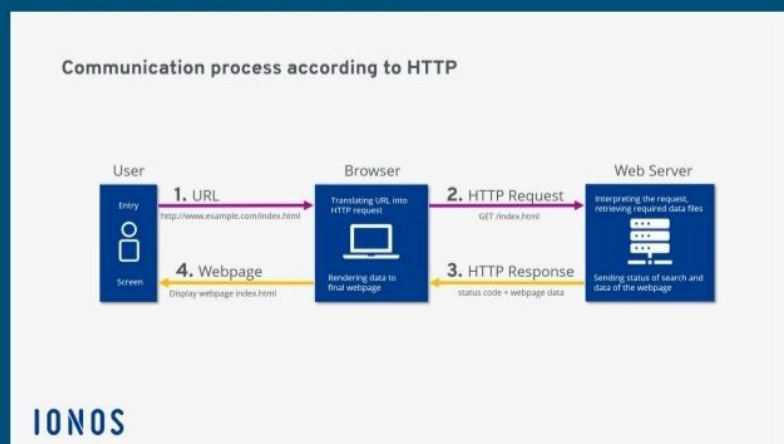
Мета роботи – розробка WEB API застосунку для взаємодії з 3rd party API для отримання і обробки даних з бази даних розважального медіаконтенту.

Актуальність роботи полягає в тому, що використання медіаконтенту постійно зростає і для роботи з великою їх кількістю необхідні засоби для швидкого і зручного доступу до конкретної інформації

2

Рисунок В.2 – Слайд № 2

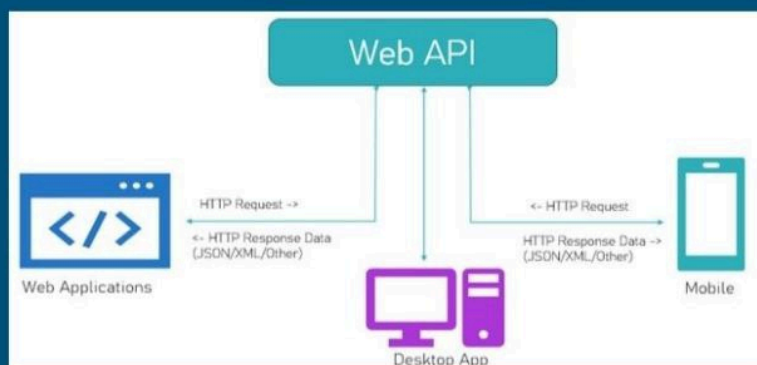
Наглядний приклад обміну даними від користувача до сервера по протоколу HTTP



3

Рисунок В.3 – Слайд № 3

Схема запитів до Web API



4

Рисунок В.4 – Слайд № 4

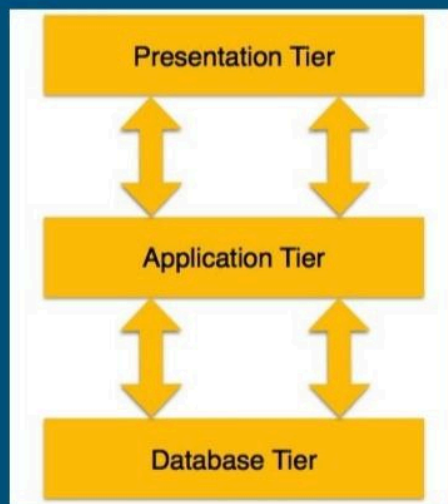
Порівняння мов програмування

Параметр	PHP	Node.js	C#
Сфера застосування	Веброзробка (сервер)	Веброзробка (сервер)	Веброзробка (сервер), мобільні та desktop додатки
Наявність CMS	Багато	Небагато	Небагато
Швидкодія	Висока	Середня	Висока
Вартість	Безкоштовний	Безкоштовний	Безкоштовний
Паралельна робота	Багатопоточність	Асинхронність одного потоку	Багатопоточність
SSR	+	+	+

5

Рисунок В.5 – Слайд № 5

Архітектура системи



Архітектура визначає як саме працюватиме застосунок, розподіл ролей, порядок обробки даних. Це спосіб структурувати елементи застосунку задля ефективної підтримки продукту.

6

Рисунок В.6 – Слайд № 6

Блок-схема роботи програми



Рисунок В.7 – Слайд № 7

Тестування програми

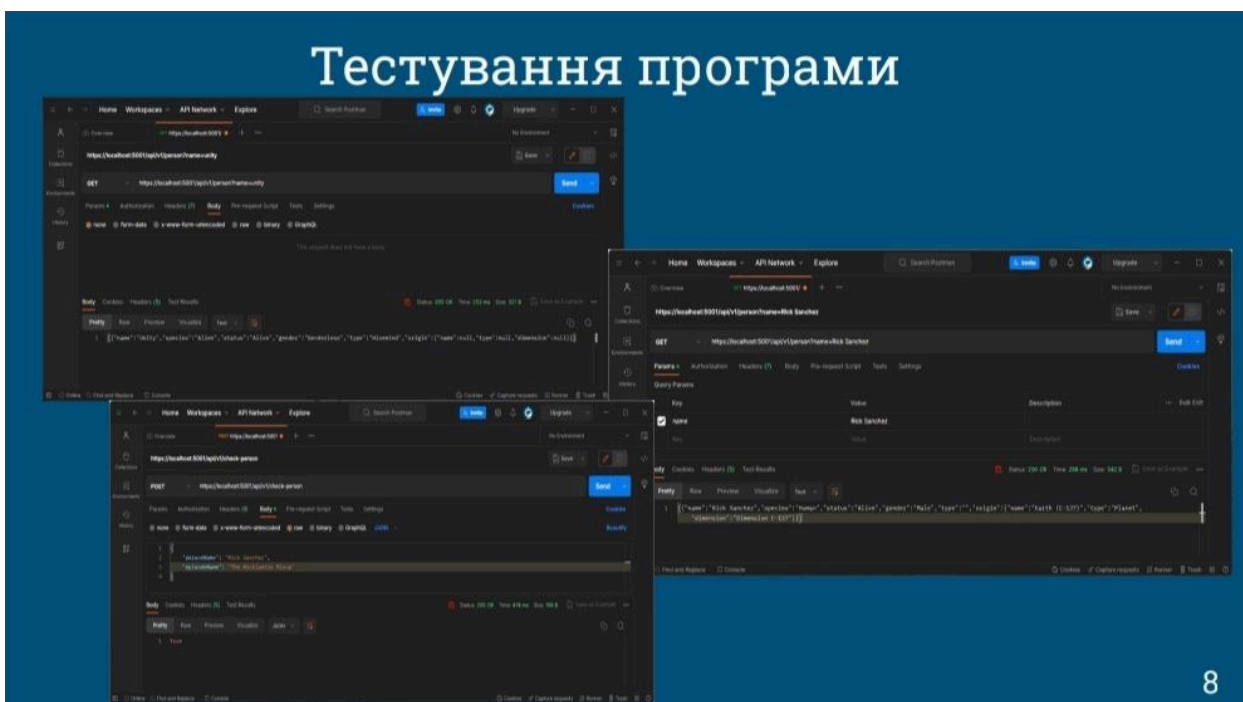


Рисунок В.8 – Слайд № 8

Висновок

Дана робота виконана за концепцією ООП та принципами SOLID з побудовою ієрархії класів та інтерфейсів, що дозволяє гнучко розширювати можливості продукту. Реалізовані класи для спілкування з іншими API з використаннями бібліотеки HttpClient, клас-обробник запитів, контроллер для взаємодії користувача з програмним продуктом.

У користувача є можливості формувати необхідні звіти по персонажам, отримувати інформацію про епізоди та локації, в яких був вперше помічений персонаж, є перевірка чи був той чи інший персонаж наявний в конкретному епізоді.

Для отримання даних про мультсеріал була використана публічна API, в якості архітектурного скелету була використана мультирівнева архітектура з побудовою за принципами SOLID.

Програмний продукт був розроблений за допомогою візуального середовища Visual Studio Code, з використанням технології GIT, користуючись платформою Github для розміщення репозиторію.

Продукт може працювати на будь-якій платформі, яка має доступ до інтернету і встановлені пакети для роботи застосунків на технологіях .NET. Публічний доступ можливий з будь-якого пристрою, який має доступ в інтернет, веббраузер та утиліти подібні до Postman або Fiddler.

Програмний засіб був протестований та виявився працездатним, має велику швидкість швидкість роботи (~350мс)

Рисунок В.9 – Слайд № 9