МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ЗАПОРІЗЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

МАТЕМАТИЧНИЙ ФАКУЛЬТЕТ
Кафедра програмної інженерії

КВАЛІФІКАЦІЙНА РОБОТА МАГІСТРА

на тему: **«РОЗРОБКА WEB-ЗАСТОСУНКУ КОНСОЛІДАЦІЇ ДАНИХ»**

Виконав: студент 2 курсу, групи 8.1219
спеціальності 121 інженерія програмного забезпечення
(шифр і назва напряму підготовки)
освітньої програми інженерія програмного забезпечення
(назва освітньої програми)
Є.О. Кандиба
(ініціали та прізвище)

Керівник декан математичного факультету, професор, д.т.н. Гоменюк С.І.
(посада, вчене звання, науковий ступінь, прізвище та ініціали)

Рецензент завідувач кафедри фундаментальної математики, доцент, д.т.н. Гребенюк. С.М.
(посада, вчене звання, науковий ступінь, прізвище та ініціали)

Запоріжжя – 2020

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ЗАПОРІЗЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

Факультет математичний

Кафедра програмної інженерії

Рівень вищої освіти магістр

Спеціальність 121 інженерія програмного забезпечення

Освітня програма інженерія програмного забезпечення

ЗАТВЕРДЖУЮ

Завідувач кафедри програмної інженерії, к.ф.-м.н., доцент

_____ А.О. Лісняк
(підпис)

« 20 » травня 2020 р.

З А В Д А Н Н Я
НА КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТОВІ(СТУДЕНТЦІ)

Кандибі Єгору Олеговичу
(прізвище, ім'я та по-батькові)

1. Тема роботи (проекту) Розробка web-застосунку консолідації даних

керівник роботи (проекту) Гоменюк Сергій Іванович, д.т.н., професор
(прізвище, ім'я та по-батькові, науковий ступінь, вчене звання)

затверджені наказом ЗНУ від « 20 » травня 2020 року № 576 - с

2. Строк подання студентом роботи 30.11.2020

3. Вихідні дані до роботи 1. Постановка задачі.
2. Перелік літератури.

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)
1. Постановка задачі.
2. Основні теоретичні відомості.
3. Реалізація програмного забезпечення.

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень)
Презентація

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв

7. Дата видачі завдання 20.05.2020

КАЛЕНДАРНИЙ ПЛАН

№	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1.	Розробка плану роботи.	04.06.2020	
2	Збір вихідних даних.	15.06.2020	
3.	Обробка методичних та теоретичних	30.06.2020	
	джерел.		
4.	Розробка першого розділу.	02.08.2020	
5.	Розробка другого розділу.	10.10.2020	
6.	Розробка третього розділу.	22.11.2020	
7.	Оформлення та нормоконтроль	30.11.2020	
	кваліфікаційної роботи		
8.	Захист кваліфікаційної роботи.	16.12.2020	

Студент _____ Є.О. Кандиба
(підпис) (ініціали та прізвище)

Керівник роботи _____ С.І. Гоменюк
(підпис) (ініціали та прізвище)

Нормоконтроль пройдено

Нормоконтролер _____ О.В. Кудін
(підпис) (ініціали та прізвище)

РЕФЕРАТ

Кваліфікаційна робота магістра «Розробка web-застосунку консолідації даних»: 63 с., 26 рис., 4 табл., 19 джерел, 1 додатки.

КОНСОЛІДАЦІЯ ДАНИХ, СИНТАКСИЧНИЙ АНАЛІЗ, СТРУКТУРУВАННЯ ДАНИХ, СТАТИЧНИЙ АНАЛІЗ КОДУ, LARAVEL FRAMEWORK.

Об’єкт дослідження – програмна реалізація консолідації зовнішніх даних.

Мета роботи: створення програмного засобу синтаксичного аналізу об’єктної моделі документу вебсайту та збір даних з нього.

Матеріали, методи та технічні засоби: структурне та об’єктно-орієнтоване програмування, фреймворк Laravel, мова програмування PHP, персональний комп’ютер з процесором i3-5600U під управлінням операційної системи Microsoft Windows 10.

Галузь використання – публікація інформації на сторінках соціальних мереж, створення програмного інтерфейсу для загального користування, створення боту для telegram для авто-інформування про вихід нових серій.

SUMMARY

Master’s Qualification Thesis «Development of web-application of data consolidation»: 63 pages, 26 figures, 4 tables, 19 references, 1 supplements.

SYNTAX ANALYSIS, DATA CONSOLIDATION, DATA STRUCTURING, STATIC CODE ANALYSIS, LARAVEL FRAMEWORK..

The object of the study is software implementation of syntactic development of external data.

The aim of the study is to create an algorithm for syntax analysis of DOM web site and to collect data from it.

Materials, methods, and tools: structural and object–oriented programming, framework Laravel, PHP programming languages, a personal computer running the i3–5600U processor running Microsoft Windows 10.

Field of use – publishing information on social media pages, creating a software interface for public use, creating a Telegram Bot for auto–informing about the release of new series.

ЗМІСТ

ПЕРЕЛІК СКОРОЧЕНЬ ТА УМОВНИХ ПОЗНАК

API – Application Programming Interface;

DOM – Document Object Model;

DRY – Don’t Repeat Yourself;

IDE – Integrated Development Environment,

JSX – Розширення Мови JavaScript;

KISS – Keep It Simpe And Straightforward;

LAMP – Linux–Apache–MySql–PHP;

MVC – Model–View–Controller;

ORM – Object–Relational Mapping;

PHP – Hypertext Preprocessor;

SMTP – Simple Mail Transfer Protocol;

SQL – Structured Query Language;

URL – Uniform Resource Locator;

ВСТУП

Збір даних – це процес, який забирає в нас достатню кількість часу, який можна було використати з користтю. В цьому нам допомогає синтаксичний аналіз даних або парсинг.

На сьогоднішний час, парсинги використовуються в пошукових системах, щоб швидше шукати сайти для користувача. Найпопулярніші пошукові системи такі як: Google, Yahoo, Bing тощо.

Мета даної кваліфікаційної роботи магістра – створення програмного інтерфейсу для парсингу даних та збору інформації про серіали, таких як: жанр, назва, статус, опис, кількість сезонів, оцінка.

Планується, що результатом роботи буде прикладний програмний інтерфейс для використання іншими розробниками. Програмний інтерфейс буде віддавати асоціативний масив даних. У цьому сервісі буде: збір даних за сторінками серіалу, збір даних за назвами серіалу, збір даних за жанром серіалу, збір даних за тегами серіалу.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

Сьогодні, оновлення інформації відбувається дуже швидко. Обробляти власноруч складно, і це займає багато часу, можна втратити важливе. Саме тому створені спеціальні програми – парсери, які в автоматичному режимі аналізують та збирають дані, що цікавлять. Вони, справляються з величезними обсягами, безперервно оновлюють значенння.

1.1 Аналіз існуючої проблеми

Парсинг – це процес збору даних, з подальшою їх обробкою та аналізом. До цього способу вдаються, коли має бути оброблено великий масив інформації, з яким складно впоратися власноруч. Програма, яка проводить збір та синтаксичний аналіз – це парсер. З її допомогою, можна спростити роботу з пошуку контенту для власного ресурсу, і проводити її в стислі терміни [2].

Парсинг – дозволяє здійснювати роботу, з даними будь-якої тематики. Серед основних сфер застосування такої технології можна виділити: пошук та наповнення ресурсів текстовим та мультимедійним контентом;

З точки зору пошукових систем, найоптимальніший варіант заповнення сайту – використовувати всі можливі види інтернет-контенту, належного рівня, щоб максимально широко розкрити масивні питання в пошуковий рядок [2].

Види контенту, формуються за способом подачі інформації:

– текстовий – найпоширеніший спосіб відображення даних (у вигляді тексту); індексується, набагато простіше і безпроблемно, ніж інші види інтернет-контенту. Має найбільший вплив, на ранжирування результатів пошукової видачі [2].

– мультимедійний – інформаційний вміст вебресурсів, який найчастіше доповнює основні види текстового контенту і сприймається користувачем візуально і / або на слух. Робить набагато менший, безпосередній вплив на

рейтинг сайту, з точки зору пошукових систем, але здатний істотно вплинути на поведінкові фактори і конверсію [2].

Можна виділити наступні види текстового контенту: динамічний і статичний контент.

Текстовий статичний контент – вся текстова інформація, розміщена власником сайту (адміністратором, копірайтером та ін.) На сторінках вебресурсу на невизначений період часу [2].

Це контент сайту, види якого широко використовуються, для заповнення сторінок і грають важливу роль в плані їх релевантності ключових запитах, за якими приходять відвідувачі з пошукових систем.

Сюди відносяться такі типи контенту, як:

– статті;

– новини;

– опису розділів, категорій, товарів, послуг, акцій.

Ці види текстового контенту (за умови оптимізації), є основним інструментом залучення відвідувачів з пошукових систем, абсолютно на будь-який сайт.

Текстовий динамічний контент – це вміст сайту, що з'являється з подачі користувачів та відвідувачів [2].

До динамічному контенту можна віднести такі типи контенту, як:

– відгуки;

– коментарі;

– обговорення.

Подібні види контенту інтернет-ресурсів, можуть успішно доповнити основний матеріал, розміщений на сторінках, посиливши їх релевантність, унікальність, корисність і цінність. Їх присутність, бажанна на сторінках практично будь-яких вебресурсів (від блогів до інтернет-магазинів) [2].

Також можно виділить і такі види мультимедійного контенту:

– зображення: використовуються щоб «урізноманітнити» основні види інтернет-контенту (текстові), і доповнити їх, в плані змістового навантаження

(рис. 1.1). Дають можливість користувачу уявити, про що саме йде мова в тексті [2];

– графічні типи контенту (фото, зображень, графіки, схеми) при правильній оптимізації, (коректні назви файлів і параметри title і alt) активно індексуються роботами пошукових систем, і дають такі необхідні додаткові пункти рейтингу. Використовуються практично на всіх сайтах [2].

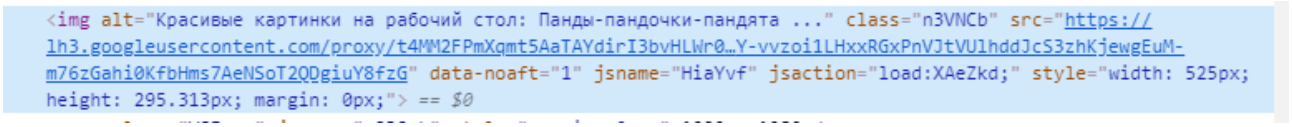


Рисунок 1.1 – Приклад зображень у HTML

– відеоконтент: використовується значно рідше, ніж текстові та графічні види контенту інтернет-ресурсів, на увазі більш складного процесу створення і проблем з індексацією пошуковими системами [2].

Доречний на таких сторінках:

– блогів як вид додаткової інформації до основного матеріалу;

– сайтів, що пропонують певні послуги, у вигляді відображення процесу їх виконання і відгуків задоволених клієнтів;

– інтернет-магазинів (відеоогляди товару);

Частіше за все, це фрейми від YouTube. Але в HTML5 є тег (рис. 1.2) для виведення відео, та він є захищеним від парсингу.

Рисунок 1.2 – Приклад відеоконтенту у HTML

– 3D моделі і анімація: порівняно рідко використовуваний контент сайту, види якого, найбільше доречні на сторінках всіляких інтернет-магазинів. Дозволяють більш детально розглянути товар перед покупкою, і значно впливають на прийняття цього рішення [2].

– аудіо – не менш популярний, ніж інші види контенту в соціальних мережах, і на музичних порталах (рис. 1.3). Але на інших типах сайтів, використовується рідко. Практично ніяк не впливає на просування, хіба що побічно через поведінкові чинники [2].

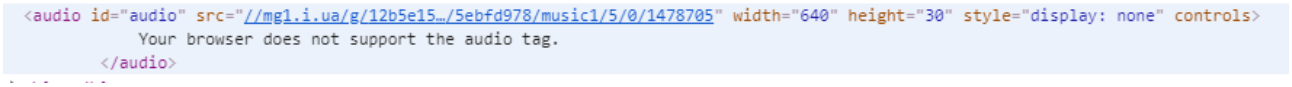


Рисунок 1.3 – Приклад аудіо контенту у HTML

Комбінуючи, всі перераховані вище види мультимедійного контенту (за умови доречності їх використання), можна значно вплинути на поведінку відвідувачів сайту: зменшити їх кількість, відлякуючи невмілим використанням (перенасичення сайту картинками і автоматично запускає відео) або ж, навпаки, збільшити приріст трафіку, частоту перетворення відвідувача в покупця та ін. [2].

1.2 Огляд існуючих рішень

На данний момент, існує величезна кількість програмних рішень парсингів. Ці рішення виконують дуже важливі задачі у всесвітній павутині, та в нашому повсякденному житті.

Це такі рішення як: пошукові боти, валідатори патернів у проекті, програми для перевірки унікальності текстів наукових робіт.

1.2.1 Пошуковий бот

Пошуковий робот – це програма, для сканування та індексації сайтів. Він дозволяє пошуковій системі, отримати відомості про вебсторінки, і внести їх в базу для подальшої видачі користувачам при запиті. Боти, не аналізують зібрані дані, а тільки передають їх на сервери пошукових систем [3].

Принцип роботи пошукового бота. Пошукова видача формується в три етапи:

– сканування – збір всіх даних з вебсторінок ботами, включаючи тексти, картинки і відеоматеріали. Даний процес, відбувається регулярно з урахуванням частоти, оновлень ресурсу;

– індексація – внесення зібраної інформації в базу даних пошукових систем, з присвоєнням певного індексу для швидкого пошуку. На великих новинних порталах контент індексується практично відразу після публікації;

– видача результатів – пошук інформації за індексом, і ранжування сторінок з урахуванням релевантності запиту.

Іноді, процес індексації сторінок відбувається навіть, без їх попереднього сканування (рис. 1.4).

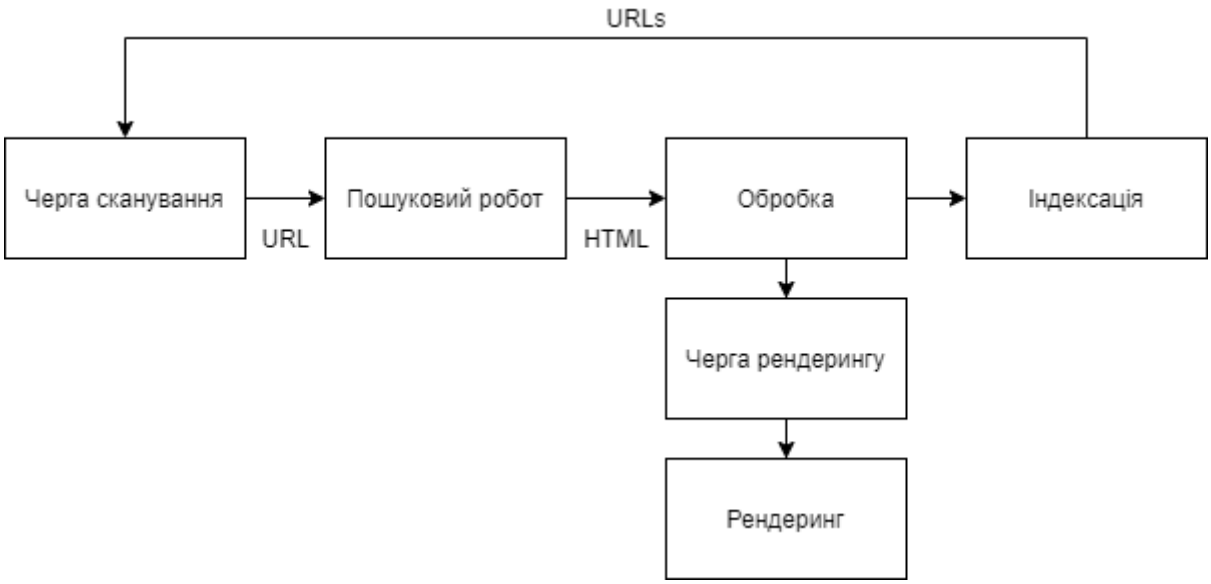


Рисунок 1.4 – Схема роботи пошукового бота

1.2.2 ESLint

ESLint – це інструмент статичного аналізу коду, для виявлення проблемних шаблонів, виявлених в коді JavaScript (рис. 1.5). Він був створений, Ніколасом С. Закасом в 2013 році. Правила в ESLint можна налаштовувати, а

правила, що налаштовуються, можна визначати та завантажувати. ESLint охоплює як якість коду, так і питання стилю кодування. ESLint підтримує поточні стандарти ECMAScript, і експериментальний синтаксис з чернеток для майбутніх стандартів. Код, з використанням JSX або TypeScript також може бути оброблений, коли використовується плагін або транспортер [4].

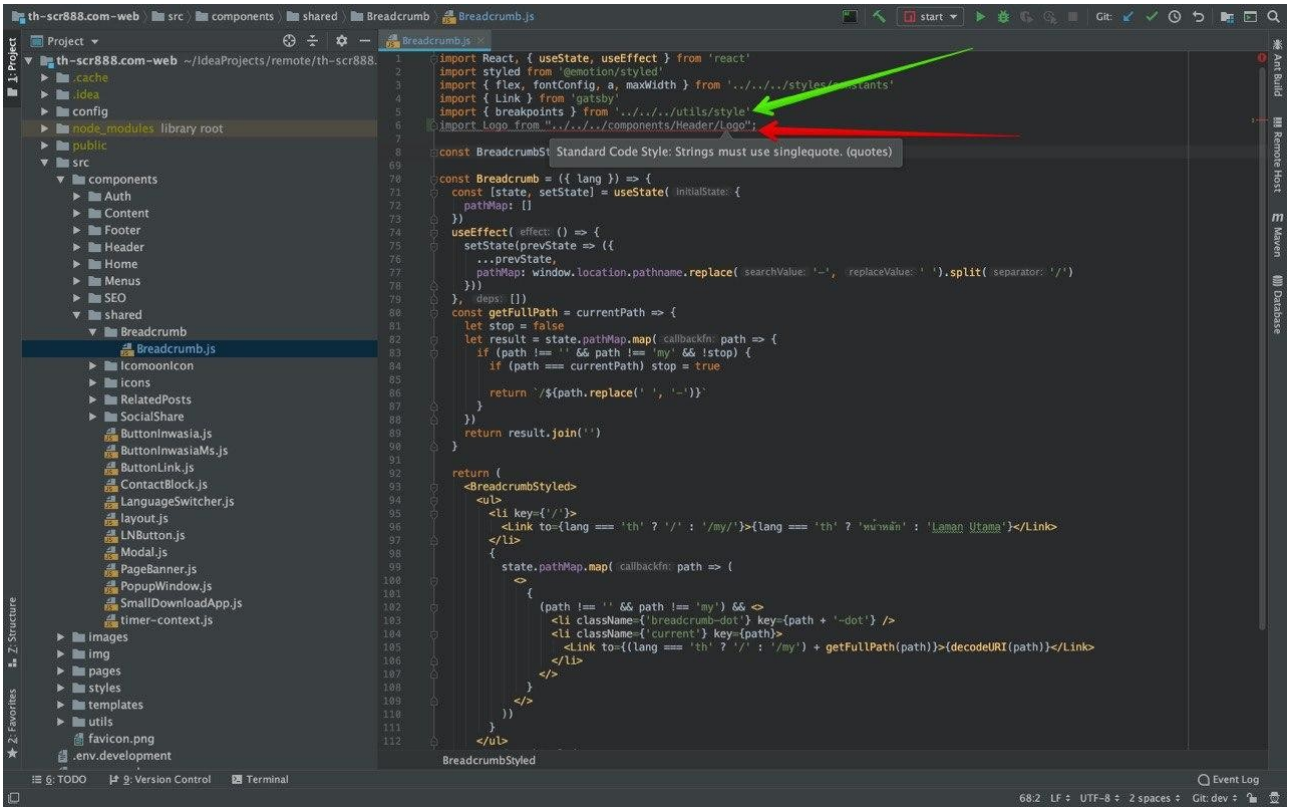


Рисунок 1.5 – Приклад роботи ESlint у коді

1.2.3 Сервіси для перевірки на унікальність

Існує кілька різних методів, але всі вони базуються на пошуку певних фраз в пошукових машинах. Надалі, у кожного сервісу свій алгоритм обробки, одержуваних з пошукових систем результатів, але так чи інакше, всі розбивають текст на фрази, шматки та тощо [5].

Кожна программа, працює по-своєму. Одні, отримують фрази з кількох слів, що йдуть один за одним, інші вилучають фрази з тексту випадково, або беруть пересічні фрази. Варіантів куди більше. Чим більше текст, тим більше

частин, які потрібно перевірити. Це перша і необхідна стадія, без неї неможливо [5].

Коли результат отриманий, пошуковики знайшли подібні фрагменти, тоді програма забирає ці тексти до себе, і обробляє за своїми алгоритмами, вираховує відсоток неунікальності, розповідає про рерайті і так далі [5].

Давайте, візьмемо текст з 1500 символів і припустимо, що вийде близько 250 слів. Розіб'ємо текст на шматки, кожен з яких буде містити 5 слів. Якщо ми, будемо брати фрази без перетинів, у нас вийде 50 фраз. А якщо зробити перетин, в 1-2 слова, то відповідно кількість фраз збільшиться в рази [5].

Тепер, для того щоб досконально перевірити текст на унікальність, нам слід перевірити всі ці фрази, таким чином нам потрібно зробити 50 запитів до пошукової системи [5].

1.2.4 Import.io сервіс для парсингу SaaS платформ

Import.io – платформа інтеграції вебданих SaaS, яка дозволяє людям перетворювати неструктуровані вебдані в структурований формат шляхом вилучення, підготовки та інтеграції вебданих для споживання в аналітичні платформи або використовуваних у бізнесі, продажах або маркетингових додатках [6].

Import.io забезпечує візуальне середовище для автоматизації робочого процесу вилучення та перетворення вебданих. Після вказівки цільової URL–адреси вебсайту модуль вилучення вебданих забезпечує візуальне середовище для проектування автоматизованих робочих процесів для збирання даних, виходячи за межі HTML – синтаксичного аналізу статичного вмісту для автоматизації взаємодії кінцевих користувачів, отримуючи дані, які в іншому випадку не будуть видно відразу [6].

Після вилучення програмне забезпечення надає повні можливості підготовки даних, які використовуються для гармонізації та очищення вебданих і пропонує бібліотеку функцій, подібних до електронних таблиць, що

дозволяють кінцевому користувачеві створювати власні формули, які можна використовувати для збагачення набору даних [6].

Для споживання результатів Import.io пропонує кілька варіантів. Він має власний модуль візуалізації та інформаційної панелі, який допомагає бізнес-аналітикам отримати уявлення, які їм потрібні, а також надає API, що пропонує повний доступ до всього, що можна зробити на їх платформі, що дозволяє інтегрувати вебдані безпосередньо у власні програми [6].

Import.io має ряд особливостей [6]:

– автовидобуток – автоматичне вилучення даних із вебсторінок у структурований набір даних;

– extractor builder – розробка власних екстракторів;

– аутентифікація – видобування даних за логіном / паролем;

– планувальник – для планування видобування за потребою;

– звітність – звіти з результатами видобування даних;

– інтернет-магазин даних – використання платформи SaaS для зберігання вилучених даних;

– пропускна здатність – швидке, паралельне збирання даних, що поширюється автоматично за допомогою масштабованої хмарної архітектури;

– час роботи – висока доступність для високого використання гучності.

1.2.5 Web Scraper – розширеня для браузеру

В сучастному світі є все для швидкого збору інформації та тільки мало хто знає про ці можливості. Web scraper – це розширеня для браузеру яке справляється з цілим рядом задач.

За допомогою цього розширення можна створити план (мапу сайту) про те, як пройти вебсайт і що потрібно витягти. Використовуючи ці мапи сайту Web scrape перейде по сайту відповідно та витягне всі дані. Видобувані дані пізніше можна експортувати як CSV [7]. Можна виділи наступні особливості [7]:

– вибір по сторінкам;

– скорочені дані зберігаються в локальному сховищі;

– кілька типів вибору даних;

– видобування даних з динамічних сторінок (JavaScript + AJAX);

– огляд видобуваних даних;

– експорт видобуваних даних у форматі CSV;

– імпорт, експорт мапи сайту.

1.2.6 Content Downloader X1 – програма для парсингу

Парсер Content Downloader X1 – це потужний, універсальний і професійних парсер контента. За словами розробника, Content Downloader створений, «щоб парсити будь-яку потрібну інформацію з будь-яких сайтів, програма виконає видобування даних де це взагалі можливо» [8].

Парсер Content Downloader використовують, щоб видобути [8]:

– товари з інтернет-магазинів у CSV таблиці (стовбці у таблицях налаштовуються під потрібні параметри);

– тексти, статті, описи;

– фотографії, картинки, зображення на комп’ютер;

– файли (наприклад, парсинг flash-ігор, рефератів, торрентів);

– контактні дані: e-mail, телефонні номери, адреси, Skype та інші (збереження отриманих даних у TXT, або HTML, або CSV формати на вибір);

– приховану (hidden) інформацію, що візуально відображається тільки після виконання дії (наприклад, натискання на кнопку «показати номер» або «показати контактні дані» – ця функція доступна тільки у ліцензії ULTIMATE);

– дані, які доступні тільки після авторизації користувача на сайті;

– посилання з сайту по заданим параметрам та фільтрам;

– окремі частини коду сторінки вебдокумента (збереження даних у потрібному форматі);

– XML-карту сайту.

Даний програмний засіб є платним але він використовує сучасні методи аналізу та збору даних, що робить его дуже швидким та зручним в використанні.

1.4 Висновки до першого розділу

Було проаналізовано види контенту на вебсторінках: текстовий та мультимедійний, та теги в яких він міститься. Також було виконано огляд та розглянуто можливості програмних засобів синтаксичного аналізу даних або програм- парсерів.

2 РОЗРОБКА АРХІТЕКТУРИ ПРОГРАМИ

Метою розробки – є створення парсеру для аналізу та збору даних з сервісу серіалів та передача вилобуваних даних через прикладний програмний інтерфейс. Ідея полягає в тому, щоб інші розробники не марнували час для збору потрібних даних та використовували прикладний програмний інтерфейс у розробці.

2.1 Огляд особливостей мов програмування

На даний час для парсингу найбільш розповсюджені наступні мови програмування: PHP, Python та Ruby.

PHP (рекурсивний акронім словосполучення PHP: Hypertext Preprocessor) – це поширена мова програмування, загального призначення з відкритим вихідним кодом. PHP спеціально сконструйований для веброзробок, та його код може впроваджуватися безпосередньо в HTML [9].

```
<html>
 <head>
  <title>Тестируем PHP</title>
 </head>
 <body>
 <?php echo '<p>Привет, мир!</p>'; ?>
 </body>
</html>
```

Рисунок 2.1 – Приклад PHP коду в HTML сторінці

Замість рутинного виводу HTML-коди командних мов (як це відбувається, наприклад, в Perl або C), скрипт PHP міститься в HTML (рис. 2.1)

з вбудованним кодом (у нашому випадку, це виходить текст "Привітання, я – скрипт PHP!"). Код PHP відокремлюється спеціальними початковими, і кінцевими тегами <? PHP ?>, які підтримують, "переключення" в "PHP-режим" і виходять з нього [9].

PHP відрізняється від JavaScript тим, що PHP-скрипти виконуються на серверах (рис. 2.2) та генерують HTML, який надається клієнту. Якщо на серверах був розміщений скрипт, то клієнт може отримати лише результат виконання скрипту, без розуміння та бачення тексту самого скрипту. Можна налаштувати свій власний сервер таким чином, щоб мати звичайні HTML-файли, які обробляє процесор PHP, так що клієнти, навіть не зможуть дізнатися, отримують чи вони звичайний HTML-файл або результати виконання сценаріїв [9].

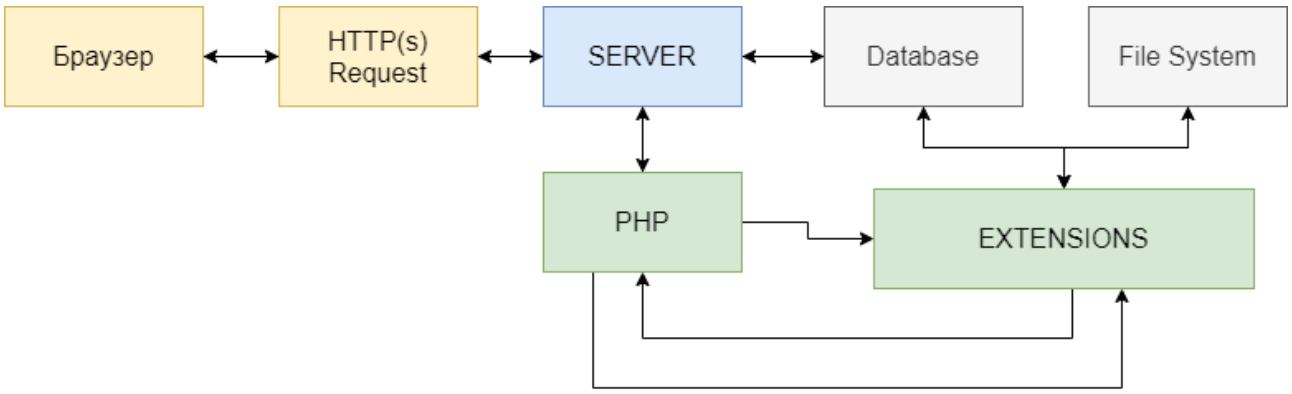


Рисунок 2.2 – Принцип роботи вебзастосунків на мові PHP [9]

Мова програмуваня PHP – це дуже популярна для розробки вебсайтів. Вона витримує дуже велике навантаження та може виконувати швидку обробку даних. Це робить її кращою для створеня складних інтернет структур таких як: інтернет-магазин, форуми та ін.

На мові PHP написано багато популярних фреймворків та бібліотек для шкидкого створеня вебрішень.

Мова програмуваня Python, як мова для парсингу. Python – високорівнева мова програмування загального призначення, орієнтована на

підвищення продуктивності розробника та читання коду. Синтаксис ядра Python мінімалістичний. У той же час стандартна бібліотека включає великий обсяг корисних функцій [10].

Для Python є дуже популярний фреймворк для парсингу – Scrapy. Scrapy використовує Spiders або автономних сканерів з певним набором інструкцій. За допомогою фреймворка легко розробити навіть великі проекти для скрапінга, так щоб і інші розробники могли використовувати цей код. Scrapy з легкістю виконує велику за обсягом роботу. Він може сканувати кілька посилань одночасно менш ніж за хвилину в залежності від загальної кількості. Це відбувається плавно завдяки Twister, який працює асинхронно (без блокування) [11].

Як висновок можна сказати, що Scrapy підходить для написання великих проєктів для парсингу. Він є дуже швидким та зручним, якщо з ним бути добре ознайомленим.

Ruby – динамічна, рефлективна, високорівнева мова програмування, що інтерпретується. Мова має незалежну від операційної системи реалізацію багатопоточності, сильну динамічну типізацію, складальник сміття та багатьо інші можливості. За особливостями синтаксису він близький до мов Perl і Eiffel, за об'єктно-орієнтованим підходом – до Smalltalk. Також деякі риси мови взяті з Python, Lisp, Dylan і Клу [12].

Для парсинга в Ruby частіш за все використовують nokogiri. Nokogiri це програмна бібліотека з відкритим вихідним кодом для аналізу HTML і XML в Ruby.

В таблиці 2.1 наведена порівняльна характеристика мов програмування

Таблиця 2.1 – Порівняння характеристика мов програмування

Характеристика	Ruby	PHP	Python
1	2	3	4
Типизація	Сильна	Слабка, неявна	Сильна, явна

Продовження табл.2.1

1	2	3	4
Об’єктно орієнтований стиль	наявний	наявний	наявний
Класи	наявне одиночне спадкування	наявне одиночне спадкування, але є trait	наявне множинне спадкування
Інтерфейси	наявне множинне спадкування	наявне множинне спадкування	наявне множинне спадкування
Багатопоточніть	наяна	відсутня	наявна
Суб’єктивна швидкість(оцінка по 10 бальній шкалі)	8	8	8

Було обрано мову програмуваня PHP тому, що вона є популярною для написання вебзастосунків та завдяки підтримці асинхроних запитів дана мова програмування краще всього підходить для збору даних з вебзастосунків.

Парсер можна написати «з нуля», використовуючи архітектурний шаблон проектування Model-View-Controller (MVC), або використовуючі бібліотеки або фреймворки. Laravel – найпопулярніший фреймворк, що використовує мову програмування PHP.

2.2 Огляд можливостей фреймворку Laravel

Laravel – безкоштовний вебфреймворк з відкритим кодом, призначений для розробки з використанням архітектурний шаблон проектування MVC. Laravel випущений під ліцензією MIT [13].

Фреймворк Laravel має ряд переваг, завдяки яким він є дуже зручним та популярним серед розробників.

Наведемо основні переваги даного фреймворку:

– високий рівень безпеки: перша загроза пов’язана, з передачею деструктивних SQL запитів. Laravel виключає цю можливість тому що він використовує ORM, яка виключає можливість “сирих” запитів; друга загроза пов’язана, з впровадженням JavaScript коду на довільну сторінку сайту. Завдяки цьому, зловмисник може зробити підміну даних, та викрасти їх у своїх цілях;

– підвищена продуктивність;

– кешування: ще одна з особливостей цього фреймворку – швидке завантаження, вебсторінок. Це досягнення noSQL баз даних таких як: Redis, MongoDB та ін. Принцип роботи цих баз в тому, що вони зберігають дані, в оперативній пам’яті;

– аутентифікація: Laravel, надає досить просунуту систему аутентифікації користувачів: як форми, так і соціальні мережі, використовуючи механізми OAuth2;

– використання шаблонізатору Blade (рис. 2.3).

```
@extends('layouts.app')
@section('content')

    <div class="main noMargin" id="domain-search">
        <div class="domain-wizard">
            <div class="container">

                @include('partials.domain_search.search_module')

                <div class="domain-wizard-hero domain-wizard-hero--success">
                    <div class="grid">
                        <div class="grid__col grid__col--md-10 inner-page-result">

                            @include('partials.domain_search.inner_page')

                        </div>
                        <div class="grid__col grid__col--md-2 domain-wizard-mobile-column
                            desktop_only">
                            @component('components.toggle_2',['class'=>'','checked'=>$vat])
                                @lang('global.with_vat')
                            @endcomponent
                                @component('components.button',[
                                  "type" => "button--color-green shopping-cart"
                                ])
                                    @lang('global.to_cart')
                                @endcomponent
                                @component('modules.helper_for_domain_search',[
                                    'domain' => $domain,
                                    'tld' => $tld,
                                    'additionalTlds' => $additionalTlds
                                ])
```

Рисунок 2.3 – Приклад написання тексту програми з використанням шаблону Blade

Завдяки цьому шаблонізатору, можна поділити HTML код на компоненти, або використання компонентного підходу в програмуванні. Це веде до оптимізації коду та можливості повторного використання цього коду;

– міграції баз даних дозволяють швидко переносити, і розгортати бази даних на серверах без втрат даних при розробці проекту локально, це скорочує час перенесення проекту на сервер замовника;

– MVC-архітектура: завдяки дотримання архітектури MVC досягається чіткий поділ між наступними абстрактними шарами програми (рис. 2.4):

– моделлю;

– контролером;

– поданням.

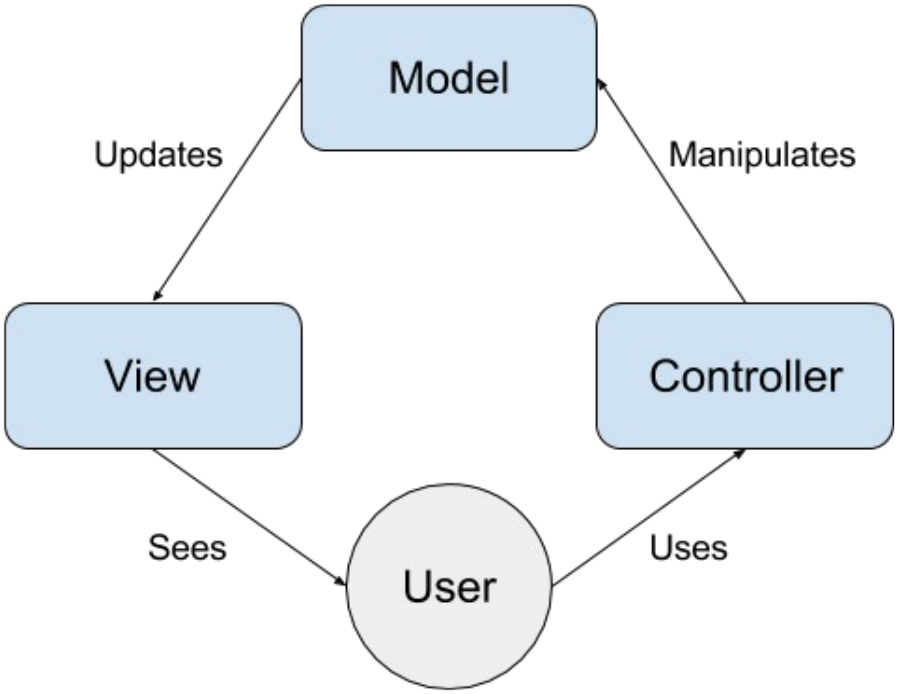


Рисунок 2.4 – Схематичне зображення роботи вебзастосунку з використанням архітектурного шаблону проектування MVC [14]

Всім відомо, що слабкий зв'язок між компонентами системи забезпечує стабільність роботи коду, і різко зменшує ймовірність регрессій. Всі ми стикались з неприємністю, коли виправлення одних багів в логіці, ламають

старі напрацювання, і призводять до ще більшого числа помилок, в багатьох місцях;

– наявність об'єктно-орієнтовани бібліотек: Laravel, поставляється з великою кількістю встановлених бібліотек. Одна з яких – система аутентифікації.

Можна виділити наступні недоліки фреймворку.

– наявність синтаксичного "цукору", що має як позитивні сторони, так може містити і негативні; дуже легко звикнути до нього і забути, як пишуться чисті запити і функції;

– сумісність: порушена зворотна сумісність, між версіями фреймворка.

В таблиці 2.2 було проведено аналіз різних фреймворків для того, щоб переконатися в тому, що фреймворк Laravel добре підходить для поставленої задачі.

Таблиця 2.2 – Порівняльна характеристика фреймворків для завдання парсингу даних

Критерій	Laravel	Zend	Symfony
Дата випуску	2011	2005	2005
Автор	Taylor Otwell	Zend Technologies	Fabien Potencier
Складність	Легка	Середня	Складна
Наявність спеціалізованих інструментів	Є	Треба інсталювати	Є
Архітектура фреймворку	MVC	MVC	MVC
Суб’єктивна оцінка (по 10 бальній шкалі)	10	8	9

За результатами порівняльної таблиці, можемо побачити, що Laravel має низький поріг входження, має набір бібліотек для парсингу та його інсталювання не займає багато часу.

На рисунку 2.5 наведено яким чином відбувається парсинг даних за запитом у фреймворку Laravel.

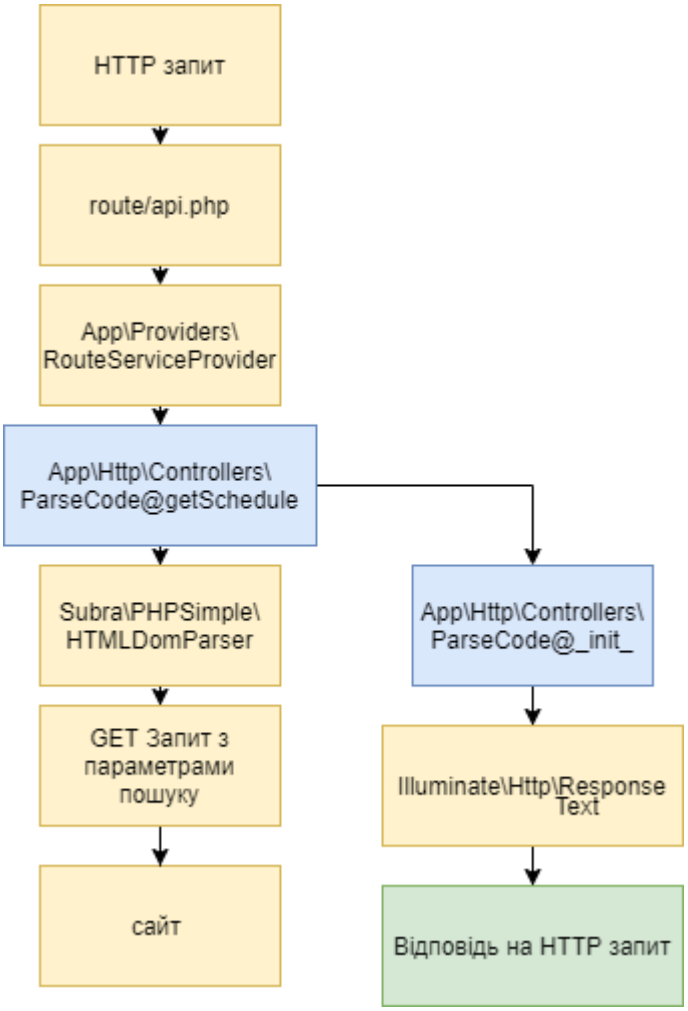


Рисунок 2.5 – Схематичне зображення принципу парсингу даних за запитом

2.3 Огляд середовищ розробки

На даний момент, у поміч розробнику приходять інтелектуальні Integrated Development Environment (IDE). Дані IDE допомогають в правописі

функцій, шкидкому форматуванні текстів програм, швидкий доступ до терміналу, портативне відображення баз даних, тощо.

Також є текстові редактори, які можуть бути налаштовані під вимоги розробника. В таблиці 2.3 наведена порівняльна характеристика середовищ розробки.

Таблиця 2.3 – Порівняльна характеристика середовищ розробки

Критерій	PHPstorm	Visual Studio Code	Sublime text 3
Тип	Інтегроване середовище розробки	Текстовий редактор	Текстовий редактор
Дата випуску	2009	квітень 2015	січня 2013
Налаштування	Мінімальне, у встановник включені необхідні пакети для розробки	Необхідно завантажити і підключити необхідні плагіни та бібліотеки	Необхідно завантажити і підключити необхідні плагіни та бібліотеки
Вбодована підтримка терміналу	Наявна	Відсутня	Відсутня
Суб’єктивна оцінка середи розробки(за 10 бальною шкалою)	9	7	6

В таблиці 2.4 наведена порівняльна характеристика системних вимог середовищ розробки.

Таблиця 2.4 – Порівняльна характеристика системних вимог середовищ

Вимоги	PHPStorm	Visual Studio Code	Sublime text 3
Мінімальне місце для інсталювання	804 Mb	200 Mb (з усіма встановленними плагінами та валідаторами 495 Mb)	34.6 Mb
Оперативна пам’ять	1 Gb – minimum, 2 Gb – recommend	1 Gb – recommend	1 Gb – recommend
Розширення Екрану	1024x768 px	1024x768 px	800x600 px
Платформи	Мультиплатформне	Мультиплатформне	Мультиплатформне

Отже, вибір середовища розробки – окремий вибір кожного розробника. Для розробки прикладного програмного інтерфейсу та програми - парсеру було обрано Visual Studio Code, як зручний в налаштувані та оптимізований програмний засіб для комп’ютерів зі низькими технічними характеристиками.

2.4 Висновки до другого розділу

У розділі було виконано огляд мов програмування та огляд можливостей фреймворку Laravel та розглянуто функціональні можливості середовищ розробки. Було обрано Visual Studio Code як засіб для розробки прикладного програмного інтерфейсу та програми - парсеру.

3 ОСНОВНІ РІШЕННЯ ЩОДО РЕАЛІЗАЦІЇ КОМПОНЕНТІВ СИСТЕМИ

Програма, що розробляється повина виконувати такі основні функції:

а) аналізувати Document Object Model (DOM) дерево сайту та виконувати збір даних:

1) збирати інформацію про серіал;

2) шукати посилання на більш детальну інформацію про серіал;

3) збирати за посиланням більш детальну інформацію про серіал;

б) виконувати парсингу за різними критеріями:

1) збір даних за жанром;

2) збір даних за тегами;

3) збір даних за ключовоми словами;

в) повертання асоціативного масиву даних про серіал.

3.1 Налаштування серверу

Для того, щоб почати розробляти парсер, потрібно підготувати сервер до інсталювання Laravel.

Для встановлення та налаштування Apache потрібно зробити декілька команд:

– sudo apt update – потрібно для оновлення локального індексу пакетів;

– sudo apt install apache2 – інсталювання apache2 до нашого комп’ютеру (рис. 3.1);

– sudo ufw allow in "Apache Full" – надавання дозволу на HTTP та HTTPS трафік;

– sudo apt install mysql–server – інсталювання MySQL;

– sudo apt install PHP libapache2–mod–PHP PHP–mysql – інсталювання PHP та пакети до нього.

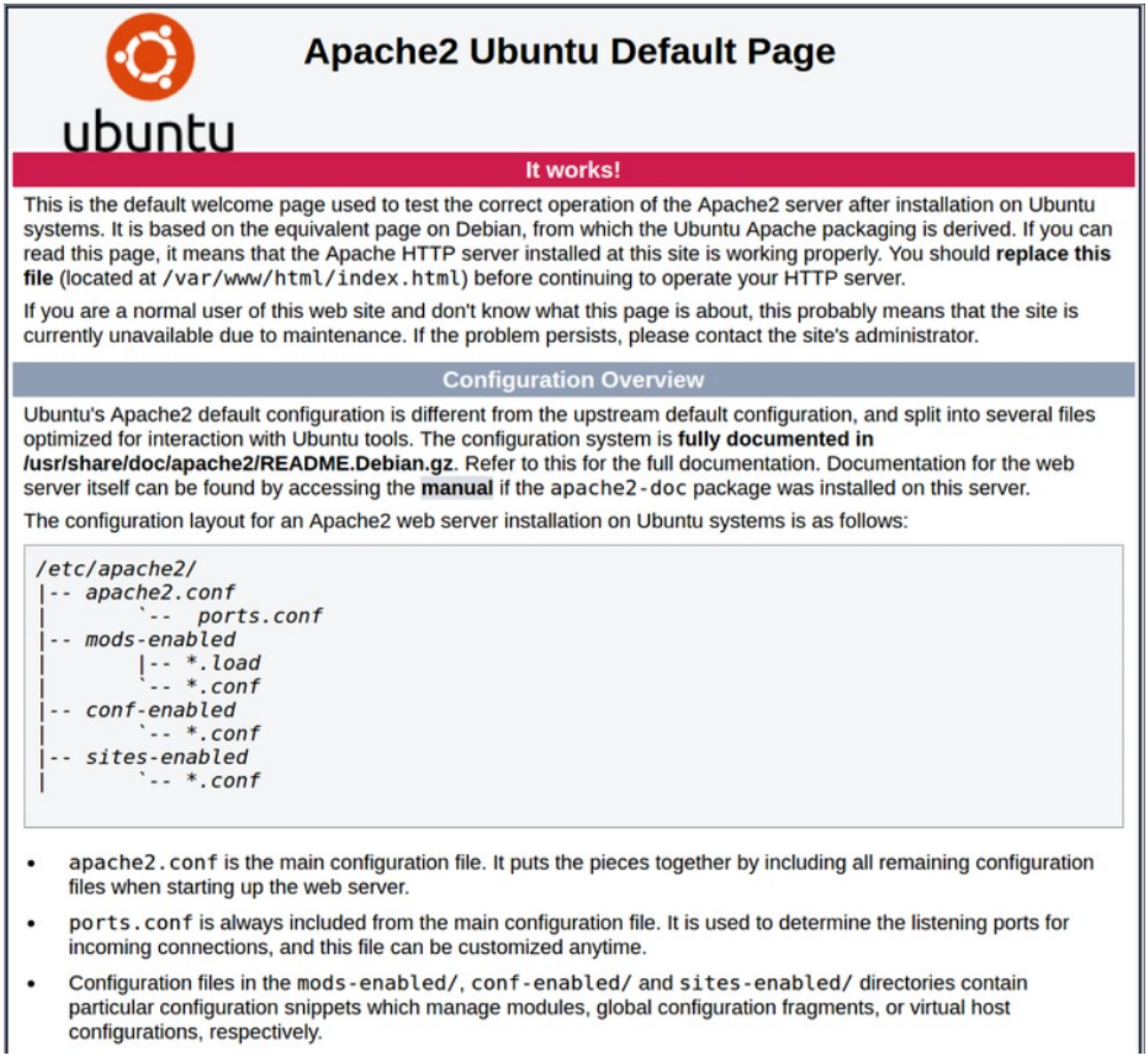


Рисунок 3.1 – Apache page після інсталювання за замовчуванням

Далі необхідно інсталювати Composer.

3.1.1 Composer

Composer – це інструмент для управління залежностями в PHP. Він дозволяє оголошувати бібліотеки, від яких залежить проєкт, та буде керувати (встановлювати / оновлювати) бібліотеками автоматично.

Composer, не є менеджером пакетів в тому ж сенсі, що і Yum або Apt. Так, він має справу з «пакетами» або бібліотеками, але керує ними окремо для

кожного проєкту, встановлюючи їх в каталог (наприклад vendor) всередині проєкту. За замовчуванням, composer нічого не встановлює глобально. Таким чином, це менеджер залежностей. Однак для зручності він підтримує «глобальний» проєкт за допомогою глобальної команди [15].

Припустимо, що:

– у вас є проєкт, який залежить від ряду бібліотек;

– деякі з цих бібліотек залежать від інших бібліотек.

Composer надає наступні можливості:

– дозволяє виконати оголошення бібліотек, від яких залежить проєкт;

– з'ясовує, які версії яких пакетів можна і потрібно встановлювати, і встановлює їх (тобто завантажує їх до проєкту).

Можна оновити всі свої залежності в одній команді.

Можна розташувати Composer PHAR де завгодно. Якщо помістити його в каталог, який є частиною PATH, то можна отримати до нього глобальний доступ. У системах Unix, можна навіть зробити його виконуванним, і викликати його, без безпосереднього використання PHP інтерпретатору [14].

Після запуску інсталятора, слідуючи інструкціям на сторінці завантаження, можна запустити його, щоб перемістити composer.phar в каталог за потрібним шляхом:

```
mv composer.phar /usr/local/bin/composer
```

Якщо потрібно встановити його тільки для свого користувача, і не вимагати прав суперкористувача, то можно використовувати *~ / .local / bin* замість нього те, що доступно за замовчуванням в деяких дистрибутивах Linux [14].

Примітка. Якщо вище описане не працює через дозволи, то потрібно використовувати команду sudo [14].

Примітка. У деяких версіях macOS / usr каталог не існує за замовчуванням. Якщо буде отримано повідомлення про помилку *«/ USR / local /*

bin / composer: Немає такого файлу або каталогу», то потрібно створити каталог вручну, перш ніж продовжити: *mkdir –p / usr / local / bin* [14].

Примітка: Для отримання інформації про зміну PATH, скористайтеся обранною пошуковою системою [14].

Для встановлення усіх залежностей для проекту, потрібно виконати наступну команду у папці проекту (рис.3.2):

```
composer install
```

Рисунок 3.2 – Приклад composer.json файлу

Також, якщо були внесені зміни до composer.json файлу, потрібно виконати команду для оновлення:

```
composer update
```

Це оновить змінити, та зробить оновлення усіх пакетів.

3.1.2 Інсталювання Laravel проекту

Laravel має кілька системних вимог. Усі ці вимоги задовольняє віртуальна машина Laravel Homestead, тому дуже рекомендується, використовувати Homestead, як місцеве середовище розробки Laravel. Але будемо використовувати LAMP для нашого майбутнього проєкту [15].

Так як, ми не будемо використовувати Homestead, то потрібно встановити ці розширення

– PHP> = 7.2.5;

– розширення BCMath PHP;

– розширення PHP Ctype;

– розширення Fileinfo PHP;

– розширення JSON PHP;

– Mbstring розширення PHP;

– розширення PHP OpenSSL;

– розширення PDO PHP;

– розширення PHP Tokenizer;

– XML-розширення PHP.

Для інсталювання Laravel будемо використовувати Composer. Спочатку треба у терміналі, перейти до нашої директорії за цим шляхом:

```
cd /var/www/diplomnaya_robota
```

Далі для завантаження інсталятора Laravel потрібно використати команду для Composer:

```
composer create-project --prefer-dist Laravel/Laravel
```

Після завершення завантаження потрібно створити .htaccess файл з наступним текстом (рис. 3.3).

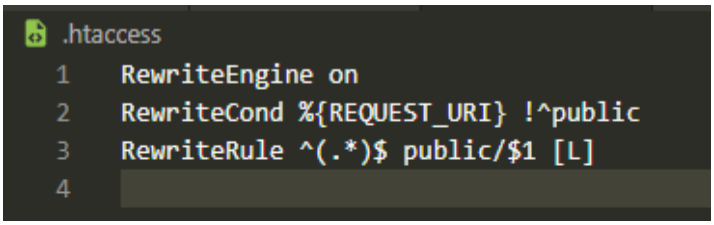


Рисунок 3.3 – Приклад .htaccess файлу

Цей код, буде виконувати переадресацію запитів Apache з кореневої папки до папки public – так як вона головна в фреймворку Laravel.

Після інсталювання, необхідно зробити команду для генерації ключа Laravel, без цього застосунок не є безпечним для користувача:

```
PHP artisan key:generate
```

Зазвичай, цей рядок має бути довжиною 32 символи. Ключ, можна встановити у .env файлі оточення.

Після встановленя ключа, потрібно дати дозвіл до папки storage, для цього потрібно виконати наступну команду:

```
sudo chmod -R 755 storage/
```

Зазвичай, потрібно зробити налаштування баз даних, але це не є необхідною умовою для роботи, через те що, не буде використуваватися база даних.

У .env файлі зберігаються усі постійні доступи до smtp серверів, такі як Application Programming Interface key від стороніх програмних інтерфейсів, також app_key от Laravel app та доступи для баз даних (рис. 3.4).

```
APP_NAME=Laravel
APP_ENV=local
APP_KEY=base64:gE1N4bgLTcRlpTGhjbPBiE8UkCpghoe+elDG8k0I3MI=
APP_DEBUG=true
APP_URL=http://localhost

LOG_CHANNEL=stack

DB_CONNECTION=mysql
DB_HOST=127.0.0.1
DB_PORT=3306
DB_DATABASE=laravel
DB_USERNAME=root
DB_PASSWORD=

BROADCAST_DRIVER=log
CACHE_DRIVER=file
QUEUE_CONNECTION=sync
SESSION_DRIVER=file
SESSION_LIFETIME=120

MIX_PUSHER_APP_KEY="${PUSHER_APP_KEY}"
MIX_PUSHER_APP_CLUSTER="${PUSHER_APP_CLUSTER}"
```

Рисунок 3.4 – .env файл проєкту

Після кожного мануального змінювання .env файлу, потрібно запускати artisan команду:

```
PHP artisan config:cache
```

без виконання цієї команди, зміни не набудуть чинності.

3.2 Структура програми

Як було описано, на початку розділу програма парсер повина мати можливість шукати серіали за різними критеріями пошуку. Для цього, був створенний один клас у якому був описанний увесь процесс за різними методами з різною областттю видимості, декількаами константами та глобальними змінними.

Для проектування програми-парсеру був обраний принцип KISS & DRY: KISS – keep it simple and straightforward (зроби простіше). Отже, принцип проектування KISS проголошує, що простота коду – понад усе, тому що простий код – найбільш зрозумілий [16].

Практично всі принципи проектування спрямовані на досягнення зрозумілості коду. Порушуючи будь-який принцип проектування можна зменшити зрозумілість коду. Незрозумілий код автоматично викликає у людини відчуття того, що код складний, так як його складно зрозуміти та модифікувати. При порушенні будь-якого з цих принципів, також порушується і принцип KISS. Тому можна говорити, що KISS включає майже всі інші принципи проектування [16].

Патерни проектування, що описують найбільш вдалі, прості і зрозумілі рішення деяких проблем. Якщо відбувається використання патерну проектування там, де немає проблеми, яку вирішує даний патерн – то відбувається порушення KISS, вносячи непотрібні ускладнення в текст програми. Якщо, не відбувається використання патерну проектування там, де є проблема, відповідна паттерну – то відбувається знову порушуння KISS, роблячи код складніше, ніж він мав би бути [16].

DRY – don’t repeat yourself (не повторюй себе). Цей принцип настільки важливий, що не вимагає повторення. Зазвичай його згадують акронімом DRY, який вперше з'явився у відомій книзі "The pragmatic programmer", але концепт, сам по собі, був відомий досить давно. Він відноситься до найдрібніших частин програми [17].

Коли потрібно розробити великий проєкт, часто доводиться зіштовхуватися з надлишковою загальною складністю реалізації. Люди, погано справляються з управлінням складних систем, їм краще вдається знаходити незвичайні рішення певних завдань. Найпростіше рішення, щодо зменшення складності – розділити систему на дрібні, незалежні модулі, якими простіше управляти [17].

Принцип роботи парсеру наведений на рис. 3.5. Розробник надсилає запит до прикладного програмного інтерфейсу або API, в залежности від його запиту, він потрапляє до одного із методів контроллеру. Далі, його запит потрапляє в обробку, і виконуються допоміжні методи і (або) повертають йому результат у вигляді зручного массиву даних. Далі, більш детально про кожен з методів.

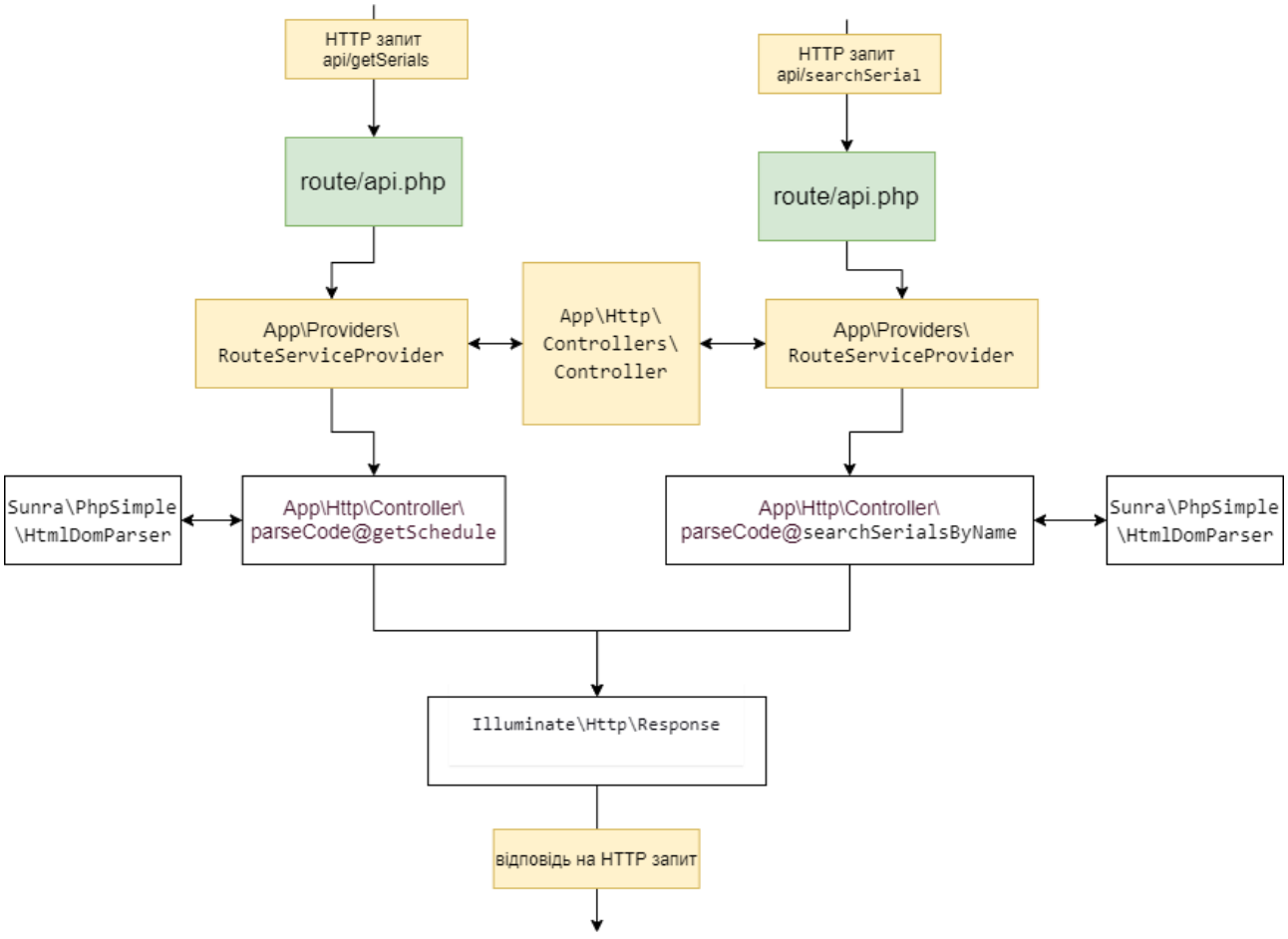


Рисунок 3.5 – Схема роботи парсеру

3.2.1 Реалізація методу parseCore\ParseCode@getSchedule

Метод parseCore\ParseCode@getSchedule – головний метод для парсингу за критеріями пошуку. Цей метод використовує внутрішні методи (або методи хелпери для розділення та простоти коду). Текст методу наведений на рис. 3.6.

```php
/**
 * @param Request $request
 * @param array $options
 * @return array
 */
public function getSchedule(Request $request,array $options = []) : array
{
    $url = $this->setOptions($request);
    $htmlObj = $this->_coreParseObj::file_get_html($url);
    $serials = $htmlObj->find('table')[1]->children;
    foreach ($serials as $key => $serial){
        $href = $htmlObj->find('table')[1]->find('tr')[$key]->find('td')[1]->find('a')[0]->attr['href'];
        $this->returnData[$key][] = $href;
        $this->returnData[$key][] = $this->_init_(str_replace("\t","",$serial->text()));
        $this->returnData[$key][] = $this->getDetailsDataAboutSerial($href);
    }
    return $this->returnData;
}
```

Рисунок 3.6 – Реалізація методу parseCore\ParseCode@getSchedule

Метод написаний за принципом Single responsibility, що означає принцип єдиної відповідальності, тобто є методи, які реалізують тільки одну функцію.

Також, в цьому методі використовується бібліотека для перенесення HTML у DOM подібний об’єкт, та має велику кількість методів для роботи з ним.

У даному методі можна побачити, що з початку потрібно проаналізувати запит на наявність критеріїв пошуку за тегом, жанром чи статусом серіалу.

У методі setOption() (рис. 3.7), описана логіка встановлення GET параметрів, до URL для пошуку за критеріями.

Завдяки статичному методу ***::file_get_html()*** бібліотеки HtmlDomParser було отримано HTML сторінки запиту. Далі, завдяки проведеному аналізу

HTML, є можливість провести синтаксичний аналіз сторінки, та почати збір необхідних даних, а саме: статус серіалу, жанр серіалу, кількість сезонів, рік випуску серіалу, опис серіалу, оцінка серіалу.

```php
/**
 * @param $request
 * @return string
 */
public function setOptions($request)
{
    $url = self::URL."/schedule.php?";
    if (isset($request->genre)){
        if (strpos("=",$url) !== false) {
            $url .= "&genre=" . $this->genre[$request->genre];
        }else{
            $url .= "genre=" . $this->genre[$request->genre];
        }
    }
    if (isset($request->tag))
    {
        if (strpos("=",$url) !== false){
            $url.="&tag=".$this->tag[$request->tag];
        }else{
            $url.="tag=".$this->tag[$request->tag];
        }
    }
    if (isset($request->status))
    {
        if (strpos("=",$url) !== false) {
            $url .= "&status=" . $this->Status[$request->status];
        }else{
            $url .= "status=" . $this->Status[$request->status];
        }
    }
    if (isset($request->year)) {
        if (strpos("=", $url) !== false) {
            $url .= "&year=" . $request->year;
        }else{
            $url .= "year=" . $request->year;
        }
    }
    if (isset($request->page)) {
        if (strpos("=", $url) !== false) {
            $url .= "&page=" . $request->page;
        }else{
            $url .= "page=" . $request->page;
        }
    }
    return $url;
}
```

Рисунок 3.7 – Реалізація методу встановлення критерію пошуку

Дану інформацію отримує окремий метод класу.

Методи, зображені на рисунку 3.8 приймають рядок, в якому міститься певна інформація. Після чого, вони його знаходять та повертають його до методу _init_(), який компонує рядки.

```php
/**
 * @param $str
 * @return false|string
 */
protected function getNameSerial($str)
{
    return trim(html_entity_decode(substr($str,0,strpos($str,"Сезонов:"))));
}

/**
 * @param $str
 * @return string|string[]
 */
protected function getNumber($str)
{
    return html_entity_decode(str_replace(" ","",substr($str,strpos($str,"Сезонов:")+16,strpos($str,"Статус")-strlen($str))));
}

/**
 * @param $str
 * @return string|string[]
 */
protected function getStatus($str)
{
    return html_entity_decode(str_replace(" ","",substr($str,strpos($str,"Статус:")+14,strpos($str,"Жанр:")-strlen($str))));
}

/**
 * @param $str
 * @return string|string[]
 */
protected function getGenre($str)
{
    return str_replace(" ","",substr($str,strpos($str,"Жанр:")+10,strlen($str)));
}
```

Рисунок 3.8 – Реалізація збору інформації зі сторінки

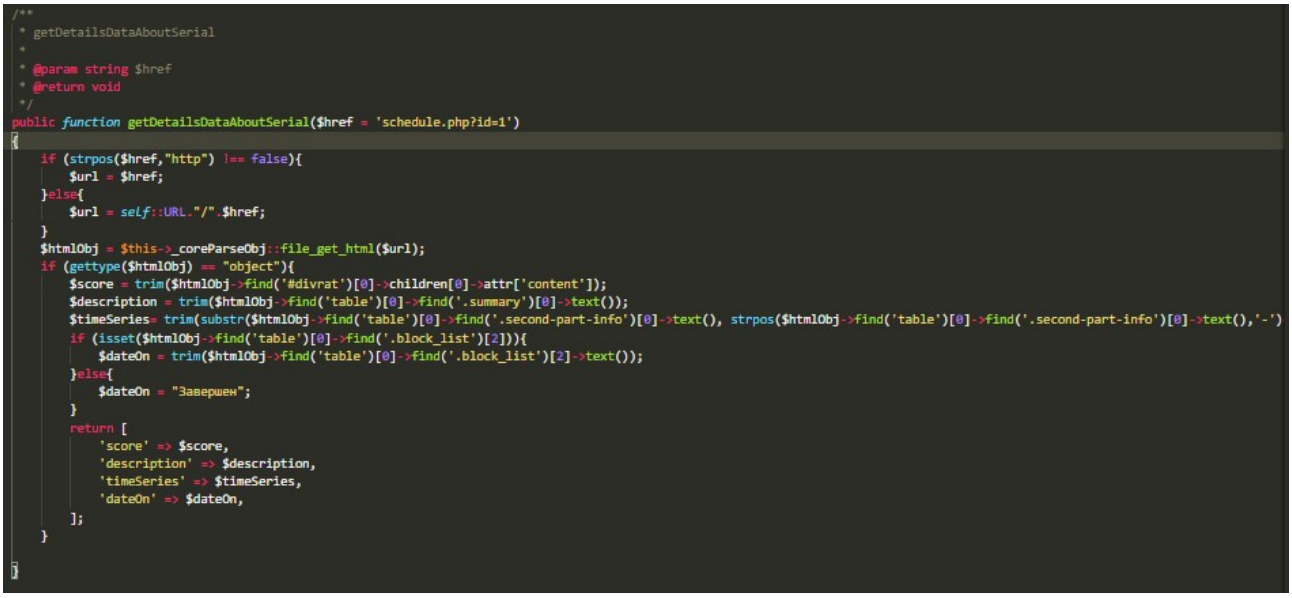
Метод _init_() приймає один головний параметр $serialString – рядок HTML, що передається в методи аналізу та збору певних даних (рис. 3.9).

```php
/**
 * @param string $serialString
 * @param string $key
 * @return array
 */
public function _init_(string $serialString ,$key = 'serials')
{
    return [
        "Name" => $this->getNameSerial($serialString),
        "NumberSeasons" => $this->getNumber($serialString),
        "Status" => $this->getStatus($serialString),
        "Genre" => $this->getGenre($serialString)
    ];
}
```

Рисунок 3.9 – Реалізація методу компонування _init_

Методи-хелпери повертають дані для об’єднання в єдиний асоціативний масив за ключами. Це зручно для компанування.

Останій метод getDetailsDataAboutSerial, який проводить більш детальний аналіз для видобування детальної інформації для серіалу (рис.3.10).



```php
/**
 * getDetailsDataAboutSerial
 *
 * @param string $href
 * @return void
 */
public function getDetailsDataAboutSerial($href = 'schedule.php?id=1')
{
    if (strpos($href,"http") !== false){
        $url = $href;
    }else{
        $url = self::URL."/".$href;
    }
    $htmlObj = $this->_coreParseObj::file_get_html($url);
    if (gettype($htmlObj) == "object"){
        $score = trim($htmlObj->find('#divrat')[0]->children[0]->attr['content']);
        $description = trim($htmlObj->find('table')[0]->find('.summary')[0]->text());
        $timeSeries= trim(substr($htmlObj->find('table')[0]->find('.second-part-info')[0]->text(), strpos($htmlObj->find('table')[0]->find('.second-part-info')[0]->text(),'-')
        if (isset($htmlObj->find('table')[0]->find('.block_list')[2])){
            $dateOn = trim($htmlObj->find('table')[0]->find('.block_list')[2]->text());
        }else{
            $dateOn = "Завершен";
        }
        return [
            'score' => $score,
            'description' => $description,
            'timeSeries' => $timeSeries,
            'dateOn' => $dateOn,
        ];
    }
}
```

Рисунок 3.10 – Реалізація методу для збору детальної інформації

У данному методі виконується пошук та збір даних за певними HTML тегами, та їх класам: оцінка, опис, дата початку серіалу, дата завершення серіалу. Також цей метод повертає масив в метод getSchedule(), та після завершення роботи, надає асоціативний масив користувачу з потрібною інформацією.

3.2.2 Реалізація методу parseCore\ParseCode@searchSerialsByName

Метод parseCore\ParseCode@searchSerialsByName використовує curl для створення запитів, для того щоб забрати HTML сторінки, та передати до статичного методу ***::str_get_html()***, що вимагає HTML рядки для створення об’єкту і (та) пошуку, збору інформації. В методі searchSerialsByName виконується видобування потрібної інформації за ключовим словом для пошуку серіалу або серіалів (рис. 3.11).

```php
public function searchSerialsByName(Request $request)
{
    $array = [];
    $url = self::URL."/search_all.php";
    $ch = curl_init();
    curl_setopt($ch, CURLOPT_URL,$url);
    curl_setopt($ch, CURLOPT_POST, 1);
    curl_setopt($ch, CURLOPT_POSTFIELDS,"value=".$request->value."&db=2");
    curl_setopt($ch, CURLOPT_RETURNTRANSFER, true);
    $server_output = curl_exec($ch);
    curl_close ($ch);
    $htmlObj = $this->_coreParseObj::str_get_html($server_output);
    foreach ($htmlObj->find('ul')[0]->find('li') as $element){
        $str = trim($element->text());
        $pos = strpos($str,")");
        $str = substr($str,"0",$pos+1);
        $array[]['name'] = $str;
    }

    foreach ($htmlObj->find('a') as $k => $element){
        $array[$k]["details"] = $this->getDetailsDataAboutSerial($element->attr['href']);
    }
    return $array;
}
```

Рисунок 3.11 – Реалізація методу searchSerialsByName

3.3 Висновки до третього розділу

Було розроблено програму-парсер для пошуку серіалів за певними параметрами. Був створений контроллер з певними методами для запиту, аналізу та компонування потрібної інформації.

Також було проаналізовано вебсторінки, з яких відбувалося видобування даних, були виявлені необхідні теги, класи, ідентифікатори та тощо за якими виконувався парсинг даних.

4 ТЕСТУВАННЯ ТА ЕКСПЛУАТАЦІЯ ПРОГРАМНОГО ИНТЕРФЕЙСУ ДЛЯ ПАРСИНГУ ДАНИХ

Для тестування програми-парсеру, використовувалися як браузерні GET запити, так і застосунок Postman для зовнішніх запитів, також для швидкої дії були написані unit тести.

4.1 Postman тестування

Postman – зручний http-клієнт для тестування вебсайтів, входить в число найкращих розширень каталогу Chrome Web Store, в категорії «інструменти для роботи» (productivity tools). За допомогою розширення, можливо складати і редагувати, прості або складні http-запити [18].

Також, призначений для перевірки запитів з клієнта на сервер, і отримання відповіді від бекенду.

Для програми - парсеру було протестовано метод parseCore\ ParseCode@getSchedule для парсингу сайту за параметрами year та genre (рис. 4.1).

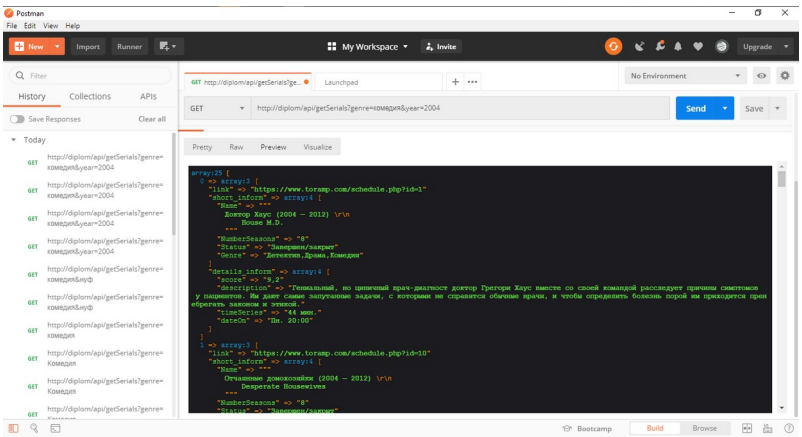


Рисунок 4.1 – Тестування роботи методу parseCore\ ParseCode@getSchedule за допомогою застосунку Postman

Тестування роботи методу parseCore\ParseCode@searchSerialsByName для пошуку серіалу за словом (рис.4.2).

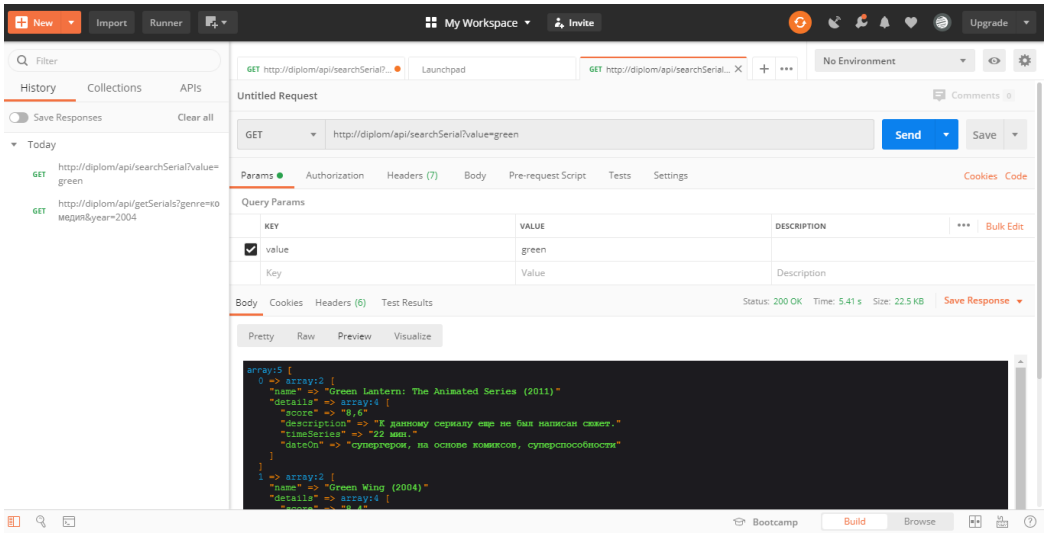


Рисунок 4.2 – Тестування роботи методу parseCore\ ParseCode@searchSerialsByName за допомогою застосунку Postman

4.2 Ручне тестування з використанням браузеру

Ручне тестування з використанням браузеру відбувалося за допомогою браузеру Google Chrome. Браузер посилає GET запит до прикладного програмного інтерфейсу (рис. 4.3).

Як результат, можна побачити всю інформацію за запитом та саму response відповідь за цим запитом. Можна побачит, що дані, які були отримані, представлені не в зручному форматі та не зрозумілі для користувача. Це тому, що дані повертаються у форматі unicode escape sequence, який відповідно до стандарту JSON потрібно перетворювати, в даному форматі символи не входять в діапазон ASCII.

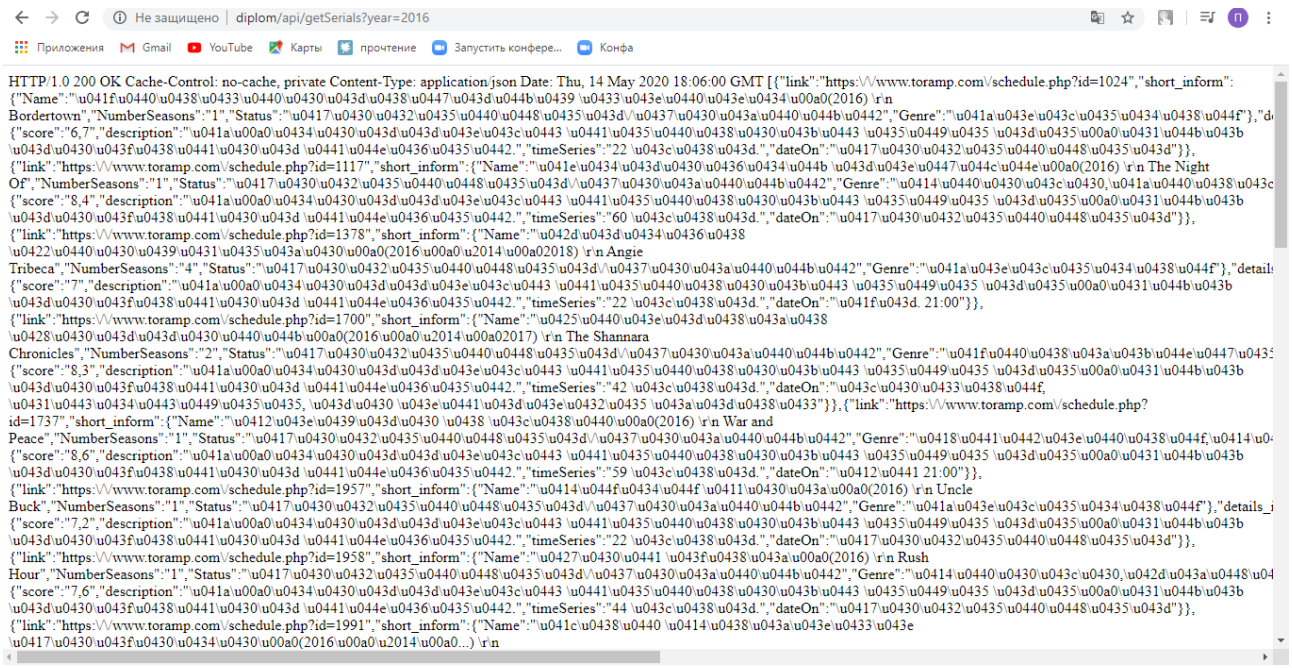


Рисунок 4.3 – Тестування HTTP GET запитом з використанням браузеру Google Chrome

4.3 Unit тестування

Модульне тестування, або unit тестування (англ. Unit testing) – процес в програмуванні, що дозволяє перевірити на коректність окремі модулі вихідного коду програми [19].

Ідея полягає в тому, щоб написати тести для кожної нетривіальної функції або методу. Це дозволяє досить швидко перевірити, чи не призвело чергова зміну коду до регресії, тобто до появи помилок в уже відтестованих місцях програми, а також полегшує виявлення та усунення таких помилок [19].

До фрейморку Laravel вже влаштован PHPunit і це дозволяє написати unit тести. Для створеня unit тесту потрібно виконати artisan команду:

```
PHP artisan make:test firstTest --unit
```

Після цього, шаблон unit тест створено та можна приступити до його налаштування.

Був написаний unit тест для перевірки, що виконується за жанрам та за ріком виходу серіалу. У разі безпомилкового відпрацюваня тесту, буде показана наступна інформація (рис. 4.4).

```php
    /**
     * A basic feature test example.
     *
     * @return void
     */
    public function testExample()
    {
        $arrayGenre = ['мистика','мюзикл','приключения','драма','криминал'];
        $arrayYear = ['2006','2010','2012','2001','2011'];

        echo "Start time testing: ". $this->getTime()."\r\n";
        for($i = 0; $i<3; $i++){
            echo "start test case : $arrayGenre[$i] : ".$this->getTime()."\r\n";
            $response = $this->json('GET','/api/getSerials',[
                'genre' => $arrayGenre[$i],
                'year' => $arrayYear[$i]
            ]);
            echo  "end test case : $arrayGenre[$i] : ".$this->getTime()."\r\n";
        }
        echo "End time testing: ". $this->getTime()."\r\n";


        $response->assertStatus(200);

    }
```

Рисунок 4.4 – Текст файлу unit тесту

Результат тестування можна побачити на рисунку 4.5

```
PHPUnit 8.5.2 by Sebastian Bergmann and contributors.

.                                                                   1 / 1 (100%) Start time testing: 2020-05-15 19:19:22
start test case : мистика : 2020-05-15 19:19:22
end test case : мистика : 2020-05-15 19:19:31
start test case : мюзикл : 2020-05-15 19:19:31
end test case : мюзикл : 2020-05-15 19:19:43
start test case : приключения : 2020-05-15 19:19:43
end test case : приключения : 2020-05-15 19:20:32
start test case : драма : 2020-05-15 19:20:32
end test case : драма : 2020-05-15 19:21:10
start test case : криминал : 2020-05-15 19:21:10
end test case : криминал : 2020-05-15 19:21:44
End time testing: 2020-05-15 19:21:44


Time: 2.52 minutes, Memory: 38.00 MB

OK (1 test, 1 assertion)
```

Рисунок 4.5 – Результати тестування

4.4 Висновки до четвертого розділу

В розділі було виконано опис різних видів тестування програми-парсеру:

– тестування з використанням застосунку postman;

– ручне тестування з використанням браузеру;

– написання unit тестів.

Тестування створено в першу чергу для виявлення помилок та для більш детального, зручного аналізу окремих методів чи компонентів розробленої програми.

ВИСНОВКИ

Результатом виконання кваліфікаційної роботи магістра є розроблений web-застосунок для видобування інформації за серіалам.

Було проаналізовано види контенту на вебсторінках: текстовий та мультимедійний, та теги в яких він міститься. Також було виконано огляд та розглянуто можливості програмних засобів синтаксичного аналізу даних або програм- парсерів.

Було розроблено програму-парсер з наступними функціональними можливостями:

а) аналіз Document Object Model (DOM) дерева сайту та виконання збору даних:

1) збір інформації про серіал;

2) пошук посилання на більш детальну інформацію про серіал;

3) збір за посиланням більш детальної інформації про серіал;

б) виконання парсингу за різними критеріями:

1) збір даних за жанром;

2) збір даних за тегами;

3) збір даних за ключовоми словами;

в) повертання асоціативного масиву даних про серіал.

В роботі було виконано огляд мов програмування та огляд можливостей фреймворку Laravel та розглянуто функціональні можливості середовищ розробки PHPstorm, Visual Studio Code та Sublime text 3.

Програма-парсер та прикладний програмний інтерфейс був реалізований з використанням мови програмування PHP, фреймворку Laravel та в середовищі розробки Visual Studio Code.

Було розроблено програму-парсер для пошуку серіалів за певними параметрами для цього був створений контроллер з певними методами для запиту, аналізу та компонування потрібної інформації.

Також було проаналізовано вебсторінки, з яких відбувалося видобування даних, були виявлені необхідні теги, класи, ідентифікатори та тощо за якими виконувався парсинг даних.

Для тестування програми-парсеру, використовувалися як браузерні GET запити, так і застосунок Postman для зовнішніх запитів, також були написані unit тести.

ПЕРЕЛІК ПОСИЛАНЬ

1. Все про парсинг [Електрон. ресурс]. – Режим доступа: https://www.seonews.ru/glossary/parsing/

2. Види контенту [Електрон. ресурс]. – Режим доступу: https://seo–akademiya.com/baza–znanij/kontent/vidyi–kontenta–na–sajte/

3. Що таке пошуковий бот [Електрон. ресурс]. – Режим доступу: https://netpeaksoftware.com/ru/blog/chto–takoe–poiskoviy–robot–i–kak–on–rabotaet

4. Що таке ESLint [Електрон. ресурс]. – Режим доступу: https://ua.wikipedia.org/wiki/ESLint

5. Сервіси для перевірки унікальності [Електрон. ресурс]. – Режим доступу: https://textbroker.ru/blog/181–kak–rabotajut–servisy–proverki–unikalnosti–teksta–i–pochemu–im–ne–vsegda–sleduet–doverjat.html

6. Import.io [Electronic resource]. – Access mode: https://en.wikipedia.org/wiki/Import.io

7. Web–scraper [Electronic resource]. – Access mode: https://chrome.google.com/webstore/detail/web–scraper/jnhgnonknehpejjnehehllkliplmbmhn?hl=ru

8. Content–DownLoader X1 [Electronic resource]. – Access mode: https://content–downloader.com/uk/

9. Що таке PHP? [Електрон. ресурс]. – Режим доступу: https://www.PHP.net/manual/ru/intro–whatis.PHP#intro–whatis

10. Python [Electronic resourse]. – Access mode: https://en.wikipedia.org/wiki/Python

11. Создание парсеров с помощью Scrapy и Python [Электрон. ресурс]. – Режим доступа: https://pythonru.com/biblioteki/sozdanie–parserov–s–pomoshhju–scrapy–i–python

12. Ruby [Електрон. ресурс]. – Режим доступу: https://ua.wikipedia.org/wiki/Ruby

13. Laravel [Electronic resource]. – Access mode: https://en.wikipedia.org/wiki/Laravel

14. Model–view–controller [Electronic resource]. – Access mode: https://en.wikipedia.org/wiki/Model-view-controller

15. Installing Laravel Framework [Electronic resource]. – Access mode: https://Laravel.com/docs/7.x/installation

16. KISS – принцип проектування, що містить всі інші принципи проектування [Електрон. ресурс]. – Режим доступу: https://habr.com/ru/post/249639/

17. Три ключові принципи ПО, які ви повинні розуміти [Електрон. ресурс]. – Режим доступу: https://habr.com/ru/post/144611/

18. Postman about [Electronic resource]. – Access mode: https://medium.com/effective–developers/postman–как–инструмент–тестирования–API–6c0f76358cf2

19. Unit тести для чайників [Електрон. ресурс]. – Режим доступу: https://habr.com/ru/post/169381/

ДОДАТОК А

Текст програми

А.1 Текст файлу index.php

```
<?PHP

/**
 * Laravel – A PHP Framework For Web Artisans
 *
 * @package  Laravel
 * @author   Taylor Otwell <taylor@Laravel.com>
 */

define('LARAVEL_START', microtime(true));

/*
|--------------------------------------------------------------------------
| Register The Auto Loader
|--------------------------------------------------------------------------
|
| Composer provides a convenient, automatically generated class loader for
| our application. We just need to utilize it! We'll simply require it
| into the script here so that we don't have to worry about manual
| loading any of our classes later on. It feels great to relax.
|
*/

require __DIR__.'/../vendor/autoload.PHP';

/*
|--------------------------------------------------------------------------
| Turn On The Lights
|--------------------------------------------------------------------------
|
| We need to illuminate PHP development, so let us turn on the lights.
| This bootstraps the framework and gets it ready for use, then it
| will load up this application so that we can run it and send
| the responses back to the browser and delight our users.
|
*/

$app = require_once __DIR__.'/../bootstrap/app.PHP';
```

```
/*
|--------------------------------------------------------------------------
| Run The Application
|--------------------------------------------------------------------------
|
| Once we have the application, we can handle the incoming request
| through the kernel, and send the associated response back to
| the client's browser allowing them to enjoy the creative
| and wonderful application we have prepared for them.
|
*/

$kernel = $app->make(Illuminate\Contracts\Http\Kernel::class);

$response = $kernel->handle(
    $request = Illuminate\Http\Request::capture()
);

$response->send();

$kernel->terminate($request, $response);
```

A.2 Текст файлу ParseCode.php

```
<?PHP

namespace App\Http\Controllers\parseCore;

use App\Http\Controllers\Controller;
use Hug\Xpath\Xpath;
use Illuminate\Http\Request;
use PHPDocumentor\Reflection\Types\Self_;
use Sunra\PHPSimple\HtmlDomParser;
use function simplehtmldom_1_5\file_get_html;

class ParseCode extends Controller
{

    const URL = "https://www.toramp.com";

    protected $_coreParseObj;

    public $returnData;

    protected $tag = [
        "all" => "любой",
        "1900–e / 1910–e" => "25",
        "20–e / 30–e" =>"26",
        "40–e / 50–e" => "27",
```

"60–е / 70–е" => "28",
"80–е" => "29",
"90–е" => "30",
"xix век" => "24",
"xvi — xvii век" => "22",
"xviii век" => "23",
"альтернативная история" => "102",
"ангелы / демоны" => "68",
"антигерой" => "46",
"антиутопия" => "56",
"антология" => "94",
"апокалиптика"=> "64",
"банды" => "40",
"библейская" => "81",
"бизнес / финансы" => "109",
"боги" => "89",
"боевые искусства" => "48",
"болезнь / вирус" => "104",
"будущее" => "55",
"вампиры" => "3",
"ведьмы / маги" => "100",
"виртуальная реальность" => "63",
"военный" => "52",
"война" => "15",
"восстание / революция" => "107",
"вторая мировая война" => "32",
"выживание" => "8",
"гангстеры / мафия" => "38",
"гении" => "87",
"гонки" => "47",
"гражданская война" => "108",
"деньги" => "41",
"дети" => "72",
"дизельпанк" => "60",
"доисторические животные" => "103",
"доисторическое время" => "19",
"драконы" => "95",
"древность" => "20",
"другие планеты" => "58",
"друзья / команда" => "71",
"еда / кухня" => "106",
"животные" => "18",
"заговоры" => "84",
"зомби" => "4",
"инопланетяне" => "12",
"искусственный интеллект" => "2",
"итальянский неореализм" => "75",
"киберпанк" => "59",
"космос" => "1",
"любовь" => "11",

"магия" => "9",
"маньяк / психопат" => "45",
"медицинский" => "36",
"месть" => "98",
"монстры" => "66",
"морской мир" => "57",
"музыка" => "13",
"на основе игр" => "79",
"на основе книг" => "77",
"на основе комиксов" => "78",
"на основе манги" => "99",
"на основе мифов / фольклора" => "80",
"на основе пьесы" => "62",
"на основе реальных событий" => "76",
"наемные убийцы" => "43",
"немое кино" => "34",
"ниндзя" => "49",
"новая волна" => "74",
"оборотни" => "69",
"ограбление / афера" => "39",
"параллельные миры" => "54",
"первая мировая война" => "31",
"пираты" => "97",
"повседневность" => "83",
"погоня / побег" => "92",
"подростки" => "73",
"поединок / соревнование" => "101",
"поиск сокровищ" => "42",
"политика" => "10",
"полицейские" => "91",
"постапокалиптика" => "6",
"призраки" => "67",
"психология" => "86",
"путешествие во времени" => "53",
"расследование / дедукция" => "17",
"религия" => "82",
"роботы / киборги" => "7",
"рыцари" => "51",
"самураи" => "50",
"семья" => "96",
"соблазнение" => "88",
"спорт" => "16",
"средневековье" => "21",
"стимпанк" => "61",
"супергерои" => "5",
"суперспособности" => "93",
"тайны / секреты" => "90",
"танцы" => "14",
"убийство" => "44",
"угнетение / дискриминация" => "105",

```
    "уменьшение / микромир" => "65",
    "фильм–путешествие" => "70",
    "холодная война" => "33",
    "черно–белое" => "35",
    "шпионы" => "85",
    "юридический" => "37"
];

protected $Status = [
        "любой" => "all",
       "выходит" =>  "o",
       "завершен/закрыт" =>  "1",
       "ждем премьеры" =>  "2",
       "решается дальнейшая судьба проекта" =>  "3",
       "вернется зимой" =>  "4",
       "вернется весной" =>  "5",
       "вернется летом" =>  "6",
       "вернется осенью" =>  "7",
       "вернется «неизвестно»" => "8"
];

protected $genre = [
        "любой" => "all",
        "биография" => "22",
        "вестерн" => "18",
        "детектив" => "28",
        "документальный" => "24",
        "драма" => "10",
        "история" => "26",
        "комедия" => "3",
        "криминал" => "8",
        "мелодрама" => "33",
        "мистика" => "7",
        "мюзикл" => "14",
        "нуар" => "29",
        "пародия" => "37",
        "пеплум" => "30",
        "приключения" => "13",
        "псевдодокументальный" => "34",
        "реалити–шоу" => "31",
        "семейный" => "25",
        "скетч–шоу" => "32",
        "ток–шоу" => "19",
        "трагедия" => "39",
        "трагикомедия" => "40",
        "тревел–шоу" => "41",
        "триллер" => "9",
        "ужасы" => "16",
        "фантастика" => "6",
        "фильм–катастрофа" => "36",
```

```
        "фэнтези" => "12",
        "черная комедия" => "38",
        "экшн"
    ];

    public function __construct()
    {
        $this–>_coreParseObj = new HtmlDomParser();
    }

    /**
     * @param Request $request
     * @param array $options
     * @return array
     */
    public function getSchedule(Request $request,array $options = []) : String
    {
        $url = $this–>setOptions($request);
        $htmlObj = $this–>_coreParseObj::file_get_html($url);
        $serials = $htmlObj–>find('table')[2]–>children;
        foreach ($serials as $key => $serial){
            $href = $htmlObj–>find('table')[2]–>find('tr')[$key]–>find('td')[2]–>find('a')[0]–
>attr['href'];
            $this–>returnData[$key]['link'] =  self::URL."/".$href;
            $this–>returnData[$key]['short_inform'] = $this–>_init_(str_replace("\t","",$serial–
>text()));
            $this–>returnData[$key]['details_inform'] = $this–>getDetailsDataAboutSerial($href);
        }
        // dd($this–>returnData);
        return response($this–>returnData,200);
    }

    /**
     * @param $request
     * @return string
     */
    public function setOptions($request)
    {
        $url = self::URL."/schedule.PHP?";
        if (isset($request–>genre)){
            if (strpos("=",$url) !== false) {
                $url .= "genre=" . $this–>genre[$request–>genre];
            }else{
                $url .= "&genre=" . $this–>genre[$request–>genre];
            }
        }
        if (isset($request–>tag))
        {
            if (strpos("=",$url) !== false){
                $url.="tag=".$this–>tag[$request–>tag];
```

```
        }else{
            $url.="&tag=".$this–>tag[$request–>tag];
        }
    }
    if (isset($request–>status))
    {
        if (strpos("=",$url) !== false) {
            $url .= "status=" . $this–>Status[$request–>status];
        }else{
            $url .= "&status=" . $this–>Status[$request–>status];
        }
    }
    if (isset($request–>year)) {
        if (strpos("=", $url) !== false) {
            $url .= "year=" . $request–>year;
        }else{
            $url .= "&year=" . $request–>year;
        }
    }
    if (isset($request–>page)) {
        if (strpos("=", $url) !== false) {
            $url .= "page=" . $request–>page;
        }else{
            $url .= "&page=" . $request–>page;
        }
    }
    return $url;
}

/**
 * @param string $serialString
 * @param string $key
 * @return array
 */
public function _init_(string $serialString ,$key = 'serials')
{
    return [
        "Name" => $this–>getNameSerial($serialString),
        "NumberSeasons" => $this–>getNumber($serialString),
        "Status" => $this–>getStatus($serialString),
        "Genre" => $this–>getGenre($serialString)
    ];
}

/**
 * @param $str
 * @return false|string
 */
protected function getNameSerial($str)
{
```

```
        return trim(html_entity_decode(substr($str,0,strpos($str,"Сезонов:"))));
    }

    /**
     * @param $str
     * @return string|string[]
     */
    protected function getNumber($str)
    {
        return html_entity_decode(str_replace(" ","",substr($str,strpos($str,"Сезонов:")
+16,strpos($str,"Статус")–strlen($str))));
    }

    /**
     * @param $str
     * @return string|string[]
     */
    protected function getStatus($str)
    {
        return html_entity_decode(str_replace(" ","",substr($str,strpos($str,"Статус:")
+14,strpos($str,"Жанр:")–strlen($str))));
    }

    /**
     * @param $str
     * @return string|string[]
     */
    protected function getGenre($str)
    {
        return str_replace(" ","",substr($str,strpos($str,"Жанр:")+10,strlen($str)));
    }

    /**
     * getDetailsDataAboutSerial
     *
     * @param string $href
     * @return void
     */
    public function getDetailsDataAboutSerial($href = 'schedule.PHP?id=1')
    {
        if (strpos($href,"http") !== false){
            $url = $href;
        }else{
            $url = self::URL."/".$href;
        }
        $htmlObj = $this–>_coreParseObj::file_get_html($url);
        if (gettype($htmlObj) == "object"){
            $score = trim($htmlObj–>find('#divrat')[0]–>children[0]–>attr['content']);
            $description = html_entity_decode(trim($htmlObj–>find('table')[0]–>find('.summary')[0]–
>text()));
```

```
        $timeSeries= html_entity_decode(trim(substr($htmlObj–>find('table')[0]
                                            –>find('.second–part–info')[0]–>text(), strpos($htmlObj
                                            –>find('table')[0]–>find('.second–part–info')[0]–
>text(),'–')+2, strlen($htmlObj–>find('table')[0]
                                            –>find('.second–part–info')[0]–>text())));
        if (isset($htmlObj–>find('table')[0]–>find('.block_list')[2])){
           $dateOn = trim($htmlObj–>find('table')[0]–>find('.block_list')[2]–>text());
        }else{
           $dateOn = "Завершен";
        }
        return [
           'score' => $score,
           'description' => $description,
           'timeSeries' => $timeSeries,
           'dateOn' => $dateOn,
        ];
     }

  }

  public function searchSerialsByName(Request $request)
  {
     $array = [];
     $url = self::URL."/search_all.PHP";
     $ch = curl_init();
     curl_setopt($ch, CURLOPT_URL,$url);
     curl_setopt($ch, CURLOPT_POST, 1);
     curl_setopt($ch, CURLOPT_POSTFIELDS,"value=".$request–>value."&db=2");
     curl_setopt($ch, CURLOPT_RETURNTRANSFER, true);
     $server_output = curl_exec($ch);
     curl_close ($ch);
     $htmlObj = $this–>_coreParseObj::str_get_html($server_output);
     foreach ($htmlObj–>find('ul')[0]–>find('li') as $element){
        $str = trim($element–>text());
        $pos = strpos($str,")");
        $str = substr($str,"0",$pos+1);
        $array[]['name'] = $str;
     }

     foreach ($htmlObj–>find('a') as $k => $element){
        $array[$k]["details"] = $this–>getDetailsDataAboutSerial($element–>attr['href']);
     }
     return $array;
  }
}
```

A.3 Текст файлу API.php

```
<?PHP
```

```
use Illuminate\Http\Request;
use Illuminate\Support\Facades\Route;

/*
|--------------------------------------------------------------------------
| API Routes
|--------------------------------------------------------------------------
|
| Here is where you can register API routes for your application. These
| routes are loaded by the RouteServiceProvider within a group which
| is assigned the "API" middleware group. Enjoy building your API!
|
*/

Route::middleware('auth:API')->get('/user', function (Request $request) {
    return $request->user();
});


 Route::get('/getSerials','parseCore\ParseCode@getSchedule');
 Route::get('/searchSerial','parseCore\ParseCode@searchSerialsByName');
```