ДЕРЖАВНИЙ УНІВЕРСИТЕТ ТЕЛЕКОМУНІКАЦІЙ

НАВЧАЛЬНО–НАУКОВИЙ ІНСТИТУТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

Кафедра комп'ютерних наук

Пояснювальна записка

до бакалаврської роботи
на ступінь вищої освіти бакалавр

на тему: «**РОЗРОБКА ТЕХНОЛОГІЇ СИНТАКСИЧНОГО АНАЛІЗУ ВМІСТУ WEB-САЙТУ МОВОЮ PYTHON»**

Виконав: студент 4 курсу, групи КНД–41
спеціальності
122 Комп'ютерні науки
(шифр і назва спеціальності)

Гребень Владислав Сергійович
(прізвище та ініціали)

Керівник Ільїн О.О.
(прізвище та ініціали)

Рецензент _____
(прізвище та ініціали)

Нормоконтроль _____
(прізвище та ініціали)

Київ – 2021

ВСТУП

В роботі досліджується використання технології синтаксичного аналізу веб-сайтів, які виникають проблеми та методи їх вирішення. Аналіз мови Python для вирішення питань з синтаксичним аналізом. Використання бібліотек Python для створення технології, яка буде включати в себе парсер контенту, базу даних та зручний засіб для користування отриманим данними. Отримані дані в подальшому будуть використовуватися за допомогою боту в Telegram за посиланням @WorldOfGamesBOT, що дасть змогу отримувати потрібну інформацію не витрачаючи на це велику кількість часу.

Об’єктом дослідження є розкриття проблем та алгоритми вирішення, які виникають при використанні синтаксичних аналізаторів для перегляду вмісту Web-сайтів. Предметом дослідження є синтаксичний аналізатор, способи збору та запису інформації, засоби для керування даними.

В роботі проводиться дослідження можливості використання технології синтаксиного аналізу вмісту веб-сайтів, та які засоби існують для створення. В процесі дослідження передбачається вирішення наступних завдань:

1. Дослідження основних етапів синтаксичного аналізу;
2. Аналіз існуючих засобів захисту від парсингу;
3. Дослідження мови Python як інструмент для створення синтаксичного аналізатора;
4. Розробка технології синтаксичного аналізу вмісту web-сайту мовою python

Метод дослідження – аналітичний, з використанням комп’ютерних технологій.

1 ПОСТАНОВКА ЗАДАЧІ ТА ОСНОВНІ ТЕОРЕТИЧНІ ВІДОМОСТІ

1.1 Постановка задачі

У наш час людство переживає науково-технічну революцію, матеріальною основою якої служить електронно-обчислювальна техніка. На базі цієї техніки з'являється новий вид технологій - інформаційні. До них належать процеси, де «вихідним матеріалом» і «продукцією» є інформація. Інформація одна з основних потреб людського життя. Найбільш поширеним джерелом інформації в сучасному світі є інтернет - всесвітня система об'єднаних комп'ютерних мереж.

Постійне оновлення контенту призводить до зростання обсягу інформації. За прогнозами IDC (International Data Corporation), кількість даних на планеті буде як мінімум подвоюватися кожні два роки. Тому людина, щоб знайти потрібну інформацію витрачає ще більше часу, адже потрібно переглянути не один сайт, наткнутися на рекламу, зрозуміти інтерфейс сайту. У зв'язку з цим з'являється необхідність впровадження спеціальних інформаційно-пошукових систем для пришвидшення процесу збору інформації по сайтах. Такими інформаційно-пошуковими програмами є пошуковики і програми, які використовують технології синтаксичного аналізу. Тому за мету в роботі визначається розробка технології, яка використовуючи синтаксичний аналіз буде збирати та зберігати дані з сайту, та в подальшому дає можливість користування їми за допомогою чат-боту. Це дасть змогу отримувати потрібну інформацію не витрачаючи на це велику кількість

1.2 Що таке синтаксичний аналіз та для чого використовують?

Інтернет постійно росте і розвивається, зрозуміло, що ваш ресурс повинен містити величезні обсяги інформації, щоб мати перевагу над конкурентами. Сьогодні контенту має бути дуже багато. А вручну заповнювати такою кількістю інформації ваш ресурс дуже важко. Людина не в змозі обслуговувати нескінченний потік постійно мінливої інформації, тому необхідний парсинг.

Синтаксичний аналіз (парсинг) (англ. parsing) — це последовне порівняння слів з правилами мови. Мова у даному понятті може бути людською, наприклад,

українська чи анлійська, яка застосовується для передачі інформації між людьми, а може бути, мова програмування. В програмуванні парсинг означає процес аналізу вхідної послідовності символів, з метою розбору граматичної структури згідно із заданою формальною граматикою. Синтаксичний аналізатор (англ. parser) — це програма або частина програми, яка виконує синтаксичний аналіз.[4]

Парсинг сайтів - це процес синтаксичного аналізу розташованої на веб-сторінках інформації. Сторінки сайтів - це набір текстових файлів, розмічених мовою HTML. Ці файли, будучи завантаженими відвідувачем на його комп'ютер, розуміються і обробляються браузером і виводяться на засіб відображення користувача.

Парсери дуже часто використовують у наступних областях:

1. Там, де швидкий автоматизований пошук інформації з формуванням сторінок. Використовується веб-майстрами для заповнення своїх сайтів чужою інформацією, скопійованою з інших джерел.
2. Там, де ручне копіювання не представляється можливим або вимагає колосальних людських витрат (наприклад, для відображення курсу валют, вартості цінних паперів або інсайдерської інформації)
3. Перевірка на антиплагіат саме використовує парсинг, швидко зіставляючи текст з інформацією на веб-сторінках.
4. Парсинг активно використовується власниками інтернет- магазинів при описуванні тисяч найменувань товарів. Технічний опис не признається інтелектуальною власністю, тому дозволяється інтернет спільнотою.
5. Використання підходу для спам розсилок. Бот запускається в соціальні мережі, збираючи адреси користувачів.
6. Збір даних для новинних сайтів, сайтів регіону.
7. Відстеження постійно мінливих погодних умов.

Отже, парсер контенту – це не що інше, як скрипт, який здатній сортувати інформацію, виділяти потрібну і обробляти її згідно з алгоритмом, створеним для

вирішення того чи іншого завдання. Скрипт можна написати за допомгою таких мов програмування: PHP, Perl, Ruby, Python, JavaScript та інші.

У порівнянні з пошуковою системою, комп'ютерна програма-парсер:

1. Має більш високою швидкістю підбору необхідної інформації.
2. Відрізняється підвищеною точністю: містить конкретні дані по заданим критеріям пошуку.

У порівнянні з людиною, комп'ютерна програма-парсер:

1. Здатна швидко проаналізувати понад тисячу веб-сторінок (за кілька секунд).
2. Безпомилково відібрати потрібну інформацію.
3. Представити кінцеві дані в необхідному вигляді (база даних або електронні таблиці).

Перелік речей, які ви можете зробити за допомогою парсеру, майже нескінченний. Зрештою, вся справа в тому, що ви можете зробити із зібраними даними та наскільки цінними ви можете їх зробити.

1.3 Засоби для створення парсеру

Створення програми-парсеру не потребує від розробника великих знань про мову програмування та необов'язково мати фундаментальні знання відомостей про сумісні технології. Але щось все-таки необхідно виділити та розібрати. Отже, перелічимо основні інструменти, які потрібно дізнатись кожному, хто забажає створювати синтаксичні аналізатори:

- Для початкового алгоритму функціонування програми потрібен ретельний аналіз вихідного коду веб-сторінки, що є донором. Тут не обійтися хоча б без середніх знань технологій верстки. Це HTML, CSS і мова JavaScript.
- Щоб зануритися в тему глибше, потрібно вивчити технологію під назвою DOM. Вона дає можливість дуже ефективно працювати з ієрархією веб-сторінки.

- Найважчий етап - написання парсеру. Тут потрібно володіти інструментом для обробки тексту. Досвідчені програмісти найчастіше використовують для цієї мети регулярні вирази, які є досить потужним засобом. Але це під силу далеко не кожному розробнику. Тут потрібна особливе мислення. Оптимальним рішенням буде використання вже готових бібліотек, які створювалися спеціально під парсинг. Що це за бібліотеки? Це упакований програмний код, який вже містить всі функції для аналізу.
- Дуже бажано розбиратися в об'єктно-орієнтованому програмуванні, яке підтримується будь-якою мовою програмування.
- Завершальний етап обробки результатів аналізу передбачає, що дані будуть структуровані та збережені. Тут не обійтися без знань баз даних.
- Потрібні знання й володіння функціями та службами для роботи з файлами. Адже дані потрібно буде записувати в ці самі файли, а потім, можливо, конвертувати в формат електронних таблиць.

1.4 Етапи парсингу

Парсинг даних можна розділити на три етапи:

1. Завантажити код інтерфейсу – значить отримати код тієї сторінки, з якої потрібно дістати інформацію. Для цього передбачені різні способи у різних мовах, наприклад, у мові програмування Python3 найчастіше використовується бібліотека «requests», яка зберігає дерево сайту в змінну.

2. Витяг інформації. Витяг інформації з html-коду зазвичай означає, що потрібно відокремити зі сторінки саме ту інформацію, заради якої й відбувається весь цей процес. Частіше - це звичайний текст, який потрібно відокремити від гіпертекстової розмітки або вистроїти ієрархічне дерево елементів документу. Для таких цілей можна звичайно використовувати регулярні вирази, проте є більш зручний шлях - спеціалізовані бібліотеки.

3. Збереження інформації. Успішно обробивши дані на веб-сторінці, потрібно їх зберегти в необхідному вигляді для подальшої обробки. Спарсені дані

зазвичай заносяться в базу даних, але конкретний формат даних залежить від того, де буде використовуватися інформація.

Зазвичай інформація, яку потрібно спарсити, знаходиться не на одній сторінці веб-сайту. Тому, після проходження всіх етапів парсингу, до алгоритму програми потрібно додати перехід на інші сторінки сайту. Зазвичай цей алгоритм складається при аналізі сайту, відповідно до того, як на сайті реалізована логіка формування посилань до сторінок, тоді можна створити функцію, яка буде генерувати посилання до нашого парсеру. Коли потрібно зібрати всю інформацію з сайту, цей спосіб не найкращий, тоді потрібно програмувати парсер таким чином, щоб він зберігав та проходив по всім внутрішнім посиланням сайту.

Отже, незалежно від того, на якій формальній мові програмування написаний парсер, алгоритм його дії залишається однаковим.

1.5. Завантаження коду інтерфейсу

Коли ви відвідуєте веб-сайт через браузер, ви відправляєте так званий HTTP-запит. По суті, це цифровий еквівалент стуку в двері з проханням увійти. У більшості випадків, ваш запит буде схвалений, ви отримаєте доступ до цього сайту і всієї інформації на ньому, проте не завжди все виявляється так просто. Деякі власники сайтів можуть навмисно ускладнювати цей процес парсеру.

Завантаження інформації інколи більш складніший етап ніж подальші аналіз та витяг. Далі ми розберемо всі способи, до яких вдаються власники сайтів, щоб зберегти інформацію, та які програмні рішення існують у розробників.

1.6. Аналіз та витяг інформації

Синтаксичний аналіз веб-сторінки дійсно важливий, оскільки той, хто витягує дані, повинен на даному етапі знайти та відокремити у завантаженому коді потрібну інформацію. Для цього єтапу у програміста зараз є багато інструментів, хоча ще недавно, єдиним було використанння регулярних виразів.

У програмуванні регулярні вирази надають стислий і гнучкий засіб для ідентифікації рядків тексту, таких, як конкретні символи, слова або шаблони

символів. Регулярні вирази записуються на офіційній мові, яка може бути інтерпретована процесором регулярних виразів або аналізує текст та ідентифікує частини, які відповідають наданій специфікації. Багато мов програмування підтримують регулярні вирази для роботи з рядками. Синтаксис регулярних виразів залежить від інтерпретатора, що використовується для їхньої обробки. Однак, із незначними відхиленнями, майже всі поширені інтерпретатори регулярних виразів мають спільні правила.

Цей метод гарно справляється з текстом, але коли ми маємо справу з некоректним html-кодом, тоді потрібно будувати дерево документу та працювати за допомогою DOM. Також нечитабельність є великою проблемою при написанні та редагуванні.

Також можна збирати дані за допомогою засобів емуляції поведінки користувача в браузері. Одним із засобів емуляції поведінки користувача в браузері є Selenium. Selenium — це масштабний open source проект, а точніше сказати browser automation framework, у межах якого розробляється серія програмних продуктів для автоматизованого тестування та збору інформації.

Одним із способів є збір данних за допомогою API. API (від англ. Application programming interface) - це інтерфейс взаємодій між сайтом, сторонніми програмами і серверами, набір готових класів, процедур, функцій, структур та констант, що надаються додатком (бібліотекою, сервісом), або операційною системою для використання у зовнішніх програмних продуктах. Вхід і реєстрація на різних онлайн-сервісах або платформах, здійснювані через акаунти в соціальних мережах, є використанням API. В даному випадку сервіси або додатки використовують бази даних соціальних мереж. При цьому сервіс може отримувати інформацію про користувача і маніпулювати нею в своїх цілях. Незважаючи на всю зручність використання API, існує одне обмеження - мережа не може віддавати всі дані, які видно користувачам в інтерфейсі.[3]

Зараз має багато гарних рішень на більшості мовах програмування - створено бібліотеки для парсингу, які полегшують роботу з синтаксичним аналізом.

1.7. Збереження даних

Дані, які ми отримали потрібно зберігати. Зазвичай це використання баз даних. База даних (БД) - упорядкований набір логічно взаємопов'язаних даних, що використовується спільно, та призначений для задоволення інформаційних потреб користувачів. У технічному розумінні включно й система управління БД.

Система керування базами даних (СКБД) - це комплекс програмних і мовних засобів, необхідних для створення баз даних, підтримання їх в актуальному стані та організації пошуку в них необхідної інформації.

Головним завданням БД є гарантоване збереження значних обсягів інформації та надання доступу до неї користувачеві або ж прикладній програмі. Таким чином БД складається з двох частин: збереженої інформації та системи управління нею.

Застосунки для роботи з базою даних можуть бути частиною СКБД або автономними. Найпопулярнішими СКБД є MySQL, PostgreSQL, SQLite, Microsoft SQL Server, Oracle, Sybase, Interbase, Firebird та IBM DB2. СКБД дозволяють ефективно працювати з базами даних, обсяг яких робить неможливим їх ручне опрацювання.

В загальному базу даних неможливо просто перемістити з однієї СКБД до іншої. Але СКБД використовують стандарти (SQL, ODBC, JDBC), які уніфікують ряд операцій по роботі з даними і дозволяють різним застосункам працювати з базами даних різних СКБД. СКБД часто класифікують за моделлю організації даних. Найвживаніші СКБД використовують реляційну модель, у якій дані подають у виді таблиць. Для кінцевого користувача (та прикладних програм) робота з базою даних напряму неможлива. Всі маніпуляції над даними здійснюють через спеціальні запити, які надсилають до СКБД. СКБД опрацьовує їх і повертає результат. Безпосередньо з базою даних працює виключно СКБД.

Сучасні СКБД забезпечують функції щодо керування даними, які можна поділити на такі групи:

- Оголошення даних — створення, зміна та видалення визначень, які описують організацію даних.
- Модифікація даних — додавання даних, їх редагування та видалення.
- Отримання даних — надання даних за запитом застосунку у формі, яка дозволяє їх безпосереднє використання. Дані можуть надаватись або у формі, в якій вони зберігаються у базі даних, або в іншій формі (наприклад, через поєднання різних даних).
- Адміністрування даних — реєстрування та відслідковування дій користувачів, дотримання безпеки роботи з даними, забезпечення надійності та цілісності даних, моніторинг продуктивності, резервне копіювання та відновлення даних тощо.[15]

Зазвичай інформацію можна відправляти до бази даних за допомогою SQL запитів, але інколи використовують JSON формат. На практиці, зазвичай, розпарсені дані імпортують у CSV файл, цей формат дуже легко переформувати у SQL запити для того, щоб відкрити в Excel.

1.7.1 SQL

SQL (Structured query language — мова структурованих запитів) — декларативна мова програмування для взаємодії користувача з базами даних, що застосовується для формування запитів, оновлення і керування реляційними БД, створення схеми бази даних та її модифікації, системи контролю за доступом до бази даних. Сама по собі SQL не є ані системою керування базами даних, ані окремим програмним продуктом. На відміну від дійсних мов програмування (C або Pascal), SQL може формувати інтерактивні запити, або, будучи вбудованою в прикладні програми, виступати, як інструкції для керування даними. Окрім цього, стандарт SQL містить функції для визначення зміни, перевірки та захисту даних.

Переваги SQL:

1. Незалежність від конкретної СУБД. Не зважаючи на наявність діалектів і відмінностей в синтаксисі, більшість текстів SQL-запитів, що містять, DDLi DML можуть бути досить легко перенесені з однієї СУБД в іншу.

2. Наявність стандартів. Наявність стандартів і наборів тестів для виявлення сумісності та відповідності конкретній реалізації SQL загальноприйнятому стандарту, тільки сприяє «стабілізації» мови.

3. Декларативність. За допомогою SQL програміст описує лише дані, які потрібно витягнути або модифікувати. Яким саме чином це зробити, вирішує СУБД безпосередньо при обробці SQL-запиту.

Оскільки SQL не є мовою програмування (тобто не надає засобів для автоматизації операцій з даними), нововведення різних виробників стосувалися в першу чергу процедурних розширень. Це збережені процедури (англ. stored procedures) і процедурні мови - «надбудови». Практично в кожній СУБД застосовується своя процедурна мова.[12]

Інколи данні з SQL доводиться конвертувати в інші формати: JSON, текстовий CSV, таблично-процесорний XLS, XML.

1.7.2 CSV

Файл, розділений комами (CSV), є текстовим файлом, який містить список даних. Ці файли часто використовуються для обміну даними між різними програмами. Наприклад, бази даних і менеджери контактів часто підтримують файли CSV.

Ці файли іноді можуть називатися значеннями, розділеними символами або файлами, розділеними комами. Для розділення (або розмежування) даних, вони в основному використовують символ коми, але іноді використовують інші символи, такі як крапки з комою (рис. 1.1). Ідея полягає в тому, що ви можете експортувати складні дані з однієї програми до файлу CSV, а отім імпортувати дані в цьому файлі CSV в іншу програму.

```
empid;empname;empgender;empsalary
1;anvesh;male;150000
2;roy;male;120000
3;jenny;female;60000
4;nupur;female;90000
```

Рисунок 1.1 - Структура файлу

У CSV купа плюсів: текстові файли прості, як гудзик, відкриваються швидко, читаються на будь-якому пристрої і в будь-якому середовищі, без додаткових інструментів, відкриваються в Excel. CSV - надпопулярний формат обміну даними, хоча йому вже років 40. Одна біда - текстового редактору для роботи з CSV замало. Ще нічого, якщо таблиця проста: в першому полі ID однієї довжини, у другому - дата одного формату, а в третьому - яка-небудь адреса. Але, коли поля різної довжини, і їх більше, ніж три, тоді аналізувати та змінювати код дуже складно. Тому CSV-файли аналізують і редагують в Excel і аналогах: Open Office, LibreOffice та інших.

1.7.3 JSON

JSON (англ. JavaScript Object Notation, укр. запис об'єктів JavaScript, вимовляється джейсо́н) — це текстовий формат обміну даними між комп'ютерами. JSON базується на тексті, може бути прочитаним людиною. Формат дає змогу описувати об'єкти та інші структури даних. Цей формат використовується переважно для передачі структурованої інформації через мережу (завдяки процесу, що називають серіалізацією).

JSON знайшов своє головне призначення в написанні веб-програм, а саме при використанні технології AJAX. JSON, що використовується в AJAX, виступає як заміна XML (використовується в AJAX) під час асинхронної передачі структурованої інформації між клієнтом та сервером. При цьому перевагою JSON перед XML є те, що він дозволяє складні структури в атрибутах, займає менше місця і прямо інтерпретується за допомогою JavaScript в об'єкти. Основними

перевагами даного методу перед тим же SQL є :можливість прямої взаємодії браузеру та серверу, що дозволяє реалізовувати алгоритми пов'язані з фоновим занесенням інформації в базу даних, роботою з файловим кешем, серіалізацією, десеріалізацією динамічних структур та ін.

1.8 Засоби захисту веб-сайтів від парсингу

Багато виробників контенту або власників сайтів, зрозуміло, хвилюються про те, що парсер збирає всі їх дані, і задаються питанням, чи є якісь технічні засоби для зупинки автоматичного збору даних. Якщо веб-сайт подає інформацію таким чином, щоб браузер міг отримати доступ до них і відтворити їх для пересічного відвідувача, то той самий вміст можна скребти за допомогою сценарію чи програми. Будь-який вміст, який можна переглянути на веб-сторінці, можна спарсити. Але є багато речей, які можуть зробити власники веб-сайтів, щоб зробити життя веб-скрапера настільки складним, що вони відмовляються й не намагаються завантажувати інформацію з сайту.

Головним завданням при видобуванні інформації з веб сайту, щоб веб-сайт не дізнався, що це робить парсер. Дізнатися, що твою програму веб-сайт заблокував чи заборонив, дуже легко. Тоді на веб-сайті, який скануємо, з’являється будь-який із наведених нижче знаків, це, як правило, знак блокування чи заборони:

- Сторінки CAPTCHA
- Незвичні затримки доставки вмісту

Часта поява кодів стану HTTP, також свідчить про блокування.

- 301 - Переміщено тимчасово
- 401 - Несанкціонований
- 403 - Заборонено
- 404 - Не знайдено
- 408 - Час очікування запиту

- 429 - Забагато запитів
- 503 - Послуга недоступна

Можна визначити такі методи, якими користуються для забезпечення захисту вмісту веб-сайту[6]:

1. Добре написаний файл robots.txt.
2. Ускладнюють перелік URL-адресів веб-сторінок .
3. Обмежують активність з однієї IP-адреси.
4. Роблять інтерфейс динамічним (JS, AJAX).
5. CAPTCHA.
6. Відстежують взаємодію з інтерфейсом (як швидко заповнюються форми, де відвідувачі натискають кнопку, чи завантажуються CSS / зображення).
7. Чорний список IP-адрес популярних проксі-сервісів.
8. User agent.
9. Файли cookie.
10. Потрібна автентифікація для всіх викликів API.
11. Оновлюють свій HTML.
12. Honeypot.

Створюючи парсер, потрібно витратити трохи часу, щоб дослідити, який механізм протидії парсингу, використовує веб-сайт, а потім запрограмуйте свою програму. Це призведе до кращого розуміння, як побудувати правильну логіку роботи парсеру.

Висновки розділу 1.

В даному розділі було проаналізовано та розглянуто актуальність даної роботи. Визначена мета роботи. Було визначено та досліджено, що таке парсинг. Визначено засоби для створення парсерів. Дослідженно основні етапи парсингу та коротко описанно основні властивості. Визначенно, які методи використовують для захисту веб-сайтів від парсингу.

2 PYTHON ЯК ІНСТРУМЕНТ ДЛЯ РОЗРОБКИ ТЕХНОЛОГІЇ ПАРСИНГУ.

Чому була вибрана мова саме Python? Python - інтерпретована об'єктно-орієнтована мова програмування високого рівня зі строгою динамічною типізацією. Розроблена в 1990 році Гвідо ван Россумом. Структури даних високого рівня разом із динамічною семантикою та динамічним зв'язуванням роблять її привабливою для швидкої розробки програм, а також як засіб поєднування наявних компонентів. Python підтримує модулі та пакети модулів, що сприяє модульності та повторному використанню коду. В мові програмування Python підтримує кілька парадигм програмування, зокрема: об'єктно-орієнтована, процедурна, функціональна та аспектно-орієнтована.

Python - мова програмування загального призначення, орієнтована на підвищення продуктивності розробника і читання коду. Подібно до англійської мови, Python має дуже базовий синтаксис. Як правило, програми, написані на Python, складають 10% коду Java. Це означає, що для того, щоб виконати те саме, що і в Java, ми можемо писати менше коду на Python. Коли ви запускаєте програму Python, вихідний код інтерпретатор перекладає та виконує рядки за рядками, порівнюючи із компільованими мовами, такими як C або C ++, де компілятор перекладає всі оператори коду одночасно. Python також зручна як мова розширення для прикладних програм, що потребують подальшого налагодження.

Еталонною реалізацією Python є інтерпретатор CPython, що підтримує більшість активно використовуваних платформ. Він поширюється під вільною ліцензією Python Software Foundation License, що дозволяє використовувати його без обмежень у будь-яких додатках, включаючи пропрієтарні. Є реалізації інтерпретаторів для JVM (з можливістю компіляції), MSIL (з можливістю компіляції), LLVM та інших [1].

Найчастіше Python використовують для таких цілей:

- створення веб-додатків на сервері
- для побудови робочих процесів Python

- для взаємодії з базами даних, а також, щоб читати або оновлювати файли
- для управління великими базами даних
- для передової математики
- широко використовується дослідниками даних для машинного навчання та штучного інтелекту
- використовується для обробки зображень, обробки тексту та графічного інтерфейсу користувача
- для тестування фреймворків
- для парсингу

Python дуже гарно підходить для побудови технології парсингу, тому що має бібліотеки для роботи над синтксаичним аналізом, управлінням базами даними та відправки, як в нашому випадку до чат-боту Telegram (рис. 2.1).

Рисунок 2.1 - Схематичне представлення технології парсингу

2.1 Методи завантаження коду сторінки

Щоб зробити синтаксичний аналіз першим єтапом буде завантаження коду сторінки. Нам потрібно встановити зв'язок з сервером сайту, та завантажити URL. Для цього в Python існує декілька бібліотек, найпоширені requests і urllib. У нашій технології будемо використовувати бібліотеку requests. Напишемо скрипт, в якому зробимо import бібліотеки requests та використаємо метод классу get().

```
def get_html(url):
return requests.get(url).text
```

Зазвичай завдяки цьому методу, передавши функції URL, ми отримаємо потрібний HTML код сторінки. Але є випадки, коли завантаження не відбувається, це свідчить про помилку. Дуже складно написати парсер, який ніколи не заблокується, але є декілька стратегій, які допоможуть нам замаскуватися під реального користувача, щоб збільшити життя свого синтаксичного аналізатора.

Для початку добавимо трохи коду до нашого методу, щоб у разі появи помилки не виконував скрипт, а повідомляв, яка саме помилка трапилась:

```
def get_html(url):
r = requests.get(url)
if r.ok: return r.text
print(r.status_code)
```

Отже цим методом ми перевіряємо, якщо сервер відповідає код 200, то в результаті отримаємо HTML код сторінки, якщо інший, то отримаємо код помилки.

Основною задачею на даному етапі буде здебільшого отримування коду 200, тому що, коли отримуємо помилку, означає, що ми невірно працюємо з сайтом. Розглянемо основні стратегії, які можна використовувати, як протидію, захисту.

Перевірка файлу Robots.txt.

Перед тим, як приступити до парсингу, потрібно подивитися на файл robot.txt. Багато веб-сайтів із великим трафіком матимуть файл robots.txt, де ми зможемо знайти конкретну інформацію, як можна збирати данні з сайту. У ньому є конкретні правила поведінки, наприклад, як часто ви можете парсити, які сторінки дозволяють парсити, а які - ні. Деякі веб-сайти дозволяють Google парсити їх веб-сайти, не дозволяючи іншим пошуковим системам. Якщо файл robots.txt існує, його можна отримати, додавши "/robots.txt" в кінці будь-якого домену, до якого ми отримуємо доступ, наприклад, "http://example.com/robots.txt". Однак, це суперечить відкритій природі інтернету і може здатися не

справедливим, але власники веб-сайту в межах своїх прав вдаються до такої поведінки. Якщо він містить такі параметри, це означає, що власник сайту не хоче, щоб його спарсили, і гугл не буде сканувати сайт[2] .

Параметри файлу robots.txt:

- User-agent - правило про те, яким браузером користується парсер.
- Disallow дає рекомендацію, яку саме інформацію не варто сканувати.
- Sitemap повідомляє парсерам, що всі URL сайту, обов'язкові для індексації, знаходяться за адресою http://site.ua/sitemap.xml.
- Crawl-delay - параметр, за допомогою якого можна задати період, через який будуть завантажуватися сторінки сайту.
- Allow дозволяє сканувати будь-який файл, директиву чи сторінку.
- Clean-param допомагає боротися з get-параметрами для уникнення дублювання контенту.

Основною функцією цих параметрів є пошук реальних користувачів веб-сайту та ботів. Отже, передав сайту потрібні параметри при завантаженні можна уникнути блокування. Розглянемо, як можна реалізувати це за допомогою Python.

User-Agent.

Найперше, про що потрібно подбати - це налаштування агенту користувача. User Agent - це інструмент, який працює від імені користувача та повідомляє серверу про те, який веб-браузер користувач використовує для відвідування веб-сайту. Багато веб-сайтів не дозволяють переглядати вміст, якщо користувацький агент не встановлений.[5]

Ідея полягає в тому, щоб передати сайту підроблене поле користувача-агента (або кілька різних агентів-користувачів у повороті) в поле заголовку, щоб обдурити сервер. Отримати рядок User-Agent можливо, написавши, у рядку пошуку Google, my user agent, і він поверне вам ваш власний User-Agent. Допрацюємо наш метод, дописавши до методу get() ще один атрибут header, який ми отримали у пошуковику:

```
def get_html(url):
header = {'user-agent': ' Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/90.0.4430.212 Safari/537.36 ' }
r = requests.get(url, headers=header)
if r.ok: return r.text
print(r.status_code)
```

Можна змінювати агента користувача шляхом редагування заголовків вручну, або шляхом написання функції, яка оновлює список агентів користувачів кожного разу, коли ми запускаємо скрипт. Регулярно потрібно оновлювати рядки агента користувача. Оскільки нові випуски браузера стають частішими, виявити застарілий рядок агента користувача простіше, ніж будь-коли, а сервери блокувати запити від зазначених агентів користувачів. Якщо виявиться, що парсер блокується навіть після введення рядка User-Agent, слід додати ще кілька заголовків запитів. Більшість сайтів потребує надіслати більше заголовків, ніж просто User-Agent.

Найбільш базові заголовки:

- User-Agent.
- Accept - це типи вмісту, прийнятні як відповідь. Існує безліч різних типів та підтипів вмісту: text / plain, text / html, image / jpeg, application / json.
- Referer - веб-сайти використовують цей заголовок, щоб змінити свою поведінку залежно від того, звідки походить користувач. Наприклад, багато веб-сайтів новин мають платну підписку і дозволяють переглядати лише 10% публікації. Мета може виглядати так, ніби ви прямуєте з Google. У цьому вам може допомогти заголовок “Referer”: “ https://www.google.com/ ”.
- Host - це доменне ім'я серверу. Якщо номер порту не вказаний, вважається, що це 80.

Робота з cookie, автентифікація.

Багато веб-сайтів мають якусь автентифікацію, про яку інколи доведеться подбати в нашому парсері. Cookie - це поле заголовка містить список пар ім'я-

значення (name1 = value1; name2 = value2). Наприклад, коли ви заповнюєте форму для входу, сервер перевірятиме, чи правильно вказані вами облікові дані. Якщо так, він перенаправить вас і введе сесійний файл cookie у ваш браузер. Потім ваш браузер надсилатиме цей файл cookie з кожним наступним запитом на цей сервер. Отже, як спарити інформацію з веб-сайт за допомогою постійних файлів cookie?[7]

Одним із способів збереження деяких параметрів і файлів cookie у запитах це використання об’єкту Session модуля python. Для тестування HTTP-запитів, які ми надсилаємо на веб-сторінку, ми можемо використовувати httpbin.org, який є «простою службою запитів та відповідей HTTP».

```
import requests
s = requests.Session()
s.get('http://httpbin.org/cookies/set/sessioncookie/1234')
header = {'user-agent': ' Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/90.0.4430.212 Safari/537.36 ', 'Accept-Encoding': 'gzip, deflate','Accept': '*/*', 'Connection': 'keep-alive', 'Cookie': 'sessioncookie=1234'}
print(page.request.headers)
```

Результат {'Cookie': 'sessioncookie = 1234'} показує нам, що файл cookie, який ми отримали з попередньої сторінки, зберігається при виконанні наступного запиту.

Використання проксі-серверів та зміна IP - адреси.

Кілька запитів, що надходять з одного і того ж IP, призведуть до того, що вас заблокують, саме тому нам потрібно використовувати кілька адрес. Коли ми надсилаємо запити з проксі-машини, цільовий веб-сайт не знатиме, звідки береться оригінальний IP, що ускладнює виявлення.[5]

Існує кілька методів, за допомогою яких можна змінити вихідний IP.

- TOP
- VPN

- Безкоштовні проксі
- Спільні проксі - найдешевші проксі, якими користуються багато користувачів. Шанси заблокувати великі.
- Приватні проксі-сервери - як правило, використовуються тільки вами, тож менші шанси заблокувати, якщо ви підтримуєте низьку частоту.
- Центри обробки даних, якщо вам потрібна велика кількість IP-адрес, найшвидші проксі-сервери, більші пули IP-адрес.

Можна почати з використання безкоштовних проксі-серверів, доступних в Інтернеті, але, потрібно пам’ятати, що оскільки ці проксі-сервери безкоштовні, то: багато людей користуються ними, вони повільні, вони часто виходять з ладу, вони вже заблоковані серверами або вони перевантажені.

Об'єкт типу requests має функціональні можливості для використання обертових проксі, для цього потрібно добавити атрибут proxies, наприклад:

r = requests.get('example.com', headers=headers, proxies={'https': proxy_url})

Отже, за допомогою зміни IP-адресу, зможемо обдурити сайт. Наприклад, побудувати алгоритм, при якому ми будемо брати безкоштовну IP-адресу, за допомогою парсингу сторінки, створювати пул IP-адрес, які будемо передавати в атрибут proxies.

Утворення штучних затримок.

Парсери переглядають данні дуже швидко, сайту дуже легко виявити, що це працює парсер, оскільки люди не можуть так швидко знайти, те що їм потрібно. Чим швидше парсер переглядає вміст сторінки, тим гірше для всіх. Якщо веб-сайт отримає забагато запитів, більше ніж він може обробити, парсер отримує помилку. Додаючи випадкові очікування часу між діями (наприклад, надсилання запитів, введення даних, клацання елементів тощо). Це дозволить рандомізувати схему перегляду та ускладнить пошук серверу, створивши розмежування між парсером та реальним користувачем. Це можна зробити, застосувавши простий тайм-аут. Як правило, гарний час очікування запиту, може становити близько 10 секунд. Функція sleep з модуля time ідеально підходить для таких простих тайм-

аутів. Функція sleep приймає ціле число або плаваюче число, яке представляє час у секундах. Записати можна таким чином:

time.sleep(5)

Добавити цю команду потрібно між діями скрипту, це означатиме на скільки часу буде відкладено виконання програми. Також більш схожим на людину, буде надсилання запитів із тимчасовими затримками на випадкові проміжки часу. Надсилання запитів з однаковим тайм-аутом може трактуватися як активність боту і блокуватися. Якщо ми замислюємось над тим, як ми переглядаємо Інтернет, запити чи клацання, як правило, відбуваються з випадковим інтервалом - ми натискаємо щось, читаємо трохи, потім клацнемо щось інше. Наступний приклад коду гарантує, що час очікування між кожною дією коливається між 5 і 10 секундами:

time.sleep(random.uniform(5, 10))

Переспрямування та Captcha.

Деякі веб-сайти перенаправляють свої посилань на новіші відображення (наприклад, перенаправляють HTTP посилання на ці HTTPS), повертаючи 3xx код у відповідь. Сучасні бібліотеки requests вже піклуються про HTTP-перенаправлення, переглядаючи їх (зберігаючи історію) і повертаючи кінцеву сторінку. Переадресація не становить особливих труднощів, якщо ми в кінцевому підсумку перенаправляємось на потрібну сторінку. Але якщо ми переспрямовані на Captcha чи на сторінку «honey pots», то це стає складно.

Captcha - це перевріка, яка використовується для обчислень вашу відповідь, у текстовому форматі де потрібно відтворити, які символи зображено на малюнку, щоб визначити, чи є користувач людиною. Досить популярний спосіб для боротьби з парсерами (рис. 2.2).

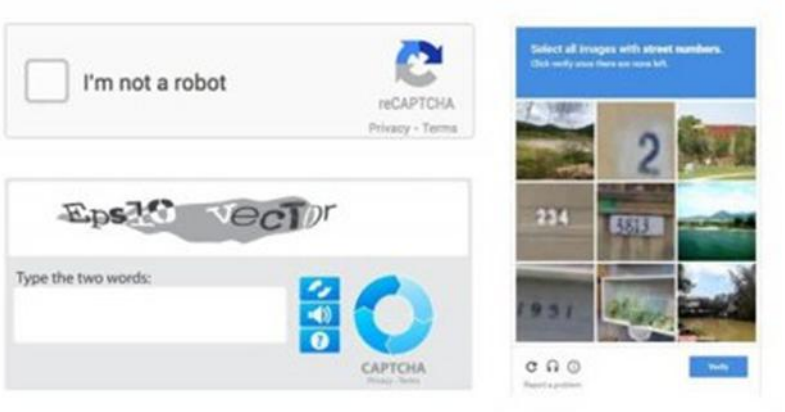


Рисунок 2.2 - Приклади Captcha

Укладнює процес розв'язання, те що зараз існує багато видів. Коли отримується Captcha, це зазвичай означає, що сайт побачив активність бота і надсилає перевірку. Їх звичайно можна вирішувати, але це забирає багато часу, краще передивитися, на якокому єтапі сайт знайшов, нашу програму. Якщо немає іншого шляху у python існує бібліотека, яка називається pytesseract, яка може вирішувати прості текстові капчі за допомогою OCR (оптичного розпізнавання символів).

Також коли буде працювати парсер є шанси, що ми зловимо так звану сторінку «honey pots». Honey pots - це HTML-посилання, яке користувач не бачить у веб-браузері, але до нього може отримати доступ веб-парсер. У випадку, коли ви намагаєтеся спарсити дані веб-сайту, надіславши запит на будь-яке з цих прихованих посилань, веб-сервер може виявити діяльність і відповідно заблокувати парсер. Простий спосіб обійти це за допомогою функції Selenium is_displayed. Оскільки Selenium відображає веб-сторінку, він перевірить наявність елемента візуально. Важливо дивитись на веб-сторінку в цілому, а не просто ігнорувати всі приховані поля. Також інколи до сайту ддодають посилання з нескінченно глибокими деревами каталогів. Тоді логічно було б обмежити кількість отриманих сторінок або обмежити глибину обходу.

На данному єтапі ми розглянули та висвітлили типові складності, які пов’язані з завнтаженням веб-сайтів, їх можливі способи вирішення, а також інструменти та бібліотеки, які ми можемо використовувати за допомогою мови Python. Парсинг трохи нагадує гру «котики-мишки», що діє в законній сірій зоні, і

може спричинити неприємності обом сторонам, якщо не робити це з повагою. Наступним шагом буде дістати потрібну нам інформацію. Отже, буде потрібно визначитись з інструментами, які є в Python для цього, та розібратися зі структурою веб-сторінки.

2.2 Витяг інформації

Зрозуміти як структурована веб-сторінка, ми можемо переглянути вихідний код. У більшості веб-браузерів вихідний код веб-сторінки можна переглянути, натиснути правою кнопкою миші на сторінці, та вибрати параметр переглянути код або натиснути клавішу F12. Перейшовши ми побачимо HTML код. Він складється з HTML тегів та інформацією, яку ми шукаємо.

Всі HTML-теги діляться на п'ять типів:

- порожні елементи - <area>, <base>,
, <col>, <embed>, <hr>, <img>, <input>, <link>, <menuitem>, <meta>, <param>, <source>, <track>, <wbr>;
- елементи з неформатований текстом - <script>, <style>;
- елементи, які виведуть звичайний текст - <textarea>, <title>;
- елементи з іншого простору імен - MathML і SVG;
- звичайні елементи - всі інші елементи.

За допомогою функції Inspector в веб-браузері можна виділяти об'єкт на сторінці та отримувати місцезнаходження потрібну інформцацію у коді(рис. 2.3).

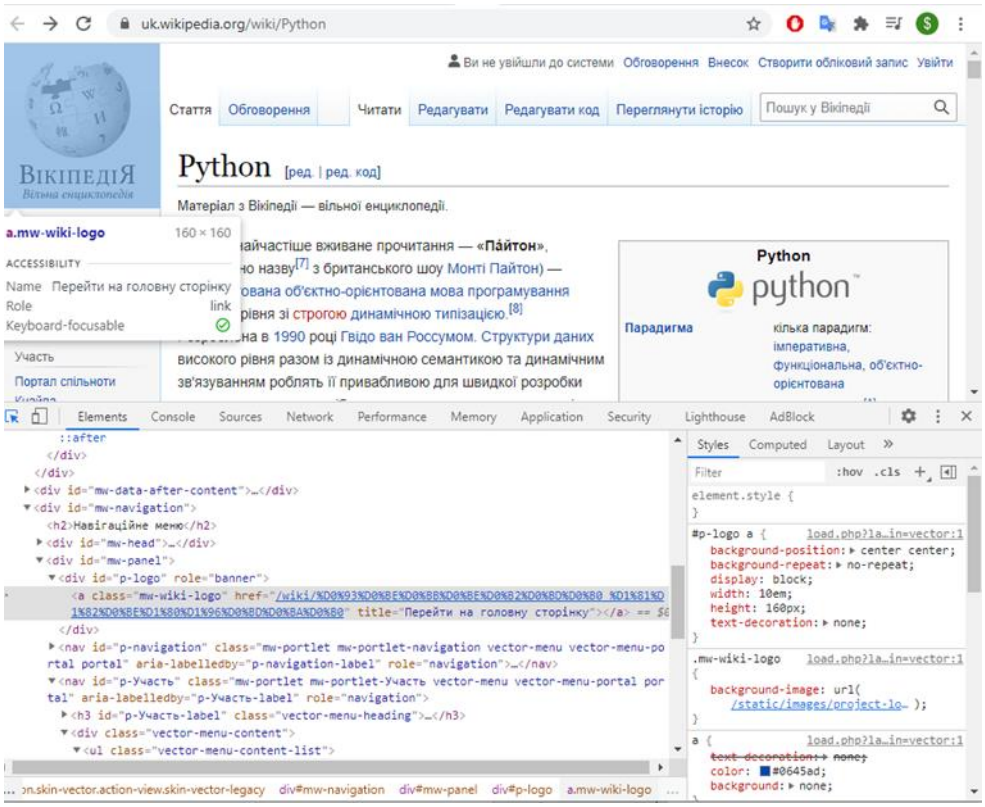


Рисунок 2.3 - Функція Inspector та код сторінки

Розібравшись зі структурою веб-сторінки, ми можемо притупити до синтасичного аналізу використовуючи три різні підходи.

Існує багато загальновживаних підходів до парсингу. Це дерево рішень допоможе нам визначитися з найкращим підходом для використання для певного веб-сайту(рис. 2.4).

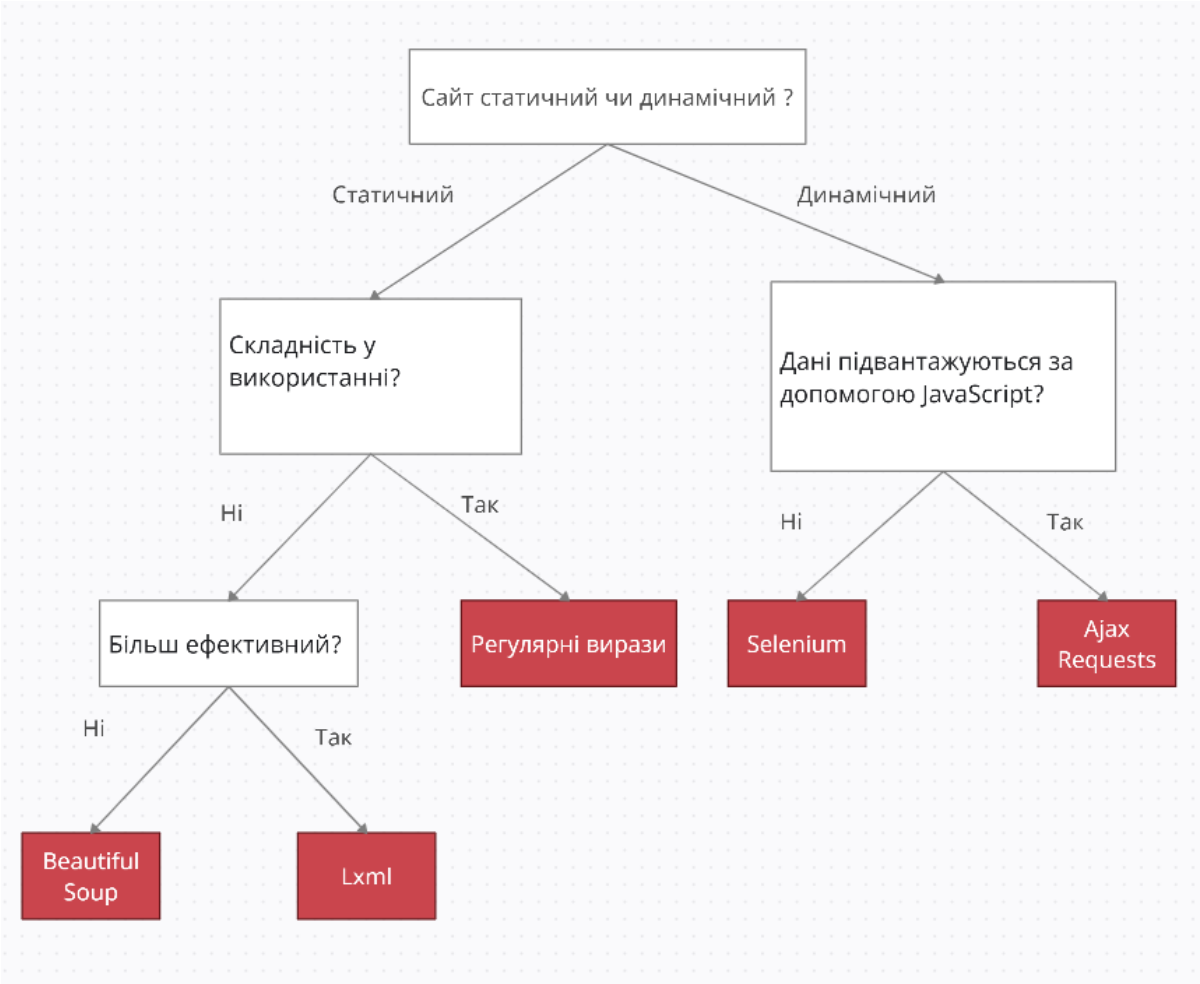


Рисунок 2.4 - Дерево рішень

Використання регулярних виразів.

Використання регулярних виразів для пошуку шаблонів HTML, як відомо, не рекомендується взагалі. Регулярні вирази - це інструмент, який недостатньо вдосконалений для розуміння конструкцій коду HTML. HTML не є звичайною мовою, тому його не можна аналізувати за допомогою регулярних виразів. Однак регулярні вирази все ще корисні для пошуку конкретних шаблонів рядків, таких як ціни, адреси електронної пошти або номери телефонів.

Імпортуйте бібліотеку та запустіть регулярний вираз у тексті програми, щоб знайти конкретні шаблони рядків:

```
import re
```

re.findall('<tr id="students_name_row"><td class="w2p_fl"><label for="students_name" id="students_name_label">Name:\</label></td><td class="w2p_fw">(.*?)</td>', html.text)

Регулярні вирази важко побудувати, так як вони нечитабельні. Крім того, наприклад, якщо розробники вирішать додати атрибут title, і на цьому наша функція вже не буде викновутись, потрібно заново переписувати регулярний вираз.

Використання BeautifulSoup.

BeautifulSoup є популярним модулем, який аналізує веб-сторінки, а також надає зручний інтерфейс для навігації контенту. Остання версія може бути встановлена за допомогою наступної команди:[11]

pip install beautifulsoup4

BeautifulSoup перетворює завантажений HTML- документ в soup- об'єкт. Більшість веб-сторінок не мають правильного HTML-коду і BeautifulSoup прекрасно це вирішує. У BeautifulSoup існує метод soup.prettify(), який генерує правильний код. BeautifulSoup може правильно інтерпретувати відсутні лапки атрибутів і закриваючі теги, а також додати недостающі теги, щоб сформувати повний HTML-документ. BeautifulSoup являє собою зручний інтерфейс, але не він не програє сильно в потужності, тому що можно використовувати встроєні синтаксичні аналізатори, такі як, lxml чи html.parser. Знайти елементи в коді можна використовуючи методи find() і find_all():

```
from bs4 import BeautifulSoup import requests
url = 'https://iqssdss2020.pythonanywhere.com/tutorial/static/views/Adams.html'
html = requests.get(url)
soup = BeautifulSoup(html.text, 'html.parser')
tr = soup.find_all(attrs={'id':'students_name_row'})
td = tr.find(attrs={'class':'w2p_fw'})
name = td.text print(name)
```

Порівнюючи код у якому використовується бібліотека BeautifulSoup з регулярними виразами, можна побачити, що код першого більш простий для розуміння та написання. Вважаючи те, що на сайтах інколи оновлюють структуру, змінювати алгоритм роботи такого скрипта, не стає серйозною проблемою.

Асинхронне завантаження.

Код сторінки, який ми отримуємо від серверу, може не містити інформації, яку ми очікували під час візуального огляду. Це відбувається тому, що інформація, яку ми насправді шукаємо, або відображається на стороні браузера такими бібліотеками, як Handlebars або React, або отримується шляхом здійснення майбутніх викликів AJAX на сервер, а потім їх надання браузером.

Кілька прикладів цього:

- Веб-сторінки з нескінченною прокруткою (Twitter, Facebook тощо)
- Веб-сторінки з попереднім завантажувачем, наприклад, відсоткові смуги або завантажувальні спінери.

Обійти асинхронне завантаження можна за допогою:

1. Використання веб-драйверу. Веб-драйвер - це як імітація браузера з інтерфейсом, яким керують за допомогою сценаріїв. Він здатний виконувати такі функції браузера, як рендеринг JavaScript, керування файлами cookie та сеансами тощо. Selenium Web Driver - це веб-система автоматизації, призначена для тестування користувацького інтерфейсу веб-сайтів, але вона також стала популярною з часом опцією для парсингу динамічно відтворюваних сайтів. Веб - драйвери симуляція браузерів ресурсномісткий та в порівнянні з бібліотеками beautifulsoup та scrapy повільніші. Selenium підтримує кілька мов для сценаріїв, включаючи Python. Зазвичай він запускає екземпляр браузера, і ми можемо бачити такі речі, як клацання та введення даних на екрані, що корисно під час тестування. Але якщо ми піклуємося лише про парсинг, ми можемо використовувати "безголові браузери", які не мають інтерфейсу користувача і швидші з точки зору продуктивності. Наприклад, Chrome Headless Headless

Firefox , PhantomJS , spynner та HtmlUnit. Деякі з них можуть вимагати встановлення віртуального фреймбуферу xvfb та його обгортки Python (xvfbwrapper або pyvirtualdisplay) для імітації відображення екрана у віртуальній пам'яті, не видаючи фактичного виводу на екран.

2. Перевірка викликів AJAX. Цей метод працює за ідеєю: "Якщо він відображається у браузері, то він повинен братись звідкись". Ми можемо використовувати інструменти розробника браузера для перевірки викликів AJAX і спробувати з’ясувати, що запити відповідають за отримання даних, які ми шукаємо. Можливо, нам доведеться встановити X-Requested-With заголовок для імітації запитів AJAX у вашому сценарії.

2. Боротьба з нескінченною прокруткою. Ми можемо вирішити нескінченну прокрутку, вводячи деяку логіку javascript у Selen. Крім того, зазвичай нескінченна прокрутка включає подальші виклики AJAX до сервера, які ми можемо перевірити за допомогою інструментів браузера та відтворити в нашому парсері.

2.3 Збереження інформації

Після обробки даних, наступне завдання - правильно зберегти отримані дані і відобразити їх користувачев. Отже, нам потрібно зберігати невелику кількість даних, а так як складних операцій з БД не передбачається, будемо використовувати SQLite базу даних.

SQLite — полегшена реляційна система керування базами даних. Особливістю SQLite є те, що вона не використовує парадигму клієнт-сервер, тобто рушій SQLite не є окремим процесом, з яким взаємодіє застосунок, а надає бібліотеку, з якою програма компілюється і рушій стає складовою частиною програми. Таким чином, як протокол обміну використовуються виклики функцій (API) бібліотеки SQLite. Такий підхід зменшує накладні витрати, час відгуку і спрощує програму. SQLite зберігає всю базу даних (включаючи визначення, таблиці, індекси і дані) в єдиному стандартному файлі на тому комп'ютері, на якому виконується застосунок. Простота реалізації досягається за рахунок того,

що перед початком виконання транзакції весь файл, що зберігає базу даних, блокується; ACID-функції досягаються зокрема за рахунок створення файлу-журналу.

У комплекті постачання йде також функціональна клієнтська частина у вигляді виконуваного файлу sqlite3, за допомогою якого демонструється реалізація функцій основної бібліотеки. Клієнтська частина працює з командного рядка, і дозволяє звертатися до файлу БД на основі типових функцій ОС.

З цією базою данних дуже зручно користуватися, за допомогою Python та бібліотеки sqlite3. SQLite3 - це C бібліотека, яка реалізує легковажну дискову базу даних (БД), яка потребує окремого серверного процесу і дозволяє отримати доступ до БД з використанням мови запитів SQL. Деякі додатки можуть використовувати SQLite для внутрішнього зберігання даних. Також можливо створити прототип додатка з використанням SQLite, а потім перенести код в більш багатофункціональну БД, таку як PostgreSQL або Oracle.

2.4 Відправити дані до Telegram

Telegram - кросплатформенний клауд-месенджер для смартфонів, планшетів та ПК, який дозволяє обмінюватися текстовими повідомленнями й медіфайлами.

Для месенджера був створений протокол MTProto, що передбачає використання декількох протоколів шифрування. MTProto API (він же Telegram API) – це API через який ваш додаток Telegram зв'язується з серерами. Telegram API має відкритий код, тому кожен може написати власний кліент месенджеру.[9]

Для написання ботів був створений Telegram Bot API - надбудова над Telegram API. Щоб використовувати Bot API, не потрібно нічого знати про те, як працює протокол шифрування MTProto - допоміжний сервер Telegram буде сам обробляти всі шифрування і зв'язок з Telegram API. Кожен розробник з'єднується з сервером через простий HTTPS-інтерфейс, який надає просту версію Telegram API.[8]

Telegram-бот повинен вміти відправляти запити до серверу і отримати від нього updates. Звичайно, зручніше використовувати бібліотеки, ніж робити http-

запити "руками". На мові Python є декілька бібліотек для керування ботами, наприклад, python-telegram-bot або telebot, але серед розробників ботів кращою вважається aiogram. Вона асинхронна, використовує декоратори і містить зручні інструменти для розробки.

Висновки розділу 2

В цьому розділі було проведено аналіз можливостей мови програмування Python для створення технології синтаксичного аналізу. Розглянуто, як за допомогою Python можна вирішити питання з обмеженнями, які створюють штучно власники сайтів. Проаналізовано бібліотеки для роботи з даними.

3 РОЗРОБКА ТЕХНОЛОГІЇ СИНТАКСИЧНОГО АНАЛІЗУ ВМІСТУ WEB-САЙТУ МОВОЮ PYTHON

Для розробки технології була вибрана мова Python, бібліотеки для парсингу requests та BeautifulSoup, база даних SQLite та для відстеження даних Telegram Bot.

Реалізована технологія реалізує систему парсингу даних та підписок користувачів в системі. Основі функції системи:

- збір та аналіз інформації на сайті
- зберігання даних користувачів (Chat id та статус підписки) у базу даних
- відправку данних за запитом або за підпискою до бота Telegram

3.1. Розробка функціоналу технології

Першим єтапом буде збір інформації з сайту та формування баз даних. Система буде включати два парсери. Перший реалізуємо наступним чином: коли на вкладці ігрові огляди публікується новий запис, то відправляємо стислу інформацію про цей запис, всім кто підписався на бота в Telegram. Другий буде збирати по всім категоріям найкращі ігри, зберігати у базі даних та відправляти перелік по запиту користувача.

Для того, щоб приступити, до написання першого парсеру спочатку подивимось на сторінку сайту і створимо алгоритм парсингу(рис. 3.1-3.2).

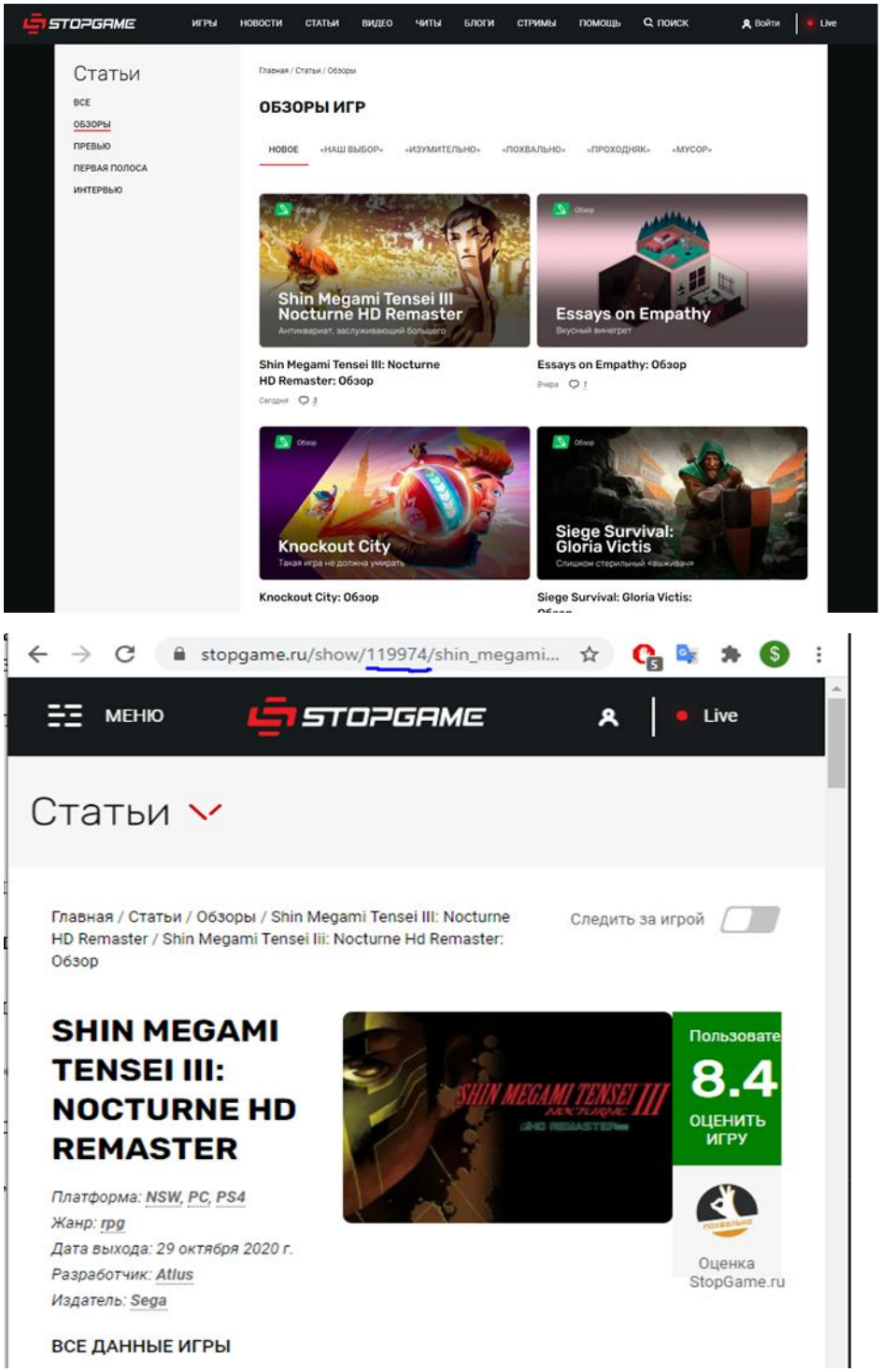


Рисунок 3.1-3.2 - Аналіз сторінок сайту для першого сайту

Проаналізувавши сайт ми побачимо, що при переході на запис у нього є власний номер. Використовуючи цю інформацію, наш парсер буде працювати наступним чином, він буде перевіряти чи є на сайті нові записи, якщо записів нових немає, то він нічого не робить, якщо нові записи з'явилися, то робимо розсилку всім підписникам. Для того, щоб дізнатися, це новий запис, чи ні, ми

будемо запам'ятовувати останій ідентифікатор запису, по якому ми робили розсилку і коли на сайті додадуть новий запис, ми порівняємо їх, і якщо вони не збігаються, тоді це новий ігровий огляд. Створимо файл gameArticles.py та імпортуємо модулі, які будемо використовувати:

```
import re
import os.path
import requests
from bs4 import BeautifulSoup as BS
from urllib.parse import urlparse
import config
```

Далі створимо класс, у якому створимо функції, для перевірки збігу:

```
class parseGame:
lastkey = ""
lastkey_file = ""
def __init__(self, lastkey_file):
    self.lastkey_file = lastkey_file
    if(os.path.exists(lastkey_file)):
        self.lastkey = open(lastkey_file, 'r').read()
    else:
        f = open(lastkey_file, 'w')
        self.lastkey = self.get_lastkey()
        f.write(self.lastkey)
        f.close()
def new_games(self):
    r = requests.get(config.url,config.header)
    html = BS(r.content, 'html.parser')
    new = []
    items = html.select('.tiles > .items > .item > a')
    for i in items:
```

```
                key = self.parse_href(i['href'])
                if(self.lastkey < key):
                        new.append(i['href'])
        return new
def get_lastkey(self):
        r = requests.get(config.url,config.header)
        html = BS(r.content, 'html.parser')
        items = html.select('.tiles > .items > .item > a')
        return self.parse_href(items[0]['href'])
def parse_href(self, href):
        result = re.match(r'\/show\/(\d+)', href)
        return result.group(1)
def update_lastkey(self, new_key):
        self.lastkey = new_key
        with open(self.lastkey_file, "r+") as f:
                data = f.read()
                f.seek(0)
                f.write(str(new_key))
                f.truncate()
        return new_key
```

Отже, коли отримали, що є нові записи, потрібно зібрати інформацію публікації. Ми будемо збирати наступну інформацію із сторнки: ідентифікатор запису, назву, посилання, зображення, оцінку та уривок від публікації. Додамо до нашого класу наступні функції:

```
def game_info(self, uri):
        link = config.host + uri
        r = requests.get(link,config.header)
        html = BS(r.content, 'html.parser')
        # parse poster image url
```

```
            poster = re.match(r'background-image:\s*url\((.+?)\)', html.select('.image-game-logo > .image')[0]['style'])
            info = {
                "id": self.parse_href(uri),
                "title": html.select('.article-title > a')[0].text,
                "link": link,
                "image": poster.group(1),
                "score": self.identify_score(html.select('.game-stopgame-score > .score')[0]['class'][1]),
                "excerpt": html.find('section',class_='article article-show').find('p').text[0:200] + '...'
            }
            return info
        def download_image(self, url):
            r = requests.get(url, allow_redirects=True)
            a = urlparse(url)
            filename = os.path.basename(a.path)
            open(filename, 'wb').write(r.content)
            return filename
        def identify_score(self, score):
            if(score == 'score-1'):
                return "Мусор  "
            elif(score == 'score-2'):
                return "Проходняк "
            elif(score == 'score-3'):
                return "Похвально  "
            elif(score == 'score-4'):
                return "Изумительно  "
```

Отже, ми створили перший парсер, перейдемо до наступного. Так як, із першим спочатку проаналізуємо сторінку з категоріями(рис. 3.3). Потрібно пройтись по всім категоріям та зібрати найкращі ігри.

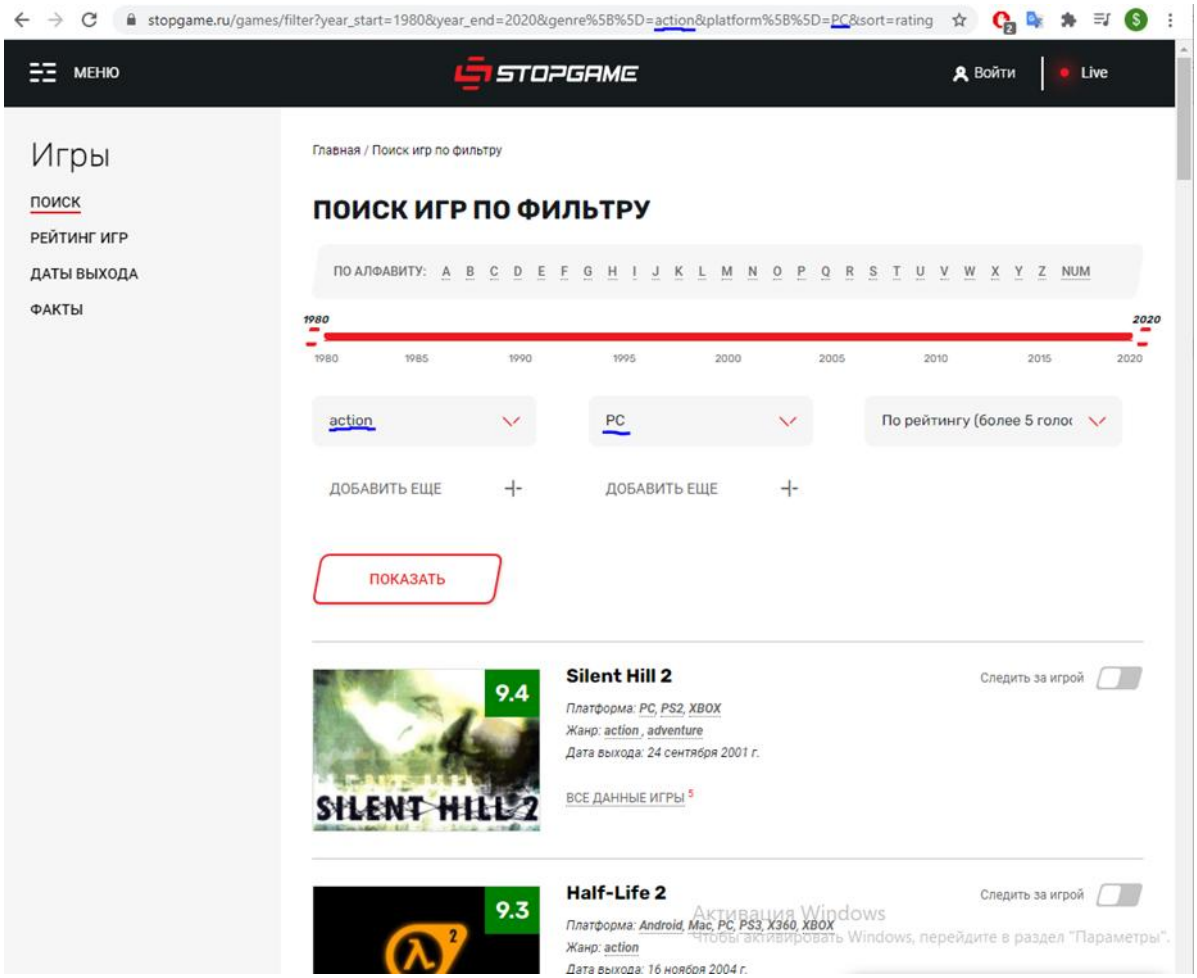


Рисунок 3.3 – Сторінка з категоріями

Проаналізувавши можна зробити висновок, що коли ми вводимо параметри використовучи інфтерфейс сайту змінюється URL адреса. Потрібно, перевірити та вручну змінити URL, щоб побачити, використовує сайт javascript чи ні. У нашому випадку можно змінювати URL відповідно до категорії.

Такі параметри, як категорії, не будуть змінюватися, а використовуються часто, то гарним тоном записати в окремий файл. Створимо файл config.py з такими параметрами:

```
host = 'https://stopgame.ru'
url = 'https://stopgame.ru/review/new'
```

header = {'user-agent': ' Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/90.0.4430.212 Safari/537.36 ' }

list= ['action','adventure','arcade','cards','educational','logic','online','racing','rpg','simulator','sport','strategy']

Отже, перейдемо до парсингу, алгоритм дуже простий, будемо проходити по всім категоріям списку list, змінючи кожною ітерацією URL адресу, парсити інформацію та записувати данні в базу даних. Це можна реалізувати наступним кодом.

```
import sqlite3
import requests
from bs4 import BeautifulSoup
import re
import config
def searchData():
    for i in config.list:
        url = 'https://stopgame.ru/games/filter?year_start=1980&year_end=2020&genre%5B%5D={}&platform%5B%5D=PC&sort=rating'.format(i)
        r = requests.get(url,config.header)
        soup = BeautifulSoup(r.text,'lxml')
        items = soup.find('div',class_= 'main-content').find('div',class_= 'simple-list games-list').find('div',class_ = 'items').find_all('div',class_='item game-summary game-summary-horiz')[:10]
        num = 0
        for item in items:
            num = num +1
            name = item.find('div',class_='details').find('div',class_='caption caption-bold').find('a').text
```

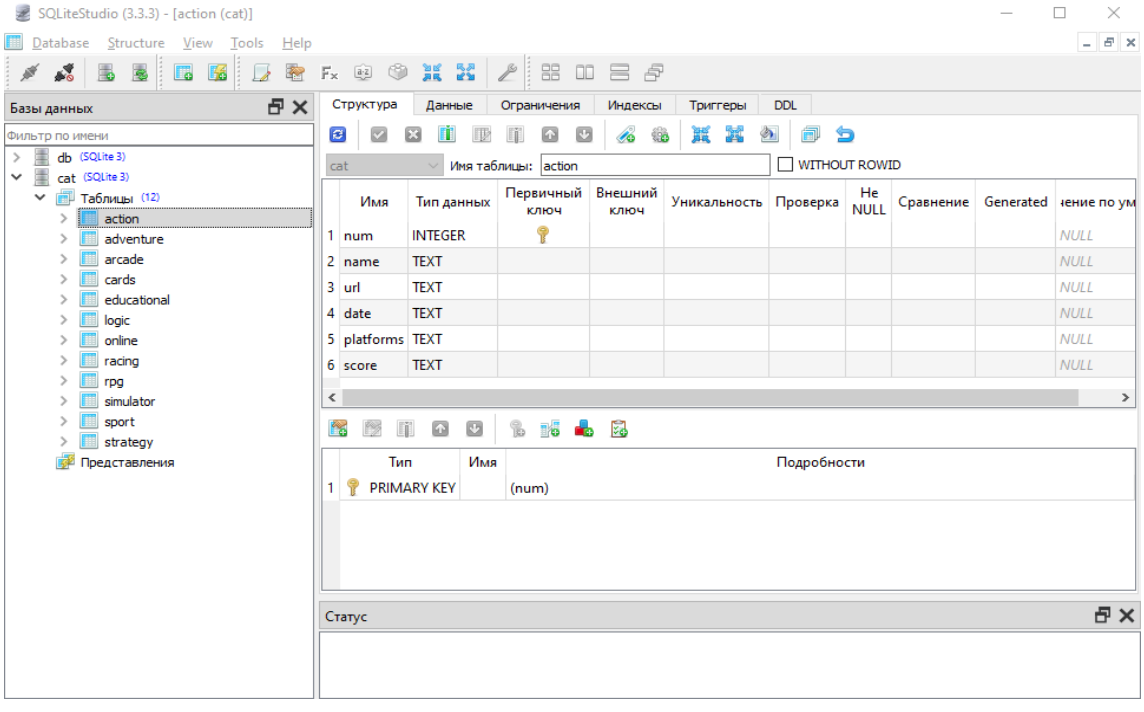
```
            url = config.host + item.find('a').get('href')
            date = item.find('div',class_='details').find_all('div',class_='game-spec')[2].find('span',class_='value').text
            platforms = item.find('div',class_='details').find('div',class_='game-spec').find('span',class_='value').text
            sc = item.find('a').find('div', class_ = 'score').text
            score = re.sub('[\n ]','',sc)
            data = ((num,name,url,date,platforms,score))
            writeData(data,i)
```

Але, щоб записати данні в базу даних, потрібно спочатко її створити. Коли створили файл "cat.db", можна підлючитися за допомогою модуля sqlite3. Будемо відправляти sql запити щоб стврити, змінити чи дістати данні.

```
def createDataBase():
    for i in config.list:
        conn = sqlite3.connect("cat.db")
        sql = "CREATE TABLE IF NOT EXISTS {} (num INTEGER, name TEXT, url TEXT, date TEXT, platforms TEXT, score TEXT, primary key(num))".format(i)
        cursor = conn.cursor()
        cursor.execute(sql)
        conn.commit()
        conn.close()
def writeData(data,i):
    conn = sqlite3.connect('cat.db')
    cursor = conn.cursor()
    cursor.execute("INSERT  INTO {} VALUES (?,?,?,?,?,?)".format(i),data )
    conn.commit()
    conn.close()
def get_subcategories(category):
```

```
conn = sqlite3.connect('cat.db')
cursor = conn.cursor()
sql = '''SELECT * FROM {} '''.format(category)
cursor.execute(sql)
rec = cursor.fetchall()
conn.commit()
conn.close()
return rec
```

Виконавши цей код ми отримаємо базу даних, яку ми відкриємо за допомгою SQLite. Вона має такий вигляд(рис. 3.4-3.5).



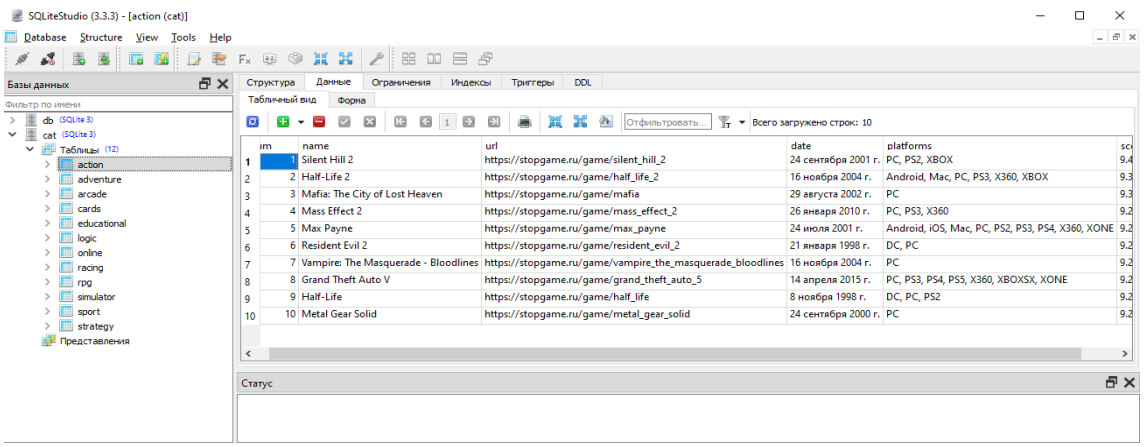


Рисунок 3.4-3.5 – Відображення бази данних в SQLiteStudio

Також розробимо окремий класс, за допомогою якого ми будемо зерігати данні про підписників у базу даних (рис. 3.6):

```
import sqlite3
class SQLighter:
    def __init__(self, database):
        self.connection = sqlite3.connect(database)
        self.cursor = self.connection.cursor()
    def get_subscriptions(self, status = True):
        with self.connection:
            return self.cursor.execute("SELECT * FROM `subscriptions` WHERE `status` = ?", (status,)).fetchall()
        def subscriber_exists(self, user_id):
            with self.connection:
                result = self.cursor.execute('SELECT * FROM `subscriptions` WHERE `user_id` = ?', (user_id,)).fetchall()
                return bool(len(result))
    def add_subscriber(self, user_id, status = True):
            with self.connection:
                return self.cursor.execute("INSERT INTO `subscriptions` (`user_id`, `status`) VALUES(?,?)", (user_id,status))
        def update_subscription(self, user_id, status):
            with self.connection:
                return self.cursor.execute("UPDATE `subscriptions` SET `status` = ? WHERE `user_id` = ?", (status, user_id))
        def close(self):
            self.connection.close()
```

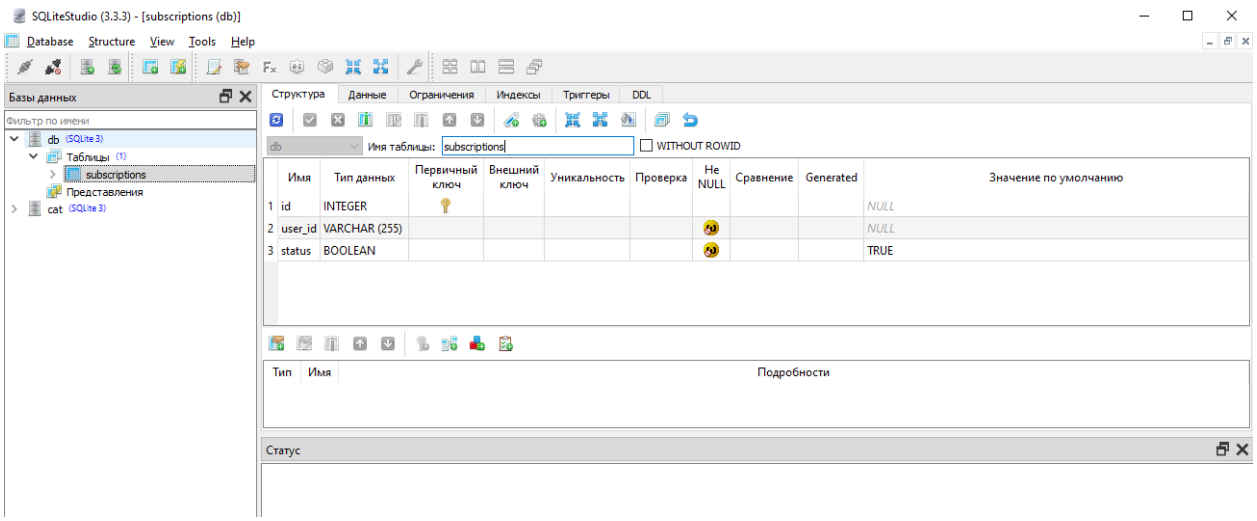


Рисунок 3.6 – Відображення бази данних в SQLiteStudio

У нас є бази данних та парсери контенту, залишилось тільки ініціювати процес та відправляти данні за допомогою бота. Для цього потрібно створити та налаштувати бота Telegram. Потрібно перейти в месенджер Telegram знайти і додати до своїх контактів @BotFather і ввести команду /start. Далі потрібно написати команду /newbot для створення нового бота. Та відправити назву для бота, це може бути будь-яке ім'я, але має закінчуватися словом bot (або Bot), це вимога BotFather. У результаті створення він дасть нам API token бота, який вже ми будемо використовувати у нашій програмі за допомгою бібліотеки Python aiogram. Перед тим, як приступити до коду створимо відразу команду menu(рис. 3.7).

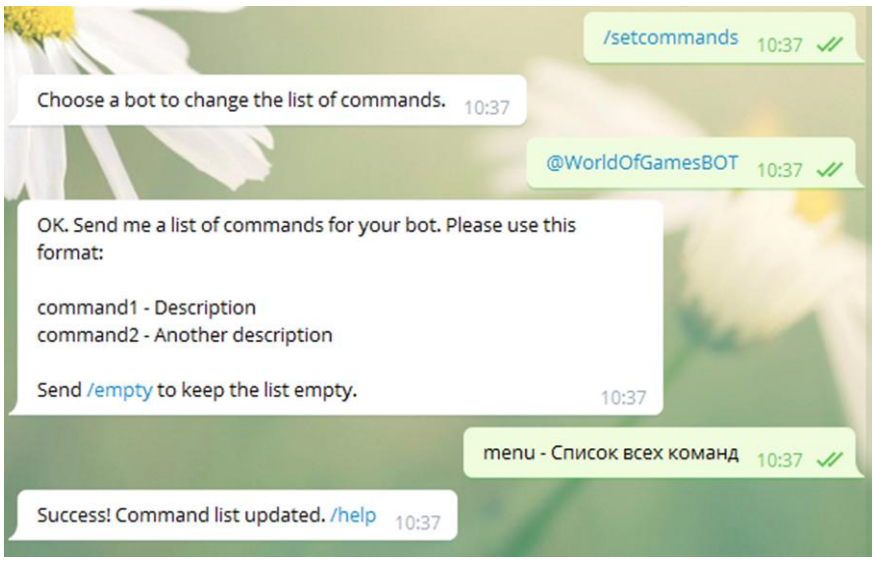


Рисунок 3.7 – Створити команду

Коли користувач відправить запит menu, ми будемо обробляти цей запит та відправляти клавіатуру з такими полями: підписатись, відписатись та категорії. Зробимо окремий файл keybord.py, імпортуємо потрібні модулі та створимо клавіатуру:

```
from aiogram.types import ReplyKeyboardMarkup, KeyboardButton
from config import list
menu = ReplyKeyboardMarkup(
keyboard = [[KeyboardButton(text = 'Подписаться'),
                KeyboardButton(text = 'Отписаться'),],
      [KeyboardButton(text = 'Категории')],  ],
resize_keyboard=True
)
```

ReplyKeyboardMarkup - це текстові повідомлення. Наприклад, бот задає користувачеві питання і пропонує варіанти відповіді. Користувач може самостійно надрукувати відповідь, або натиснути на готову кнопку. Така клавіатура показується замість основної. Отже, така клавіатура підходить також для категорій, тому добавимо до цього коду, ще кнопки:

```
button1 = KeyboardButton('{}'.format(list[0]))
button2 = KeyboardButton('{}'.format(list[1]))
…
button11 = KeyboardButton('{}'.format(list[10]))
button12 = KeyboardButton('{}'.format(list[11]))
markup_all=
ReplyKeyboardMarkup(resize_keyboard=True,one_time_keyboard=True).insert(
    button1).insert(button2).insert(button3).insert(
    button4).insert(button5).insert(button6).insert(
    button7).insert(button8).insert(button9).insert(
    button10).insert(button11).insert(button12)
```

Залишилось тільки зібрати все до купи. Створимо окремий файл bot.py та імпортуємо всі потрібні модулі та створимо об'єкти класів: \

```
import config
import categories
import keybord
from sqlighter import SQLighter
from gameArticles import parseGame
import logging
import asyncio
from aiogram import Bot, Dispatcher, executor, types
bot = Bot(token=config.API_TOKEN)
dp = Dispatcher(bot)
db = SQLighter('db.db')
pg = parseGame('lastkey.txt')
kb = keybord
```

Далі потрібно створити для кожного запиту обробники, які відповідно будуть відправляти в telegram повідомлення і наш шаблон клавітури. Спочатку напишимо обробники для таких команд /start та /menu(рис. 3.8):

```
@dp.message_handler(commands=['start'])
async def start_message(message: types.Message):
await message.answer('Добро пожаловать.Это новостной бот нажимай /menu')
@dp.message_handler(commands=['menu'])
async def start_message(message: types.Message):
await message.answer( '…', reply_markup=kb.menu)
```

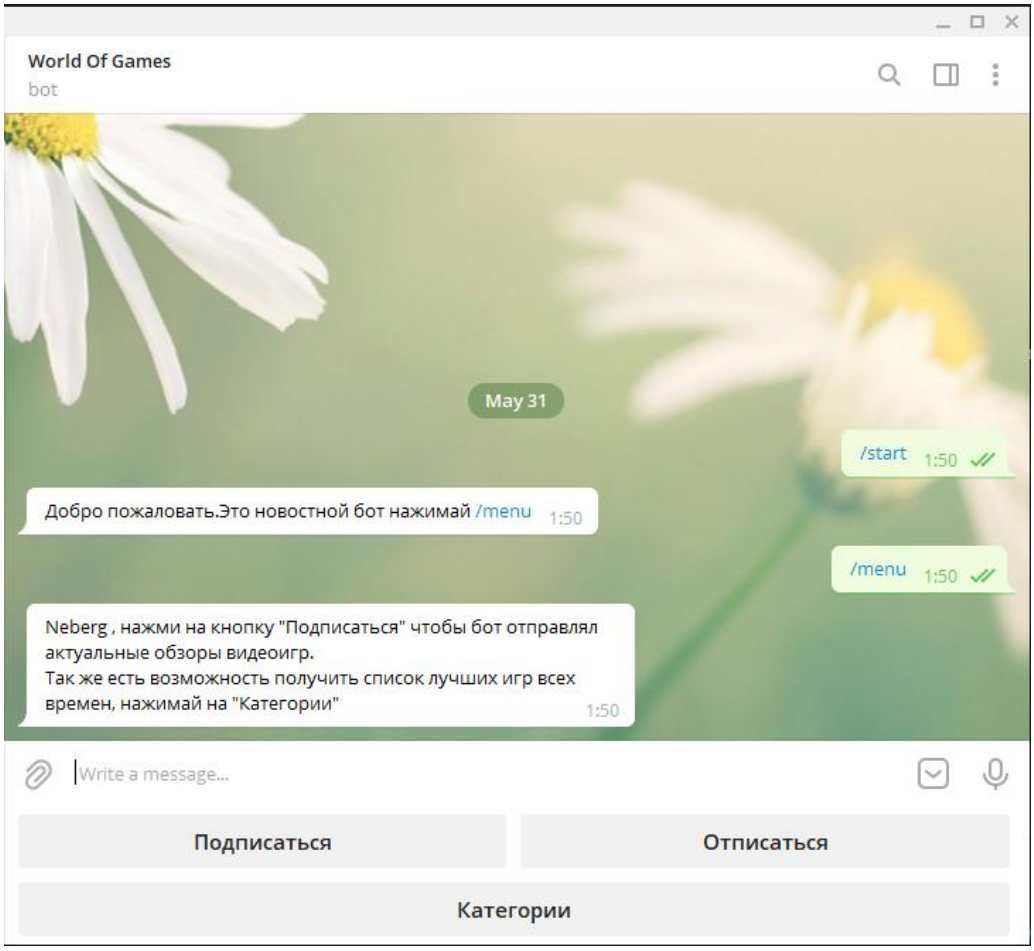
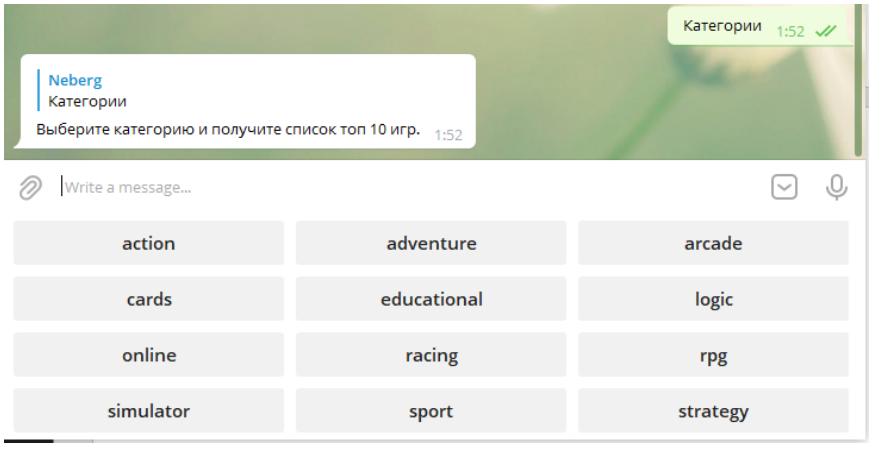


Рисунок 3.8 – Відображення роботи запитів та клавіртури menu

Наступним кроком буде створення обробників для кожного натискання на кнопки клавіатури. При натисканні підписатись, користувач передає Chat id, який ми записуєм в базу db.db та оновлюємо статус підписки. Кнопка відписатись, змінює статус підписки на 0. Натискаючи на кнопку категорія, користувач отримає клавіатуру з категоріями (рис. 3.9-3.10). Користувач при натисканні на одну з категорій буде отримувати інформацію з бази данних cat.db.

```
@dp.message_handler(text=['Категории'])
async def best(message: types.Message):
    await message.reply("Выберите категорию и получите список топ 10 игр. ", reply_markup=kb.markup_all)
@dp.message_handler(filters.Text)
async def unsubscribe(message: types.Message):
if message.text == "Подписаться":
```

```
            if(not db.subscriber_exists(message.from_user.id)):
                db.add_subscriber(message.from_user.id)
            else:
                db.update_subscription(message.from_user.id, True)
            await message.answer("Вы успешно подписались на рассылку!\nЖдите, скоро выйдут новые обзоры и вы узнаете о них первыми")
        elif message.text == "Отписаться":
            if(not db.subscriber_exists(message.from_user.id)):
                db.add_subscriber(message.from_user.id, False)
                await message.answer("Вы и так не подписаны.")
            else:
                db.update_subscription(message.from_user.id, False)
                await message.answer("Вы успешно отписаны от рассылки.")
        elif message.text == "action":
            await message.reply(categories.create_text(message.text))
        elif message.text == "adventure":
            await message.reply(categories.create_text(message.text))
        …
        else:
            await message.answer('Нажмите /menu')
```



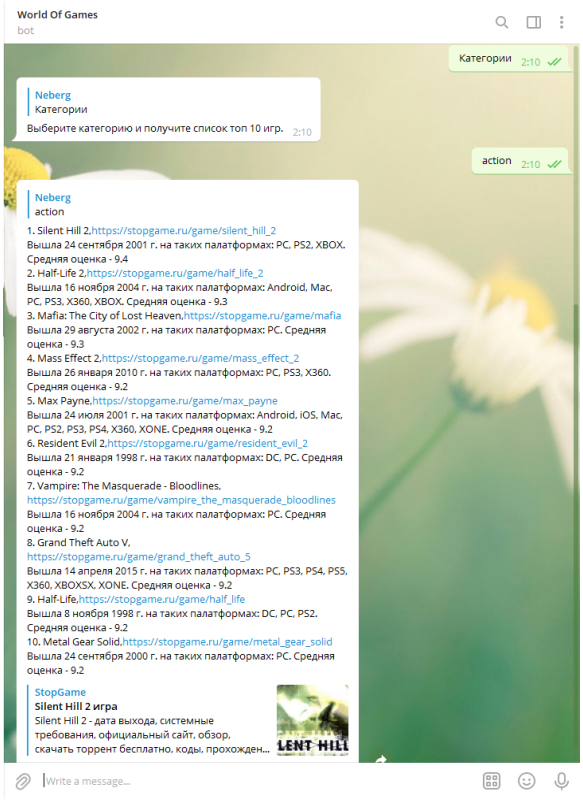


Рисунок 3.9-3.10 - Відображення роботи клавітури категорій

Залишилось ініціювати та описати функцію, яка пеервіряє нові статті та робить розсилку всім кто має підписку:

```
async def scheduled(wait_for):
while True:
        await asyncio.sleep(wait_for)
        new_games = pg.new_games()
        if(new_games):
            new_games.reverse()
            for ng in new_games:
                nfo = pg.game_info(ng)
                subscriptions = db.get_subscriptions()
                with open(pg.download_image(nfo['image']), 'rb') as photo:
                    for s in subscriptions:
                        await bot.send_photo(
                            s[1],
```

```
                                        photo,
                                        caption = nfo['title'] + "\n" + "Оценка: " +
nfo['score'] + "\n" + nfo['excerpt'] + "\n\n" + nfo['link'],
                                        disable_notification = True
                                    )
                                pg.update_lastkey(nfo['id'])
```

Коли виходить нова публікація, користувачі, які підписані отримують повідомлення про них (рис. 3.11).

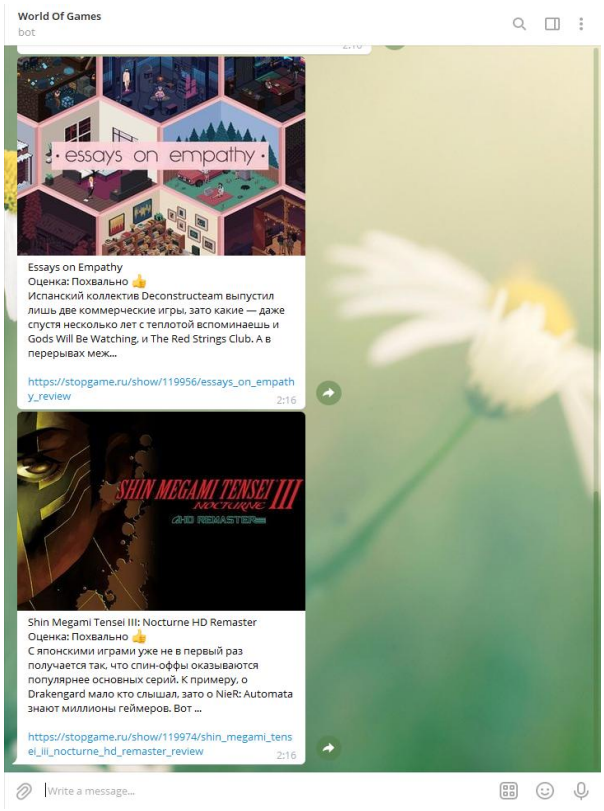


Рисунок -3.11 - Приклад повідомлення при нових публікаціях.

1.4 Висновки розділу 3

В третьому розділі дипломної роботи було створено технологію на мові програмування Python, що робить синтаксичний аналіз потрібних сторінок сайту, записує данні у базу даних та відправляє за запитом користувача данні до Telegram Bot.

Технологія реалізує такі функції:

- збір та аналіз інформації на сайті
- зберігання даних користувачів (Chat id та статус підписки) у базу даних
- відправку данних за запитом або за підпискою до бота Telegram

Описано структуру програми. Показано результати роботи програми.

ВИСНОВКИ

В роботі досліджено використання технології синтаксичного аналізу веб-сайтів. Досліджено проблеми, які можуть виникнути при синтаксичного аналізі та методи вирішення цих проблем за допогою мови Python. Створенно технологію з використанням бібліотек Python, яка включатє в себе парсер контенту, базу даних та зручний засіб для користування отриманим данними. Отримані дані використовуються за допомогою боту в Telegram за посиланням @WorldOfGamesBOT, що дає змогу отримувати потрібну інформацію не витрачаючи на це велику кількість часу.

В першому розділі було проаналізовано та розглянуто актуальність даної роботи. Визначена мета роботи. Було визначено та досліджено, що таке парсинг. Визначено засоби для створення парсерів. Дослідженно основні етапи парсингу та коротко описанно основні властивості. Визначенно, які методи використовують для захисту веб-сайтів від парсингу.

В другому розділі було проведено аналіз можливостей мови програмування Python для створення технології синтаксичного аналізу. Розглянуто, як за допомогою Python можна вирішити питання з обмеженнями, які створють штучно власники сайтів. Проаналізовано бібліотеки для роботи з даними.

В третьому розділі дипломної роботи було створено технологію на мові програмування Python, яка робить синтаксичний аналіз потрібних сторінок сайту, записує данні у базу даних та відправляє за запитом користувача данні до Telegram Bot.

Технологія реалізує такі функції:

- збір та аналіз інформації на сайті
- зберігання даних користувачів (Chat id та статус підписки) у базу даних
- відправку данних за запитом або за підпискою до бота Telegram

Описано структуру програми. Показано результати роботи програми.