МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ ВОДНОГО ГОСПОДАРСТВА ТА ПРИРОДОКОРИСТУВАННЯ
Навчально-науковий інститут автоматики, кібернетики та обчислювальної техніки
Кафедра комп'ютерних наук та прикладної математики

«До захисту допущена»
Завідувач кафедри комп'ютерних наук та прикладної математики
_____Турбал Ю.В.
«_____» _____ 2023 р.

КВАЛІФІКАЦІЙНА РОБОТА

на тему:

«Дослідження методів видобування даних з веб-ресурсів. Система бронювання віз на сайті vfsglobal.com»

Виконав: Власик Денис Вікторович _____
(прізвище, ім'я, по батькові) (підпис)
студент групи ІПЗ-41

Керівник: к.т.н., доц. Климюк Ю. Є. _____
(науковий ступінь, вчене звання, прізвище, ініціали) (підпис)

Рівне – 2023

ЗМІСТ

РЕФЕРАТ

Кваліфікаційна робота: 61 сторінка, 9 рисунків, 1 додаток, 13 джерел.

Актуальність: Зростаюча залежність від веб-ресурсів для різних цілей, в тому числі для подання візових заявок, вимагає проведення дослідження ефективних і дієвих методів видобування даних. Актуальність теми полягає у вирішенні проблеми, з якою стикаються користувачі під час запису на візовий прийом через веб-сайт vfsglobal.com, що полягає в його обмеженій доступності, та розробці бота, який може автоматизувати сам процес бронювання. Це дослідження має на меті надати практичне рішення, яке заощадить час і покращить досвід клієнтів.

Мета дослідження. Метою цього дослідження є вивчення та аналіз методів отримання даних з веб-ресурсів, з особливим акцентом на системі бронювання віз на веб-сайті vfsglobal.com. Основним завданням є розробка бота, який може ефективно та швидко забронювати відповідні дати візових зустрічей для користувачів, які стикаються з обмеженою доступністю та часовими обмеженнями на платформі vfsglobal.com.

Об’єкт дослідження. Об'єктом дослідження в цьому дослідженні є система бронювання віз на сайті vfsglobal.com. Основна увага приділяється розумінню обмежень і проблем, пов'язаних з доступом і витяганням даних з цього конкретного веб-ресурсу. Вивчаючи тонкощі системи бронювання візових зустрічей, це дослідження має на меті виробити комплексне розуміння процесу вилучення даних та визначити потенційні сфери для вдосконалення.

Предмет дослідження. Предметом дослідження є методи вилучення даних з веб-ресурсів, з особливим акцентом на веб-сайті vfsglobal.com та його системі бронювання віз. У дослідженні розглядаються різні методи, інструменти та стратегії, що використовуються для вилучення даних з веб-

ресурсів, з метою розробки бота, який може ефективно переміщатися по веб-сайту, отримувати відповідні дані про зустрічі та автоматизувати процес бронювання.

Методи дослідження. Для досягнення вищезазначених цілей буде використано поєднання якісних та кількісних методів дослідження. Дослідження включатиме ретельний огляд літератури, щоб отримати уявлення про існуючі дослідження та підходи до вилучення веб-даних. Крім того, збір даних включатиме розробку та впровадження бота, який взаємодіє з веб-сайтом vfsglobal.com і збирає відповідні дані про візові зустрічі. Ефективність роботи бота буде оцінюватися за допомогою тестування та аналізу, включаючи кількісні показники, такі як рівень успішності, час відгуку та точність.

Методи вивчення – чисельні методи аналізу та оптимізації даних: попередня обробка даних, інтелектуальний аналіз та вилучення даних, статистичний аналіз, алгоритми оптимізації, обчислювальні інструменти та бібліотеки.

Новизна та практична цінність одержаних результатів. Новизна цього дослідження полягає в розробці спеціалізованого бота для автоматизації процесу бронювання віз на сайті vfsglobal.com. Хоча методи вилучення веб-даних досліджувалися в різних контекстах, конкретне застосування до бронювання візових зустрічей та веб-сайту vfsglobal.com є унікальним внеском у цю сферу. Практична цінність отриманих результатів полягає в розробці ефективного бота, який може заощадити час і зусилля осіб, які записуються на візові зустрічі. Уроки, винесені з цього дослідження, можуть бути застосовані до подібних сценаріїв, покращуючи розуміння та впровадження методів вилучення веб-даних для покращення користувацького досвіду та оптимізації процесів.

Ключові слова: видобування даних, веб-скрапинг, веб-ресурси, vfsglobal.com, бот, аналіз даних, динамічний вміст, заходи захисту від ботів,

рендеринг JavaScript, веб-інтерфейси API, збір даних, аналіз веб-даних, бронювання, автоматизація, python, анти-бот системи, Cloudflare, Perimeter X, Akamai, ja3, фингерпринти.

ВСТУП

У сучасну цифрову епоху Інтернет став незамінним ресурсом для доступу до величезних обсягів інформації. Веб-ресурси слугують сховищами даних і надають можливість окремим особам та організаціям отримувати цінну інформацію. Метою цього дослідження є вивчення методів отримання даних з веб-ресурсів, з особливим акцентом на систему бронювання віз, доступну на сайті vfsglobal.com.

Процес вилучення даних з веб-ресурсів привертає значну увагу завдяки своєму потенціалу для оптимізації операцій, покращення процесу прийняття рішень та отримання цінної інформації. У контексті веб-сайту vfsglobal.com основна увага приділяється розробці бота, який може швидко забронювати відповідні візові зустрічі від імені користувачів. Мотивація для цього дослідження випливає зі спостереження, що обмежена кількість вільних місць на сайті часто створює проблеми для людей, які не можуть знайти відповідні місця в бажані терміни.

Основна мета цієї кваліфікаційної роботи - вивчити та проаналізувати різні методи вилучення даних з веб-ресурсів, зокрема, дослідити систему бронювання віз на сайті vfsglobal.com. Досліджуючи різні методи і стратегії, я прагну спроектувати і зробити бота, який може ефективно переміщатися по сайту, отримувати необхідну інформацію і автоматизувати процес бронювання віз. Це дослідження зробить внесок у сферу вилучення веб-даних, надаючи уявлення про виклики та можливості, пов'язані з доступом та використанням даних з веб-ресурсів.

Кваліфікаційна буде поділена на кілька розділів, кожен з яких зосереджується на різних аспектах дослідження. У наступних розділах буде зроблено огляд суміжних робіт та існуючих підходів до вилучення веб-даних, обговорено методологію, використану в цьому дослідженні, представлено деталі реалізації бота, оцінено його продуктивність і

завершено обговоренням наслідків, обмежень та майбутніх напрямків дослідження.

Загалом, ця робота має на меті надати комплексне розуміння методів вилучення даних з веб-ресурсів, з конкретним застосуванням до системи бронювання віз на сайті vfsglobal.com. Розробляючи ефективного бота, який може допомогти користувачам швидко зарезервувати відповідні зустрічі, ми прагнемо вирішити проблеми, пов'язані з обмеженою доступністю та часовими обмеженнями. Результати цього дослідження не лише принесуть користь особам, які шукають візові зустрічі, але й зробили внесок у ширшу сферу вилучення веб-даних, поглиблюючи наше розуміння методів, які можуть бути застосовані до подібних сценаріїв у майбутньому.

Актуальність: Зростаюча залежність від веб-ресурсів для різних цілей, в тому числі для подання візових заявок, вимагає проведення дослідження ефективних і дієвих методів видобування даних. Актуальність теми полягає у вирішенні проблеми, з якою стикаються користувачі під час запису на візовий прийом через веб-сайт vfsglobal.com, що полягає в його обмеженій доступності, та розробці бота, який може автоматизувати сам процес бронювання. Це дослідження має на меті надати практичне рішення, яке заощадить час і покращить досвід клієнтів.

Мета: Метою цього дослідження є вивчення та аналіз методів отримання даних з веб-ресурсів, з особливим акцентом на системі бронювання віз на веб-сайті vfsglobal.com. Основним завданням є розробка бота, який може ефективно та швидко забронювати відповідні дати візових зустрічей для користувачів, які стикаються з обмеженою доступністю та часовими обмеженнями на платформі vfsglobal.com.

Завдання:

- Провести комплексний огляд пов'язаних робіт та існуючих підходів у сфері вилучення веб-даних.

- Дослідити та оцінити різні методи, техніки та інструменти для вилучення даних з веб-ресурсів.
- Спроектувати та розробити бота, здатного здійснювати навігацію на сайті vfsglobal.com, витягувати відповідні дані про візові зустрічі та автоматизувати процес бронювання.
- Оцінити продуктивність та ефективність розробленого бота за допомогою широкого тестування та оцінки.
- Проаналізувати виклики та можливості, пов'язані з видобутком веб-даних в контексті веб-сайту vfsglobal.com.
- Надати рекомендації щодо підвищення ефективності та надійності методів вилучення веб-даних для подібних сценаріїв.

Об’єкт дослідження. Об'єктом дослідження в цьому дослідженні є система бронювання віз на сайті vfsglobal.com. Основна увага приділяється розумінню обмежень і проблем, пов'язаних з доступом і витяганням даних з цього конкретного веб-ресурсу. Вивчаючи тонкощі системи бронювання візових зустрічей, це дослідження має на меті виробити комплексне розуміння процесу вилучення даних та визначити потенційні сфери для вдосконалення.

Предмет дослідження. Предметом дослідження є методи вилучення даних з веб-ресурсів, з особливим акцентом на веб-сайті vfsglobal.com та його системі бронювання віз. У дослідженні розглядаються різні методи, інструменти та стратегії, що використовуються для вилучення даних з веб-ресурсів, з метою розробки бота, який може ефективно переміщатися по веб-сайту, отримувати відповідні дані про зустрічі та автоматизувати процес бронювання.

Методи дослідження. Для досягнення вищезазначених цілей буде використано поєднання якісних та кількісних методів дослідження. Дослідження включатиме ретельний огляд літератури, щоб отримати уявлення про існуючі дослідження та підходи до вилучення веб-даних. Крім

того, збір даних включатиме розробку та впровадження бота, який взаємодіє з веб-сайтом vfsglobal.com і збирає відповідні дані про візові зустрічі. Ефективність роботи бота буде оцінюватися за допомогою тестування та аналізу, включаючи кількісні показники, такі як рівень успішності, час відгуку та точність.

Методи вивчення – чисельні методи аналізу та оптимізації даних: попередня обробка даних, інтелектуальний аналіз та вилучення даних, статистичний аналіз, алгоритми оптимізації, обчислювальні інструменти та бібліотеки.

Новизна та практична цінність одержаних результатів. Новизна цього дослідження полягає в розробці спеціалізованого бота для автоматизації процесу бронювання віз на сайті vfsglobal.com. Хоча методи вилучення веб-даних досліджувалися в різних контекстах, конкретне застосування до бронювання візових зустрічей та веб-сайту vfsglobal.com є унікальним внеском у цю сферу. Практична цінність отриманих результатів полягає в розробці ефективного бота, який може заощадити час і зусилля осіб, які записуються на візові зустрічі. Уроки, винесені з цього дослідження, можуть бути застосовані до подібних сценаріїв, покращуючи розуміння та впровадження методів вилучення веб-даних для покращення користувацького досвіду та оптимізації процесів.

РОЗДІЛ І. ВИДОБУВАННЯ ДАНИХ

Уявімо, що ви працюєте над великим дослідницьким проектом. Наприклад, вам може знадобитися дослідити, як певна тема висвітлюється в онлайн-медіа по всьому світу або як на неї реагує громадськість. В Інтернеті є багато інформації, яку потрібно знайти, дослідити, впорядкувати та обробити, і у вас є лише кілька днів на це. Дослідження даних є прикладом великого, але простого завдання, яке можна доручити іншому експерту. Воно передбачає відвідування численних веб-сайтів і копіювання відповідного контенту. Крім того, ви можете скористатися спеціалізованим скриптом або сервісом, який може отримати набагато більше інформації набагато швидше. Правильні налаштування мають вирішальне значення.

Автоматизований збір і структурування даних з Інтернету називається парсингом, а парсер - це комп'ютер (або скрипт), який збирає ці дані відповідно до заздалегідь визначеної методології. Каталоги, форуми, блоги, інтернет-магазини та інші веб-ресурси, а також окремі сторінки всередині них можуть бути об'єктами парсингу [1].

1.1. Використання парсингу

Зазвичай аналіз сайту використовується для двох цілей:

1. Спеціалізоване дослідження вашого веб-активу для виявлення помилкових перенаправлень, непрацюючих з'єднань, розпізнавання мета-міток копій, застарілих або неправильних даних та іншої інформації, необхідної для SEO.

2. Парсинг з метою просування в торгівлі. У цьому випадку парсер інформації використовується для:

- швидкого підрахунку замовлень

- збору даних з веб-сайтів конкурентів для ознайомлення з їхніми перевагами та недоліками, збору та виокремлення основних моментів
- рекламувати запити та елементи змін (дослідження витрат, запити, пропозиції щодо певних продуктів або адміністрування)
- збір відгуків та коментарів;
- наповнення невикористаного інтернет-магазину картками товарів (для ілюстрації - копіювання каталогу віддаленого сайту та адаптація його до свого ресурсу);
- створення бази даних лідів (парсер може виявити дані про те, які дії певна категорія клієнтів виконує на вашому сайті) [2].

Парсинг також полегшує метод перенесення сайту на вільне місце. Одне з найважливіших завдань спеціалізованого майстра при зміні URL-адреси - обміняти всі записи і бази даних, щоб сайт працював без перешкод. Для базових цільових сторінок контент копіюється фізично, а для багатосторінкових локалізацій найкращою альтернативою є парсинг. При написанні коду або виборі готового бенефісу враховуйте, наскільки точно вам потрібно обмінюватися контентом - ідентично поточному вигляду ресурсу (на стародавньому просторі), або внести певні зміни (наприклад, об'єднати дані з декількох категорій в одну) - у випадку вибору моменту вам знадобиться більш "просунутий" і складний скрипт.

1.2. Приклади розробки простих прасерів

Існує безліч прикладів використання веб-стретчінгу: вам може знадобитися зібрати ціни з різних напрямків електронної комерції для сайту порівняння цін. Або, можливо, вам потрібен розклад авіарейсів і

бронювання готелів/AirBNB для певного місця подорожі. Можливо, вам потрібно зібрати адреси електронної пошти з різних реєстрів для пошуку потенційних клієнтів або використати інформацію з Інтернету для підготовки моделей машинного навчання/ШІ. В іншому випадку, можливо, вам дійсно хочеться побудувати двигун пошуку, як у Google [3]!

Почати роботу з веб-скрапінгом дуже просто, і цей задачу можна розбити на дві основні частини:

- отримання інформації за допомогою бібліотеки HTML-запитів або безголового браузера,
- та аналіз інформації для отримання точних даних, які вам потрібні.

Поки ми розглянемо це за допомогою поширених модулів запитів-обіцянок Node.js, CheerioJS та Puppeteer. Працюючи над кейсами в цьому посібнику, ви зможете дізнатися всі поради та пастки, які вам знадобляться, щоб стати професіоналом у зборі будь-якої інформації за допомогою Node.js!

1.2.1. Парсинг html-сторінок

Давайте скористаємося Chrome DevTools, щоб знайти синтаксис коду, який ми хочемо проаналізувати, щоб ми могли витягти ім'я та день народження за допомогою Cheerio.js.

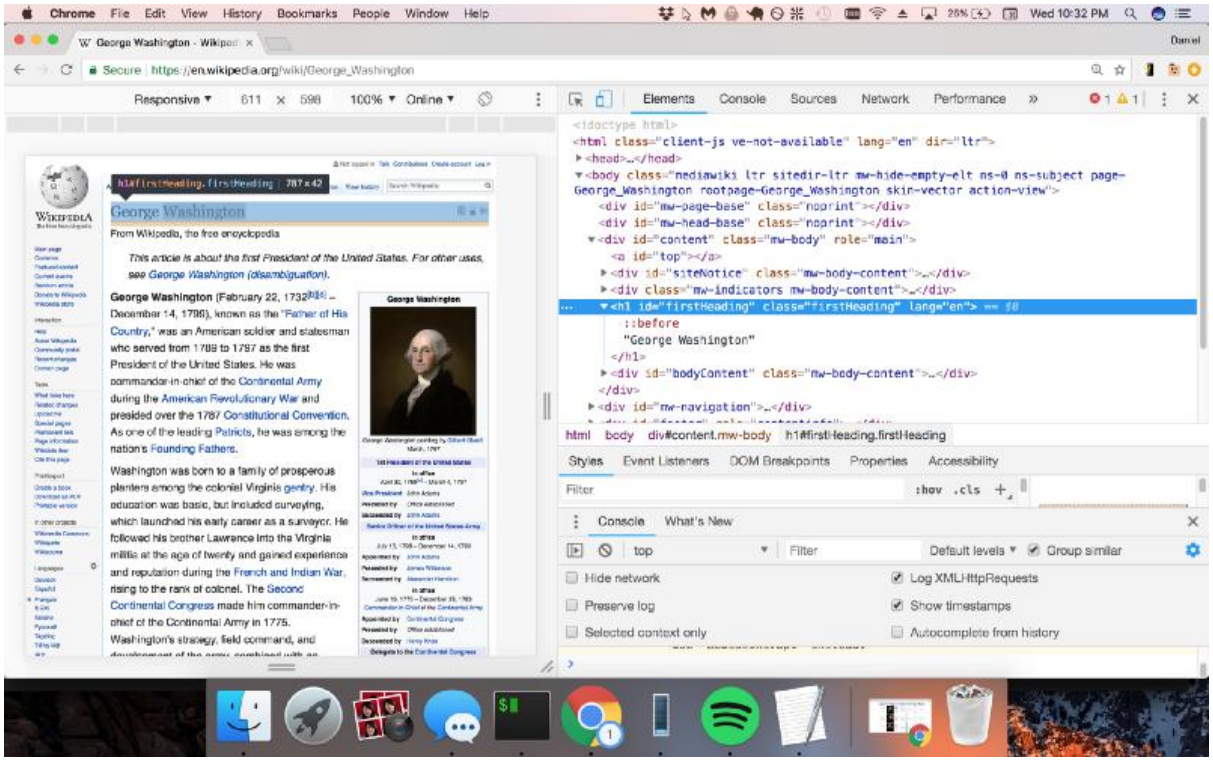


Рис. 1.1. Сторінка у вікіпедії

Отже, ми бачимо (рис. 1.1.), що ім'я знаходиться у класі з назвою "firstHeading", а день народження - у класі з назвою "bday". Давайте змінимо наш код, щоб використовувати Cheerio.js для вилучення цих двох класів.

Тепер давайте перетворимо це на функцію і експортуємо її з цього модуля.

module:

```
const rp = require('request-promise');

const $ = require('cheerio');



const potusParse = function(url) {

return rp(url)

.then(function(html) {
```

```
return {

name: $('.firstHeading', html).text(),

birthday: $('.bday', html).text(),

};

})

.catch(function(err) {

//handle error

});

};



module.exports = potusParse;
```

Тепер повернемося до нашого початкового файлу cheerParserMain.js і викличмо модуль cheerParserModule.js. Потім ми застосуємо його до списку wiki-адрес.

```
const rp = require('request-promise');

const $ = require('cheerio');

const potusParse = require('./potusParse');

const                                url                                =
'https://en.wikipedia.org/wiki/List_of_Presidents_of_the_United_States';




rp(url)

    .then(function(html) {

        //success!

        const wikiUrls = [];
```

```
        for (let i = 0; i < 45; i++) {
            wikiUrls.push($('big > a', html)[i].attribs.href);
        }
        return Promise.all(
            wikiUrls.map(function(url) {
                return potusParse('https://en.wikipedia.org' + url);
            })
        );
    })
    .then(function(presidents) {
        console.log(presidents);
    })
    .catch(function(err) {
        //handle error
        console.log(err);
    });
```

Маємо:

```
[
{ name: 'George Washington', birthday: '1732-02-22'
},
…]
```

Список імен та днів народження всіх 45 президентів США. Використовуючи лише модуль request-promise та Cheerio.js, так можна проскребти переважну більшість сайтів в інтернеті [4].

1.2.2. Рендеринг JavaScript-сторінок

Останнім часом багато сайтів почали використовувати JavaScript для генерації динамічного контенту на своїх сайтах. Це створює проблему для request-promise та інших подібних бібліотек HTTP-запитів (таких як axios і fetch), оскільки вони отримують лише відповідь на початковий запит, але не можуть виконати JavaScript так, як це може зробити веб-браузер.

Таким чином, для сканування сайтів, які вимагають виконання JavaScript, нам потрібне інше рішення. У нашому наступному прикладі ми отримаємо заголовки всіх постів на головній сторінці Reddit. Давайте подивимося, що станеться, коли ми спробуємо використати запит-обіцянку, як ми це робили в попередньому прикладі [4].

Маємо:

```
const rp = require('request-promise');

const url = 'https://www.reddit.com';



rp(url)

.then(function(html){

//success!

console.log(html);

})

.catch(function(err){

//handle error

});

}
```

Ось як виглядає результат:

```
<!DOCTYPE html><html

lang="en"><head><title>reddit: the front page of
the

internet</title>

...
```

Рис. 1.2. Пуста сторінка Реддіт

На рис. 1.2. не зовсім те, що ми хотіли. Це тому, що для отримання фактичного вмісту потрібно запустити JavaScript на сторінці! З Puppeteer це не проблема.

Puppeteer - це надзвичайно популярний новий модуль, створений командою розробників Google Chrome, який дозволяє керувати безголовим браузером. Він ідеально підходить для програмного скрапінгу сторінок, які вимагають виконання JavaScript. Давайте отримаємо HTML з головної сторінки Reddit, використовуючи Puppeteer замість запиту-обіцянки.

```
const puppeteer = require('puppeteer');

const url = 'https://www.reddit.com';
```

```
puppeteer

.launch()

.then(function(browser) {

return browser.newPage();

})

.then(function(page) {

return page.goto(url).then(function() {

return page.content();

});

})

.then(function(html) {

console.log(html);

})

.catch(function(err) {

//handle error

});
```

Вивід:

```
<!DOCTYPE html><html lang="en"><head><link

href="//c.amazon-adsystem.com/aax2/apstag.js"
rel="preload"

as="script">
```

Тепер ми бачимо всю сторінку на рис. 1.3.

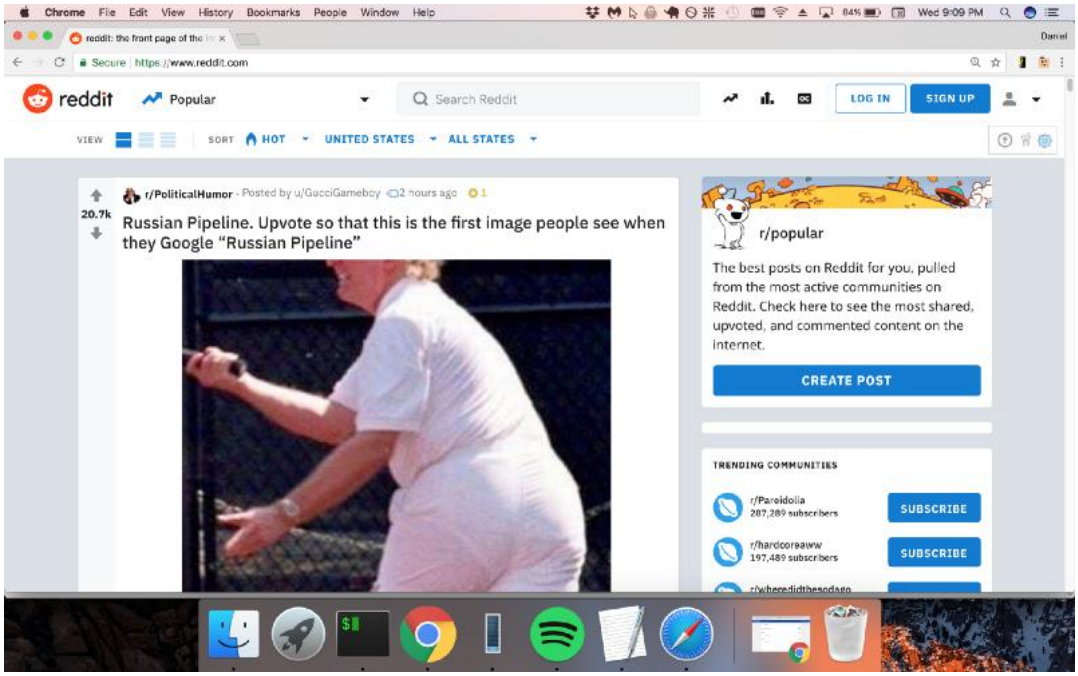


Рис. 1.3. Загружена сторінка реддіт

Тепер ми можемо використовувати Chrome DevTools, як і в попередньому прикладі.

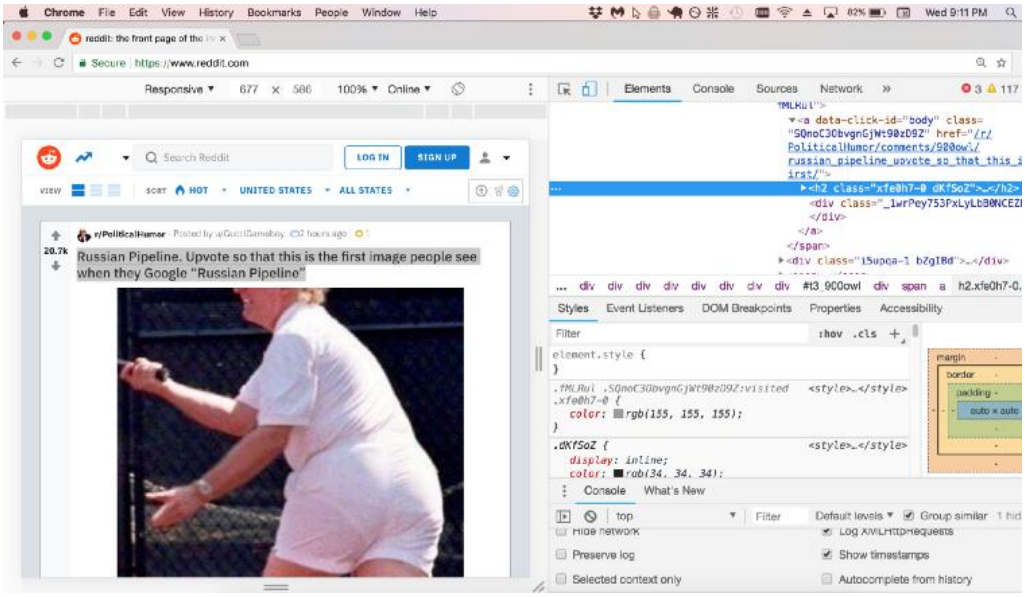


Рис. 1.4. ChromeDevTools на стронки реддіта

Схоже, що Reddit поміщає заголовки всередині тегів "h2" (рис. 1.4.) .Давайте використаємо Cheerio.js для вилучення тегів h2 зі сторінки.

```
const puppeteer = require('puppeteer');

const $ = require('cheerio');

const url = 'https://www.reddit.com';

puppeteer

.launch()

.then(function(browser) {

return browser.newPage();

})

.then(function(page) {

return page.goto(url).then(function() {

return page.content();

});

})

.then(function(html) {

$('h2', html).each(function() {

console.log($(this).text());

});

})

.catch(function(err) {

//handle error

});
```

Маємо:

```
John F. Kennedy Jr. Sitting in the pilot seat of the Marine One circa 1963

I didn't take it as a compliment

How beautiful is this

…
```

РОЗДІЛ ІІ. ПАРСИНГ БЕЗ БЛОКУВАННЯ

Однією з найбільших проблем у веб-скрапінгу є блокування, яке може бути викликане сотнями різних причин. Як би там не було, ми можемо звести всі ці причини до єдиної реальності - з'єднання веб-скрепера виглядає інакше, ніж у веб-браузері. Що ж робить роботу веб-скрапера таким простим для розпізнавання? Зараз детально розберем кожний його пункт.

2.1. Блокування по заголовках

Найпростіший спосіб виявити з'єднання веб-скрапінгу - це аналіз заголовків запитів. Заголовки є частиною кожного з'єднання і містять важливі метадані. Якщо наш веб-скрепер з'єднується з заголовками, які відрізняються від заголовків веб-браузера, то його можна легко ідентифікувати. Для цього нам потрібно зрозуміти, як працюють заголовки, як вони представлені у веб-браузерах і як ми можемо повторити це в нашому коді веб-скрапинга [5].

Отже:

- потрібно переконатися, що значення заголовків відповідають загальному веб-браузеру
- для значень змінних - треба звести до загальних значень, таких як Chrome в Windows або Safari в MacOs
- рандомізувати деякі значення змінних під час скрапінгу в масштабі
- переконатися, що порядок заголовків відповідає порядку у веб-браузері, а ваш HTTP-клієнт отримує правильний порядок заголовків.

Приклад заголовків Chrome на Windows:

```
GET / HTTP/1.1

Host: 127.0.0.1:65432

Connection: keep-alive

Cache-Control: max-age=0

sec-ch-ua: " Not A;Brand";v="99", "Chromium";v="99", "Google Chrome";v="99"

sec-ch-ua-mobile: ?0

sec-ch-ua-platform: "Windows"

Upgrade-Insecure-Requests: 1

User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like
Gecko) Chrome/99.0.4844.51 Safari/537.36

Accept:
text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/a
png,*/*;q=0.8,application/signed-exchange;v=b3;q=0.9

Sec-Fetch-Site: none

Sec-Fetch-Mode: navigate

Sec-Fetch-User: ?1

Sec-Fetch-Dest: document

Accept-Encoding: gzip, deflate, br

Accept-Language: en-US,en;q=0.9
```

2.2. Блокування за IP-адресою

Ще одна миттєва інформація про метадані, яка міститься в кожному HTTP-з'єднанні, - це IP-адреса. Під час веб-скрепінгу ми часто використовуємо проксі-сервери, щоб уникнути надсилання нелюдської кількості запитів через одне з'єднання. Це чудово допомагає уникнути виявлення при аналізі трафіку, але не всі IP-адреси однакові. Деякі з них працюють набагато краще у веб-скрепінгу, ніж інші [6].

Анти-бот системи використовують ці дві деталі IP - адресу і метадані - для генерації початкової оцінки довіри до з'єднання для кожного клієнта, яка використовується для визначення того, чи є клієнт бажаний чи ні.

Наприклад, якщо ви підключаєтеся зі своєї чистої домашньої мережі, сервіс може почати з оцінки 1 (надійний) і пропустити вас без особливих зусиль, не запитуючи розгадування капчі.

З іншого боку, якщо ви підключитеся до публічного Wi-Fi, оцінка буде трохи нижчою (наприклад, 0,5), що може час від часу викликати невеликі запити на введення капчі.

У найгіршому випадку, якщо ви підключається із завантаженого загальнодоступного IP-адреси дата-центру, ви отримаєте дуже низьку оцінку, що може призвести до кількох запитів на введення captcha або навіть повного блокування.

Отже, які дані про IP-адресу найбільше впливають на цей показник?

По-перше, це сама адреса. Всі сервіси відстеження ведуть базу даних про IP-з'єднання, наприклад, IP X підключався N разів за минулу добу і так далі. Важливо відзначити, що ці дані мають широку мережу взаємозв'язків. Отже, на оцінку однієї IP-адреси можуть впливати її сусіди та родичі.

Яскравим прикладом цього є той факт, що IP-адреси продаються не по одній, а блоками. Це означає, що одне гниле яблуко часто псує весь букет. IP-адреси зазвичай продаються блоками /24, що означає 256 адрес або, іншими словами, 1 підмережу (3-й номер IPv4). Отже, якщо ми бачимо кілька незвичайних з'єднань з таких адрес, як 1.1.1.2, 1.1.1.43, 1.1.1.15, ми можемо припустити, що весь блок 1.1.1.X належить одній особі. Це часто призводить до того, що вся підмережа блокується або знижується її рейтинг довіри. Ми можемо розширити цю ідею володіння блоком ще більше, подивившись на метадані IP-адреси.

Найпоширенішою точкою даних для цього є автономний системний номер (ASN), який є номером, що присвоюється кожному зареєстрованому

власнику IP-адреси. Таким чином, кілька "гнилих яблук" одного конкретного ASN можуть знизити оцінку з'єднання для всіх IP-адрес під тим самим ASN. Існують різні онлайн-бази даних, які дозволяють перевіряти номери ASN на предмет присвоєння їм IP-номерів, наприклад, bgpview.io.

Ще одним елементом метаданих, який зазвичай використовується при розрахунку показників довіри, є сам тип IP-адреси. Хоча в метаданих явно не вказано, чи є адреса житловою, мобільною або центром обробки даних, про це можна дізнатися з відомостей про власника. Таким чином, IP-адреса дата-центру отримає нижчий бал лише тому, що вона, найімовірніше, є роботом, тоді як мобільні та домашні IP-адреси будуть оцінені набагато справедливіше.

Отже:

- уникати використання IP-адрес центрів обробки даних.
- диверсифікувати пул IP-адрес, включивши в нього багато підмереж, а не тільки адреси.
- Перевіряйте метадані про IP-адреси, щоб ще більше диверсифікувати пули IP-адрес за ASN та іншими ідентифікаторами власників.

2.3. Блокування за TLS “рукостисканням”

Безпека на транспортному рівні (TLS) - це наскрізний протокол шифрування, який використовується у всіх HTTPS-з'єднаннях.

Транспортний рівень безпеки - це те, що забезпечує роботу всіх HTTPS-з'єднань. Саме він забезпечує наскрізний зашифрований зв'язок між клієнтом і сервером. У контексті веб-скрепінгу нас рідко хвилює, чи використовує веб-сайт з'єднання HTTP або HTTPS, оскільки це не впливає на нашу логіку збору даних. Однак нова технологія зняття відбитків пальців

використовує цей етап з'єднання не лише для відстеження користувачів, але й для блокування веб-скрапера. Процес рукостискання TLS може призвести до ідентифікації веб-скрепера та зняття відбитків пальців. Це пов'язано з тим, що кожен HTTP-клієнт, будь то бібліотека програмування або веб-браузер, виконує початкове рукостискання TLS-з'єднання трохи по-різному. Ці невеликі відмінності збираються і компілюються в відбиток, який називається JA3.

2.3.1. Відбитки “рукостискання” TLS

TLS - досить складний протокол, і нам не потрібно розуміти його повністю, щоб визначити нашу проблему, але деякі основи допоможуть.

На початку кожного HTTPS-з'єднання клієнт і сервер повинні привітати один одного і домовитися про те, як буде захищене з'єднання. Це називається клієнтським рукостисканням. З точки зору даних це виглядає приблизно так:

Рис. 2.1. Аналіз TLS-рукостискання в Wireshark. Відмічені точки використовуються для зняття відбитків пальців.

Тут багато даних (рис. 2.1.), і саме тут починається гра "знайди відмінності": які значення цього рукостискання можуть відрізнятися в різних HTTP-клієнтах, таких як веб-браузери або бібліотеки програмування?

Перше, на що слід звернути увагу, це те, що існує кілька версій TLS: зазвичай це або 1.2, або 1.3 (остання).

Ця версія визначає решту даних, що використовуються під час рукостискання. TLS 1.3 надає додаткові оптимізації і менше даних, тому його легше захистити, але незалежно від того, чи це 1.2, чи 1.3, наша мета залишається незмінною - зробити його схожим на справжній веб-браузер. Насправді, нам доводиться зміцнювати кілька версій, тому що деякі веб-сайти ще не підтримують TLS 1.3.

У нас також є найважливіший напрямок: Набори шифрів.

Це поле являє собою список алгоритмів шифрування, які підтримують сторони переговорів. Цей список впорядкований за пріоритетом, і обидві сторони погоджуються на перше значення, що збігається.

Тому нам потрібно переконатися, що список наших HTTP-клієнтів збігається зі списком звичайного веб-браузера, включно з порядком.

Подібно до списку наборів шифрів, у нас є список дозволених розширень.

Ці розширення вказують на функції, що підтримуються клієнтом, і деякі метадані, такі як доменне ім'я сервера. Як і у випадку з наборами шифрів, ми повинні переконатися, що ці значення і їх порядок відповідають звичайному веб-браузеру.

2.3.2. JA3 Фингерпринт

Як ми бачимо, є кілька значень, які можуть сильно відрізнятися у різних клієнтів. Для цього часто використовується метод відбитків JA3, який по суті являє собою рядок значень, що змінюються[7]:

- TLS-версія,
- Шифри,
- Розширення,
- support_groups (раніше EllipticCurves),
- EllipticCurvePointFormats,

Кожне значення відокремлюється символом ‘,’ і значення масиву символом ‘-‘.

Ось, наприклад, цей профіль веб-браузера Chrome у Linux:

```
Handshake Type: Client Hello (1)

Length: 508

Version: TLS 1.2 (0x0303) #1 (note that 0x0303 is hex for 771)

Cipher Suites Length: 32

Cipher Suites (16 suites)

#2.1 Cipher Suite: Reserved (GREASE) (0x1a1a)

#2.2 Cipher Suite: TLS_AES_128_GCM_SHA256 (0x1301)

#2.3 Cipher Suite: TLS_AES_256_GCM_SHA384 (0x1302)

#2.4 Cipher Suite: TLS_CHACHA20_POLY1305_SHA256 (0x1303)

#2.5 Cipher Suite: TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256 (0xc02b)

#2.6 Cipher Suite: TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256 (0xc02f)

#2.7 Cipher Suite: TLS_ECDHE_ECDSA_WITH_AES_256_GCM_SHA384 (0xc02c)

#2.8 Cipher Suite: TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384 (0xc030)

#2.9 Cipher Suite: TLS_ECDHE_ECDSA_WITH_CHACHA20_POLY1305_SHA256 (0xcca9)

#2.10 Cipher Suite: TLS_ECDHE_RSA_WITH_CHACHA20_POLY1305_SHA256 (0xcca8)
```

```
#2.11 Cipher Suite: TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA (0xc013)

#2.12 Cipher Suite: TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA (0xc014)

#2.13 Cipher Suite: TLS_RSA_WITH_AES_128_GCM_SHA256 (0x009c)

#2.14 Cipher Suite: TLS_RSA_WITH_AES_256_GCM_SHA384 (0x009d)

#2.15 Cipher Suite: TLS_RSA_WITH_AES_128_CBC_SHA (0x002f)

#2.16 Cipher Suite: TLS_RSA_WITH_AES_256_CBC_SHA (0x0035)

Extensions Length: 403

Extension: Reserved (GREASE) (len=0)

#3.1 Type: Reserved (GREASE) (56026)

Extension: server_name (len=16)

#3.2 Type: server_name (0)

Extension: extended_master_secret (len=0)

#3.3 Type: extended_master_secret (23)

Extension: renegotiation_info (len=1)

#3.4 Type: renegotiation_info (65281)

Extension: supported_groups (len=10)

#3.5 Type: supported_groups (10)

Supported Groups (4 groups)

#4.1 Supported Group: Reserved (GREASE) (0x7a7a)

#4.2 Supported Group: x25519 (0x001d)

#4.3 Supported Group: secp256r1 (0x0017)

#4.4 Supported Group: secp384r1 (0x0018)

Extension: ec_point_formats (len=2)

#3.6 Type: ec_point_formats (11)

Elliptic curves point formats (1)

#5 EC point format: uncompressed (0)

Extension: session_ticket (len=0)

#3.7 Type: session_ticket (35)

Extension: application_layer_protocol_negotiation (len=14)
```

```
#3.8 Type: application_layer_protocol_negotiation (16)

Extension: status_request (len=5)

#3.9 Type: status_request (5)

Extension: signature_algorithms (len=18)

#3.10 Type: signature_algorithms (13)

Extension: signed_certificate_timestamp (len=0)

#3.11 Type: signed_certificate_timestamp (18)

Extension: key_share (len=43)

#3.12 Type: key_share (51)

Extension: psk_key_exchange_modes (len=2)

#3.13 Type: psk_key_exchange_modes (45)

Extension: supported_versions (len=7)

#3.14 Type: supported_versions (43)

Extension: compress_certificate (len=3)

#3.15 Type: compress_certificate (27)

Extension: application_settings (len=5)

#3.16 Type: application_settings (17513)

Extension: Reserved (GREASE) (len=1)

#3.17 Type: Reserved (GREASE) (27242)

Extension: padding (len=44)

#3.18 Type: padding (21)

Extension: pre_shared_key (len=156)

#3.19 Type: pre_shared_key (41)
```

Це дасть нам такі відбитки пальців:

```
771,6682-4865-4866-4867-49195-49199-49196-49200-52393-52392-49171-
49172-156-157-47-53,56026-0-23-65281-10-11-35-16-5-13-18-51-45-43-27-
17513-27242-21-41,31354-29-23-24,0
```

Відбитки пальців JA3 часто піддаються додатковому хешуванню md5 для зменшення довжини відбитків, для зручності:

```
dbe0907495f5e986a232e2405a67bed1
```

Отже:

- можна проаналізувати рукостискання TLS, щоб зрозуміти, чим відрізняється рукостискання веб-браузера від рукостискання http-клієнта. Звичайними винуватцями є поля "Cipher Suite" і "Extensions", які залишають унікальний відбиток кожної клієнтської бібліотеки.
- методика JA3 добре підходить для відстеження програмного забезпечення, але не окремих комп'ютерів. Основна мета - уникнути ідентифікації як загальновідомої машини, наприклад, фреймворку або бібліотеки для веб-скрапінгу.

2.4. Блокування по Javascript

Відбитки пальців і блокування на основі Javascript здебільшого застосовуються до веб-скреперів, які використовують технології автоматизації браузерів, такі як Selenium, Playwright або Puppeteer.

Javascript дозволяє серверам виконувати віддалений код на клієнтській машині, і це, мабуть, найпотужніший метод ідентифікації веб-скреперів. Клієнтське середовище Javascript відкриває тисячі різних змінних, які залежать від самого веб-браузера, операційної системи та технології автоматизації браузера (наприклад, Selenium).

Деякі з цих змінних можуть миттєво ідентифікувати нас як нелюдські з'єднання, а деякі можуть надати унікальні артефакти відстеження для зняття відбитків пальців. При цьому більшість з цих витоків можна закрити або підробити, а це означає, що не все ще втрачено!

2.4.1. Фингерпринт браузера

Javascript у браузері може отримати доступ до тисяч різних деталей середовища, таких як змінні часу виконання javascript, можливості відображення, такі як роздільна здатність і колірний відбиток і так далі. Вся ця інформація може бути використана для ідентифікації та блокування веб-скреперів, тому давайте розглянемо, як працює фінгерпринт і як ми, розробники веб-скреперів, можемо її уникнути [8].

Є дві різні концепції використання JavaScript для виявлення веб-скреперів:

Витік відбитків пальців робота, коли середовище javascript можна використовувати для визначення того, чи є щось людиною або роботом.

Відбитки пальців ідентичності, коли середовище JavaScript використовується для створення унікальної ідентичності для відстеження користувачів. У веб-скрепінгу це здебільшого означає, що якщо наш скрепер відстежується, то його можна ідентифікувати після того, як він створить занадто багато неприродних з'єднань. Іншими словами, якщо ID 1234 переглядає веб-сторінку з нелюдською швидкістю, сервер може впевнено зробити висновок, що клієнт не є людиною.

Ці дві концепції тісно пов'язані між собою, однак приховування ідентичності робота значно важливіше, оскільки це дійсно поширений спосіб блокування скраперів на базі Playwright/Puppeteer/Selenium.

2.4.2. Зняття фінгерпринту автоматизації браузера

Технологія зняття відбитків пальців може бути досить потужною, щоб миттєво ідентифікувати веб-скрепера. На жаль, багато інструментів

автоматизації веб-браузерів зливають інформацію про себе в контекст виконання javascript - це означає, що javascript може легко визначити, що браузером керує програма, а не людина. Це має стати нашим першим кроком до зміцнення відбитків пальців - нам потрібно приховати сліди, залишені нашим скребковим середовищем.

Браузери, керовані скрепером, часто містять додаткову інформацію про середовище javascript, яка вказує на те, що браузер працює без елементів графічного інтерфейсу (так званий "безголовий") або працює на рідкісних операційних системах (наприклад, Linux).

Наприклад, найбільш відомим витоком, коли мова йде про інструменти автоматизації браузерів, такі як Selenium, Playwright або Puppeteer, є витік navigator.webdriver, коли в автоматизованому браузері значення navigator.webdriver встановлюється в незвичне значення true:

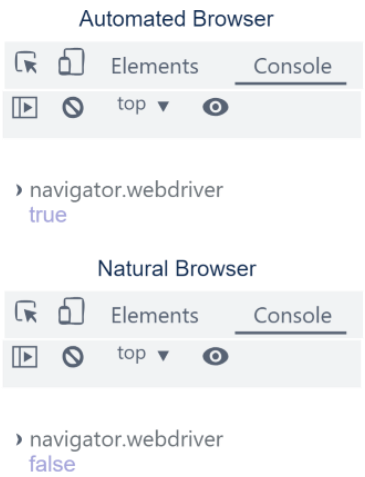


Рис. 2.2. Різниця між витоком navigator.webdriver автоматизованого і реального браузера

Вище ми бачимо (рис. 2.2.), що керований браузер має значення navigator.webdriver рівним true, тоді як природні браузери завжди мають

значення false. Ці змінні можуть бути прочитані будь-яким веб-сайтом, що робить ідентифікацію роботів надзвичайно простою!

Підтасовка фингерпринта автоматизованого браузера

Підтасувавши фінгерпринт, ми можемо впевнено боротися зі значними витоками інформації. Хоча не всі витоки є бінарними, деякі з них є такими. Системи захисту від ботів спрямовані на уникнення хибних спрацювань, а різноманітність веб-простору дає певну свободу дій. Закриття основних витоків має вирішальне значення для веб-скреперів, які намагаються отримати доступ до захищених веб-сайтів.

Інструменти для виявлення витоків

Існує багато онлайн-інструментів, які можуть проаналізувати веб-браузер на наявність найпоширеніших витоків і відбитків пальців:

- https://bot.sannysoft.com/
- http://arh.antoinevastel.com/bots/areyouheadless
- https://antoinevastel.com/bots/
- https://github.com/paulirish/headless-cat-n-mouse
- https://abrahamjuliot.github.io/creepjs/

Варто зазначити, що жоден з цих інструментів не є досконалим, і оскільки середовище веб-браузерів постійно розвивається і змінюється з кожним новим випуском браузера, ми повинні підтверджувати всі ці дані самостійно. Для багатьох відомих дірок існують інструменти в бібліотеках автоматизації браузерів:

- puppeteer-stealth - плагін для Puppeteer
- playwright-stealth - плагін для Playwright
- selenium-stealth - плагін для Selenium
- headless-cat-and-mouse - досліджує витоки як з боку клієнта, так і з боку сервера

Однак бібліотеки з відкритим вихідним кодом часто відстають від оновлень браузерів, що ускладнює пошук комплексних рішень. Тим не менш, ми можна розбирати плагіни типу puppeteer-stealth на предмет важливих методів і стратегій. Тестування можливостей браузера є поширеним способом ідентифікації та зняття відбитків пальців з браузерів. Наприклад, змінні navigator.plugins і navigator.mimetypes в Chrome розкривають інформацію про плагіни і підтримувані типи документів. При використанні безголових браузерів для веб-скрепінгу ми прагнемо імітувати значення браузерів з великою кількістю елементів.

JavaScript-середовища браузерів можуть відрізнятися залежно від того, чи захищене поточне з'єднання за допомогою SSL чи ні. Наприклад, змінна Notification.permission в Chrome відрізняється для захищених і незахищених веб-сайтів. Щоб уникнути виявлення, важливо забезпечити відповідність наших налаштувань скрепінгу поведінці реальних браузерів.

Специфічні для браузера об'єкти JavaScript - це ще одна область, де відбувається зняття відбитків пальців. У браузерах Chrome об'єкт Chrome використовується розширеннями, але в безголовій версії він відсутній. Потрібно вручну відтворювати цей об'єкт, щоб він відповідав версії з головою.

Браузери можна налаштовувати за допомогою прапорів запуску, але деякі прапори, додані інструментами автоматизації браузерів, можуть призвести до незвичної поведінки та витоку ідентифікаційних даних. Рекомендується видалити такі прапори, як "--disable-extensions", "--disable-default-apps" і "--disable-component-extensions-with-background-pages".

Відповідність заголовка агента користувача і можливостей браузера важлива для створення фінгерпринту JavaScript. Модифікація просторів імен JavaScript (наприклад, navigator.platform) для відображення правильної операційної системи і використання правильного рядка агента користувача

мають вирішальне значення. Команда Network.setUserAgentOverride Chrome Developer Protocol може оновити заголовок User-Agent і пов'язані з ним деталі JavaScript.

Є тисячі деталей ідентифікаторів, які розкриваються за допомогою javascript. На щастя, нам не потрібно все рандомізувати. Змінивши лише кілька деталей, ми можемо зробити наш відбиток достатньо унікальним, щоб уникнути виявлення. До того ж, детальне створення відбитків пальців є ресурсоємною операцією, і більшість веб-сайтів не можуть дозволити собі, щоб їхні користувачі чекали 2 секунди, поки контент завантажується.

Отже, ми повинні почати з рандомізації основ:

Заголовки - рядок user-agent, який є найважливішим, але ми також можемо змінити деякі деталі, такі як мовні заголовки, можливості кодування тощо.

Область перегляду - хоча 1920x1080 є найпопулярнішою роздільною здатністю, вона не єдина. Ми можемо легко рандомізувати найпопулярніші роздільні здатності і піти далі, оскільки багато звичайних користувачів не використовують повноекранні браузери.

Локація, часовий пояс, геолокація - ще один простий спосіб внести випадковість у відбиток браузера. При використанні проксі, слід дотримуватися геолокації проксі [9].

Чим далі масштабується середовище веб-скрапінгу, тим важливішою стає рандомізація профілю.

2.5. Анти-бот системи

Всі ці різні системи використовуються численними сайтами для захисту від скрапінгу:

2.5.1. Perimeter X

PerimeterX - один з найпопулярніших анти-бот-сервісів на ринку, що пропонує широкий спектр захисту від ботів і скребків. Продукти PerimeterX - Bot Defender, Page Defender та API Defender - використовуються для блокування веб-скреперів.

PerimeterX (також відомий як Human) - це веб-сервіс, який захищає веб-сайти, додатки та API від автоматизованих атак, таких як скрепери. Він використовує комбінацію веб-технологій і поведінкового аналізу, щоб визначити, чи є користувач людиною або ботом.

Його використовують такі популярні веб-сайти, як Zillow.com, fiverr.com та багато інших, тому, зрозумівши, як обійти PerimeterX, ми зможемо протистояти веб-скрепінгу на багатьох популярних сайтах.

PerimeterX використовує комбінацію відбитків пальців і аналізу з'єднань для розрахунку показника довіри для кожного клієнта. Цей показник визначає, чи може користувач отримати доступ до веб-сайту чи ні.

На основі остаточної оцінки довіри користувачеві або дозволяється доступ до веб-сайту, або він блокується на сторінці блокування PerimeterX, яку в подальшому можна обійти, вирішивши проблеми з javascript (наприклад, кнопка "натиснути і утримувати") [10].

2.5.2. Akamai

Akamai пропонує набір веб-сервісів, а служба Bot Manager використовується для визначення того, чи є користувач, який підключається, людиною чи автоматизованим процесом. Хоча цей сервіс використовується для захисту веб-сайтів від зловмисних ботів, він також блокує веб-скрепери від доступу до публічних даних.

Akamai Bot Manager в основному використовується великими веб-сайтами, такими як Ebay.com, Airbnb.com, Amazon.com, що робить веб-скрапінг цих цілей складним, але можливим.

Більшість блокувань ботів Akamai призводять до HTTP-кодів статусу 400-500. Найпоширеніший код статусу 403 з повідомленням "Вибачте за переривання" або "У доступі відмовлено". Однак, щоб відлякати ботів, Akamai може також повертати код статусу 200 з тими ж повідомленнями [11].

2.5.3. Cloudflare

Cloudflare Bot Management - це веб-сервіс, який намагається виявити і заблокувати доступ веб-скрепперів та інших ботів до веб-сайту.

Це складний багаторівневий сервіс, який зазвичай використовується для захисту від ботів і спаму, але стає все більш популярним способом блокування доступу веб-скреперів до публічних даних.

Приблизно 40% веб-сайтів використовують мережу доставки контенту Cloudflare (CDN), тому обхід системи захисту від ботів Cloudflare став важливою вимогою для розробників, які хочуть отримати доступ до найпопулярніших веб-сайтів в Інтернеті.

Обійти захист від ботів Cloudflare можливо. Однак це непросте завдання. Існує кілька підходів до обходу Cloudflare, кожен з яких має свої плюси і мінуси. Вони варіюються від простих, таких як використання готових інструментів, до надзвичайно складних, таких як повне перепроектування того, як Cloudflare виявляє і блокує скрепери.

Для початку давайте розглянемо деякі поширені помилки Cloudflare, з якими стикаються скрепери, і що вони означають.

Більшість блоків бота Cloudflare призводять до HTTP-кодів статусу 403 (найчастіше), 401, 429 і 502. Хоча, що найважливіше, тіло повідомлення містить фактичні коди помилок та їх визначення. Ці коди можуть допомогти нам зрозуміти, що відбувається, і допомогти нам обійти помилки Cloudflare 403 [12].

Існує кілька різних повідомлень про помилки, які вказують на те, що нам не вдалося обійти Cloudflare:

- Помилка Cloudflare 1020: Access Denied
- Помилка Cloudflare 1009 супроводжується повідомленням "...заборонив країну або регіон вашої IP-адреси".
- Помилка Cloudflare 1015: You are being being rate limited означає, що скрепер працює занадто швидко.
- Помилка Cloudflare 1010: Access Denied викликана заблокованим відбитком пальця браузера.

Варіанти обходу захисту клаудфлейр

Варіант #1: Надсилання запитів на сервер походження

Це не завжди можливо, але є одним з найпростіших способів обходу Cloudflare - відправити запит безпосередньо на IP-адресу сервера походження веб-сайту, а не на CDN-мережу Cloudflare.

У цьому випадку замість того, щоб обманювати Cloudflare, змушуючи його думати, що запити надходять від реального користувача, ми повністю обходим Cloudflare, знаходячи IP-адресу вихідного сервера, на якому розміщений веб-сайт, і надсилаючи свої запити на нього.

Cloudflare - це складна система захисту від ботів, але вона налаштовується людьми, які:

- Не до кінця розуміють Cloudflare
- Припускаються помилок при налаштуванні свого сайту на Cloudflare.

Через це, іноді, трохи пошукавши, можна знайти IP-адресу сервера, на якому розміщена основна версія веб-сайту.

Знайшовши цю IP-адресу, можна налаштувати свої скрапери так, щоб вони надсилали запити на цей сервер, а не на сервери Cloudflare, які мають активний захист від ботів.

Існує кілька способів знайти IP-адресу сервера походження веб-сайту. Ось 3 найкращі з них:

Спосіб 1: SSL-сертифікати

Якщо цільовий веб-сайт використовує SSL-сертифікати (більшість сайтів використовують), то ці SSL-сертифікати зареєстровані в базі даних Censys.

Хоча веб-сайти розгорнули свій веб-сайт на Cloudflare CDN, іноді їхні поточні або старі SSL-сертифікати зареєстровані на оригінальному сервері. Можна знайти веб-сайт у базі даних Censys і перевірити, чи не розміщений на якомусь із цих серверів веб-сайт-джерело.

Спосіб 2: DNS-записи інших служб

Іноді інші субдомени, поштові сервери (MX), FTP/SCP-сервіси або імена хостів розміщуються на тому ж сервері, що й основний веб-сайт, але не захищені мережею Cloudflare.

Тут можна перевірити DNS-записи для інших субдоменів або A, AAAA, CNAME і MX DNS-записи, які звільняють IP-адресу основного сервера, використовуючи базу даних Censys або Shodan.

За умови, що веб-сайт не використовує стороннього постачальника послуг електронної пошти, один з трюків полягає в тому, щоб надіслати електронного листа на неіснуючу адресу електронної пошти на цільовому веб-сайті fakeemail@targetwebsite.com, і якщо доставка не вдасться, отримаю повідомлення від поштового сервера, яке буде містити IP-адресу.

Спосіб 3: Старі записи DNS

Історія DNS кожного сервера доступна в Інтернеті, тому іноді трапляється так, що веб-сайт все ще розміщується на тому ж сервері, що і

до його розгортання на Cloudflare CDN. В результаті, можна використовувати такий інструмент, як CrimeFlare, щоб знайти його.

CrimeFlare підтримує базу даних ймовірних серверів походження веб-сайтів, розміщених на Cloudflare, на основі поточних і старих записів DNS.

Варіант #2: Витягнути версію кешу з Google

Залежно від того, наскільки свіжими мають бути ваші дані, ще один варіант - витягти дані з кешу Google, а не з самого веб-сайту.

Коли Google сканує Інтернет для індексації веб-сторінок, він створює кеш знайдених даних. Більшість веб-сайтів, захищених Cloudflare, дозволяють Google сканувати їхні веб-сайти, щоб ми могли отримати цей кеш замість них.

Вилучення кешу Google може бути простішим, ніж вилучення кешу веб-сайту, захищеного Cloudflare, але це доцільний варіант, якщо дані на веб-сайті, який ми хочемо вилучити, змінюються не так часто. Щоб отмати кеш сайту від Google, просто додайте https://webcache.googleusercontent.com/search?q=cache: до початку URL-адреси.

Варіант #3: Скребти за допомогою вдосконалених безголових браузерів

Інший варіант - виконати всю роботу зі скрапінгу за допомогою безголового браузера, який був удосконалений, щоб виглядати як справжній користувацький браузер.

- Puppeteer: Стелс-плагін для puppeteer.
- Playwright: Стелс-плагін для Playwright.
- Selenium : Оптимізований хромдрайвер undetected-chromedriver.

Варіант #4: Реверс-інжиніринг анти-бот захисту Cloudflare

Останній і найскладніший спосіб обійти захист від ботів Cloudflare полягає в тому, щоб здійснити реінжиніринг системи захисту від ботів Cloudflare і розробити обхід, який пройде всі перевірки Cloudflare без необхідності використання повністю захищеного екземпляра безголового браузера.

Цей підхід працює (і це те, що роблять багато розумних проксі-рішень), однак він не для людей зі слабкими нервами.

Переваги: Перевага цього підходу полягає в тому, що якщо ви працюєте у великих масштабах і не хочете запускати сотні (якщо не тисячі) дорогих екземплярів повноцінного безголового браузера. Замість цього ви можете розробити найбільш ресурсоефективний обхід Cloudflare. Той, що призначений виключно для проходження тестів Cloudflare JS, TLS та IP-відбитків.

Недоліки: Недоліком цього підходу є те, що вам доведеться глибоко зануритися в анти-бот-систему, яку навмисно зробили складною для розуміння ззовні, і провести спліт-тестування різних методів, щоб обдурити їх систему верифікації. Потім підтримувати цю систему, оскільки Cloudflare продовжує розвивати свій захист від ботів.

Коли ми говоримо, що хочемо обійти Cloudflare, ми насправді маємо на увазі, що хочемо обійти їхній менеджер ботів, який є частиною їхнього брандмауера веб-додатків (WAF).

Система, призначена для пом'якшення атак зловмисних ботів, не впливаючи на реальних користувачів.

Систему виявлення ботів Cloudflare можна розділити на дві категорії:

- Внутрішні методи виявлення: Це методи фінгерпрінту ботів, які виконуються на внутрішньому сервері.
- Методи виявлення на стороні клієнта: Це методи виявлення ботів, які виконуються в браузері користувача (на стороні клієнта).

Щоб обійти Cloudflare, потрібно пройти обидва набори перевірок.

Проходження методів виявлення Cloudflare на стороні бекенд

Одним з основних тестів, які проводить Cloudflare, є обчислення оцінки репутації IP-адреси для IP-адрес, які використовуються для відправки запитів. При цьому враховуються такі фактори, як приналежність адреси до відомих бот-мереж, її місцезнаходження, провайдер, історія репутації.

Щоб отримати найвищий бал репутації IP-адреси, слід використовувати домашні/мобільні проксі-сервери замість проксі-серверів центрів обробки даних або будь-яких проксі-серверів, пов'язаних з VPN. Однак, проксі-сервери центрів обробки даних можуть працювати, якщо вони якісні.

Cloudflare також аналізує заголовки HTTP, які надсилаються разом із запитами, і порівнює їх з базою даних відомих шаблонів заголовків браузерів.

Більшість HTTP-клієнтів за замовчуванням надсилають user-agents та інші заголовки, які чітко ідентифікують їх, тому потрібно перевизначити ці заголовки і використовувати повний набір заголовків браузера, які відповідають типу браузера, в якості якого бот буде відображатися.

Більш складна система виявлення відбитків пальців, яку використовує Cloudflare - це TLS & HTTP/2 відбитки пальців. Кожен клієнт HTTP-запиту генерує статичний відбиток TLS і HTTP/2, який Cloudflare може використовувати, щоб визначити, чи йде запит від реального користувача або бота.

Різні версії браузерів і HTTP-клієнтів, як правило, мають різні відбитки TLS і HTTP/2, які Cloudflare може порівняти з заголовками браузера, які відправляються, щоб переконатися, що я дійсно є тим, за кого себе видаю у встановлених заголовках браузера.

Проблема полягає в тому, що підробити відбитки TLS і HTTP/2 набагато складніше, ніж просто додати підроблені заголовки браузера до запиту. Спочатку потрібно перехопити і проаналізувати пакети від браузерів, які видаємо за справжні, а потім змінюємо відбитки TLS і HTTP/2, які використовуються для виконання запиту.

Проходження методів виявлення Cloudflare на стороні клієнта

Ці клієнтські перевірки відбуваються, коли Cloudflare показує сторінку безпеки перед тим, як надавати доступ до веб-сайту. Коли скрепер вперше відвідуєте веб-сайт, Cloudflare відображає сторінку, а у фоновому режимі браузер вирішує різні завдання, щоб довести Cloudflare, що ви не робот. Оцінка ризику запиту, отримана під час тестів на стороні сервера, може вплинути на те, які тести перевірки на стороні клієнта будуть запущені. Найголовніше, чи вимагає він від вас розв'язання CAPTHCA чи ні.

Існує три загальних підходи до вирішення проблем з анти-ботами на стороні клієнта, які виникають під час очікування на цій сторінці:

- Автоматизований браузер: Як згадувалося раніше, якщо використовується захищений браузер для відкриття сторінки, він візьме на себе більшу частину важкої роботи по вирішенню проблем з JavaScript Cloudflare.
- Емуляція браузера в пісочниці: Можете емулювати браузер в пісочниці за допомогою бібліотеки на кшталт JSDOM, яка буде менш ресурсоємною і дасть більш точний контроль над тим, що хочеться, щоб сайт відображав.
- Можна зробити алгоритм який буде проходити перевірки без браузера. Це найскладніший підхід, оскільки вам потрібно повністю зрозуміти перевірки на стороні клієнта Cloudflare, деобфускацію викликів Javascript, а потім створити алгоритм для їх вирішення.

РОЗДІЛ ІІІ. СИСТЕМА БРОНЮВАНННЯ ВІЗ НА САЙТІ VFSGLOBAL.COM

3.1. Огляд сайту vfsglobal.com

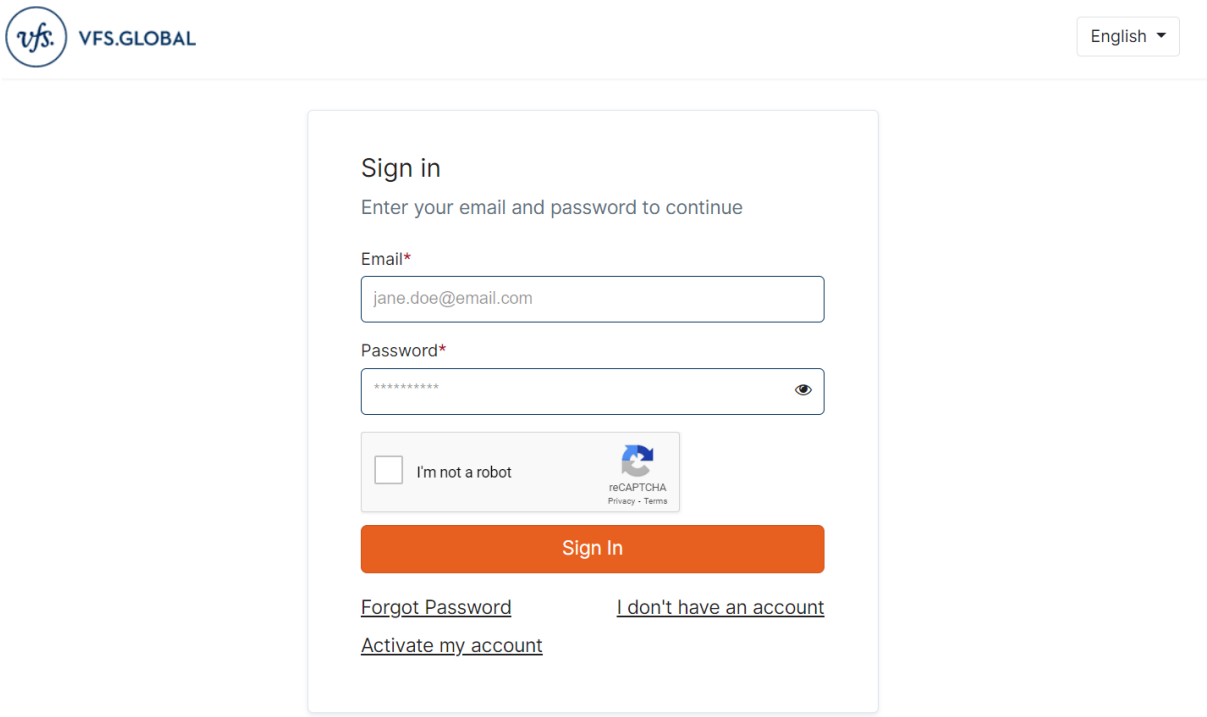


Рис. 3.1. Стартова сторінка сайту vfsglobal.com

VFS Global (рис. 3.1.) - найбільший у світі спеціаліст з візового аутсорсингу та технологічних послуг для урядів і дипломатичних місій по всьому світу. Компанія виконує адміністративні та неупереджені завдання, пов'язані з візовими, паспортними та консульськими послугами для урядів своїх клієнтів. Це дозволяє їм повністю зосередитися на критично важливому завданні оцінки. Маючи 3361 візовий центр у 145 країнах на 5 континентах, VFS Global обслуговує інтереси 68 урядів країн-клієнтів. З моменту свого заснування у 2001 році компанія успішно опрацювала понад 264 мільйони заявок, а з 2007 року - понад 120,62 мільйона біометричних реєстрацій.

Веб-сайт vfsglobal.com слугує провідною платформою для візових послуг, пропонуючи спрощений процес для осіб, які подають заяви на отримання віз до різних країн. Завдяки широкому спектру послуг, включаючи планування зустрічей, подачу документів та відстеження заявок, vfsglobal.com став важливим центром для шукачів віз по всьому світу. Однак, вилучення даних з vfsglobal.com створює унікальні проблеми через заходи безпеки сайту та методи протидії вилученню даних [13].

Вилучення даних з vfsglobal.com вимагає складних методів та інструментів через складний характер структури сайту та наявних механізмів захисту. Веб-сайт використовує різні заходи безпеки для захисту своїх послуг і запобігання несанкціонованому доступу. Ці заходи включають виклики CAPTCHA, системи людської верифікації, обмеження швидкості та блокування IP-адрес. Крім того, vfsglobal.com використовує динамічні методи відображення контенту, що ускладнює ефективне вилучення інформації.

Складність вилучення інформації з vfsglobal.com пов'язана з необхідністю переміщатися по складних веб-елементах і обходити захисні бар'єри. Веб-сайт використовує елементи на основі AJAX, які завантажують дані динамічно, що робить необхідним обробку асинхронних запитів для точного вилучення потрібної інформації. Крім того, включення викликів CAPTCHA і систем перевірки людиною має на меті відрізнити користувачів від автоматизованих ботів, що додає додатковий рівень складності до процесу скрепінгу.

Розробка бота для vfsglobal.com передбачає розуміння робочого процесу бронювання на сайті та вимог до даних. Це передбачає впровадження методів веб-скрепінгу для вилучення даних з різних сторінок, аналіз вилучених даних для визначення доступних дат зустрічей та автоматизацію процесу бронювання. Створення ефективного бота

вимагає врахування безпеки даних, конфіденційності та масштабованості для забезпечення безперебійної роботи користувачів.

Для подолання проблем зі скрепінгом, пов'язаних з vfsglobal.com, розробники використовують різні стратегії. Ці стратегії включають обхід CAPTCHA і систем верифікації людини за допомогою алгоритмів розпізнавання зображень або використання сторонніх сервісів, управління динамічним відображенням контенту шляхом імітації взаємодії з користувачем, використання ротації IP-адрес і проксі-сервісів для уникнення обмеження швидкості і блокування IP-адрес. Оскільки технології продовжують розвиватися, майбутнє систем бронювання віз і бот-технологій має великий потенціал. Досягнення в галузі автоматизації, машинного навчання та обробки природної мови можуть ще більше розширити можливості ботів для бронювання віз, підвищити їхню точність, ефективність та адаптивність.

3.2 Середовище програмування

Версія операційної системи: Windows 11;

Розрядність ОС: x64.

Для виконання поставлених завдань було використано наступні інструменти, програми та програмні компоненти:

Середовище розробки: Pycharm(Build #PC-231.9011.38);

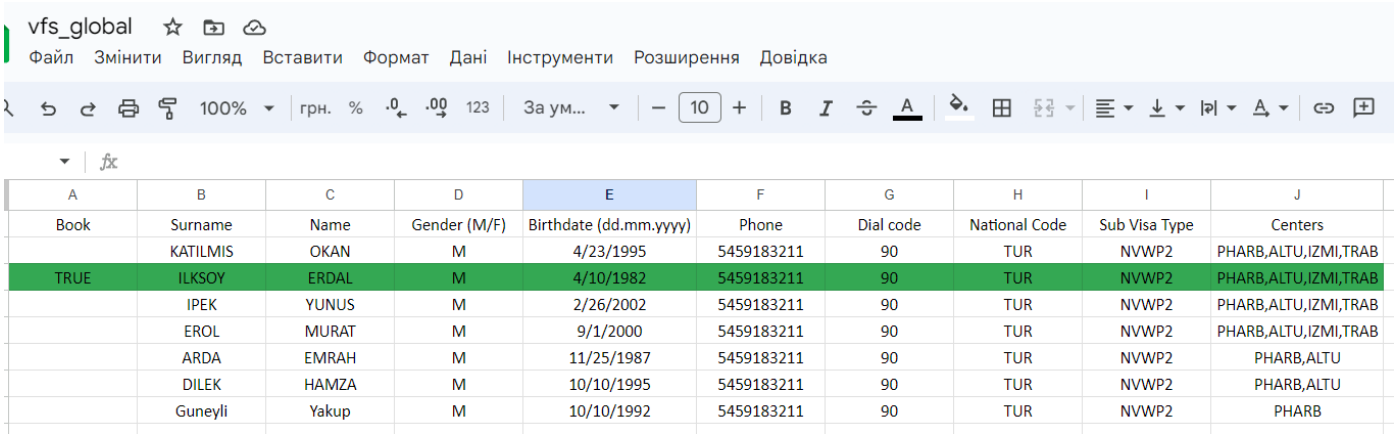
Мова програмування: Python;

Допоміжні бібліотеки:

1. google-api-core,
2. gspread;
3. loguru;
4. requests;
5. tls-client
6. colorama;
7. urllib3;

3.3 Користування системи

Ввід даних (рис. 3.3.1.) відбувається через google sheet з використанням бібліотеки gspread через google api. Система працює в багато потоці тому можна без проблем водити різні центри на різних людей і не пропускати потрібні візи. Для зручного використання воду даних було обрано саме гугл ексель, адже так можна буде користуватися ботом з любого пристрою та будь де, головне підключення до інтернету відповідно.



A	B	C	D	E	F	G	H	I	J
Book	Surname	Name	Gender (M/F)	Birthdate (dd.mm.yyyy)	Phone	Dial code	National Code	Sub Visa Type	Centers
	KATILMIS	OKAN	M	4/23/1995	5459183211	90	TUR	NVWP2	PHARB,ALTU,IZMI,TRAB
TRUE	ILKSOY	ERDAL	M	4/10/1982	5459183211	90	TUR	NVWP2	PHARB,ALTU,IZMI,TRAB
	IPEK	YUNUS	M	2/26/2002	5459183211	90	TUR	NVWP2	PHARB,ALTU,IZMI,TRAB
	EROL	MURAT	M	9/1/2000	5459183211	90	TUR	NVWP2	PHARB,ALTU,IZMI,TRAB
	ARDA	EMRAH	M	11/25/1987	5459183211	90	TUR	NVWP2	PHARB,ALTU
	DILEK	HAMZA	M	10/10/1995	5459183211	90	TUR	NVWP2	PHARB,ALTU
	Guneyli	Yakup	M	10/10/1992	5459183211	90	TUR	NVWP2	PHARB

Рис. 3.3.1. Ввід даних клієнтів

В гугл таблиці для коректної роботи програми потрібно заповнити 9 полів, які запрошує сайт, а саме:

- Book(не обов’язкове поле) - відповідає чи заброньовано людино на потрібні центри, TRUE якщо заброньована і пусто якщо ще ні
- Surname - прізвище клієнта
- Name - ім’я клієнта
- Gender - стать, якщо чоловік то “M”, якщо жінка то “F”
- Birthdate (dd.mm.yyyy) - дата народження в форматі - 11/25/1987
- Phone - номер телефону(5459183211)
- Dial code - телефонний код
- National Code - код країни, де проживає клієнт

- Sub Visa Type - код під-візи
- Centers - коди потрібних центрів

Якщо бот забронював потрібний центр, то прийде відповідне сповіщення в телеграмі для користувача, а також змінюється колір на зелений в гугл таблиці як на картинці (рис. 3.3.1.).

3.4 Технічна частина роботи бота

Бот написаний на мові програмування пайтон і працює на запитах, не зважаючи, на те що на сайті майже кожний день відбувається його оновлення, а також присутня куча всіляких захисних механізмів таких як антибот система Cloudflare, яку ми розглянули раніше. Нижче рис. 3.4.1., продемонстровано файлову систему бота.

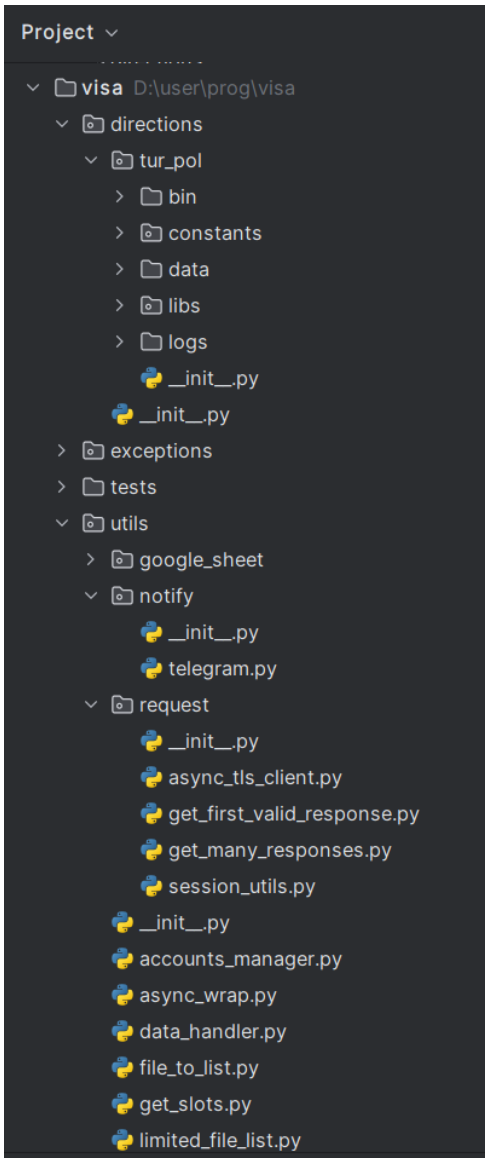


Рис. 3.4.1. Структура файлової системи бота

Отже роботу бота можна поділити на декілька частин, а саме:

1. Головною і початковою проблемою для всіх програмістів які стараються зробити швидкий софт на поточний сайт стала антибот система Cloudflare. Найлегший спосіб це було використати end-to-end технологію типу selenium для проходження Cloudflare, але це дуже повільно і так, як я мав на меті зробити швидкий бот, то одразу взявся за проходження цієї системи(як уже було зазначено раніше).
2. Логін, де потрібно було б на фронтенді розв’язати рекаптчу в2, але на щастя на бекенді немає перевірки валідної відповіді рекаптчі тому якщо відправляти напряму запит то можна слати рандомну строку, яка підійде. Юрл логіна маємо - https://lift-api.vfsglobal.com/user/login. У відповіді ми отримуємо токен авторизації який і використовуємо далі у всіх наступних запитах.
3. Далі їде запит на отримання дат, а саме ендпоінт - https://lift-api.vfsglobal.com/appointment/slots. Методом спроб і помилок я дійшов до висновку, що ідеальна ротація відповідей буде становити 2-3 сек. на центр. Цей запит становить 99% роботи бота, адже він показує коли вільна віза з’явилася.
4. Після того як з’явилася вільна віза я відправляю запит на отримання айді поточної людини, на поточний лінк - https://lift-api.vfsglobal.com/appointment/applicants
5. Далі вже я надсилаю запит на бронювання поточної людини на поточну візу через айді який я отримав у минулому запиті. Тут я відсилаю 30 запитів одночасно. Щоб за 2-3 секунди хоч якийсь із них забронював візу.
6. Далі вже йде оплата яка здійснюється або оффлайн в банку користувачем або онлайн ботом.

В середньому на кожний ендпоінт дається лімітована кількість запитів після яких потрібно змінювати всі відбитки користувача(хеадерси, айпі, тлс і т.д.). Наразі відбувається один запит на логін, потім 8 запитів на перевірку чи є вільні візи, а потім(якщо немає вільної візи) знову перелогін і по кругу. На сайті кожний день працюють якщо не тисячи то сотні ботів-скраперів, для ловлі потрібної візи, що звісно дуже сильно ускладнює процес розробки, але тим не менш бот справляється і бере потрібні візи.

ВИСНОВКИ

Дослідження було зосереджене на вивченні методів отримання даних із системи бронювання віз на сайті vfsglobal.com. Розробка бота, який ефективно бронює відповідні дати для користувачів, була здійснена з метою вирішення проблеми обмеженої доступності візових зустрічей на сайті та часових обмежень, з якими стикаються люди.

Дослідження висвітлило складнощі та проблеми, пов'язані з вилученням даних з vfsglobal.com. Для захисту своїх послуг від несанкціонованого доступу сайт застосовує різні заходи безпеки, включаючи складні CAPTCHA, системи людської верифікації та обмеження швидкості. Крім того, динамічний рендеринг контенту та механізми захисту від ботів, такі як антибот Cloudflare, створювали значні перешкоди для вилучення даних.

Для подолання цих проблем було розроблено бота на мові Python з використанням таких бібліотек, як gspread, requests та tls-client. Для введення даних бот використовував платформу Google Sheets, що дозволило користувачам зручно вводити інформацію та отримувати доступ до бота з будь-якого пристрою, підключеного до Інтернету. Багатопотоковість дозволила боту працювати з кількома візовими центрами одночасно, що підвищить ефективність та гарантувало, що користувачі не пропустили потенційні візові зустрічі.

Технічна реалізація бота включає навігацію через антибот систему Cloudflare, вирішення проблем капчі під час входу в систему, а також запити щодо доступних дат видачі віз та бронювання зустрічей. Процес оплати здійснювався як офлайн, так і онлайн, залежно від уподобань користувача.

Крім того, дослідження підкреслило важливість дотримання безпеки та конфіденційності даних під час роботи ботів. Дотримання правових та етичних міркувань має важливе значення для забезпечення цілісності процесу скрапінгу та захисту інформації користувачів.

Загалом, розроблений бот продемонстрував багатообіцяючі результати у швидкому бронюванні відповідних візових дат для користувачів. Однак постійний розвиток веб-сайту vfsglobal.com з його постійними оновленнями та додатковими захисними заходами створює постійні виклики для розробки та обслуговування бота.

Результати цього дослідження є внеском у сферу вилучення веб-даних і надають цінну інформацію про методи отримання даних зі складних веб-ресурсів, таких як vfsglobal.com. Розроблений бот слугує практичним інструментом для осіб, які бажають отримати візу, спрощуючи процес та підвищуючи зручність для користувачів.

Оскільки технології продовжують розвиватися, з'являються можливості для подальших досліджень і вдосконалення бот-технологій. Постійна співпраця між розробниками, адміністраторами веб-сайтів і політиками має вирішальне значення для досягнення балансу між ефективним вилученням даних і забезпеченням цілісності веб-ресурсів.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Wikipedia [Електронний ресурс] : [Веб-сайт] – Web Scraping – Електронні дані. Режим доступу: https://en.wikipedia.org/wiki/Web_scraping (дата звернення 09.03.2023) - Wikipedia

2. ParseHub Blog [Електронний ресурс] : [Веб-сайт] – What is Web Scraping – Електронні дані. Режим доступу: https://www.parsehub.com/blog/what-is-web-scraping/ (дата звернення 20.05.2023) - ParseHub Blog

3. Towards Data Science [Електронний ресурс] : [Веб-сайт] – Web Scraping Basics – Електронні дані. Режим доступу: https://towardsdatascience.com/web-scraping-basics-82f8b5acd45c (дата звернення 22.05.2023) - Towards Data Science

4. freeCodeCamp [Електронний ресурс] : [Веб-сайт] – The Ultimate Guide to Web Scraping with Node.js – Електронні дані. Режим доступу: https://www.freecodecamp.org/news/the-ultimate-guide-to-web-scraping-with-node-js-daa2027dcd3/ (дата звернення 23.05.2023) - freeCodeCamp

5. Scrapfly Blog [Електронний ресурс] : [Веб-сайт] – How to Scrape Without Getting Blocked? – Електронні дані. Режим доступу: https://scrapfly.io/blog/how-to-scrape-without-getting-blocked-tutorial/ (дата звернення 24.05.2023) - Scrapfly Blog

6. Scraping Robot Blog [Електронний ресурс] : [Веб-сайт] – IP Blocks: How To Get Around Them While Web Scraping – Електронні дані. Режим доступу: https://scrapingrobot.com/blog/ip-blocks (дата звернення 25.05.2023) - Scraping Robot Blog

7. Cloudflare Developers [Електронний ресурс] : [Веб-сайт] – JA3 Fingerprint – Електронні дані. Режим доступу: https://developers.cloudflare.com/bots/concepts/ja3-fingerprint/ (дата звернення 01.06.2023) - Cloudflare Developers

8. LinkedIn [Електронний ресурс] : [Веб-сайт] – HOW DOES BROWSER FINGERPRINT IMPACT ON WEB SCRAPING – Електронні дані. Режим доступу: https://www.linkedin.com/pulse/how-does-browser-

fingerprint-impact-web-scraping-sandeep-suthar/ (дата звернення 08.06.2023) - LinkedIn

9. Zenrows Blog [Електронний ресурс] : [Веб-сайт] – Anti-bot: What Is It and How to Get Around – Електронні дані. Режим доступу: https://www.zenrows.com/blog/anti-bot (дата звернення 09.06.2023) - Zenrows Blog

10. Scrapfly Blog [Електронний ресурс] : [Веб-сайт] – How to Bypass PerimeterX when Web Scraping – Електронні дані. Режим доступу: https://scrapfly.io/blog/how-to-bypass-perimeterx-human-anti-scraping/ (дата звернення 10.06.2023) - Scrapfly Blog

11. Scrapfly Blog [Електронний ресурс] : [Веб-сайт] – How to Bypass Akamai when Web Scraping – Електронні дані. Режим доступу: https://scrapfly.io/blog/how-to-bypass-akamai-anti-scraping/ (дата звернення 22.06.2023) - Scrapfly Blog

12. Scrapfly Blog [Електронний ресурс] : [Веб-сайт] – How to bypass Cloudflare when web scraping – Електронні дані. Режим доступу: https://scrapfly.io/blog/how-to-bypass-cloudflare-anti-scraping/ (дата звернення 23.06.2023) - Scrapfly Blog

13. VFS Global [Електронний ресурс] : [Веб-сайт] – VFS Global Official Website – Електронні дані. Режим доступу: https://www.vfsglobal.com/ (дата звернення 25.06.2023) - VFS Global Official Website

ДОДАТКИ

Додаток 1. Лістинг коду.

```
import asyncio
import random
import traceback



from .sdk import (
PersonDetails,
Book,
ConfirmPayment,
Login
)
from utils import (
logger,
DataHandler,
file_to_list,
)
from utils.request import SessionUtils
from directions.tur_pol.constants import BIN_PATH
from utils.notify import Telegram

from .new_identity import Identity
class Booking(Identity, PersonDetails, Book, ConfirmPayment, Login,
Telegram):
def __init__(self, slot, allocation_id: int, person: dict, session: dict):
super().__init__()
```

```
self.sub_visa_type = slot["sub_visa_type"]
self.center = slot["center"]
self.fee_amount = slot["fee_amount"]
self.data_handler = slot["data_handler"]


self.allocation_id = allocation_id


self.person = person


self.email = session["email"]
self.proxy = session["proxy"]
self.session_token = session["session_token"]


self.headers = {
"authorize": self.session_token,
}


self.urn = None
self.payment_type = None
self.passport_number = None


async def get_urn(self, amount: int):
```

```
resp_json = await self.add_person_details(
amount=amount, statement_func=lambda result:
result is not None and result["success"] and
result["data"].get("error") is None
)


return resp_json["urn"]


async def get_payment_credentials(self, amount: int):
res_json = await self.book(
amount=amount, statement_func=lambda result:
result is not None and result["success"] and
result["data"].get("error") is None
)


return res_json


async def try_to_book(self):
try:
self.urn = await self.get_urn(amount=5)


self.payment_type = "ONLINE" # BANK
```

```
book_result = await self.get_payment_credentials(amount=10)


logger.info(f"Booked!! {book_result} {self.email}")


if self.payment_type.upper() == 'ONLINE':
await self.pay_for_slot(book_result)


await self.logs(book_result)


return True
except:
logger.error(f"booking exp {traceback.format_exc()}")
return False


async def pay_for_slot(self, book_result: dict):
payment_endpoint = await
self.get_payment_endpoint(book_result["RequestRefNo"], book_result["URL"],
book_result["DigitalSignature"])
payment_url = f"https://online.vfsglobal.com{payment_endpoint}"
logger.info(f"payment_url {payment_url}")


Booking.add_to_payment_queue(rf"{BIN_PATH}\booking.txt", payment_url)
```

```
async def logs(self, book_result: dict):
with open(rf"{BIN_PATH}\book_result.txt", "a") as book_result_file:
book_result_file.write(f"{self.email}|{book_result}\n")


if book_result["IsAppointmentBooked"] or not book_result["error"]:
logger.success(f"{self.email}  booked!")
if not DataHandler.test:
msg = f"Gotcha: {self.email} | {self.center}"
chat_id = -1001879372138
await self.notify(msg, chat_id)
else:
logger.error(f"Person is not booked {self.email}")


@staticmethod
def add_to_payment_queue(file_path: str, payment_url: str):
with open(file_path, "a") as f:
f.write(f"{payment_url}\n")


@staticmethod
async def wait_for_payment(file_path: str):
completed_payment_urls = file_to_list(file_path)
logger.info("Waiting for payment url")


if completed_payment_urls:
```

```
open(file_path, 'w').close() # clean file
url = completed_payment_urls[0]
bank_ref_no = url.split("TransactionId=")[1]
request_ref_no = url.split("RequestRefNo=")[1].split("&")[0]
logger.info(f"wait_for_payment {url} {bank_ref_no} {request_ref_no}")
return bank_ref_no, request_ref_no


await asyncio.sleep(1)
return await Booking.wait_for_payment(file_path)
```