РАЗРАБОТКА УНИВЕРСАЛЬНОГО ВЕБ-СКРАПЕРА НА БАЗЕ ФРЕЙМВОРКА NESTJS И БРАУЗЕРА CHROMIUM HEADLESS

Давлатов Ш. Р.
Кафедра защиты информации,
Белорусский государственный университет информатики и радиоэлектороники
Минск, Республика Беларусь
E-mail: shohrukh.92@gmail.com

В работе рассматриваются наиболее популярные методы и средства веб-скрапинга, которые широко применяются для автоматизации процесса извлечения данных из различных ресурсов сети Интернет. Предлагается новая архитектура универсального веб-скрапера на основе браузера Chromium Headless и фреймворка NestJS. Решается задача автоматического парсинга сайтов из заданного списка доменов для последующих процессов анализа и получения статистики используемых технологий, уязвимостей, географического распределения серверов и многое другое.

Введение

Веб-скрапинг это автоматическое считывание различных данных, выложенных на определенных страницах в Интернете и их последующая систематизация для того, чтобы разработать прикладные приложения на основе полученной информации. Таким образом, можно программно извлекать, обрабатывать, систематизировать и сохранять данные тысяч веб-страниц за считанные минуты. Например, инструменты веб-скрапинга позволяют автоматически получать информацию о товарах и ценах из онлайн-магазинов или собирать данные из новостных сайтов. Как правило, постановка задачи веб-скрапинга выглядит следующим образом: есть исходные данные, доступные только на определенных веб-ресурсах в сети, требуется программно получить информацию из заданного списка ресурсов, преобразовать в нужный формат и сохранить в локальном хранилище для последующего процесса обработки данных [1]. Рассмотрим основные области применения веб-скрапинга:

- Сбор и анализ данных для маркетинговых исследований рынка
- Извлечение контактных данных (электронные почты, номера телефонов и т.д.)
- Обнаружение потенциальных уязвимостей веб-ресурсов
- Определение технологий, на базе которых был разработан веб-ресурс.

I. Веб-скрапинг

Рассмотрим простейший подход веб-скрапинга, который основан на статическом анализе HTML контента веб страницы. Как правило, для подобного рода задач требуется незначительная компьютерная мощность и минимум времени. Данный способ работает только в том случае, если исходный HTML код веб-страницы содержит все необходимые данные. Примером таких приложений являются клиент-серверные решения на базе шаблона проектирования MVC, где серверная часть отправляет браузеру HTML файл с готовыми данными. Первым шагом скрапинга является написание CSS селектора для получения необходимых блоков информации (это могут быть ячейки таблицы, заголовки меню и т.п.). Для реализации данного подхода, необходимо отправить HTTP GET запрос на URL-адрес страницы и получить обратно исходный HTML код. В экосистеме платформы NodeJS можно использовать инструмент CheerioJS для парсинга необработанного HTML кода и извлечение данных с помощью CSS селектора. Данная библиотека позволяет работать с данными, используя синтаксис, аналогичный jQuery. Отметим, что во многих случаях получение доступа к информации из необработанного HTML-кода является сложной задачей, ввиду того, что DOM дерево манипулируется сценариями JavaScript, выполняемые в фоновом режиме. Типичным примером является одностраничное веб-приложение, где HTML-документ содержит минимальный объем информации, а код JavaScript заполняет его во время выполнения. В этом случае, одним из возможных решений является создание DOM дерева и выполнение JavaScript сценариев, как это делают обычные браузеры. После данного процесса необходимая информация может быть извлечена из объекта DOM с помощью заданных CSS селекторов. Одним из возможных решений задачи автоматического создания DOM дерева является запуск сайта на Headless браузере, который обладает таким же набором функций что и обычный браузер, только без графического пользовательского интерфейса. Он работает в фоновом режиме, и управляется программно с помощью веб-драйвера и определенных программных сценариев. Для работы обычного браузера требуется гораздо больше вычислительной мощности, чем для отправки простого запроса GET и анализа ответа с помощью Headless браузера. Следовательно, использование подобных браузеров позволяет намного повысить быстродействие про-

грамм веб-скраперов. С другой стороны, этот метод является очень гибким и можно использовать его для навигации по страницам, имитации щелчков мыши, использования клавиатуры, заполнения форм, создания скриншотов страниц и выбора элементов для извлечения текстового содержимого. Headless Chromium является одним из самых популярных Headless браузеров, которая разрабатывается компанией Google [2]. Данная библиотека предоставляет API высокого уровня для программного управления Chromium в автономном режиме.

II. Архитектура приложения

Рассмотрим детально архитектуру разработанного приложения (рис. 1), которая основана на шаблоне проектирования «Цепочка обязанностей». Данный паттерн позволяет структурировать модули системы таким образом, чтобы запросы передавались последовательно по цепочке обработчиков. Каждый последующий модуль решает, может ли он обработать запрос сам и стоит ли передавать запрос дальше по цепи. Первое звено цепочки состоит из модуля для получения и первичной обработки списка исходных доменов. Для каждого элемента списка параллельно запускается процесс для отправки GET запроса и получения HTML ответа от соответствующих серверов. В качестве библиотеки для отправки асинхронных запросов была выбрана node-fetch, которая идентична по функциональности и совместимости методу fetch в обычных браузерах.

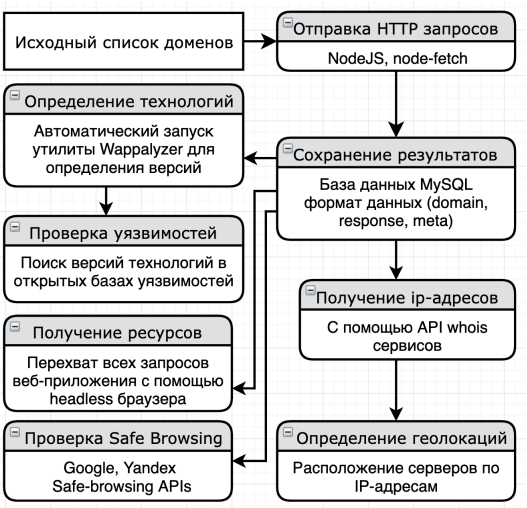

Рис. 1 – Архитектура приложения

После получения ответов от сервера, вся информация хранится в локальной базе данных MySQL для последующих процессов обработки и анализа [3]. Структура таблиц базы данных выглядит следующим образом: таблица доменов, категорий, заголовки HTTP ответов сервера, мета-теги страницы и HTML-контент. Для повышения производительности результат запроса из базы данных разбивается на отдельные блоки фиксированного размера и для каждого блока параллельно запускаются последующие процессы. Далее, с помощью утилиты Wappalyzer определяются версии технологий, на базе которых были разработаны веб-ресурсы [1]. На основе полученных версий ПО и общедоступных баз уязвимостей можно найти потенциально зараженные ресурсы в сети Интернет. С помощью whois сервисов определяется географическое расположение серверов по ip-адресу. Также, на базе публичных API для проверки безопасности веб-ресурсов проверятся каждый домен в исходном списке. Подобные сервисы есть у Google и Yandex (`https://developers.google.com/safe-browsing/v4` и `https://tech.yandex.com/safebrowsing` соответственно). Данные API позволяют проверять безопасность ресурса в сети по его URL. Разработчики браузеров, почтовых программ и клиентов обмена сообщениями могут использовать API для предупреждения пользователей, когда они пытаются перейти по опасной ссылке. Владельцы форумов, социальных сетей и сервисов сокращения ссылок могут воспользоваться автоматической проверкой каждого опубликованного URL-адреса. А системные администраторы корпоративных сетей могут применять централизованный подход к защите сотрудников от нежелательных веб-сайтов.

III. Заключение

В работе изучаются популярные инструменты веб-скрапинга, которые широко применяются для автоматического поиска и обработки необходимой информации из различных веб-ресурсов. Предложенное решение на базе технологий NestJS и Chromium Headless дает возможность достаточно гибко настроить систему под любые нужды пользователей, что позволяет проводить детальный анализ веб-ресурсов из определенной предметной области. Это достигается путем использования декларативного стиля создания модулей программы на основе декораторов фреймворка NestJS. Следующим шагом разработки станет создание онлайн SaaS платформы, основанная на разработках данной статьи.

IV. Список литературы

1. Learning Web Scraping with JavaScript / M. Mekhatria. – Packt Publishing, 2018. – 320 c.
2. The Chromium Projects website [Electronic resource] / Google Inc. – Mode of access: https://www.chromium.org/Home/. – Date of access: 09.09.2019.
3. NestJS framework documentation [Electronic resource] / MIT by Kamil Mysliwiec, 2019. – Mode of access: https://docs.nestjs.com. – Date of access: 05.09.2019.