НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ імені ІГОРЯ СІКОРСЬКОГО»
ФАКУЛЬТЕТ ІНФОРМАТИКИ ТА ОБЧИСЛЮВАЛЬНОЇ ТЕХНІКИ
Кафедра автоматизованих систем обробки інформації та управління

До захисту допущено:
В.о. завідувача кафедри
_____ *Олександр ПАВЛОВ*
(підпис) (вл.ім'я, прізвище)
"___"_____2021 р.

Дипломний проєкт

на здобуття ступеня бакалавра

за освітньо-професійною програмою «Інформаційні управляючі системи та технології»
спеціальності 126 «Інформаційні системи та технології»

на тему: «Система добування текстової інформації з множини веб-сторінок для побудови графу знань» *(комплексна тема)*
Розробка серверної частини
Частина 2

Виконав: студент *IV* курсу, групи *ІС-73*
Дуда Володимир Олександрович _____
(прізвище, ім'я, по батькові) (підпис)

Керівник *доц., к.т.н., доц. Мажара Ольга Олександрівна* _____
(посада, науковий ступінь, вчене звання, прізвище, ім'я, по батькові) (підпис)

Консультант з графічної документації *доц., к.т.н., доц. Сперкач Майя Олегівна* _____
(посада, науковий ступінь, вчене звання, прізвище, ім'я, по батькові) (підпис)

Рецензент *доц., к.т.н., доц. Шаповалова Світлана Ігорівна* _____
(посада, науковий ступінь, вчене звання, прізвище, ім'я, по батькові) (підпис)

Засвідчую, що у цьому дипломному проєкті немає запозичень з праць інших авторів без відповідних посилань.
Студент _____
(підпис)

Київ – 2021 року

Пояснювальна записка
до дипломного проєкту

на тему: Система добування текстової інформації з множини веб-сторінок для побудови графу знань.

Розробка серверної частини

Київ – 2021 року

АНОТАЦІЯ

Структура та обсяг роботи. Пояснювальна записка дипломного проєкту складається з п’яти розділів, містить 14 рисунків, 1 таблиця, 1 додаток, 14 джерел.

Індивідуальна частина дипломного проєкту присвячена розробці фреймворку, який дозволяє динамічно завантажувати запити, їх редагувати та створювати пайплайн на основі об'єднання декількох запитів. Для цієї системи було реалізовано проксі серверу на мові Python та сервер для Node.js на мові JavasScript.

У розділі постановки задачі були визначені необхідні для розробки компоненти та вимоги.

У розділі програмного та технічного забезпечення описуються етапи розробки та засоби, які були використані, з обґрунтуванням. Були визначені вимоги та архітектура, наведена графічна інформація.

У технологічному розділі надано інструкцію по встановленню, керівництво користувача для запуску та роботи з системою. Було визначено тестовий сценарій та проведені по ньому тести.

ЕКСТРАКЦІЯ ДАНИХ, ПАРСЕР, ТЕКСТОВІ ДАНІ, ГРАФ ЗНАНЬ.

ABSTRACT

Structure and scope of work. The explanatory note of the diploma project consists of five sections, contains 14 figures, 1 table, 1 appendix, 14 sources.

The individual part of the diploma project is dedicated to the development of a framework that allows you to dynamically download requests, edit them and create a pipeline based on the combination of several requests. A Python proxy server and a Javascript Node.js server were implemented for this system.

In the problem statement section, the components and requirements were identified.

The software and hardware section describes the stages of development and the tools that were used, with description. Requirements and architecture were defined, graphic information was given.

The technology section provides installation instructions, a user guide for starting and work with system. The test scenario was determined and tests were performed on it.

DATA EXTRACTION, PARSER, TEXT DATA, KNOWLEDGE GRAPH.

ЗМІСТ

ВСТУП

Індивідуальна частина роботи присвячена розробці фреймворку для аналізу запитів користувача з метою визначення параметрів, що впливають на завантаження динамічних даних. В рамках роботи реалізовано проксі-сервер для налаштування авторизації та серверну частину застосунку. Модульна архітектура системи дозволяє реалізувати їх у вигляді незалежних компонентів.

Для реалізації проєкту використовувалися програмні засоби з відкритими ліцензіями. Інформація про них представлена на сайтах розробників. Це дозволило створити відкриту до модифікації та доповнення систему. Враховуючи складність системи було виокремлено підвищені вимоги до засобів розробки, серед яких швидкодія, ресурсоємність, надійність, а також простота та зручність інтерфейсу.

Практичне значення одержаних результатів. Практичне значення одержаних результатів полягає у розробці фреймворку, який дозволяє аналізувати запити користувача, виокремлювати їх значимі параметри та створювати системи екстракції даних, які не потребують візуалізації сторінки для завантаження динамічного контенту.

Публікації. Результати роботи були опубліковані у тезах доповіді на науково-технічній конференції «Інформатика та обчислювальна техніка-IOT-2021» [1].

1 ПОСТАНОВКА ЗАДАЧІ

1.1 Призначення розробки

Призначенням підсистеми, розробленої мною, в межах даної частини є розробка проксі серверу та серверної частини з технологією WebSocket.

1.2 Цілі та задачі розробки

Ціллю розробки - було створення серверу для обробки запитів, які він отримає від проксі серверу та подальших змін чи запитів користувача.

Щоб досягти цілі необхідно було провести такі етапи розробки підсистеми:

- налаштувати проксі серверу;
- створити обробник «сирих» запитів на стороні проксі сервера;
- визначити, як будуть надсилатись запити на сервер та їх подальша обробка та зберігання;
- налаштування обміну командами з клієнтом на основі системи подій, для відтворення архітектурного рішення, яке б забезпечило б одночасне існування багатьох клієнтів;
- створення системи для автоматичного під завантаження команд та їх обробників;
- перевірка працездатності усіх компонентів;
- перевірка працездатності у сукупності з клієнтською частиною та внесення корективів;
- запустити сервер на хостингу.

2 ПРОГРАМНЕ ТА ТЕХНІЧНЕ ЗАБЕЗПЕЧЕННЯ

2.1 Засоби розробки

Щоб визначити логіку та поведінку користувача було спочатку створено макет у сервісі Figma[1], графічному онлайн-редакторі для спільної роботи 2 або більше розробників. У ньому можна створити як дизайн веб-сайту, або просто схематичний компонент архітектурного рішення. Використання його значно пришвидшує обговорення головних ідей, так як одночасно можна малювати певні об’єкти та обговорювати їх з колегами.

Для написання коду був використаний безкоштовний текстовий редактор від компанії Microsoft – Visual Studio Code[2]. Стильний, швидкий, так як не перевантажений функціоналом редактор. Щоб зробити розробку комфортною були завантажені плагіни для розпізнання коду на мовах Python[3], JavaScript[4], TypeScript[5].

Для поєднання серверу та клієнту була використана технологія WebSocket[6].

Аналіз запитів користувача в браузері було створено на основі mitmproxy[7]. Для обробки запитів в цьому інструменті використовувались скрипти на мові Python[3]

Клієнтська частина була реалізована на веб-фреймворці React[8], мовою програмування TypeScript[5].

Серверна частина використовує мову JavaScript[4] та оперує V8 Virtual Machine[8] через бібліотеку vm[14] для запуску певних скриптів.

Для збереження даних в PostgreSQL[9] було використано ORM[10] систему Sequelize[11].

Для підняття REST серверу була використана бібліотека fastify[12] та її під модуль fastify-websocket[13] для WebSocket[6].

2.1.1 Figma

Хмарний інструмент дизайну, який за функціональністю схожий на простий аналог потужного редактору Photoshop, але із певних відмінностей він є кращим для співпраці в команді. Figma спрощує процес проєктування та ефективніше інших програм допомагає дизайнерам та командам ефективно співпрацювати.

Figma працює з будь-якою операційною системою, яка запускає веб-браузер. Це єдиний інструмент дизайну такого типу, який так златний, а отже користувач, що працює під різними операційними системами, все одно може спільно використовувати, відкривати та редагувати файли Figma.

Використовуючи Figma, керівник проєкту може переглянути, над чим команда працює в режимі реального часу, лише відкривши спільний файл. Якщо дизайнер десь помилився, то ця функція дозволяє іншим втрутитися та відредагувати, що зекономить незліченні години, які в іншому випадку були б витрачені даремно.

2.1.2 Visual Studio Code

У своїй основі Visual Studio Code – блискавичний та гнучкий редактор коду, який дуже зручний для повсякденного використання. Ним можна редагувати будь який текстовий файл, або навіть бінарний. Це все із-за підтримки майже усіх мов програмування, VS Code допоможе збільшити продуктивність за усіх обставин: підсвічений синтаксис, відповідність розташування дужок відповідно до позиції та мови, автоматичний відступу до типу файлу, виділення відповідних елементів: вікон чи фрагментів… З перших хвилин знайомства з редактором можна легко запам’ятовувати багато простих та зручних комбінацій клавіш, швидко налаштувати під себе, або створити нові для пришвидшення роботи та для зручності орієнтації в коді.

VS Code зразу ж з коробки має підтримку автоматичного заповнення коду IntelliSense, потужний аналізатор архітектури та семантики коду, який може легко бути використаний для рефакторингу коду.

VS Code має інтерактивний налагоджувач, який показує статус змінних, тому можна переглядати їх значення, дивитись на стеки викликів та та виконання команд в даній області чи консолі.

Дивлячись масштабно, то VS Code має в собі найкраще для веб та інших мов програмування. Так як його архітектура використовує Electron (JavaScript фреймворк), VS Code ідейно поєднує в собі веб-застосунки та технології, такі як мову JS та Node.js, при цьому не залежачи від них. Більше, VS Code має архітектурно розроблений за стосунок, як службу інструментів, що дозволяє інтегрувати його з іншими технологіями, такими як, Roslyn для .NET, TypeScript, тощо.

VS Code працює на загальнодоступнії (open) модель розширюваності, яка дозволяє розробникам створювати та використовувати розширення та багато налаштовувати їх досвід редагування-побудови-налагодження.

2.1.3 Python

Python простий за архітектурою, що робить з нього одного з найшвидших мов / інструменту з точки зору швидкості розвитку та розробки.

Зручне для користувача середовище означає менше часу на боротьбу з алгоритмами і більше часу, витраченого на планування архітектури.

Крім того, зразу можна отримати багатий вибір фреймворків та бібліотек. Вони позбавлять від клопоту з кодуванням вручну та пришвидшивши час релізу.

Для ситуацій, коли продуктивність дійсно важлива, Python пропонує випробувані рішення для включення в код інших, більш швидких мов - наприклад, Cython.

Синтаксис Python чіткий і стислий. Мова розроблена для читання та наближення до фактичної англійської, що полегшує її аналіз. Для написання програми мовою Python треба менше рядків коду для досягнення певного результату порівняно з іншими мовами, такими як C / C++ / C# / Java.

Простота Python дуже допомагає, коли доводиться читати написаний код іншого розробника.

Завдяки коду, який простіше зрозуміти та орієнтуватися, можна зменшити обсяг роботи, необхідної для підтримки та розширення проєкту.

2.1.4 JavaScript та V8

Гнучкість JavaScript легко зрозуміла для багатьох розробників. Мова просто допомагає зробити проєкт, дозволяючи розробнику зосередитися на вирішенні проблеми. Розробники можуть використовувати комбінацію плагінів та власні фрагменти коду.

JavaScript може працювати скрізь, включаючи:

- Такі пристрої, як мобільні телефони, планшети та ноутбуки
- На стороні клієнта, а також на стороні сервера

V8 - це високопродуктивний механізм JavaScript та WebAssembly JIT з відкритим кодом від компанії Google, написаний на мові C ++. Він використовується в таких поширених середовищах, як Chrome та Node.js. V8 може працювати автономно або може бути вбудований у будь-яку програму на C ++.

Як видно з усього попереднього, то JavaScript можна використовувати як:

- Front end (браузери)
- Back end (Node.js)
- Android/iOS (React Native, NativeScript…)
- Desktop (Electron фреймворк)

2.1.5 TypeScript

TypeScript - це ООП мова програмування, розроблена корпорацією Microsoft, яка і такоє займається її підтримкою. Це надбудова JavaScript, а отже і містить усі її елементи.

Серед переваг можна виділити:

- спрощення коду JavaScript, полегшуючи читання та налагодження.
- надання високопродуктивні засоби розробки для середовищ.
- переваги ES6 (ECMAScript 6), в додаток до більшої продуктивності.
- TypeScript може допомогти нам уникнути «динамічних» помилок, які розробники часто стикаються під час написання JavaScript.
- потужна система типу, включаючи generics.

2.1.6 WebSocket

WebSocket - це технологія, яка дозволяє клієнту встановити двосторонній зв'язок із сервером.

За допомогою WebSocket і клієнт, і сервер можуть запускати зв'язок один з одним, і обидва можуть одночасно надсилати повідомлення.

Його реалізації як серверу, так і клієнту є на усіх популярних мовах. Також є підтримка в усіх сучасних браузерах.

2.1.7 mitmproxy

Використовуватися для аналізу, перехоплення, перевірки, зміни та відтворення веб-трафіку за такими веб-протоколами: HTTP/1, HTTP/2, WebSocket, або будь-яких інших протоколів, захищених SSL/TLS. Є можливість впорядковування та декодування різних сторінок, запитів, повідомлень, починаючи від HTML/text-plain і закінчуючи бінарними, як

Protobuf, перехоплювати певні запити на льоту, змінювати їх до того, як вони досягнуть клієнта, і показати їх на девайсі користувача.

Головні переваги:

- перехоплення запитів та відповідей HTTP і HTTPS та зміна їх на льоту;
- збереження запитів HTTP для подальшого відтворення та аналізу;
- реверс проксі-режиму для переадресації трафіку на вказаний сервер;
- вносення зміни до трафіку HTTP за допомогою сценарію на мові Python;
- сертифікати SSL / TLS для аналізу можна генерувати на льоту.

2.1.8 React

React.js - це фреймворк, написаний на мові JavaScript з доступним кодом, який слугує для побудови інтерфейсів для як для веб-додатків так і для специфічних за стосунків по типу Elctron. За основу в ньому використовують обробник layer of view в веб-застосунках і мобільних додатках. Дозволяє писати універсальні «функції» - компоненти, які використовуються для побудови інтерфейсу користувача, як будинок з цеглинок.

React пропонує писати веб-додатки, які змінюють лише динамічні дані, без перезавантаження усіх інформації сторінки. За мету розробники фреймворку React поставили 3 правила: швидкість, масштабованість і простота.

Він відповідає моделі розробки, яка складається з моделі, її відображення та контролеру,який пов’язує ці об’єкти, тобто MVC. Через його гнучкість React можна використовувати в екосистемі, яка складається з

набору інших бібліотек написаних на мові JavaScript, наприклад View.js, Angular JS, тощо.

З певних переваг можна виділите те, що в ньому замість, або разом з, JS для побудови шаблонів використовують JSX. JSX - це такий же самий JS, але за наявності HTML-тегу дозволяє оперувати HTML та візуально відображати структуру HTML-коду, з явним виділенням підкомпонентів.

2.1.9 PostgreSQL

Ідея PostgreSQL спрямована на те, щоб створювати відмовостійкі середовища, для яких розробники створювали додатки, а також користувачу мати доступ до даних незалежно від того, наскільки великий об'єм самої БД.

Серед переваг PostgreSQL можна виділите те, що він є безкоштовним та дуже розширюваний, з повністю відритим кодом. На льоту можна сворювати структури, типи даних, писати власні функції, на кодах для різних мов програмування, без перекомпіляції.

Як і усі шановна БД, PostgreSQL відповідає стандарту SQL, та вносить багато своїх. Це дещо може налякати новачка, але такі можливості надають змогу створювати потужні системи використовуючи обчислення самої БД.

2.1.10 Sequelize

Sequelize - це Node.js ORM для таких БД Postgres(безкоштовна), MySQL(безкоштовна), MariaDB(безкоштовна), SQLite(безкоштовна та вбудована) та БД від компанії Microsoft - Microsoft SQL Server. Він має надійну підтримку транзакцій, відносин, eager and lazy loading, реплікації читання тощо.

2.1.11 Fastify

Fastify - це фреймворк для мови JS, який допомагає розробникам з найменшими витратами досягти результату своєї ідеї, використовуючи потужну архітектуру плагінів.

Fastify побудований як веб-фреймворк загального призначення, але він виділяється при створенні надзвичайно швидких HTTP API, які використовують JSON як формат даних. Вони надзвичайно поширені як в архітектурі веб-додатків, так і в мобільному, тому Fastify може покращити пропускну здатність більшості програм.

2.2 Вимоги до технічного забезпечення

2.2.1 Загальні вимоги

Мінімальні вимоги користувача:

- браузер – Chrome 79, Edge, Firefox 65, Opera 65, Safari 13, Yandex.browser;
- екран – 1280 * 720 px (краще, якщо буде 1920*1280)

Програмні засоби, що використовуються для проведення тестування:

- ОС - Windows 10, VS Code;

Технічні засоби, що використовуються для проведення тестування:

- оперативна пам’ять – 8 Гб;
- процесор – 4 ядра та 2Ггц +.

2.3 Архітектура програмного забезпечення

2.3.1 Діаграма класів

Діаграму класів було наведено у графічних матеріалах. Вона описує взаємодію усіх функціональних компонентів, розроблених в межах серверної частини.

2.3.2 Діаграма послідовності

Діаграми послідовності описують процеси завантаження запитів в систему, їх обробку, взаємодію користувача з ними, створення пайплайну та його обробка. Діаграми були наведені у графічних матеріалах.

2.3.3 Діаграма компонентів

Діаграму компонентів було наведено у графічних матеріалах.

2.3.4 Специфікація запитів

Таблиця 2.1 – Таблиця, яка описує специфікацію запитів

Вхідний запит	Вхідні дані	Вихідний запит	Вихідні дані	Опис
GET_ CONFIG		CONFIG	Налаштування користувача	Отримати налаштування користувача
GET_ REQUEST	id: ідентифікатор запиту	REQUEST	Запит	Отримати запит
GET_ PIPELINE	id: ідентифікатор пайплайну	PIPELINE	Пайплайн	Отримати пайплайн
GET_ALL_ REQUESTS		ALL_ REQUESTS	Список запитів	Отримати усі запити
GET_ALL_ PIPELINES		ALL_ PIPELINES	Список пайплайнів	Отримати усі пайплайни

ADD_ PIPELINE	Пайплайн	NEW_ PIPELINE	id пайплайну	Створити або зберегти пайплайн
UPDATE_ CONFIG	Налаштування користувача	CONFIG	Налаштування користувача	Редагувати налаштування користувача

Продовження таблиці 2.1

UPDATE_ PIPELINE	Пайплайн	UPDATED_ PIPELINE	Пайплайн	Редагувати пайплайн
DELETE_ PIPELINE	id пайплайну	ALL_ PIPELINES	Список пайплайнів	Видалити пайплайн
DELETE_ ALL_ REQUESTS		ALL_ REQUESTS	Список запитів (пустий)	Видалити усі запити
MAKE_ REQUEST	Запит	RESULT_ REQUEST	Результат запиту	Виконати запит та повернути результат
RUN_ SCRIPT	script: код на мові JavaScript	RESULT_ SCRIPT	Результат виконання скрипту	Запустити скрипт та повернути його результат
RUN_ PIPELINE	Пайплайн	RESULT_ PIPELINE	Результат виконання пайплайну	Запустити виконання пайплайну
STOP_ PIPELINE				Зупинити (терміново) виконання пайплайну

Висновок до розділу

У розділі програмного і технічного забезпечення описуються інструменти, бібліотеки та засоби, що використовуються. Були визначені їх переваги та недоліки. Була виділена архітектура на різних рівнях та інструменти з якими вона взаємодіє.

В графічному розділі були наведені діаграми, що дозволяють просто й швидко зрозуміти про архітектуру та з яких елементів вона складається.

ТЕХНОЛОГІЧНИЙ РОЗДІЛ

2.4 Керівництво користувача

Етапи по встановленню застосунку:

- завантаження репозиторію за допомогою команди git clone;
- встановлення Python версії 3.6 або більше за посиланням https://www.python.org/;
- встановлення Node.js версії 10 або більше за посиланням https://nodejs.org;
- встановлення всіх модулей за допомогою команд:

python -m pip install -r requirements.txt

npm install;

- встановлення mitmproxy за посиланням https://mitmproxy.org/;
- запуск проксі серверу:

mitmdump -s .\proxy.py;

- запуск серверу:

npm run dev.

Щоб усі запити були надіслані на сервер треба налаштувати підключення до проксі серверу в нашому браузері. На прикладі використаний Mozilla Firefox.

Спочатку треба найди пагін, який дозволить встановити налаштування нашого серверу в браузер.

Один з кращих є плагін FoxyProxy, який дозволяє швидко налаштувати його та вмикати / вимикати в будь-який час.

Рисунок 3.1 – Сторінка плагіну в сервісі завантаження Firefox Add-ons

Після його встановлення можна побачити його логотип в правому верхньому куту браузера. Це – посилання на його налаштування.

Рисунок 3.2 – Розташування кнопки налаштування

Далі в налаштування вводимо значення розташування нашого серверу (виділені поля на Рисунку 3.3)

Рисунок 3.3 – Налаштування проксі серверу в браузері

Обираємо наше налаштування й усі запити йдуть через проксі сервер.

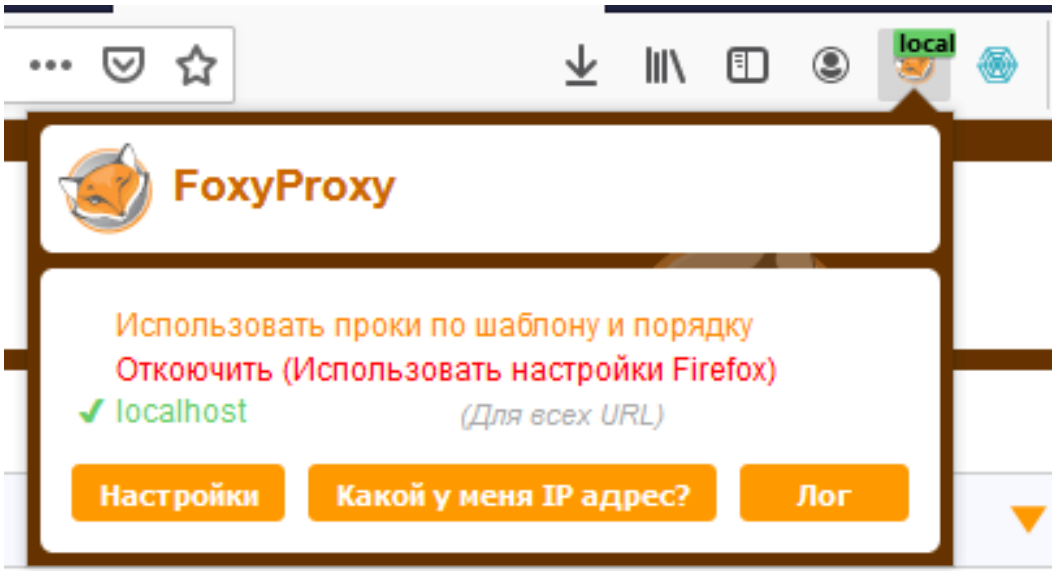

Рисунок 3.4 – Вибір налаштування localhost

Так як проксі сервер аналізує запити, то авторизуватись в додатку не потрібно, всього лиш треба перейти за його посиланням.

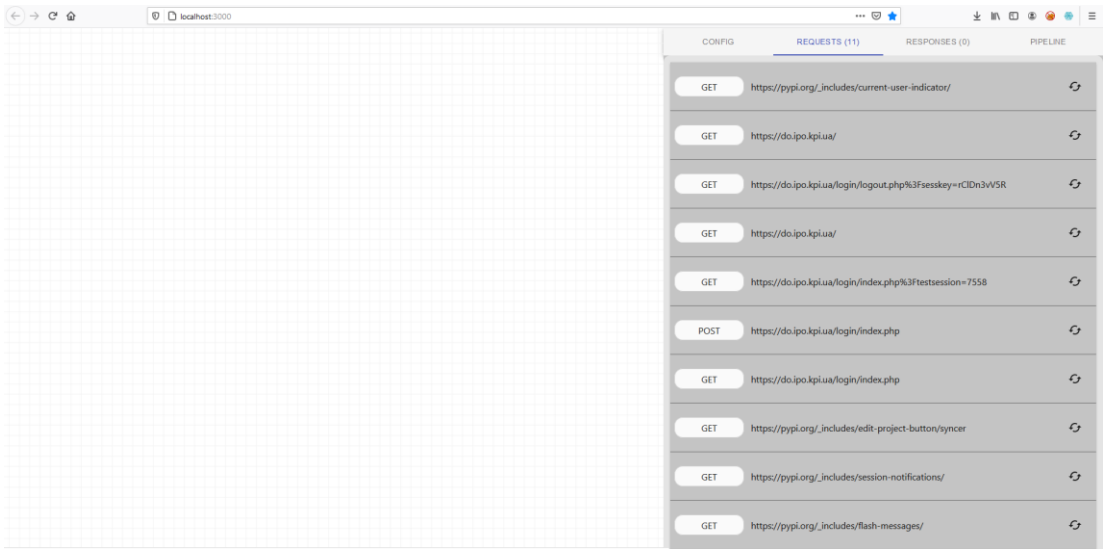

Рисунок 3.4 – Головна сторінка за посиланням http://localhost:3000/

Так як за стосунок складається з компонентів, то розглянемо можливості кожного окремо.

В компоненті налаштувань (CONFIG) можна:

- включити/відключити зберігання запитів, які йдуть через проксі сервер за допомогою кнопки Pause Flow / Resume Flow;
- видалити усі збережені запити - Delete All;
- обмежити кількість запитів для зберігання – Max Requests (усі старі запити, для яких не має місця в цій квоті видаляються, а нові зберігаються);
- встановити фільтр для проксі серверу за часткою тексту в посиланні. Усі фільтри працюють за логікою OR, тобто умовно «текст1 чи текст2 чи текст3 є в посиланні, то запит зберігається».

Рисунок 3.5 – Панель налаштування (CONFIG)

В панелі запитів (REQUESTS) можна побачити усі запити, які були зроблені користувачем. Кожен з них може бути розкритий.

Запит має поля, які можна змінити відповідно до частини. Можна вимкнути поле, потім увімкнути, додати та відредагувати.

Є поля, які представляють певний об’єкт і відображені у вигляді JSON. Редагуючи їх треба пам’ятати, про те, що він має певну структуру й повинен редагуватись за стандартами, які описують JSON.

Також є поле під скрипт. Написання в ньому скрипту повністю стандартизоване під JavaScript.

При натисненні кнопки Make Request (Зробити Запит) на сервері буде виконаний запит і його результат буде доданий до списку відповідей (RESPONSES), де кожен можна розгорнути та переглянути.

Рисунок 3.6 – Панель запитів

Рисунок 3.7 – Розгорнений запит

Рисунок 3.8 – JSON поле

Рисунок 3.9 – Поле для скрипту й кнопка, щоб зробити запит

В компоненті налаштування пайплайнів (PIPELINES) є можливість:

- створити новий пайплайн з назвою;
- зберегти завантажений в область пайплайн;
- завантажити в область пайпайл зі списку справа;
- видалити обраний пайплайн;
- запустити виконання пайплайну;
- зупинити виконання пайплайну.

Рисунок 3.10 – Панель пайплайнів

Також є область для введення глобальних значень на мові JavaScript

Рисунок 3.11 – Приклад змінних

Є виведення усієї статистики: як запуску усього пайплайну (кількість виконаних запитів, скриптів, помилок), так і індивідуальної для кожного запиту(чи вдалий запит, швидкість його виконання, швидкість виконання скрипту для обробки та локальні змінні скрипту)

Requests: 14 Scripts: 0 Fails: 0

BlockId	RequestId	Success	Request perfomance	Script perfomance	Response context
1	2039	true	135.52930000051856	0.000700000673532486	{}
0	2039	true	178.78220000118017	0.000700000673532486	{}
1	2039	true	168.24070000089705	0.0006999988108873367	{}
0	2039	true	166.92200000025332	0.000700000673532486	{}
1	2039	true	113.4229000005871	0.0006999988108873367	{}

Рисунок 3.12 – Приклад виведення статистичних даних

Щоб сформувати пайплайн із запитів, треба із компоненту з запитами пересунути блок із запитом в область редагування. В цій області можна додавати запити/їх видаляти та поєднувати їх один з одним, як вершини графу.

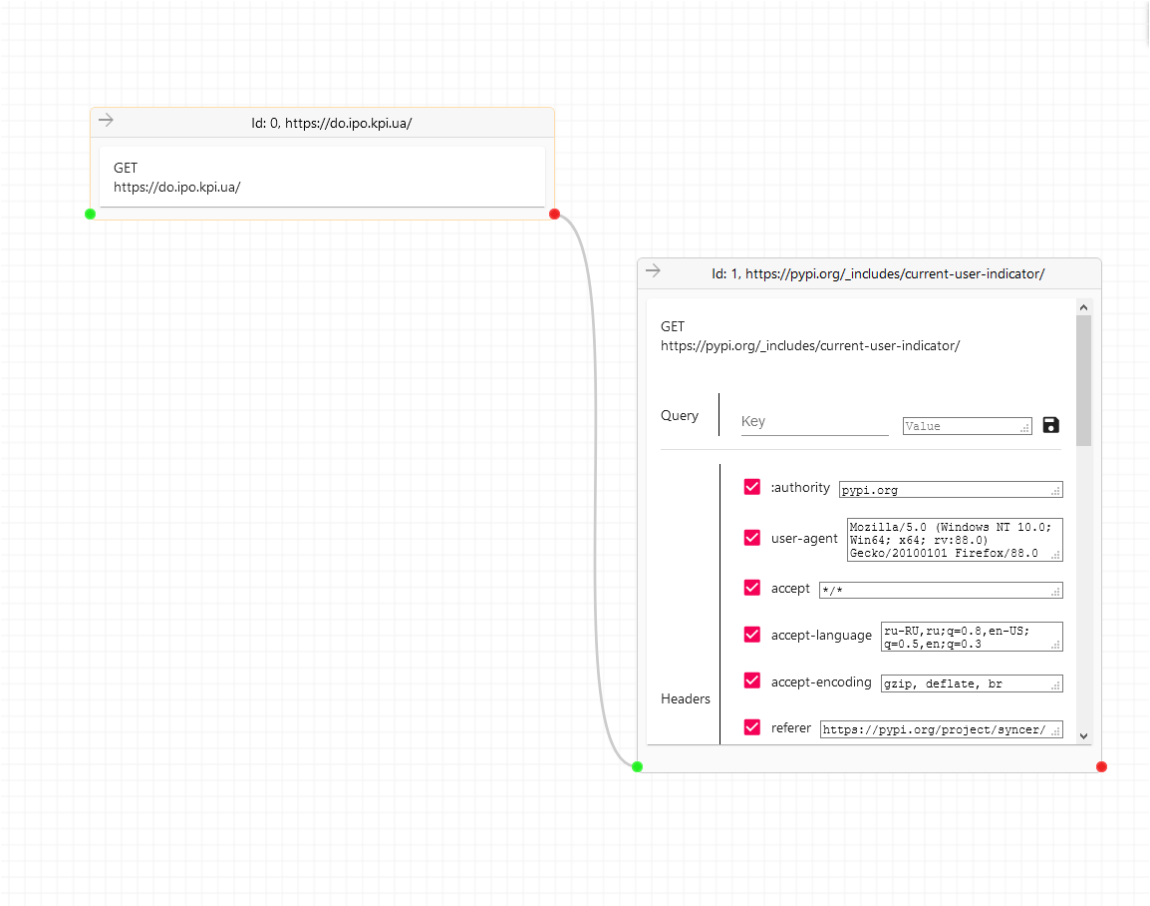

Рисунок 3.13 – Приклад в області редагування

2.5 Випробування програмного продукту

Для визначення працездатності системи було проведено тестування по запуску пайплайну, як головного елемента, що і є результатом нашої роботи.

Тести були проведені на множині сайтів, які були обрані раніш

2.5.1 Мета випробувань

Метою випробувань являється перевірка відповідності функцій комплексу задач для обробки пайплайну вимогам технічного завдання.

2.5.2 Загальні положення

Випробування проводяться на основі наступних документів:

– ГОСТ 34.603–92. Інформаційна технологія. Види випробувань автоматизованих систем;
– ГОСТ РД 50-34.698-90. Автоматизовані системи вимог до змісту документів.

2.5.3 Результати випробувань

В результаті тестів отримали відповідь у вигляді збереженого файлу зі списком інформації зі статей сайту DTF.

Потім ці данні були використані для створення графу знань, наведеного як графічний додаток.

Висновок до розділу

В цьому розділі є керівництво по встановленню та роботі з системою для користувача. Це все допоможе йому швидко та без проблем запустити та зразу ж почати працювати. Для більшого розуміння були виділені компоненти, кожен з яких був описаний. Також були додані приклади з тестовими даними, які були використані для тестів, в результаті яких отримали позитивну відповідь.

ЗАГАЛЬНІ ВИСНОВКИ

В роботі представлено фреймворк для аналізу запитів користувача з метою визначення інформації, необхідної для завантаження динамічного контенту без перезавантаження сторінки. Представлено архітектуру серверної частини та її реалізацію. Обґрунтовано вибір програмних засобів. Наведено приклади використання фреймворку. Розроблений фреймворк дозволяє користувачеві з незначним досвідом веб-розробки створювати застосунки для екстракції текстової інформації.

ПЕРЕЛІК ПОСИЛАНЬ

1. Figma [Електронний ресурс]: [Веб-сайт] – Електронні дані. – Режим доступу: https://www.figma.com/
2. Visual Studio Code [Електронний ресурс]: [Веб-сайт] – Електронні дані. – Режим доступу: https://code.visualstudio.com/
3. Python [Електронний ресурс]: [Веб-сайт] – Електронні дані. – Режим доступу: https://www.python.org/
4. JavaScript [Електронний ресурс]: [Веб-сайт] – Електронні дані. – Режим доступу: https://developer.mozilla.org/ru/docs/Web/JavaScript
5. TypeScript [Електронний ресурс]: [Веб-сайт] – Електронні дані. – Режим доступу: https://www.typescriptlang.org
6. WebSocket [Електронний ресурс]: [Веб-сайт] – Електронні дані. – Режим доступу: https://developer.mozilla.org/ru/docs/Web/API/WebSocket
7. React [Електронний ресурс]: [Веб-сайт] – Електронні дані. – Режим доступу: https://reactjs.org/
8. V8 Virtual Machine [Електронний ресурс]: [Веб-сайт] – Електронні дані. – Режим доступу: https://v8.dev/docs/embed
9. PostgreSQL [Електронний ресурс]: [Веб-сайт] – Електронні дані. – Режим доступу: https://www.postgresql.org
10. ORM [Електронний ресурс]: [Веб-сайт] – Електронні дані. – Режим доступу: https://ru.wikipedia.org/wiki/ORM
11. Sequelize [Електронний ресурс]: [Веб-сайт] – Електронні дані. – Режим доступу: https://sequelize.org
12. fastify [Електронний ресурс]: [Веб-сайт] – Електронні дані. – Режим доступу: https://www.fastify.io
13. fastify-websocket [Електронний ресурс]: [Веб-сайт] – Електронні дані. – Режим доступу: https://github.com/fastify/fastify-websocket

14. vm [Електронний ресурс]: [Веб-сайт] – Електронні дані. – Режим доступу: https://nodejs.org/api/vm.html

Додаток А

Тексти програмного коду

Система добування текстової інформації з множини веб-сторінок для побудови графу знань (комплексна тема)

(Найменування програми (документа))

DVD-R

(Вид носія даних)

19 арк, 1488 Кб

(Обсяг програми (документа) , арк., Кб)

Київ – 2021 року

1) Scrapping DTF.ru with selectors

```
// parseDtf.js
// Created by Duda Volodymyr 30.04.2021
// Copyright © 2021 Duda Volodymyr. All rights reserved
const cheerio = require('cheerio');
const fetch = require('node-fetch');
const querystring = require('querystring')

const deleteSpaces = (text) => text.replace(/\ {2,}/g, '')

const parseTitle = (text) => deleteSpaces(text).replace('\n', '').split('\n')[0]

const page = (url) => fetch(url).then(res => res.text()).then(pageData => {
   const $ = cheerio.load(pageData);
   return {
      title: parseTitle($('.content-title').text()),
      content: $('p', '.content').text()
   }
}).catch(() => console.error('Error:', url))

const categories = () => fetch('https://dtf.ru/initialData', {
   method: 'POST',
   body: 'updateRecentLastSeen=false&mode=raw',
   headers:
      { 'X-This-Is-CSRF': 'THIS IS SPARTA!' }
}
```

```
    ).then(res => res.json()).then(data =>
data.data['module.siteHeader'].subscriptions.map(({ value, url }) => ({
        name: value,
        url: url
    })))

    const category = (url, maxIter, last_sorting_value) => fetch(url + '/more?' +
querystring.stringify({ last_id: 1, last_sorting_value })).then(res => res.json())
        .then(pageData => {
            const $ = cheerio.load(pageData.data.items_html);
            const parsed = $('.feed__item').map(function () {
                const item = $(this)
                return {
                    title: parseTitle($('.content-title', item).text()),
                    url: $('.content-feed__link', item).attr('href')
                }
            }).get()
            if (maxIter && maxIter > 1) {
                return category(url, maxIter - 1,
pageData.data.last_sorting_value).then(p => parsed.concat(p))
            }
            return parsed
        })

    const parseCategory = (url, pages) => category(url, pages || 0).then(pages =>
Promise.all(pages.map(p => page(p.url))))

    const parseAll = () => categories().then(cats => Promise.all(cats.slice(1,
6).map(cat => category(cat.url))))
        .then(pages => pages.reduce((now, all) => all.concat(now)))
```

```
    .then(pages => Promise.all(pages.map(p => page(p.url))))

const cat = 'https://dtf.ru/cinema'

const pcount = 1

parseCategory(cat, pcount).then(data => {
    for (const d of data) {
        console.log(d)
    }
})
```

2) Server framework

```
// proxy.py
// Created by Duda Volodymyr 10.05.2021
// Copyright © 2021 Duda Volodymyr. All rights reserved
from mitmproxy import http
import re
import os
import os.path
import requests
from threading import Thread
import time

from config import siteUrl as server_url

request_url = server_url + '/request'

proxy_script_path = 'proxy_scipt.js'
```

```
token_key = 'Token'


def create_token(flow):
    addr, port, _, _ = flow.client_conn.address
    return '{}'.format(addr)


def requestheaders(flow: http.HTTPFlow):
    # print(flow.request.host)
    if flow.request.host and flow.request.host in server_url:
        flow.request.headers.add(token_key, create_token(flow))


def response_cookies_to_dict(cookie):
    return {key: value[0] for key, value in cookie.items()}


def body_check(body):
    if body:
        try:
            return body.decode()
        except:
            pass
```

```
        def excludeHeaders(header):
            return header in ['Cookie', 'cookie', 'Host', 'host', 'Connection',
'connection']


        def flow_to_data(flow):
            requestHeaders = flow.request.headers

            responseHeaders = flow.response.headers

            return {
                'user': create_token(flow),
                'time': round(time.time() * 1000),
                'host': flow.request.host,
                'method': flow.request.method,
                'path': flow.request.path,
                'query': dict(flow.request.query),
                'port': flow.request.port,
                'requestType': requestHeaders.get('content-type'),
                'requestHeaders': dict({key: value for key, value in
requestHeaders.items() if not excludeHeaders(key)}),
                'requestBody': body_check(flow.request.content),
                'requestCookies': dict(flow.request.cookies),
                'responseType': responseHeaders.get('content-type'),
                'responseHeaders': dict({key: value for key, value in
responseHeaders.items() if key != 'Cookie'}),
                'responseBody': body_check(flow.response.content),
                'responseCookies': response_cookies_to_dict(flow.response.cookies)
```

```
    }


def make_request(flow):

    requests.post(request_url, json=flow_to_data(flow))


def response(flow: http.HTTPFlow) -> None:

    if not flow.request.host in server_url:

        Thread(target=make_request, args=(flow,)).start()
```

```
// proxy_script.js
// Created by Duda Volodymyr 11.05.2021
// Copyright © 2021 Duda Volodymyr. All rights reserved
const frame_src = '{siteUrl}'

if (!window.location.href.startsWith(frame_src)) {
    const frame = document.createElement('iframe')
    frame.style.cssText =
'position:fixed;top:40px;right:40px;height:300px;width:300px;z-index:9999;'
    frame.src = frame_src
    document.body.appendChild(frame)
}

// request.js
// Created by Duda Volodymyr 15.05.2021
// Copyright © 2021 Duda Volodymyr. All rights reserved
```

```
const { Request } = require('../../db')
const { sendToClient } = require('../websocket/pool')
const { minifiRequest, getToken, stringToData } = require('../utils')
const getConfig = require('../websocket/getConfig');
const { exclude } = require('../../config');



const prepareRequest = (data, user) => {
    data.user = user
    return Object.assign(data, {
        requestData: stringToData(data.requestBody, data.requestType) || null
    })
}

const checkStatic = (path) => {
    path = path.split(/[\?\&]/)[0]
    for (const p of exclude.files) {
        if (path.endsWith(p)) {
            return true
        }
    }
}

const checkPath = (path, filters) => {
    if (checkStatic(path)) {
        return false
    }
    if (!filters.length) {
        return true
    }
```

```
    for (const filter of filters) {
        if (path.match(filter)) {
            return true
        }
    }
    return false
}

const howMuchToDelete = (count, maxCount) => {
    const toDel = count - maxCount
    if (toDel < 0) {
        return 0
    }
    return toDel
}

const checkRequest = (request, config) => {
    for (const method of exclude.methods) {
        if (request.method === method) {
            return false
        }
    }
    if (!checkPath(request.host + (request.path || "/"), config.filter)) {
        return false
    }
    for (const type of exclude.types) {
        if (request.responseType?.match(type)) {
            return false
        }
    }
```

```
    return true
}

module.exports = async (fastify, options) => {
    fastify.post('/request', async (request, reply) => {
        const user = getToken(request)
        const data = request.body
        const config = await getConfig(user)
        if (config.active && checkRequest(data, config)) {
            const req = prepareRequest(data, user)
            const count = await Request.count({
                where: { user }
            })
            const toDel = howMuchToDelete(count + 1, config.maxRequests)
            if (toDel) {
                await Request.destroy({
                    where: { user },
                    limit: toDel
                })
            }
            await Request.create(req)
            sendToClient(user, {
                event: "NEW_REQUEST",
                data: minifiRequest(req)
            })
        }
        return {
            ok: true
        }
    })
```

```
}

// pooljs
// Created by Duda Volodymyr 17.05.2021
// Copyright © 2021 Duda Volodymyr. All rights reserved
const connections = {}

const listeners = {}

const parseMessage = (message) => {
    try {
        return JSON.parse(message)
    } catch {
        return null
    }
}

const addTo = (obj, key, value) => {
    if (obj[key]) {
        obj[key].push(value)
    } else {
        obj[key] = [value]
    }
}

const deleteFrom = (obj, key, value) => {
    const list = obj[key]
    if (list) {
        const index = obj[key].indexOf(value)
        if (index >= 0){
```

```
            delete list[index]
        }
    }
}

const stringifyMessage = (message) => {
    try {
        return JSON.stringify(message)
    } catch {
        return '{"ok": false}'
    }
}

module.exports = {
    addClient: (client_id, connection) => {
        addTo(connections, client_id, connection)
        connection.socket.on('message', (message) => {
            if (listeners[client_id]) {
                const parsedMessage = parseMessage(message)
                listeners[client_id](parsedMessage)
            }
        })
        connection.socket.on('close', () => {
            deleteFrom(connections, client_id, connection)
        })
    },
    addListener: (client_id, listener) => {
        listeners[client_id] = listener
    },
    sendToClient: (client_id, data) => {
```

```
            if (connections[client_id]) {
                for (const connection of connections[client_id]) {
                    connection?.socket?.send(stringifyMessage(data))
                }
            }
        }
    }

    // requestPipeline.js
    // Created by Duda Volodymyr 17.05.2021
    // Copyright © 2021 Duda Volodymyr. All rights reserved
    const runScript = require('./runScript')
    const makeRequest = require('./makeRequest')
    const { responseToContext } = require('./utils')
    const defaultConfig = require('./defaultPipeline.json')
    const { performance } = require('perf_hooks');
    const { maxDepth } = require('../config')

    const getCombinations = (arr, pre = []) => {
        if (!arr.length) {
            return pre;
        } else if (arr.length === 1) {
            return arr[0]?.reduce((accum, value) => {
                return accum.concat(getCombinations(arr.slice(1), [[...pre, value]]));
            }, []);;
        }
        return arr[0]?.reduce((accum, value) => {
            return [...accum, ...getCombinations(arr.slice(1), [...pre, value])]
        }, []);
    }
```

```
const parseVariables = (string) => {
    return [...string.matchAll(/\$\{([^}]+)\}/g)].map(([_, v]) => v)
}

const variableRegExp = (variable) => new RegExp(`\\\$\\\{${variable}\\\}`,
'g')

const checkVariable = (variable, recurse = false) => {
    if (typeof variable === 'string') {
        return variable
    } else if (typeof variable === 'number') {
        return variable.toString()
    } else if (typeof variable === 'object') {
        if (!recurse && Array.isArray(variable)) {
            return variable.map(v => checkVariable(v, true)).filter(v => v)
        }
        try {
            return JSON.stringify(variable)
        } catch {
            return
        }
    }
}

const toOneArray = (list) => list && list.length ? list.reduce((p, c) => [...p,
...c]) : []

const findVariablesInObject = (obj, path) => {
    if (typeof obj === 'string') {
```

```
            return parseVariables(obj).map((v) => ({ name: v, path: path }))
        } else if (obj && typeof obj === 'object') {
            return toOneArray(Object.entries(obj).map(([key, value]) =>
findVariablesInObject(value, [...path, key])))
        } else {
            return []
        }
    }

    const findAllVariables = (request) => {
        return [
            ...findVariablesInObject(request.path, ['path']),
            ...findVariablesInObject(request.query, ['query']),
            ...findVariablesInObject(request.requestHeaders, ['requestHeaders']),
            ...findVariablesInObject(request.requestData, ['requestData']),
            ...findVariablesInObject(request.requestCookies, ['requestCookies'])
        ]
    }

    const makeParametersList = (varsContext) =>
(Array.isArray(varsContext.value) ? varsContext.value : [varsContext.value])
        .map((v) => ({ ...varsContext, value: v }))

    const prepareParameters = (findedVars, variables) => {
        const vars = findedVars.map((v) => ({ ...v, value:
checkVariable(variables[v.name]), regex: variableRegExp(v.name) })).filter((v) =>
v)
        const combinatorInput = vars.map(makeParametersList)
        return getCombinations(combinatorInput)
    }
```

```
const treeObjectIterator = (obj, path, callback) => {
    if (obj) {
        if (path.length <= 1) {
            obj[path[0]] = callback(obj[path[0]])
        } else {
            treeObjectIterator(obj[path[0]], path.slice(1), callback)
        }
    }
}

const replaceStringInObject = (obj, { path, value, regex }) => {
    treeObjectIterator(obj, path, (string) => string.replace(regex, value))
}

const setParametersToRequest = (request, parameters) => {
    const requestCopy = { ...request }
    for (const par of parameters) {
        replaceStringInObject(requestCopy, par)
    }
    return requestCopy
}

const prepareRequestParameters = (request, parameters) => {
    if (parameters.length > 0) {
        return parameters.map((params) => setParametersToRequest(request,
params))
    } else {
        return [request]
    }
```

```
    }

    const prepareRequest = (request, variables) => {
      const findedVariables = findAllVariables(request)
      const params = prepareParameters(findedVariables, variables)
      return prepareRequestParameters(request, params)
    }


    module.exports = async (pipeline, onAction = () => { }, controlContext =
{}, cons0le = console) => {

      let allRequestsCount = 0
      let allRequestsErrors = 0
      let allScriptsRunned = 0

      const oneRequest = async (reqData) => {
        if (!controlContext.stop) {
          allRequestsCount++
          try {
            return makeRequest(reqData)
          } catch (error) {
            cons0le.error(error)
            allRequestsErrors++
          }
        }
      }

      const runRequest = async (requests) =>
        Promise.all(requests.map(oneRequest))
```

```
const { global, startId, globalKey, resultKey, blocks, globalScript } = {
...defaultConfig, ...pipeline };

if (globalScript?.length) {
   Object.assign(global, runScript(globalScript, {}, {}, cons0le) || {})
}

const globalContext = {}
const resultContext = {}
const globalStaticContext = { ...global, [globalKey]: globalContext,
[resultKey]: resultContext }

const variables = (dynamicContext) => {
   return { ...global, ...globalContext, ...dynamicContext }
}

const runBlock = async (block_id, dynamicContext, depth = 0) => {

   const block = blocks[block_id]

   if (block) {

      const { request, script, next } = block

      if (request) {

         const vars = variables(dynamicContext)

         const req = prepareRequest(request, vars)
```

```
const reqStart = performance.now()

const results = (await runRequest(req, vars)).filter(r => r)

const reqFinish = performance.now()

const scriptStart = performance.now()

let newDynamicContext = {}

let isStop = false

if (script?.length) {

    const contexts = results.map((res) => {

        allScriptsRunned++

        return runScript(script,
            {
                ...globalStaticContext,
                ...responseToContext(res),
                stop: () => isStop = true
            }
            , {}, cons0le)
    })

    newDynamicContext = contexts.reduce((prev, curr) => (curr ? {
...prev, ...curr } : { ...prev }), {})
```

```
        }

        const scriptFinish = performance.now()

        if (controlContext.stop || depth >= maxDepth) {
            isStop = true
        }
    return resultContext
```