Центральноукраїнський національний технічний університет
Механіко-технологічний факультет
Кафедра кібербезпеки та програмного забезпечення

”Допущено до захисту”
Завідувач кафедри кібербезпеки
та програмного забезпечення
д.т.н., професор
_____ Олексій СМІРНОВ
“____” _____ 2022 р.

ВИПУСКНА КВАЛІФІКАЦІЙНА РОБОТА
за другим (магістерським) рівнем вищої освіти
на тему
“Дослідження та програмна реалізація хмарного сервісу для парсингу сайтів на основі технології Octoparse”

Виконав здобувач вищої освіти
ІІ курсу, групи КН-21М-1,4
ОПП «Комп’ютерні науки»
спеціальності 122 «Комп’ютерні науки»
_____ Калюжний Р.І.
«____» _____ 2022 р.

Керівник проекту
доктор технічних наук, професор
_____ Смірнов О.А.
«____» _____ 2022 р.
Рецензент _____
_____

м. Кропивницький

Центральноукраїнський національний технічний університет

Факультет *Механіко-технологічний*

Кафедра *Кібербезпеки та програмного забезпечення*

Рівень вищої освіти *магістр*

Галузь знань _12 _____*"Інформаційні технології"*_____

Спеціальність 122 *"Комп'ютерні науки"*

Освітньо-професійна (освітньо-наукова) програма _*"Комп'ютерні науки"*_

ЗАТВЕРДЖУЮ
Завідувач кафедри
д.т.н., проф.
_____ Олексій СМІРНОВ
« 6 » вересня 2022 року

ЗАВДАННЯ НА ВИПУСКНУ КВАЛІФІКАЦІЙНУ РОБОТУ ЗА ДРУГИМ (МАГІСТЕРСЬКИМ) РІВНЕМ ВИЩОЇ ОСВІТИ ЗДОБУВАЧА ВИЩОЇ ОСВІТИ

Калюжному Ростиславу Ігоровичу
(прізвище, ім'я, по батькові)

1. Тема роботи *Дослідження та програмна реалізація хмарного сервісу для парсинга сайтів на основі технології Octoparse*

2. Керівник роботи *Смірнов Олексій Анатолійович, д.т.н., професор,*
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)
затверджені наказом вищого навчального закладу № 18-13 від 17.08.2022 року

3. Строк подання студентом роботи до захисту *10.12.2022 р.*

4. Мета та завдання випускної кваліфікаційної роботи*: Метою розробки є дослідження та програмна реалізація хмарного сервісу для парсингу сайтів на основі технології Octoparse*

5. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)

1. Призначення та область використання.	*6. Наукова новизна.*
2.Перегляд аналогічних існуючих систем.	*7. Економічна ефективність розробленої програми.*
3. Опис і обґрунтування проектних рішень.	*8. Заходи з охорони праці та техніки безпеки.*
4. Етапи програмування системи.	*9. Висновки.*
5. Впровадження системи в промислову експлуатацію	

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

Наукова новизна	*1 аркуш*
Структурна схема системи	*1 аркуш*
Функціональна схема системи	*1 аркуш*
Діаграма процесів	*1 аркуш*
Блок-схема алгоритму роботи додатку	*2 аркуша*
Показники економічної ефективності	*1 аркуш*

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Економічний	Савеленко Г.В.	05.10.2022	14.11.2022
Охорона праці	Оришака О.В.	06.10.2022	16.11.2022

7. Дата видачі завдання « 6 » вересня 2022 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів випускної кваліфікаційної роботи за другим (магістерським) рівнем вищої освіти	Строк виконання етапів випускної кваліфікаційної роботи за другим (магістерським) рівнем вищої освіти	Примітка
1.	Аналіз існуючих систем	10.10.2022 р.	
2.	Постановка задачі, оформлення ТЗ	15.10.2022 р.	
3.	Розробка моделі компонента	20.10.2022 р.	
4.	Розробка структур даних	25.10.2022 р.	
5.	Розробка алгоритмів зв’язку та відображення	30.10.2022 р.	
6.	Програмування алгоритмів	10.11.2022 р.	
7.	Розрахунок економічної ефективності	13.11.2022 р.	
8.	Розрахунки з охорони праці та техніки безпеки	15.11.2022 р.	
9.	Оформлення ПЗ	17.11.2022 р.	
10.	Попередній захист роботи	10.12.2022 р.	

Дата видачі завдання
« 6 » вересня 2022 р.

Підпис керівника
_____ Смірнов О.А.
(прізвище та ініціали)

Завдання прийнято до виконання
« 6 » вересня 2022 р.

Підпис здобувача
_____ Калюжний Р.І.
(прізвище та ініціали)

АНОТАЦІЯ

Калюжний Р.І. Дослідження та програмна реалізація хмарного сервісу для парсингу сайтів на основі технології Octoparse. 122 Комп'ютерні науки. Центральноукраїнський національний технічний університет. Кропивницький. 2022.

В даній випускній кваліфікаційній роботі за другим (магістерським) рівнем вищої освіти розроблено хмарний сервіс, який призначений для парсингу сайтів на основі технології Octoparse.

Метою розробки є дослідження та програмна реалізація хмарного сервісу для парсингу сайтів на основі технології Octoparse.

Об'єктом дослідження є процес парсингу сайтів на основі технології Octoparse.

Предметом дослідження є методи парсингу сайтів на основі технології Octoparse.

Методи дослідження базуються на методах аналізу та збіру інформації, методах розробки програмного забезпечення.

Результат роботи – програмна реалізація хмарного сервісу для парсингу сайтів на основі технології Octoparse.

В процесі роботи над програмною моделлю виконано аналіз існуючих апаратних та програмних засобів. В повній мірі описані всі компоненти розробленого програмного забезпечення.

Розроблено зручний інтерфейс користувача. Наведені інструкції по роботі з програмними засобами.

Програма може використовуватися на ПЕОМ архітектури IBM PC, з будь-якою ОС, за умов встановлення платформи Node.js.

Програму розроблено в середовищі Node.js.

Ключові слова: комп'ютерні науки, хмарні сервіси, Octoparse

ABSTRACT

Kaliuzhnyi R.I. Research and software implementation of a cloud service for site parsing based on Octoparse technology. 122 Computer Science. Central Ukrainian National Technical University. Kropyvnytskyi. 2022.

In this graduation thesis for the second (master's) level of higher education, a cloud service designed for site parsing based on Octoparse technology was developed.

The purpose of the development is research and software implementation of a cloud service for site parsing based on Octoparse technology.

The object of research is the process of site parsing based on Octoparse technology.

The subject of the research is methods of parsing sites based on Octoparse technology.

Research methods are based on methods of analysis and information collection, software development methods.

The result of the work is a software implementation of a cloud service for site parsing based on Octoparse technology.

In the process of working on the software model, an analysis of existing hardware and software was performed. All components of the developed software are fully described.

A convenient user interface has been developed. Instructions for working with software tools are provided.

The program can be used on an IBM PC, with any OS, provided the Node.js platform is installed.

The program was developed in the Node.js environment.

Keywords: computer science, cloud services, Octoparse

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ І ТЕРМІНІВ

AWS	–	Amazon Web Service
UX	–	User Experience
UI	–	User Interface
CL	–	Command Line
DOM	–	Document Object Model
HTML	–	HyperText Markup Language
GUI	–	Graphical User Interface
http	–	HyperText Transfer Protocol – протокол передачі гіпертексту
HTTPS	–	розширення протоколу HTTP, підтримуюче шифрування
UA	–	User Agent
AJAX	–	Asynchronous JavaScript and XML
NPM	–	Node Package Manager, менеджер пакетів в Node.js.
API	–	Application Programming Interface
JSON	–	JavaScript Object Notation
CDP	–	Chrome Devtools Protocol
XPath	–	XML Path Language
SQL	–	Structured Query Language
БД	–	база даних
NSG	–	Network Security Group
TSP	–	Time-Stamp Protocol
URL	–	Uniform Resource Locator
SDK	–	Software Development Kit

ВСТУП

Актуальність теми. Отримувати доступ до Інтернету тільки через браузер – означає втрачати масу можливостей. Основне призначення браузерів – виконання скриптів JavaScript, вивід зображень та подання об'єктів у зрозумілій для людини формі. В свою чергу веб-скрапери набагато краще справляються зі швидким збором та обробкою великих обсягів даних. Замість перегляду сторінки за сторінкою на екрані монітора, можна читати відразу цілі бази даних, у яких зберігаються тисячі і навіть мільйони сторінок.

Інтернет містить багато цікавих джерел даних. На жаль, поточна неструктурована природа Інтернету не завжди дозволяє легко збирати або експортувати ці дані в простий спосіб. Так, при пошуку в Google "найдешевших авіарейсів " ви отримаєте купу посилань на рекламні оголошення та популярні сайти пошуку авіарейсів. Google знає лише те, що повідомляється на сторінках змісту цих сайтів, а зовсім не точні результати різних запитів, введених для пошуку рейсів. Однак правильно побудований веб-скрапер здатний створити графік зміни вартості перельоту до конкретного міста на різних сайтах і визначити дати, коли можна купити найвигідніший квиток.

В ході роботи, мого сервісу, доведеться робити високопродуктивні обчислення та зберігати великі обсяги даних. Ефективним рішенням стануть хмарні сервіси, які швидко надають доступ до необхідних ресурсів, дозволяють легко обмінюватися даними та зберігати їх поза локальними комп'ютерами.

Хмарні технології вже сьогодні широко використовуються в математиці, фізиці, археології, дослідженнях атмосфери та клімату, медицині, космічних дослідженнях, енергетиці та інших галузях. Вони розширюють можливості наукових досліджень за рахунок власних потужностей. Хмарні сервіси також впроваджують у невеликі стартапи та великі бізнеси з великою кількістю співробітників, клієнтів та постачальників, оскільки вони забезпечують постійну

синхронізацію та мобільність. В даний час найбільшими хмарними провайдерами є Amazon Web Services (AWS), Microsoft Azure, Google Cloud, IBM Bluemix, Oracle.

Мета й завдання дослідження. Метою роботи є дослідження та програмна реалізація хмарного сервісу для парсинга сайтів на основі технології Octoparse.

Для досягнення поставленої мети складена програма дослідження, що складається з наступних завдань:

– Огляд та аналіз існуючих Web-додатків для парсинга сайтів.

– Огляд існуючих хмарних провайдерів та їх сервісів.

– Програмна реалізація хмарного сервісу для парсинга сайтів на основі технології Octoparse.

Об'єктом дослідження є процес парсинга сайтів на основі технології Octoparse.

Предметом дослідження є методи для парсинга сайтів на основі технології Octoparse, розміщеному в хмарному сервісі.

Методи дослідження базуються на методах пошуку та обробки інформації, хмарних технологій та методах розробки програмного забезпечення.

Наукова новизна отриманих результатів. У процесі рішення завдань, обумовлених цілями дослідження, отримані наступні результати:

– Удосконалено метод для парсингу сайтів.

– Сервіс буде розміщений за допомогою одно з хмарних провайдерів, тому він буде більш продуктивним ніж існуючі аналоги.

Практична цінність отриманих результатів полягає в тому, що розроблений сервіс дозволяють успішно вирішувати задачі парсингу даних з сайтів.

1 ПРИЗНАЧЕННЯ ТА ОБЛАСТЬ ВИКОРИСТАННЯ

1.1 Призначення системи

Існує багато практичних застосувань доступу до даних у мережі та їх збору, багато з яких належать до сфери науки про дані. У наведеному нижче списку описано кілька цікавих випадків використання в реальному житті:

– Багато продуктів Google отримали вигоду від основного бізнесу Google – сканування Інтернету. Google Translate, наприклад, використовує текст, що зберігається в Інтернеті, для навчання та вдосконалення.

– Скрапінг часто застосовується в аналітиці кадрів і співробітників. Стартап hiQ із Сан-Франциско спеціалізується на продажі аналізу співробітників шляхом збору та вивчення загальнодоступної інформації профілю, наприклад, із LinkedIn (який був незадоволений цим, але досі не зміг запобігти цій практиці після судового розгляду).

– Цифрові маркетологи та цифрові художники часто використовують дані з Інтернету для різноманітних цікавих і творчих проектів. «We Feel Fine» Джонатана Гарріса та Сепа Камвара, наприклад, збирав різні сайти блогів на фрази, що починалися з «я відчуваю», результати яких могли візуалізувати, як світ почувався протягом дня.

– Зібрані з Twitter, блогів та інших соціальних медіа, були зібрані для створення набору даних, який використовувався для побудови прогнозної моделі для виявлення депресії та суїцидальних думок. Це може бути безцінним інструментом для постачальників допомоги, хоча, звичайно, він вимагає також ретельного розгляду питань, пов’язаних із конфіденційністю.

– Еммануель Сейлз також скрапив Twitter, хоча тут з метою зрозуміти своє власне коло спілкування та часову шкалу публікацій. Цікавим зауваженням є те, що автор спочатку розглядав можливість використання API Twitter, але виявив, що «Twitter сильно обмежує це: якщо ви хочете отримати список підписок

користувача, ви можете робити це лише 15 разів кожні 15 хвилин, з чим незручно працювати».

– У документі під назвою «The Billion Prices Project: Using Online Prices for Measurement and Research» веб-скрапінг використовувався для збору набору даних інформації про онлайнові ціни, яка використовувалася для побудови надійного щоденного індексу цін для кількох країн.

– Банки та інші фінансові установи використовують веб-збирання для аналізу конкурентів. Наприклад, банки часто сканують сайти конкурентів, щоб отримати уявлення про те, де відкриваються чи закриваються філії, або щоб відстежити пропоновані кредитні ставки – усе це є цікавою інформацією, яку можна включити у їхні внутрішні моделі та прогнози. Інвестиційні фірми також часто використовують веб-збирання, наприклад, щоб відстежувати новинні статті щодо активів у своєму портфелі.

– Соціологи-політологи збирають соціальні веб-сайти, щоб відстежити настрої населення та політичну орієнтацію. Відома стаття під назвою «Дисектування найбільш шалених онлайн-фоловерів Трампа» аналізує дискусії користувачів на Reddit за допомогою семантичного аналізу для характеристики онлайн-фоловерів і шанувальників Дональда Трампа.

– У статті «Analyzing 1000+ Greek Wines With Python» Флорентс Целай збирає інформацію про тисячу сортів вина з грецького винного магазину (див. https://tselai.com/greek-wines-analysis.html), щоб проаналізувати їх походження, рейтинг, тип і міцність.

1.2 Область застосування

Застосування та причини використання такі ж безмежні, як і використання Всесвітньої павутини. Веб-скрапери можуть робити будь-що, наприклад, замовляти їжу в Інтернеті, сканувати веб-сайт онлайн-магазинів для вас і купувати квитки на матч, коли вони доступні, тощо, як це може робити людина. Деякі з області застосування:

– Веб-сайти електронної комерції: можуть збирати дані, спеціально пов’язані з ціною певного продукту, з різних веб-сайтів електронної комерції для їх порівняння.

– Агрегатори контенту: широко використовується агрегаторами, такими як агрегатори новин і агрегатори вакансій, для надання оновлених даних своїм користувачам.

– Маркетингові кампанії та кампанії з продажу: можна використовувати для отримання таких даних, як електронні адреси, номер телефону тощо, для кампанії з продажу та маркетингу.

– Пошукова оптимізація (SEO): широко використовується такими інструментами SEO, як SEMRush, Majestic тощо, щоб повідомити бізнесу, який рейтинг вони мають для ключових слів пошуку, які для них важливі.

– Дані для проектів машинного навчання. Отримання даних для проектів машинного навчання залежить від веб-парсинга.

2 ПЕРЕГЛЯД АНАЛОГІЧНИХ ІСНУЮЧИХ СИСТЕМ

2.1 Огляд існуючих систем, технологій, архітектур, програмних рішень за профілем теми випускної кваліфікаційної роботи за другим (магістерським) рівнем вищої освіти

Терміни веб-кравлінг(Web Crawling) та веб-скрапінг(Web Scraping) часто використовуються як синоніми, оскільки їх основною концепцією є вилучення даних. Однак вони відрізняються один від одного. Ми можемо зрозуміти основну різницю з їхніх визначень. Веб-кравлінг в основному використовується для індексування інформації на сторінці за допомогою ботів, які називаються сканерами. Його ще називають індексуванням. З іншого боку, веб-скрапінг – це автоматизований спосіб вилучення інформації за допомогою ботів, які називаються скребками. Це також називається вилученням даних.

Методи збирання даних

Ось кілька методів, які зазвичай використовуються для збирання даних із веб-сайтів. Загалом, усі методи копіювання веб-сайтів отримують вміст із веб-сайтів, обробляють його за допомогою механізму копіювання та генерують один або кілька файлів даних із витягнутим вмістом.

Розбір HTML. Аналіз HTML передбачає використання JavaScript для націлювання на лінійну або вкладену сторінку HTML. Це потужний і швидкий метод вилучення тексту та посилань (наприклад, вкладеного посилання чи адреси електронної пошти), сканування екранів і отримання ресурсів.

Розбір DOM. Об'єктна модель документа (DOM) визначає структуру, стиль і вміст файлу XML. Скребки зазвичай використовують парсер DOM для глибокого перегляду структури веб-сторінок. Парсери DOM можна використовувати для доступу до вузлів, які містять інформацію, і аналізу веб-сторінки за допомогою таких інструментів, як XPath. Для динамічно створеного вмісту скребки можуть вбудовувати такі веб-браузери, як Firefox і Internet

Explorer, щоб видобувати цілі веб-сторінки (або їх частини).

Вертикальна агрегація. Компанії, які використовують велику обчислювальну потужність, можуть створювати платформи вертикального агрегування для націлювання на певні вертикалі. Це платформи збору даних, які можна запускати в хмарі та використовувати для автоматичного створення та моніторингу ботів для певних галузей з мінімальним втручанням людини. Боти генеруються відповідно до інформації, необхідної кожній вертикалі, і їх ефективність визначається якістю даних, які вони витягують.

XPath. XPath є скороченням від XML Path Language, яка є мовою запитів для документів XML. XML-документи мають деревоподібні структури, тому скребки можуть використовувати XPath для навігації по них, вибираючи вузли відповідно до різних параметрів. Скрепер може поєднувати синтаксичний аналіз DOM із XPath, щоб видобувати цілі веб-сторінки та публікувати їх на цільовому сайті.

Таблиці Google. Google Таблиці – популярний інструмент для збирання даних. Скапери можуть використовувати функцію IMPORTXML у Таблицях, щоб отримати дані з веб-сайту, що корисно, якщо вони хочуть витягти певний шаблон або дані з веб-сайту. Ця команда також дає змогу перевірити, чи веб-сайт можна скопіювати чи він захищений.

Інші технології веб-скрапінгу :

Selenium. Selenium – це інструмент автоматизації веб-браузера, який дозволяє вам робити багато попередньо встановлених речей, як за допомогою бота. навчитися використовувати Selenium допоможе вам зрозуміти, як працюють веб-сайти. Його використання може імітувати звичайне відвідування сторінки людиною за допомогою звичайного веб-браузера, і це дозволить отримати точні дані. Часто він також використовується для емуляції викликів ajax у веб-збиранні. Будучи потужним інструментом автоматизації, він може надати вам більше, ніж просто можливість виконувати веб-збирання. Ви також можете тестувати веб-сайти та автоматизувати будь-які трудомісткі дії в Інтернеті.

Boilerpipe. Якщо вам потрібно витягти текст разом із пов’язаними

заголовками, Boilerpipe стане в нагоді. Boilerpipe – це бібліотека Java, створена для вилучення даних з Інтернету, як структурованих, так і неструктурованих. Він усуває небажані HTML-теги та інші шуми, виявлені на веб-сторінках, залишаючи вам чистий текст. Приваблива особливість цієї технології полягає в тому, що вона виконує швидке сканування веб-сторінок без втручання користувача. Швидкість не впливає на точність даних, оскільки він має високу точність, і все це робить його одним із найпростіших доступних інструментів копіювання.

Nutch. Коли згадується тема золотого стандарту веб-збирання, Nutch є одним із найкращих варіантів, які будуть представлені. Це веб-сканер із відкритим кодом, який блискавично витягує дані з веб-сторінок. Nutch сканує, витягує та зберігає дані після того, як їх було запрограмовано. Його потужний алгоритм виділяє його як один із найкращих інструментів веб-збирання.

Щоб створити scrap за допомогою Nutch, веб-сторінки потрібно закодувати в Nutch вручну. Коли це буде зроблено, він просканує сторінки, отримає необхідні дані та збереже їх на сервері.

Watir. Watir (що вимовляється як вода) – це сімейство бібліотек Ruby з відкритим вихідним кодом, яке є хорошим вибором для автоматизації веб-браузера, оскільки воно просте у використанні та є дуже гнучким. Однією з причин, чому він ідеально підходить для використання в веб-збиранні, є те, що він взаємодіє з веб-браузерами так само, як і люди. Його можна використовувати для натискання посилань, заповнення форм, натискання кнопок і всього, що ви можете собі уявити, що людина робить на веб-сторінці. Ruby робить використання Watir дуже приємним і легким, оскільки Ruby, як і інші мови програмування, дає вам можливість читати файли даних, експортувати XML, підключатися до баз даних і писати електронні таблиці.

Selerity. Це обгортка JRuby, створена навколо HtmlUnit (безголовий Java-браузер із підтримкою JavaScript). Його API простий у використанні та дозволяє легко переміщатися по веб-додатках. Він працює з дуже вражаючою швидкістю, оскільки не вимагає тривалого рендерингу GUI або непотрібних завантажень. Оскільки він масштабований і ненав’язливий, він може безшумно працювати у

фоновому режимі після початкового налаштування. Celerity – це хороший інструмент автоматизації веб-переглядача, який можна використовувати для ефективного сканування веб-сторінок.

Збирання даних і кібербезпека

Інструменти збирання даних використовуються різними компаніями, не обов'язково зі зловмисною метою. До них належать маркетингові дослідження та бізнес-аналітика, веб-вміст і дизайн, а також персоналізація.

Однак сканування даних також створює проблеми для багатьох компаній, оскільки воно може використовуватися для розкриття та неправомірного використання конфіденційних даних. Веб-сайт, який збирається, може не знати, що його дані збираються або що саме збирається. Подібним чином легітимний сканер даних може не зберігати дані безпечно, дозволяючи зловмисникам отримати до них доступ.

Якщо зловмисники можуть отримати доступ до даних, зібраних за допомогою веб-скраппінгу, вони можуть використовувати їх у кібератаках. Наприклад, зловмисники можуть використовувати зібрані дані, щоб:

– Фішингові атаки – зловмисники можуть використовувати зібрані дані, щоб вдосконалити свої методи фішингу. Вони можуть дізнатися, які співробітники мають права доступу, на які вони хочуть націлити, або чи є хтось більш сприйнятливим до фішингової атаки. Якщо зловмисникам вдасться дізнатися особи старшого персоналу, вони зможуть здійснити фішингові атаки, адаптовані до своєї мети.

– Атаки зі злому паролів – зловмисники можуть зламати облікові дані, щоб зламати протоколи автентифікації, навіть якщо паролі не розкриваються безпосередньо. Вони можуть вивчати загальнодоступну інформацію про ваших співробітників, щоб вгадувати паролі на основі особистих даних.

Техніки проти веб-скрапінгу і як їх можна уникнути

IP. Відстеження IP-адреси є одним із найпростіших способів веб-сайту виявити дії веб-збирання, оскільки воно може визначити, чи є IP-адреса роботом чи людиною на основі поведінки. IP-адреса блокується, якщо веб-сайт отримує

багато запитів протягом короткого періоду часу, оскільки така поведінка є неприродною. Для веб-сайту, який має сканер для захисту від сканування, враховуються кількість і частота відвідувань за одиницю часу, і ось деякі сценарії, з якими ви можете зіткнутися під час сканування:

– Сценарій 1: людині неможливо відвідати веб-сайт кілька разів за кілька секунд. Тож під час сканування, якщо ваш сканер часто надсилає запити на веб-сайт, веб-сайт блокуватиме вашу IP-адресу та позначатиме його як робота. Рішення: штучно зменшити швидкість парсера. Налаштування часу затримки між двома кроками завжди спрацює, щоб запобігти виявленню вашої IP-адреси як робота.

– Сценарій 2: відвідування веб-сайту з таким самим темпом і таким самим шаблоном не є людським. Деякі веб-сайти відстежують частоту надсилання запитів і аналізують шаблони надсилання запитів. Якщо вони відбуваються з тією самою схемою, що й раз на секунду, ймовірно, що механізм захисту від подряпин буде активований і заблокує вас. Рішення: встановлюючи час затримки між кожним кроком, виберіть випадковий час. З довільною швидкістю скребка ваш сканер буде більш схожим на людину.

– Сценарій 3: деякі веб-сайти використовують високорівневі методи захисту від сканування, які використовують складні алгоритми для відстеження та аналізу запитів від різних IP-адрес. Якщо запит з IP-адреси незвичайний, наприклад повторюваний і передбачуваний, його буде заблоковано. Рішення: періодично міняйте свій IP. Більшість проксі-сервісів, таких як limeproxies, надають IP-адреси, які ви можете змінювати. Якщо запити надсилаються через ці змінені IP-адреси, ваш веб-сканер буде поводитися не як бот, а як людина. Завдяки такій непередбачуваній поведінці ймовірність блокування вашої IP-адреси зменшується.

Captcha. Переглядаючи веб-сайт, ви напевно натрапляли на такі зображення:

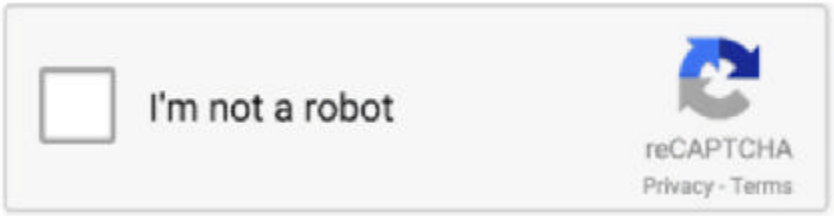

Рисунок 2.1 – Один із видів капчі в Інтернеті

Ці зображення називаються captcha (повністю автоматизований публічний тест Тюрінга для розрізнення комп'ютерів і людей). Це автоматична програма, яка використовується для визначення, чи є користувач людиною чи роботом. Він надає користувачеві різні завдання, такі як погіршення зображення, рівняння, заповнення пропусків тощо, які, як кажуть, вирішує лише людина.

Тести Captcha постійно вдосконалювалися, і багато веб-сайтів використовують їх як техніку проти скрапінгу. Колись було важко передавати captcha безпосередньо, але тепер за допомогою деяких інструментів з відкритим кодом проблеми з captcha можна вирішити під час збирання. Для цього, звичайно, знадобляться більш просунуті навички програмування, і деякі люди йдуть так далеко, що створюють бібліотеки функцій і створюють методи розпізнавання зображень за допомогою машинного або глибокого навчання, щоб проходити перевірку captcha. Однак легше уникнути запуску тесту captcha, ніж вирішити його. Налаштування часу затримки запиту та використання змінних IP-адрес є ефективними способами зменшення кількості тестів captcha.

Log in. Платформи соціальних медіа та багато інших веб-сайтів нададуть вам доступ до інформації лише після того, як ви ввійдете в систему, тому, щоб сканувати такі сайти, веб-сканерам потрібно буде навчитися входити в систему.

Після входу сканер має зберегти файли cookie. Файл cookie – це невеликий фрагмент даних, який зберігається веб-сайтом для користувачів, щоб веб-сайт запам'ятав вас і не вимагав повторного входу. Деякі веб-сайти, все одно дадуть вам доступ до часткової інформації навіть після входу.

Сервіс по веб-скапінгу повинен вміти входити на веб-сайт:

– Імітуйте операції клавіатури та миші. Робот-сканер повинен мати можливість імітувати процес входу, як-от клацання текстового поля та кнопок входу за допомогою миші або введення інформації для входу в обліковий запис за допомогою клавіатури.

– Зберігайте файли cookie після першого входу. деякі веб-сайти дозволяють файли cookie, і це дозволить веб-сайту запам'ятовувати користувачів. Зі збереженими файлами cookie вам не потрібно входити повторно через короткий час. Завдяки цьому вашому веб-сканеру не доведеться кожного разу проходити процедуру входу в систему, і він зможе видаляти стільки даних, скільки вам потрібно.

UA. UA (User agent) означає агент користувача та є заголовком веб-сайту. Він містить інформацію про користувача, наприклад операційну систему, версію, тип ЦП, браузер, версію браузера, мову браузера, плагін браузера тощо.

Прикладом UA є: Mozilla/5.0 (Macintosh; Intel Mac OS X 10_7_0) AppleWebKit/535.11 (KHTML, як Gecko) Chrome/17.0.963.56 Safari/535.11.

Якщо ваш веб-скрапер не має заголовка під час запису, він буде визначений лише як сценарій. Запити від сценаріїв дійсно блокуються, тому, щоб уникнути цього, скрапер має вдавати, що це браузер із заголовком UA, щоб мати доступ.

Іноді веб-сайти надають різну інформацію з однієї сторінки в різні браузери або різні версії, навіть якщо ви відвідуєте їх за тією самою URL-адресою. Це тому, що ви бачите ту інформацію, яка сумісна з браузером, а решта блокується. Тому, щоб переконатися, що ви отримуєте те, що вам потрібно, знадобиться кілька браузерів і версій.

Змінюйте інформацію UA, доки не знайдете потрібну, яка відображає всю необхідну інформацію. Ваш доступ може бути заблокований деякими конфіденційними веб-сайтами, якщо ви продовжуєте використовувати той самий UA протягом тривалого часу, тому, щоб уникнути цього, вам доведеться час від часу змінювати інформацію UA.

AJAX. Більшість веб-сайтів зараз розробляється з використанням AJAX замість традиційних методів веб-розробки. AJAX розшифровується як

Asynchronous JavaScript and XML, і це техніка, яка оновлює веб-сайт асинхронно. Це означає, що весь веб-сайт не потрібно перезавантажувати, коли всередині сторінки відбуваються невеликі зміни.

Для веб-сайтів без AJAX сторінка буде оновлена, навіть якщо ви внесете лише невелику зміну. Під час сканування таких веб-сайтів ви дізнаєтеся, як шаблон URL-адреси змінюється при кожному перезавантаженні. Генерувати URL-адреси пакетами та безпосередньо видобувати інформацію легше, а не вчити сканера навігації веб-сайтами, як люди.

Для веб-сайту з AJAX зміниться лише місце, на якому ви клацаєте, а не вся сторінка, тому ваш сканер працюватиме з цим простим способом. Використання браузера з вбудованими операціями JS автоматично розшифрує веб-сайт AJAX і витягне потрібні вам дані.

Модель хмарного сервісу

Сьогодні хмарні послуги бувають публічні, приватні або гібридні. Вибрана хмарна модель залежить від багатьох факторів, наприклад, які функції ви вважаєте найважливішими, скільки ви хочете інвестувати тощо.

Публічна хмара є найбільш економічно ефективним варіантом. Він простий в управлінні, більш масштабований і надійний. Приватна хмара дає більше контролю над тим, де зберігаються дані, і зберігає їх більш обмеженими за ціною вищої вартості налаштування. Гібридна хмара пропонує більше гнучкості, поєднуючи елементи обох типів моделей. Але він також має кілька недоліків, особливо якщо розглядати розташування кількох точок доступу (AP). Вибір між публічним, приватними та гібридними хмарними рішеннями не є ситуацією або/або. Багато організацій використовують усі три типи хмарних рішень, враховуючи притаманні ціннісні пропозиції та компроміси.

Одна хмара чи багато хмар

Все більше компаній використовують різні рішення від різних постачальників хмарних технологій. Мультихмарність зростає. Дані Forrester показують, що 75% глобальних керівників інфраструктури описали б свою хмарну стратегію як гібридну. Майже кожна компанія дотримується багатохмарної

стратегії через застарілу інфраструктуру («у нас є всього потроху») або щоб уникнути прив'язки до постачальника.

Незалежно від того, використовуєте ви одного хмарного постачальника чи керуєте кількома хмарними платформами, кожна стратегія має свої плюси та мінуси. З одним хмарним постачальником існує ризик блокування постачальника. За допомогою багатохмарної стратегії ви можете змусити свій код працювати з кількома хмарними постачальниками та збалансувати навантаження між кількома хмарними платформами. Цей варіант дорожчий і складніший, оскільки кожен провайдер пропонує різні послуги та інструменти для управління, але він дає певний ступінь свободи та гнучкості.

Види хмарних обчислень

Переважна більшість організацій (майже 90%) використовують ту чи іншу форму хмарних обчислень. Залежно від послуги це – може бути, інфраструктура як послуга (IaaS), програмне забезпечення як послуга (SaaS), платформа як послуга (PaaS) або функція як послуга (FaaS). Звіт IDC визначає SaaS як найбільшу категорію витрат, охоплюючи більше половини всіх витрат на публічні хмари протягом прогнозованого періоду. Повідомляється, що IaaS є другою за величиною категорією витрат і є категорією, що розвивається найшвидше, з прогнозованим CAGR за п'ять років у 32,0%. PaaS – це найнижча категорія витрат із другим за величиною CAGR за п'ять років у 29,9%.

Хмарні провайдери

На сьогодні існує понад 100 постачальників хмарних послуг, але на ринку домінують AWS, Azure, Google Cloud, IBM Cloud і Oracle Cloud. Згідно зі звітом Canalys and Synergy Research Group за 4 квартал 2019 року, AWS є явним світовим лідером на ринку хмарних послуг (32,3%), за нею йдуть Microsoft з 17% і Google з 6%.

З 2006 року, коли AWS була вперше запущена, вона швидко перетворилася на домінуючу хмарну платформу. Хоча GCP і Azure вийшли на ринок кількома роками пізніше, вони склали жорстку конкуренцію AWS як надійні публічні хмарні постачальники як серед стартапів, так і серед підприємств.

Node.js

Node.js - це середовище виконання, яке базується на двигуні Chrome V8. Технологія була вперше представлена в 2009 році Райаном Далем на щорічному європейському форумі JSConf і відразу ж була визнана найвагомішою частиною програмного забезпечення в сучасному всесвіті JavaScript.

Використовуючи Node.js як основу, ви автоматично отримуєте всі переваги повнофункціональної розробки на JavaScript, серед яких:

– оптимальна швидкість роботи додатків;

– можливість спільного та повторного використання;

– безліч безкоштовних інструментів;

– кросплатформеність.

Node.js працює швидко. Це доводять результати продуктивності тестів, представлені toptal.com. Дослідники порівняли, як популярні мови, такі як GO, PHP, Java та Node.js, як вони обробляють одночасні запити. Останній демонструє найкращі результати практично за всіма показниками завдяки використанню двигуна V8.

Ще однією важливою перевагою є асинхронна обробка запитів. У контексті серверної частини синхронна обробка передбачає, що код виконується послідовно. Таким чином, кожен новий запит блокує потік решти, а інші команди почнуть виконуватись лише після того, як буде виконано попередню. У свою чергу, асинхронний метод, який використовується в Node.js, максимально використовуючи однопоточну обробку, що скорочує час відгуку в кілька разів.

Третій аспект – це модель подій. При використанні однієї мови як на стороні клієнта, так і в бекенді синхронізація відбувається максимально швидко, що особливо корисно для додатків реального часу, заснованих на подіях.

Будучи простим та легким середовищем програмування, Node.js став ідеальним рішенням для так званої архітектури мікросервісів. Цей підхід передбачає розробку одного додатка як набору невеликих сервісів, кожен з яких працює з власними процесами та взаємодіє з легковажними механізмами, часто з API ресурсів HTTP.

NPM – менеджер пакетів Node.js за замовчанням – також є основною платформою для інструментів JavaScript з відкритим вихідним кодом, які відіграють важливу роль у розвитку цієї мови програмування. З урахуванням того, що на даний момент у реєстрі npm доступно близько мільйона бібліотек і щотижня публікується понад 10 000 нових, екосистема Node.js досить різноманітна.

Amazon Web Service

AWS має найбільшу спільноту клієнтів і партнерів у всьому світі, що робить її номером один серед інших постачальників хмарних послуг. AWS представлена в 77 зонах у 24 географічних регіонах по всьому світу та оголосила про плани створення ще дев'яти зон доступності та ще трьох регіонів AWS в Індонезії, Японії та Іспанії. AWS – це безпечне середовище хмарних обчислень. AWS Security Hub надає вам низку інструментів безпеки, починаючи від брандмауерів і захисту кінцевих точок до сканерів вразливостей і відповідності. Він надає бази даних як SQL, так і NoSQL. Amazon RDS – це служба реляційної бази даних, яка має шість механізмів баз даних: Amazon Aurora, MySQL, PostgreSQL, Oracle, MariaDB і Microsoft SQL Server. Рішення NoSQL від Amazon – DynamoDB. Крім того, він підтримує MongoDB, Redis, Mongo 3, Memcached, Cassandra.

Основна сила AWS полягає в публічному хмарному ринку. Тому він менш відкритий для приватних або сторонніх хмарних провайдерів. Він забезпечує гібридну підтримку через партнерство з локальними постачальниками. За останні роки він значно розвинув напрямок гібридної хмари за допомогою таких рішень, як VMware і Outposts.

Що стосується ціноутворення, AWS має модель оплати за використання, і ви платите за годину.

Відокремлю основні переваги AWS:

– Найширша мережа в дата-центрах по всьому світу: вона пропонує високу надійність та безпеку у своїх послугах та у мережній інфраструктурі, яку він пропонує своїм клієнтам. Його великий досвід також дозволив розширити спектр послуг, які він пропонує порівняно з іншими аналогічними платформами.

– Гнучкість для проектів: незалежно від того, який тип продукту, проекту або рішення вам потрібно інтегрувати у вашу робочу команду або бізнес, ви знайдете в AWS всі сервіси, необхідні для цього. Інструменти легко адаптуються та легко налаштовуються відповідно до потреб кожного середовища.

– Бюджетний та можливість економії: позбавляючись необхідності вкладати кошти в обслуговування та оновлення обладнання та програмного забезпечення, ви можете зосередитися на основних цілях свого бізнесу або організації. Крім того, AWS стягує плату тільки за використання сервісів і може масштабуватися, наприклад, ціни та функції, які пропонують Amazon Elastic Compute Cloud.

2.2 Обґрунтування вибору засобів для побудови системи та мови програмування

Для реалізації програми мною була використана мова програмування JavaScript та платформа NodeJS та хмарне середовище Amazon Web Services. Для скрапінгу – C# чи Python підходять краще, ніж будь-яка інша мова програмування. Є багато бібліотек у цих мовах, які облегчать збирання даних. Незважаючи на це в NodeJS, також є потрібні інструменти та бібліотеки для реалізації веб-скрапера.

Найбільшим недоліком Node.js зараз є його нездатність обробляти важкі завдання, пов'язані з тяжкими обчисленнями з процесором. Частково вирішує проблему введена в середовище багатопоточність.

У 2018 році багатопоточність була представлена як експериментальна функція Node.js версії 10.5.0. Так званий модуль робочих потоків може задіяти додаткові потоки, але тільки в процесорах з кількома ядрами. За останні кілька років ця функція значно вдосконалилася, та стала пріорітетною в багатьох програмах проте в обробці важких завдань рішення, як і раніше, поступається альтернативам.

Що стосується хмарного сервісу то AWS має більш ніж 10-річний досвід

роботи в хмарних рахуваннях, що гарантує безліч переваг платформи. Недаром багато відомих організацій довіряють цьому сервісу свої дані, проекти, аналізи, звіти та навіть всю свою інфраструктуру. Але не забуду відокремити деякі мінуси:

– Він не адаптований до конкретного середовища: будучи такою великою та різноманітною платформою, дуже легко загубитися у кількості пропонованих послуг та залишити осторонь роботу, проект чи потреби компанії.

– Складання бюджету та планування витрат утруднені: у деяких випадках плата за користування може бути перевагою, в інших – ні. Коли планування витрат є суворим, це може бути дуже складно визначити загальну суму комісійних за придбані послуги, особливо коли робочий процес більш напружений.

2.3 Розгорнута постановка завдання

Згідно з технічним завданням на випускну кваліфікаційну роботу за другим (магістерським) рівнем вищої освіти, реалізується хмарний сервіс для парсингу сайтів на основі технології Octoparse.

В процесі розробки випускної кваліфікаційної роботи за другим (магістерським) рівнем вищої освіти необхідно виконати наступний обсяг роботи:

а) провести аналіз існуючих систем парсингу сайтів та хмарних сервісів для виявлення їх позитивних і негативних якостей. Результати аналізу врахувати в подальших розробках;

б) вибрати та обґрунтувати методику побудови системи сервісу. Розробити функціональну та структурну схеми системи;

в) розробити програмне забезпечення системи, що дозволить реалізувати поставлену технічним завданням задачу. Побудувати блок-схеми алгоритмів програми та підпрограми;

г) організувати інтерфейс користувача;

д) розробити рекомендації по організаційних та методичних заходах, які забезпечать впровадження системи в промислову експлуатацію та її подальшу

успішну експлуатацію;

е) провести розрахунки по визначенню економічної ефективності розробленої системи;

ж) розробити заходи по охороні праці при впровадженні та експлуатації системи, а також розробити заходи з цивільного захисту;

з) сформувати висновки про виконаний обсяг робіт та одержані результати.

3 ОПИС І ОБҐРУНТУВАННЯ ПРОЕКТНИХ РІШЕНЬ

3.1 Опис функціонування системи

Робота парсера

Веб-парсер можна визначити як програмне забезпечення або сценарій, який використовується для завантаження вмісту кількох веб-сторінок і вилучення з нього даних.

NodeJS у веб-парсинга в основному відомий завдяки набору інструментів автоматизації браузера Puppeteer. Використання автоматизації веб-браузера для веб-скрапінгу має багато переваг, однак це дійсно складний і ресурсно-важкий підхід до веб-парсинга. За допомогою невеликого зворотного проектування та деяких розумних пакетів nodeJS ми можемо досягти подібних результатів без повних витрат на веб-браузер.

Перш за все я зосереджусь на кількох інструментах зокрема – для з’єднання буду використовувати HTTP-клієнт axios, а для синтаксичного аналізу зосереджусь на синтаксичному аналізаторі HTML-дерева cheerio.

Двома найбільшими проблемами веб-парсинга є масштабування та уникнення блокування. Для швидкого сканування нашим скраперам потрібно використовувати проксі-сервери, щоб уникнути різноманітних обмежень швидкості, які накладає веб-сайт. Крім того, веб-сайти можуть заблокувати нас у будь-який час, якщо виявлять, що ми робот, а не справжній користувач.

Створення ресурсів у хмарі

Хмарні провайдери мають в основі своїх сервісів величезні дата-центри, чиї обчислювальні ресурси за допомогою системи віртуалізації поділяються на невеликі частини: голі віртуальні машини різних розмірів із встановленою операційною системою (IaaS) та групи віртуальних машин із встановленим софтом, що надають доступ лише до своїх можливостей (PaaS). Так ось, створити хмарний ресурс – означає надіслати запит контролеру ресурсів,

розміщеному в хмарному ЦОДі, на виділення необхідних обчислювальних ресурсів з доступних пулу. Тобто, по суті, ресурс не створюється з нічого, а лише виділяється на вимогу. І тут можлива ситуація (рідко, але буває), коли користувач попросив ресурси у контролера, а вони не з'явилися. Це відбувається через те, що фізичні ресурси, на яких розміщуються віртуальні, вже задіяні іншими користувачами.

Завдання оптимального розподілу доступних ресурсів між користувачами вирішує контролер. І якщо користувач при створенні ресурсів зіткнувся з проблемою, він повинен повторити спробу, вдавшись до різних варіацій (повторити через деякий час, змінити регіон і повторити, змінити обліковий запис і повторити тощо). З точки зору користувача, виділення ресурсів виглядає як створення ресурсів: він виконав низку дій на веб-порталі, і в останньому з'явилися ресурси.

Насправді, звичайно, вони були виділені, і про це не варто забувати. Однак для простоти та наочності я застосовуватиму термін «створення». Існує чотири способи керування хмарною інфраструктурою.

Перший, найпростіший і очевидніший – задіяти веб-портал. При цьому користувач повинен мати відповідні права на створення ресурсів. Ручний спосіб дуже простий: у всіх хмарних провайдерів є зручні портали, велика документація, відео-інструкції та ін. Не потрібні додаткові сервіси, SDK та ін.

Однак цей спосіб має недоліки:

– тривалий час створення інфраструктури;

– недостатня надійність (у разі проблем із ресурсами їх доведеться перетворювати вручну, з усіма ручними налаштуваннями, конфігуруванням тощо);

– трудність перенесення інфраструктури в новий регіон або обліковий запис - її знадобиться вручну клонувати або копіювати (даний недолік частково згладжується тим, що хмарні провайдери дозволяють копіювати або клонувати ресурси, але ця процедура все одно вимагає ручного ініціювання);

– процес створення ресурсів у цьому випадку неможливо автоматизувати.

Другий спосіб - застосувати програмні бібліотеки (Software Development Kit, SDK), що забезпечують доступ до ресурсів хмари з коду програм користувача. Як правило, SDK є набором класів і методів, що полегшують програмні операції з ресурсами хмари. Щоб забезпечити доступ до таких ресурсів, програма з хмарним SDK повинна містити ключі облікового запису, який матиме доступ до хмари. У хмарі ці ключі зареєстровані у вигляді користувача в активному каталозі хмарного облікового запису, що має права виконувати програмне маніпулювання ресурсами хмари (такий користувач називається принципалом – service principal). І керування цими обліковими записами відбувається так само, як і обліковими записами користувачів хмарного веб-порталу. Серед переваг такого підходу – можливість створення програм, які самі собі створюють хмарні ресурси, а також автоматизованого управління хмарним акаунтом.

До третього способу створення хмарних ресурсів відносять спеціалізовані розширення для мов командного рядка – shell, CMD та ін. Для підключення цих розширень до хмарних ресурсів необхідно імпортувати ключі або виконати вхід до облікового запису через форму введення логіна/паролю. Як і у випадку SDK для сценарних мов програмування, SDK для командної оболонки дозволяє описувати хмарну інфраструктуру як набору команд, кожна з яких створює чи конфігурує відповідний хмарний сервіс. І SDK, і команди оболонки оперують зрештою з API хмарного провайдера (як правило, REST API), доступ до яких також дозволить маніпулювати ресурсами хмари.

Четвертий спосіб створення хмарних ресурсів – застосувати шаблони. У цьому випадку всі необхідні ресурси та зв'язки між ними описуються за допомогою текстового файлу у форматі YAML або JSON. Такий шаблон може бути завантажений у хмарний сервіс безпосередньо через веб-портал або через CLI-команди.

На веб-порталі AWS є спеціальний редактор, що спрощує створення та конфігурування шаблону. Останній може бути завантажений у файлове сховище S3, репозиторій CodeCommit або інше місце, доступне для сервісу

CloudFormation. Цей сервіс створює стек – набір ресурсів, керованих спільно (створення, видалення та оновлення).

Безпека хмарних ресурсів

Поряд із незаперечними перевагами, зберігання та обробка даних у хмарних середовищах потенційно може доставити низку проблем, яких немає (або, вірніше, вони виявляються не так чітко) у разі розміщення та обробки даних у власних дата-центрах. Це зумовлено низкою причин. По-перше, хмарні середовища самі собою публічно доступні і всі сервіси, якщо явно не налаштовано інше, доступні для всіх в Інтернеті. По-друге, захист даних та інфраструктури від ненавмисних дій користувачів лежить поза компетенцією хмарного провайдера. Крім того, хмарні інфраструктури, що працюють з великими даними, часто містять у своєму складі великі кластери віртуальних машин, що потребує застосування спеціальних заходів для забезпечення надійної роботи системи.

Найбільш поширений спосіб захисту кінцевих точок хмарних сервісів – обмеження доступу до них за допомогою механізмів автентифікації та створення списків дозволених IP-адрес, з яких можна отримати доступ до точок.

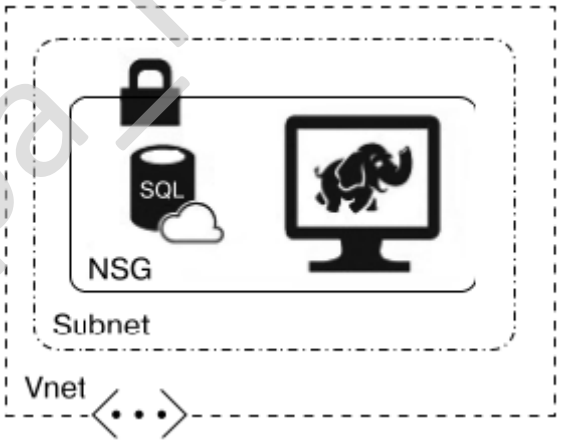

Рисунок 3.1 – Обмеження доступу до кінцевих точок хмарних сервісів за допомогою мережевих груп безпеки

Сервіси, що стосуються IaaS, а також у ряді випадків до PaaS, вимагають для свого створення налаштованої хмарної віртуальної приватної мережі (VNet,

VPC), розбитої на підмережі. Доступ до кінцевих точок сервісів, розташованих у цих підмережах, можна регулювати за допомогою конфігурування мережевих груп безпеки (Network Security Group, NSG) на рисунку 3.1, які є списками контролю доступу, ACL.

Отже, віртуальна частина мережі – один із базових сервісів IaaS. Він є хмарним аналогом локальної мережі і служить для надання діапазону IP-адрес для розміщення в них ресурсів. Віртуальну приватну мережу можна розділити на підмережі (subnet), а між ними встановити правила маршрутизації IP-пакетів. Крім того, на підмережі можна встановити списки контролю доступу, які називаються мережними групами безпеки. Це дозволяє логічно розділяти архітектури інформаційних систем на різні рівні (наприклад, рівень даних, бізнес-логіки, фронтенд) шляхом розміщення кожного рівня своєї підмережі та встановлення правил маршрутизації.

Ще один «ешелон» захисту даних у хмарних ресурсах – це їхнє шифрування. Поширений підхід у разі – про прозоре шифрування даних (transparent data encryption, TDE). Суть його полягає в тому, що все шифрування та дешифрування даних відбувається «за лаштунками», без участі користувача, за допомогою ключів шифрування, які генеруються та зберігаються самим хмарним обліковим записом. У разі Azure ці ключі зберігаються в Azure KeyVault, а у випадку AWS – AWS KMS.

3.2 Розробка структурної схеми

Ми можемо зрозуміти роботу веб-скрапера за допомогою простих кроків.

– Крок 1: Завантаження вмісту з веб-сторінок. На цьому кроці веб-скребок завантажить потрібний вміст із кількох веб-сторінок.

– Крок 2: Вилучення даних. Дані на веб-сайтах є HTML і здебільшого неструктуровані. Отже, на цьому етапі веб-скрапінгу розбере та витягне структуровані дані із завантаженого вмісту.

– Крок 3: Зберігання даних. Тут веб-скрапер зберігатиме витягнуті дані в будь-якому форматі, наприклад CSV, JSON або в базі даних.

– Крок 4: Аналіз даних. Після успішного виконання всіх цих кроків веб-скребок проаналізує отримані таким чином дані.

Рисунок 3.2 – Проста схема роботи веб-скрапера

Веб-краулинг

Веб-краулери – основа того, що рухає багатьма сучасними веб-технологіями, і для їх використання не обов'язково мати велике сховище даних. Щоб виконати будь-який крос-доменний аналіз даних, необхідно розробити веб-краулери, здатні інтерпретувати та зберігати дані, зібрані з багатьох інтернет-сторінок. Веб-краулери, які ми маємо намір створити, переходитимуть за посиланнями від сторінки до сторінки, формуючи карту мережі. Вони не ігноруватимуть зовнішні посилання, а, навпаки, переходитимуть по них. Не обов'язково зовнішні посилання виявляться на першій сторінці сайту. У цьому випадку для пошуку зовнішніх посилань використовується метод, аналогічний тому, який застосовувався в попередньому прикладі веб-краулінгу для

рекурсивного переходу в глибину сайту, доки не буде знайдено зовнішнє посилання.

Розподіл задач на прості функції, такі як «знайти всі зовнішні посилання на сторінці», має перевагу: згодом код можна легко перебудувати для виконання інше завдання веб-парсингу. Наприклад, якщо мета полягає в тому, щоб переглянути сайт, знайти всі зовнішні посилання та зберегти їх.

Перед реалізацією веб-парсера, який, переходитиме по всіх зовнішніх посиланнях, слід поставити собі кілька запитань.

– Які дані я маю намір зібрати? Чи достатньо для цього виконати веб-парсинг кількох заздалегідь визначених сайтів (що майже завжди простіше зробити), чи мій парсер повинен мати можливість виявляти нові сайти, про які я, ймовірно, не підозрюю?

– Потрапивши на певний сайт, чи повинен мій краулер відразу переходити за наступним зовнішнім посиланням на інший сайт, чи йому слід затриматися на якийсь час на цьому і вивчити його більш поглиблено?

– Чи існують якісь умови, за яких я б не хотів виконувати веб-скрапінг поточного сайту? Чи цікавить мене контент іноземними мовами?

– Як я захищатимусь від судових позовів, якщо мій веб-краулер приверне увагу веб-майстра одного з сайтів, на які потрапить.

Схема роботи хмарного сервісу

NSG (з рисунка 3.1) є список доступу, що містить набір записів. Кожен запис складається з таких елементів, як:

– назва;

– число, що визначає пріоритет перегляду списку записів;

– діапазон IP-адрес (для однієї конкретної адреси це /32);

– номер порту;

– дія – ALLOW або DENY («Дозволити» або «Відхилити») стосовно запиту, що надійшов з цієї адреси.

Крім того, вказується протокол, до якого застосовується дія ALLOW або DENY (TCP, UDP, ICMP тощо). Безпека кінцевих точок у разі забезпечується

обмеженням до них доступу ззовні. Крім NSG, ряд хмарних сервісів, що не вимагають віртуальної приватної мережі (наприклад, Azure SQL), мають фаєрволи списки «дозволених» і «заборонених» діапазонів. Хорошою практикою є повсюдне використання NSG та фаєрволів. При цьому необхідно, щоб усі порти, що стосуються віддаленого доступу/керування (наприклад, 22 для SSH, 3388 для RDP) або безпосередньо до сервісу (скажімо, 1433 для MS SQL) були недоступні з Інтернету поза діапазоном адрес віртуальної приватної мережі. Для отримання ж доступу до сервісів із «дозволеної» локальної мережі або з дозволеного комп'ютера слід встановити VPN-шлюз з локальної мережі або з комп'ютера до віртуальної приватної мережі або безпосередньо до екземпляра сервісу. Крім шлюзу, при з'єднанні локальної мережі з віртуальною приватною мережею необхідно застосувати проміжний хост, проксі-хост (рис 3.3), який транслюватиме запити та дозволятиме приватні адреси хмарних ресурсів з локальної мережі. Проксі-хост у різних реалізаціях можна розмістити як у мережі хмар, так і в локальній. В останній може розташовуватися контролер домену, сервер БД з даними, які не можуть бути розміщені у хмарі, а також інші сервери, які можуть бути розміщені лише у локальній мережі.

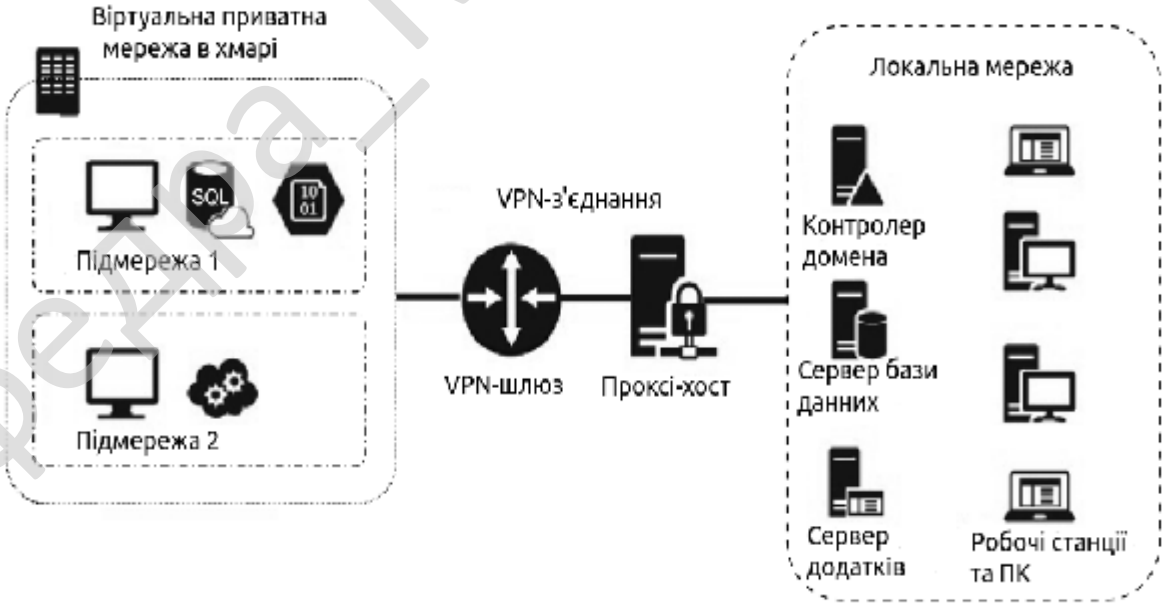

Рисунок 3.3 – Структурна схема системи

3.3 Розробка функціональної схеми

У якості функціональної схеми, так як вона є складовою частиною структурної схеми, наведемо функціональну схему роботи сервісу.

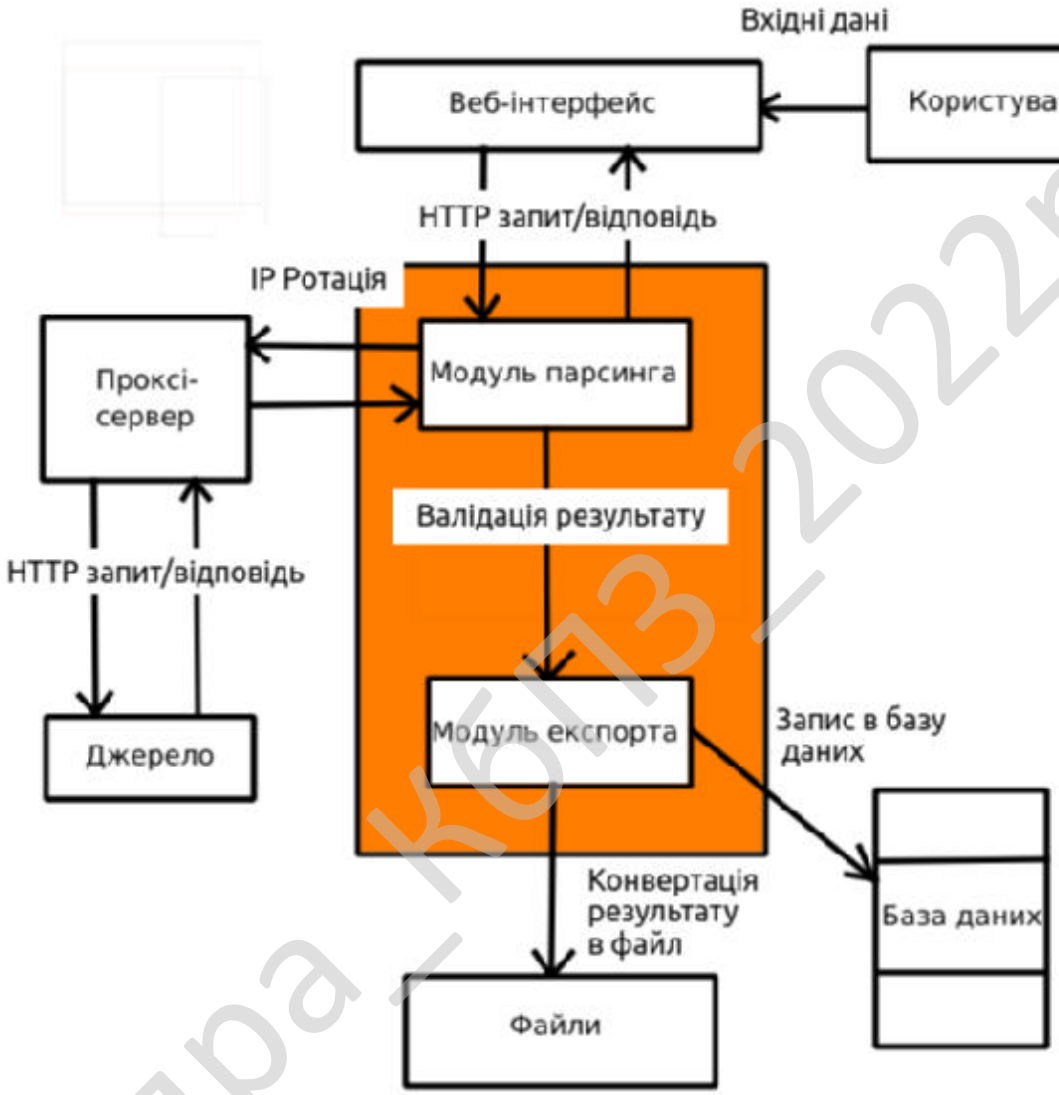

Рисунок 3.4 – Функціональна схема роботи системи

Робота відбувається наступним чином:

1. Користувач вказує початкові вхідні дані, такі як URL сайту та параметри парсингу.

2. Веб-інтерфейс приймає дані, в виді веб-форми, та відправляє її на сервер.

3. Модуль парсингу генерує динамічний IP-адрес та відправляє його в проксі-сервер.

4. Проксі-сервер завантажує код сторінки сайту, з яким працює спеціальний скрипт. Він поділяє весь код на лексеми, оцінює, які відомості потрібні користувачеві.

5. На етапі валідації парсер знайде у коді сторінки місце із вхідних параметрів користувача. Потім визначить, де розташовані потрібні дані, і сформує файл виключно з ними.

6. Після отримання необхідних даних із сайту їх потрібно зберегти. Як правило, часто їх заносять до таблиць, щоб бачити наочно.

7. Можна заносити інформацію до бази даних. У цьому випадку користувач вибирає найзручніший варіант.

Є кілька способів розповсюдження даних, але найбільш релевантними технологіями є XML/HTML, AJAX і JSON.

Другий стовп технологій збору веб-даних потрібен для отримання інформації з файлів, які ми збираємо. Залежно від техніки, яка використовувалася для збору файлів, існують спеціальні інструменти, які підходять для отримання даних із цих джерел. Першим інструментом у нашому розпорядженні є мова запитів XPath. Він використовується для вибору певних частин інформації XPath із розмічених документів, таких як HTML, XML або будь-який їх варіант, наприклад SVG або RSS. У типовому завданні веб-копіювання даних виклик веб-сторінок є важливим, але зазвичай лише проміжним кроком на шляху до добре структурованих і очищених наборів даних.

Головним завданням веб-парсинга є збір необхідної інформації для нашої дослідницької проблеми з купи текстових даних. Зазвичай ми дбаємо про систематичні елементи в текстових даних, особливо якщо ми хочемо застосувати кількісні методи до отриманих даних. Систематичними структурами можуть бути числа або назви, наприклад країни чи адреси. Одним із методів, який ми можемо застосувати для вилучення систематичних компонентів інформації, є регулярні вирази. По суті, регулярні вирази – це абстрактні послідовності рядків, які

відповідають конкретним, повторюваним шаблонам у тексті. Окрім використання їх для вилучення вмісту з документів із звичайним текстом, ми також можемо застосувати їх до документів HTML і XML, щоб ідентифікувати та витягти частини документів, які нас цікавлять.

Нарешті, третій стовп технологій збору веб-даних стосується засобів зберігання даних. Загалом, зв'язок між технологіями вилучення інформації та технологіями зберігання даних менш очевидний. Найкращий спосіб зберігання даних не обов'язково залежить від їх походження. Прості та повсякденні процеси, як-от покупки в Інтернеті, перегляд бібліотечних каталогів, перерахунок грошей через SQL або навіть купівля солодощів у супермаркеті, – все це пов'язано з базами даних. Ми навряд чи усвідомлюємо, що бази даних відіграють таку важливу роль, оскільки ми не взаємодіємо з ними безпосередньо – бази даних люблять працювати за лаштунками. Щоразу, коли дані є ключовими для проекту, веб-адміністратори покладаються на бази даних через їхню надійність, ефективність, багатокористувацький доступ, практично необмежений розмір даних і можливості віддаленого доступу. Що стосується автоматизованого збору даних, бази даних представляють інтерес з двох причин: по-перше, іноді нам може бути надано прямий доступ до бази даних, і ми повинні впоратися з нею. По-друге, незважаючи на те, що R має багато засобів керування даними, можливо, краще зберігати дані в базі даних, а не в одному з рідних форматів. Наприклад, якщо ви працюєте над проектом, де дані потрібно зробити доступними в Інтернеті, або якщо у вас є різні сторони, які збирають певні частини ваших даних, база даних може забезпечити необхідну інфраструктуру. Крім того, якщо дані, які вам потрібно зібрати, є великими, і вам доводиться часто множити дані та маніпулювати ними, також має сенс налаштувати базу даних для швидкості, з якою до них можна запитувати.

3.4 Розробка діаграми процесів

Розглянемо процеси, що відносяться до життєвого циклу роботи сервісу. Після отримання вхідних даних, відбувається аналіз заданого URL адреса, отримання масиву даних їхня агрегація заданими параметрами та зберігання в потрібному форматі.

Розглянемо процеси, що стосуються безпосередньо парсингу.

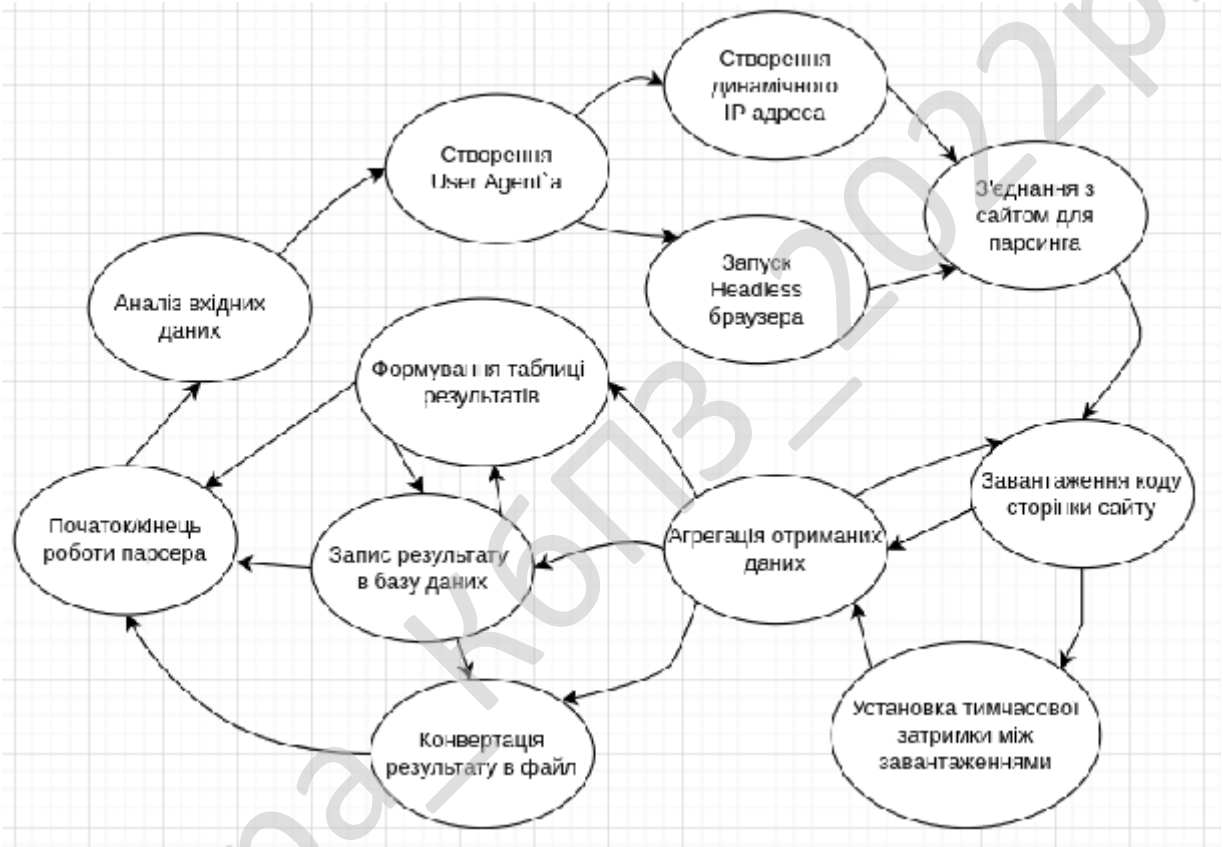

Рисунок 3.5 – Діаграма процесів системи

Таким чином, розглянувши опис системи, структурну, функціональну схеми системи, та діаграму взаємодії процесів перейдемо до опису блок-схем основної програми, та підпрограм, які використовуються, для реалізації системи.

4 РЕАЛІЗАЦІЯ РОБОТИ. РОЗРАХУНКИ І ЕКСПЕРИМЕНТАЛЬНІ ДАНІ, ЩО ПІДТВЕРДЖУЮТЬ ВІРНІСТЬ ПРОЕКТНИХ ТА ПРОГРАМНИХ РІШЕНЬ

4.1 Блок-схеми та опис алгоритмів функціонування системи

Після запуску розробленої системи на екран виводиться основне вікно програми. Перед з'єднанням з сервером користувач повинен ввести параметри. Зазвичай існує два підходи до збору даних: надсилання HTTP-запитів і використання засобів автоматизації браузера для керування веб-браузерами. Хоча підхід HTTP є дуже ефективною формою збору даних, часто від нас вимагається інвестувати додатковий час у розробку, щоб виконати зворотне проектування та зрозуміти нашу мету.

Згадана підсистема дозволить зробити інтерфейс користувача більш простішим за рахунок автоматизації процесу парсингу. Автоматизація веб-переглядача споживає більше ресурсів і працює повільніше, заощаджує нам багато зусиль і є більш доступною формою автоматизації. Я розгляну Puppeteer, який є чудовою бібліотекою для автоматизації браузера для JavaScript (NodeJS).

Скрапер на основі веб-браузера бачать те, що бачать користувачі. Іншими словами, браузер відтворює всі сценарії, зображення тощо, що означає, що розробка веб-скребка легша, оскільки ми працюємо з тими самими даними, що й користувач веб-сайту. Скрапер на основі веб-браузера важче виявити та заблокувати. Оскільки ми виглядаємо як звичайні користувачі веб-сайту, нас набагато важче ідентифікувати.

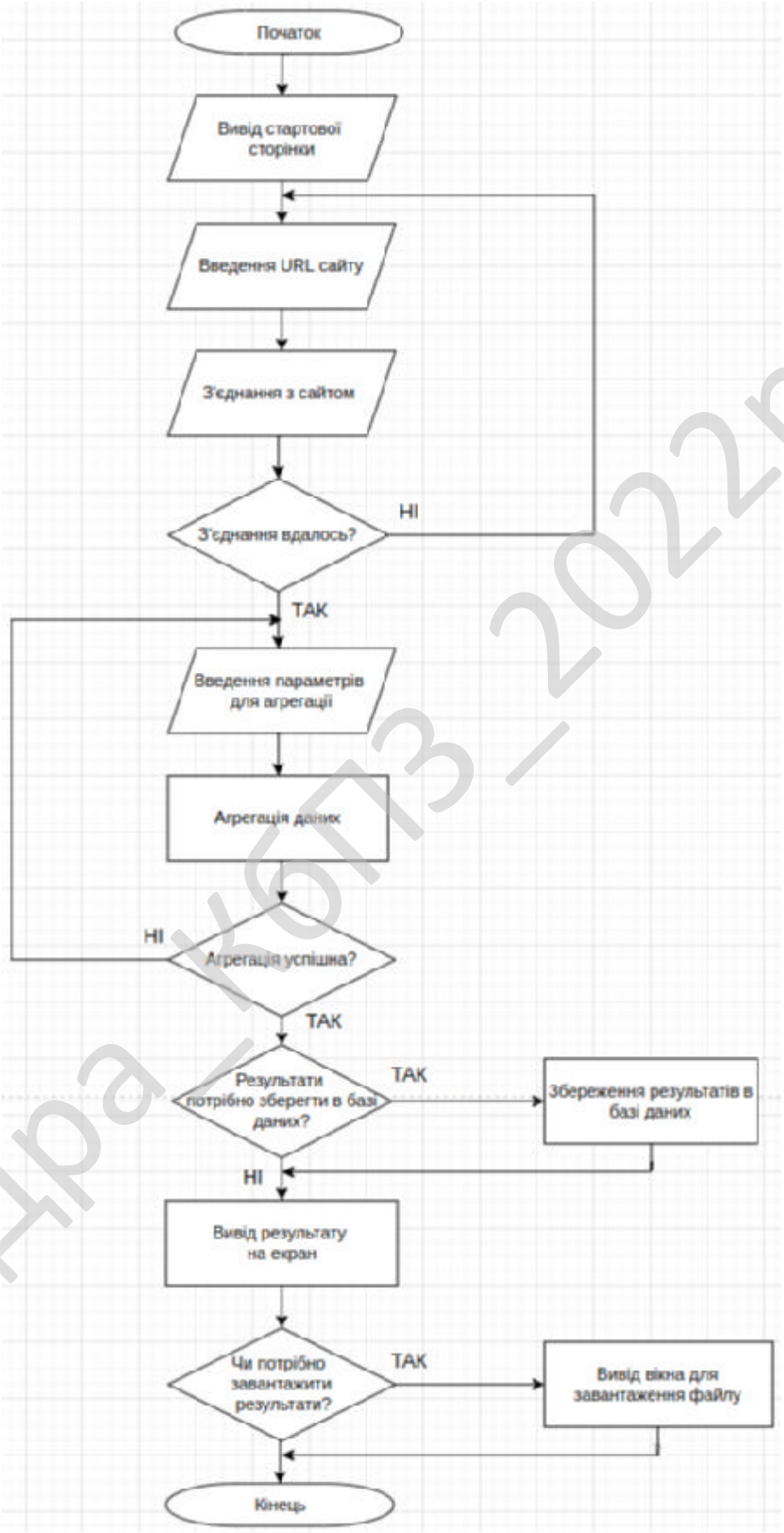

Рисунок 4.1 – Блок-схема основного алгоритму програми

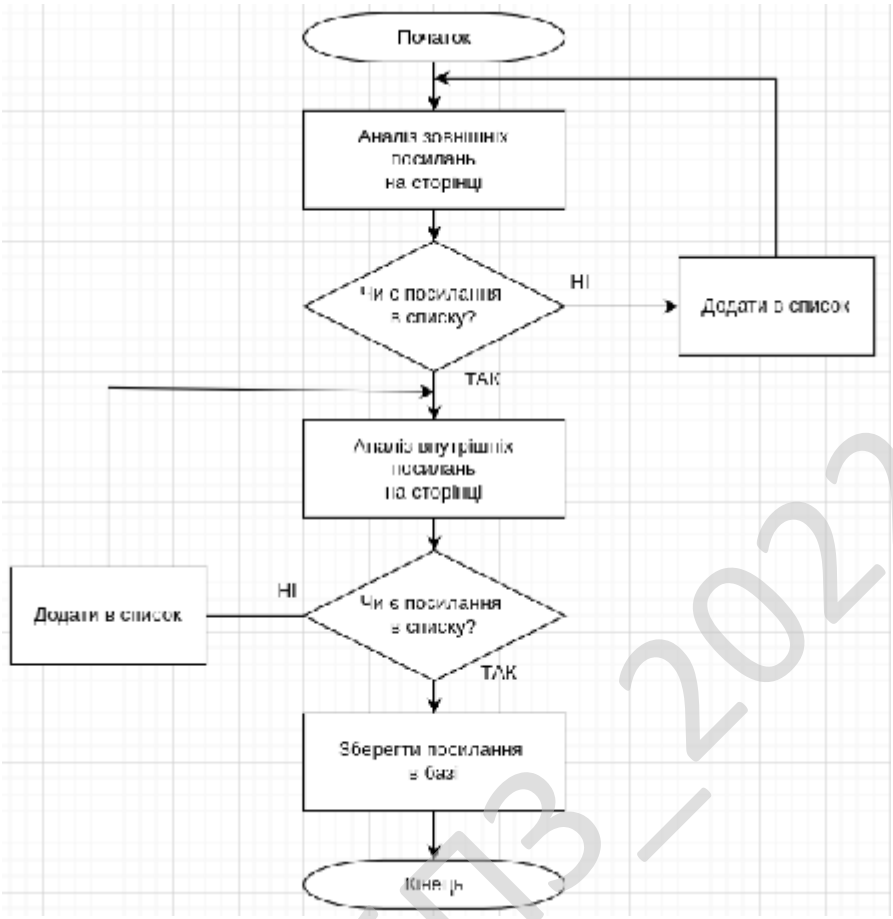

Рисунок 4.2 – Блок-схема роботи підпрограми

Puppeteer

Сьогодні сучасні веб-браузери містять спеціальні засоби доступу, призначені для автоматизації та міжпрограмного зв'язку. Зокрема, Chrome Devtools Protocol (він же CDP) – це високорівневий протокол API, який дозволяє програмам керувати екземплярами веб-браузера Chrome або Firefox через сокет з'єднання. Іншими словами, ми можемо написати програму, яка підключається до екземпляра браузера Chrome/Firefox і наказує йому щось робити.

З огляду на це, є деякі негативи. Браузери – це справді складні програмні проекти, що означає, що вони споживають багато ресурсів. У свою чергу, більша складність також може означати більші витрати на обслуговування наших веб-скраперів.

Бібліотеку Puppeteer node.js можна встановити через менеджер пакетів NodeJS npm за допомогою цих команд терміналу:

```
$ npm install puppeteer
```

Перше, що ми повинні відзначити, це те, що Puppeteer є асинхронною бібліотекою вузлів. Це означає, що ми працюватимемо в контексті Promises і async/await програмування.

Тепер, коли наш пакет готовий, почнемо з найпростішого прикладу. Ми запустимо безголовий веб-браузер Chrome (безголовий режим означає спеціальну версію веб-переглядача, яка не має елементів графічного інтерфейсу користувача), накажемо йому перейти на деякі веб-сайти, зачекаємо, поки він завантажиться, і отримаємо джерело сторінки HTML:

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

async function run(){
    const browser = await puppeteer.launch({
        headless: false,
        ignoreHTTPSErrors: true,
    }

    let page = await browser.newPage();
    await page.goto('https://openweathermap.org/full-price#current
', {
        waitUntil: 'domcontentloaded',
    });
    console.log(await page.content());
    await page.close();
    await browser.close();
}
run();
```

У цьому базовому прикладі ми створюємо видимий екземпляр браузера, відкриваємо нову вкладку, переходимо на веб-сторінку https://openweathermap.org/full-price#current і друкуємо її вміст. За допомогою ключового слова await робимо наш код синхронним, так як деякі операції в браузері потребують часу для виконання.

Під час копіювання за допомогою Puppeteer ми здебільшого працюватимемо з налаштуваннями браузера та з їхніми об'єктами Page, які, по

суті, є вкладками веб-браузера. У цьому прикладі ми використовуємо два методи: goto(), який повідомляє вкладці, куди потрібно перейти, і content(), який повертає вихідний код веб-сторінки.

Маючи ці базові знання, ми можемо почати досліджувати загальні шаблони використання Puppeteer, давайте почнемо з базового аналізу.

У цьому базовому сценарії ми стикаємося з нашою першою проблемою: як ми дізнаємося, що сторінка завантажена та готова до аналізу даних?

У цьому прикладі ми використали аргумент waitUntil, щоб повідомити браузеру чекати сигналу domcontentloaded, який запускається, коли браузер читає вміст HTML сторінки. Однак це може працювати не для кожної сторінки, оскільки динамічні сторінки можуть продовжувати завантажувати вміст, навіть якщо сторінку HTML читає браузер.

Вибір вмісту

Оскільки Puppeteer працює у повноцінному браузері, ми маємо доступ до селекторів CSS і XPATH, які дозволяють нам вибирати певні частини сторінки та або витягувати відображені дані, або надсилати події, такі як кліки та введення тексту. Давайте подивимося, як це використовується в веб-збиранні.

Об’єкт Page постачається з кількома методами, які дозволяють нам знаходити об’єкти ElementHandle, які ми можемо витягнути або використати як ціль кліку/введення:

```
await page.setContent(`
<div class="links">
  <a href="https://twitter.com/@scrapfly_dev">Twitter</a>

  <a href="https://www.linkedin.com/company/scrapfly/">LinkedIn
</a>
</div>
`);

await (await page.$('.links a'))
.evaluate( node => node.innerText);
await (await page.$('.links a'))
.evaluate( node => node.getAttribute("href"));
```

```
let links = await page.$$('.links a');

for (const link of links){
    console.log(await link.evaluate( node => node.innerText));
    console.log(await link.evaluate( node => node.getAttribute("href")));
}
```

Як видно, об’єкт Page надає нам доступ як до селекторів CSS, так і до XPATH. Ми можемо витягнути перший знайдений елемент за допомогою методу .$ або всі відповідні елементи за допомогою .$$.

Коли справа доходить до автоматизованого сканування веб-переглядача, існує два типи проблем: швидкість сканування та виявлення ботів.

Розглянемо загальні поради та прийоми, які ми можемо застосувати в нашому бібліотечному веб-скребку Puppeteer, щоб вирішити ці дві проблеми.

Оптимізація швидкості копіювання/ресурсів

Найефективніше, що ми можемо зробити, це вимкнути завантаження вбудованих зображень/відео. Під час аналізу веб-сторінок нам байдуже, чи завантажуються зображення на веб-сторінку – ми просто хочемо збирати дані.

Ми можемо налаштувати безголові браузери Puppeteer із правилами, які блокуватимуть зображення та аналітичний трафік:

```
const blockResType = [
  'font',
  'image',
  'imageset',
  'media',
  'object',
  'texttrack',
];

const blockResName = [
  'adition',
  'adzerk',
  'analytics',
  'clicksor',
  'clicktale',
  'doubleclick',
  'exelator',
  'facebook',
```

```
    'fontawesome',
    'google',
    'google-analytics',
    'googletagmanager',
    'mixpanel',
    'quantserve',
    'sharethrough',
    'tiqcdn',
    'zedo',
  ];

  const page = await browser.newPage();
  await page.setRequestInterception(true);
  page.on('request', request => {
    const requestUrl = request._url.split('?')[0];
    if (
      (request.resourceType() in blockedResourceType) ||
      blockResourceName.some(resource => requestUrl.includes(resource))
    ) {
      request.abort();
    } else {
      request.continue();
    }
  });
  }
```

У цьому прикладі ми додаємо до нашої сторінки розширення, яке вимикає завантаження заблокованих ресурсів і типів ресурсів. Це значно пришвидшить сканування веб-сайтів, на веб-сайтах із великою кількістю медіафайлів це може бути до 10 разів! Не тільки це, але це заощадило б нашому скребку багато пропускної здатності.

Уникнення виявлення ботів

Хоча ми використовуємо справжній веб-переглядач, визначити, людина ми чи бот, не так вже й складно для нашого веб-сайту. Оскільки ваш безголовий браузер виконує весь JavaScript і працює на одній IP-адресі, веб-сайти можуть використовувати різні методи, такі як аналіз з'єднання та відбитки пальців профілю, щоб визначити, чи є ми ботом чи ні. Щоб підвищити наші шанси, ми

можемо зробити дві речі: використовувати проксі та застосувати невидимі латки до нашого браузера.

Використання проксі. Спочатку розглянемо проксі. Стандартним способом використання проксі-серверів із Puppeteer є застосування їх до об'єкта Browser:

```
const browser = await puppeteer.launch({
   args: [ '--proxy-server=http://12.34.56.78:5000' ]
});
```

Однак цей підхід має кілька підводних каменів. По-перше, це означало б, що кожного разу, коли ми хочемо змінити проксі-сервер, нам потрібно буде перезавантажити веб-переглядач - що, якщо ми потрапили в щось? Для невеликих веб-скребків може бути достатньо одного проксі-сервера, але після масштабування нам знадобиться щось краще.

На жаль, Puppeteer не може встановити проксі для кожного запиту чи навіть сторінки. Існують деякі рішення, як-от puppeteer-page-proxy та puppeteer-proxy, але ці розширення просто перехоплюють безголові запити браузера та роблять їх через http-клієнт NodeJS, що збільшує ймовірність виявлення як бота.

Найкращий спосіб використовувати кілька проксі-серверів у Puppeteer – запустити власний проксі-сервер. Таким чином веб-скребок підключатиметься до одного проксі-сервера, а наш проксі-сервер замість цього вибере випадковий проксі зі списку. Наприклад, ми можемо досягти цього за допомогою пакета nodeJS проксі-ланцюжка:

```
const puppeteer = require('puppeteer')
const ProxyChain = require('proxy-chain');

const proxies = [
  'http://user:pass@11.11.11.11:4000',
  'http://user:pass@22.22.22.22:3000',
  'http://user:pass@33.33.33.33:7000',
const server = new ProxyChain.Server({
  port: 8000,
  prepareRequestFunction: ({request}) => {
    let randomProxy = proxies[proxies.length * Math.random() | 0];
    return {
      upstreamProxyUrl: randomProxy,
    };
```

```
  });
});

server.listen(() => console.log('Proxy server started on 127.0.0.1:8000'));

const browser = await puppeteer.launch({
   args: [ '--proxy-server=http://127.0.0.1:8000' ]
});
```

За такого підходу наш браузер використовуватиме наш проксі-сервер як свій проксі, який, у свою чергу, вибирає випадковий проксі для кожного запиту! Однак цей підхід не є куленепробивним і може призвести до деяких помилок, якщо використовувати його з низькоякісними проксі-серверами. Тому, якщо ви все ж використовуєте ланцюжки проксі, обов'язково приділіть додаткові зусилля моніторингу.

Stealthing Puppeteer

Коли ми виконуємо парсинг за допомогою веб-браузера, ми надаємо повний доступ для виконання коду на веб-сайті. Це означає, що веб-сайти можуть використовувати різні сценарії javascript для збору інформації про наш браузер. Це називається відбитком пальця, і коли веб-сайти успішно ідентифікують нас як нелюдей, вони можуть заблокувати нас.

Щоб обійти відбитки пальців, ми можемо зміцнити наші безголові браузери, щоб вони фактично брехали, що це таке. Це дійсно трудомісткий процес, який виходить за рамки одного розробника, але існують інструменти, які підтримує спільнота, як-от плагін puppeteer-stealth.

```
const puppeteer = require('puppeteer-extra')

const StealthPlugin = require('puppeteer-extra-plugin-stealth')
puppeteer.use(StealthPlugin())

puppeteer.launch({ headless: true }).then(async browser => {
  console.log('Running tests..')
  const page = await browser.newPage()
  await page.goto('https://bot.sannysoft.com')
  await page.waitForTimeout(5000)
  await page.screenshot({ path: 'testresult.png', fullPage: true })
  await browser.close()
```

```
  console.log(`All done, check the screenshot`)
})
```

Є кілька способів, за допомогою яких цільовий веб-сайт може легко виявити використання puppeteer. Додавання HeadlessChrome до агента користувача є лише найбільш очевидним.

Мета цього це бути безсумнівним супутником puppeteer, щоб уникнути виявлення, застосовуючи нові методи, коли вони з'являються.

Оскільки ця гра в котів і мишок перебуває в зародковому стані та швидко розвивається, плагін залишається максимально гнучким, щоб підтримувати швидке тестування та ітерації.

Я вважаю це дружнім змаганням у досить цікавій грі в кішки-мишки. Якщо інша команда (wave) хоче виявити безголовий хром, все ще є способи зробити це.

Ймовірно, неможливо запобігти всім способам виявлення безголового хрому, але має бути можливість зробити це настільки складним, що це стане непомірно дорогим або спричинить занадто багато хибних спрацьовувань, щоб бути можливим.

Підсумовуючи плагін Stealth — це зручна оболочка, яка автоматично потребує кількох методів ухилення та поставляється зі стандартними налаштуваннями.

4.2 Захист розробленого програмного забезпечення

Створений сервіс не надає фізичних чи електронних копій програмного продукту, тому захист, моєї програми лежить виключно на системі захисту хмарного сервісу AWS. Розглянемо, детальніше, вивчивши документацію AWS.

Хмарне шифрування

Хмарне шифрування засекречує дані перед збереженням їх у хмарних базах даних. Так, у разі витоку інформації, сторонні особи не зможуть скористатися цими даними у зловмисних цілях. Організація використовує "Сервіс керування ключами AWS", щоб контролювати шифрування даних у робочих

навантаженнях AWS.

Безпека даних

Безпека даних захищає дані в місцях зберігання за допомогою надійної системи зберігання та безпечної передачі даних. Розробники використовують такі захисні заходи, як шифрування та ізольоване резервне копіювання, для забезпечення операційної відмовостійкості до можливих порушень безпеки даних. У деяких випадках розробники використовують систему AWS Nitro для забезпечення конфіденційності сховища та обмеження доступу оператора.

Аварійне відновлення та планування безперервних бізнес-процесів

Ці стратегії є планами дій у надзвичайних ситуаціях, що дозволяють організаціям оперативно реагувати на інциденти кібербезпеки і при цьому продовжувати працювати лише з невеликими збоями або взагалі без них. Крім того, організації впроваджують політики відновлення даних, щоб уникнути втрати даних.

Система виявлення вторгнень

Організація використовує системи виявлення вторгнень для ідентифікації кібератак та швидкого реагування на них. Сучасні рішення для забезпечення безпеки використовують машинне навчання та аналітику даних для виявлення прихованих загроз у обчислювальних інфраструктурах організацій. За допомогою механізму захисту від вторгнень, що збирає дані про інциденти, команди з питань безпеки можуть визначати їхні джерела.

Як AWS допомагає з кібербезпекою?

Як клієнт AWS, ви отримуєте вигоду від використання можливостей центрів обробки даних AWS та мережі, розробленої для захисту вашої інформації, особистих даних, програм та пристроїв. AWS надасть вам можливості для задоволення ключових вимог до безпеки та відповідності, таких як вимоги до розміщення даних, захисту та конфіденційності, за допомогою комплексного набору сервісів та функцій. Крім того, AWS дозволяє автоматизувати завдання безпеки, що вирішуються вручну, щоб ви могли сконцентруватися на питаннях розвитку та впровадження інновацій у ваш бізнес.

AWS надає різні послуги кібербезпеки, зокрема:

– захист даних, акаунтів та робочих навантажень від несанкціонованого доступу;

– управління посвідченнями, ресурсами та дозволами в будь-якому масштабі;

– застосування детальної політики безпеки у точках контролю мережі в межах організації;

– безперервне відстеження активності в мережі та поведінки в облікових записах хмарного середовища;

– комплексне уявлення про статус відповідності вимогам за допомогою автоматизованих перевірок

5 МЕТОДИКА ВПРОВАДЖЕННЯ СИСТЕМИ В ПРОМИСЛОВУ ЕКСПЛУАТАЦІЮ

Весь функціонал програми має API інтерфейс, завдяки чому ними зручно користуватися під час написання власних сайтів чи додатків. API – популярна у всьому інтернеті технологія, що облегчує процеси обміну інформацією. Дані видаються в форматі JSON. Стандартний запит виглядає приблизно так:

https://localhost/parse&url=”https://en.wikipedia.org/wiki/Ukrainian_Air_Force”&className=”aircraft-name”

Подібним чином можна відправити запит на будь-якій мові. Відповідь на запит буде приблизно така інформація:

{

Mikoyan MiG-29

Sukhoi Su-24

Sukhoi Su-25

Sukhoi Su-27

}

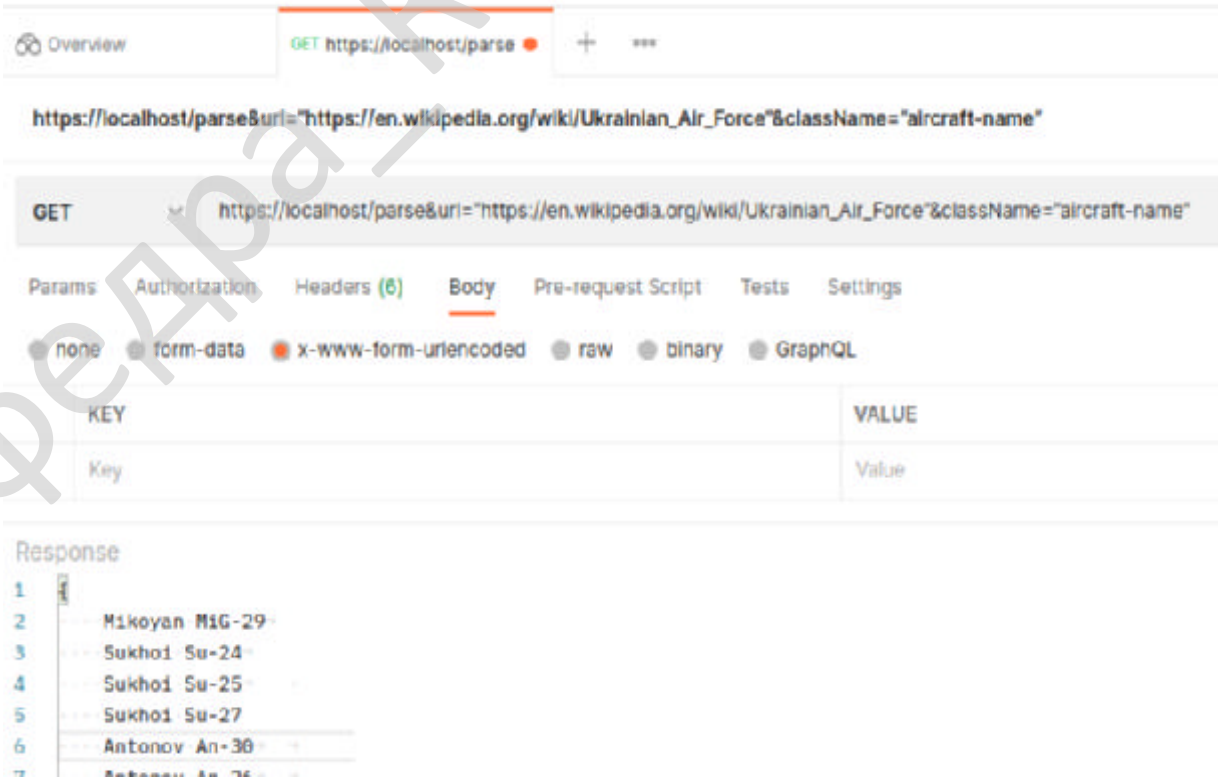

Рисунок 5.1 – Приклад відповіді сервера на запит

Для більш детального ознайомлення функціоналом сервісу, було створено і графічний варіант.

На головній сторінці користувачу пропонується ввести URL сайту, з якого буде проводитись парсинг.

Рисунок 5.2 – Головна сторінка

Далі при введені URL, та після натискання кнопки “Парсити”, через декілька секунд з’явиться кнопка та випадаючий список.

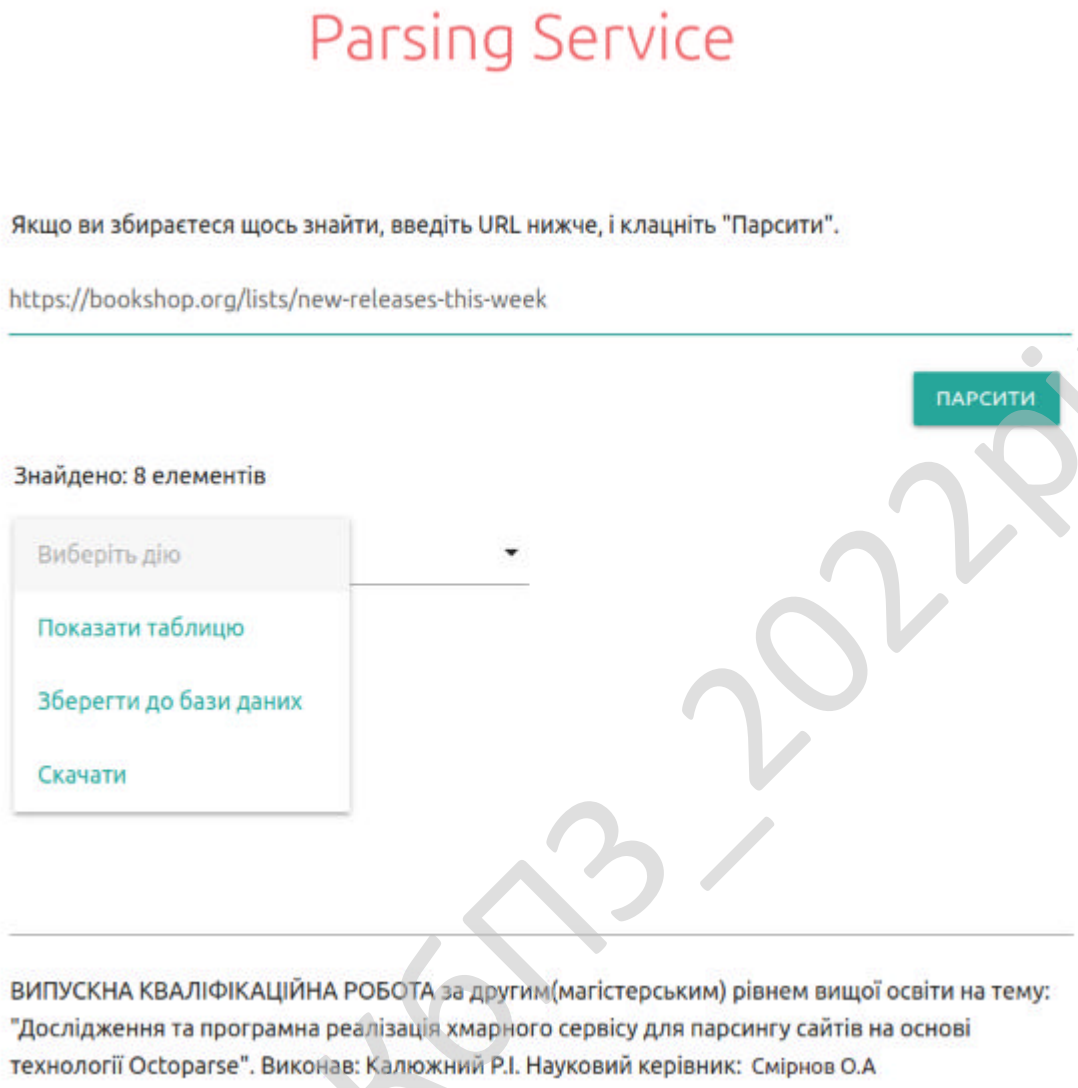

Рисунок 5.3 – Випадаючий список, після вдалого парсингу

Після обраної дії натискаємо кнопку “Обрати”.

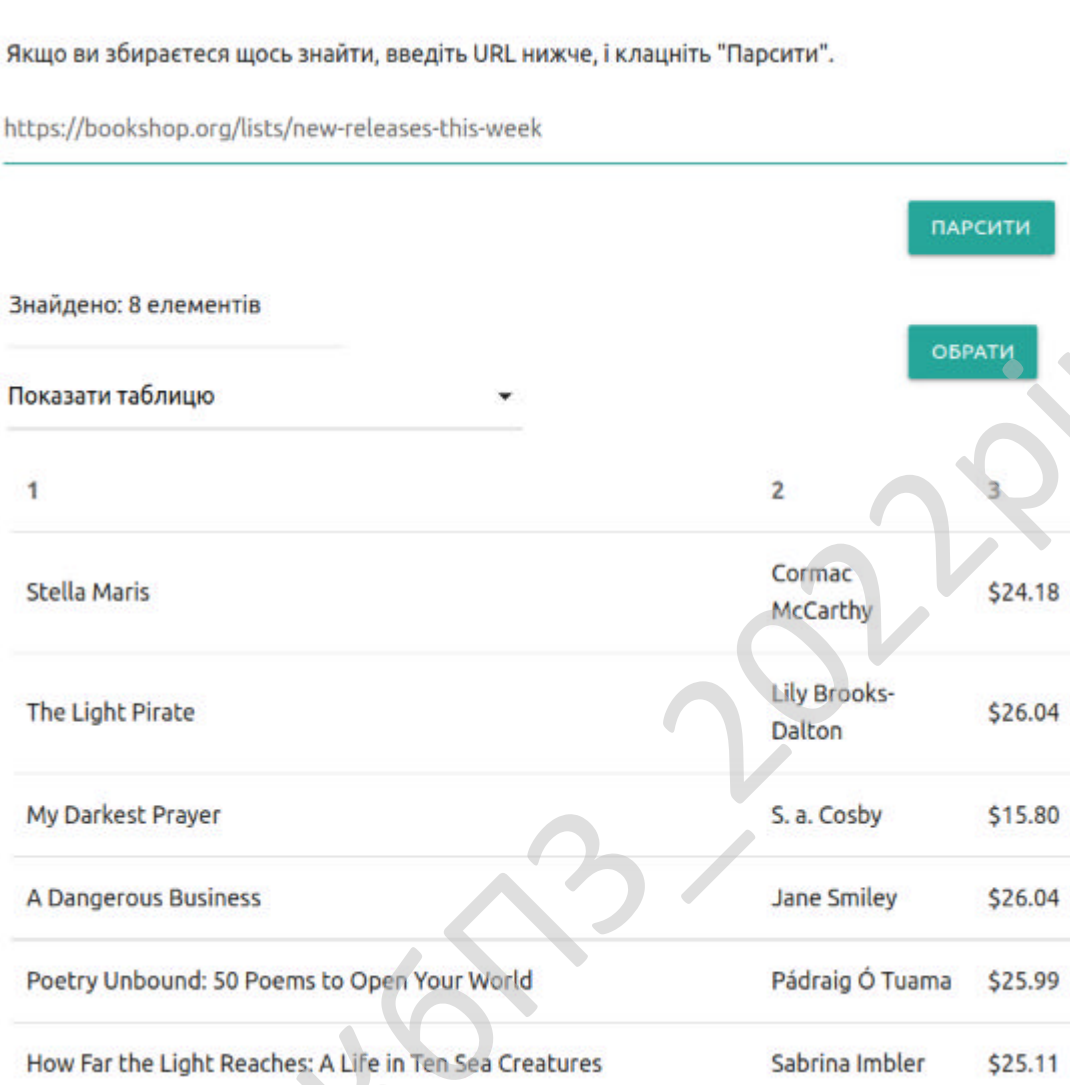

Рисунок 5.4 – Таблиця, з результатами парсингу

6 НАУКОВА НОВИЗНА

У випускній кваліфікаційній роботі за другим (магістерським) рівнем вищої освіти розроблено програмне забезпечення хмарного сервісу, яке призначено для парсингу сайтів на основі технології Octoparse.

Метою роботи є дослідження та програмна реалізація хмарного сервісу для парсингу сайтів на основі технології Octoparse.

Об'єктом дослідження є процес парсингу сайтів на основі технології Octoparse.

Предметом дослідження є методи для парсингу сайтів на основі технології Octoparse, розміщеному в хмарному сервісі.

Методи дослідження базуються на методах пошуку та обробки інформаці хмарних технологій та методах розробки програмного забезпечення.

Наукова новизна отриманих результатів. У процесі рішення завдань та досліджень , розроблені та впроваджені наступні переваги над іншими системами системами парсингу, такі як:

– Удосконалено метод для парсингу сайтів.

– Сервіс буде розміщений за допомогою одно з хмарних провайдерів, тому він буде більш продуктивним ніж існуючі аналоги.

7 ЕКОНОМІЧНА ЕФЕКТИВНІСТЬ РОЗРОБЛЕНОЇ ПРОГРАМИ

7.1 Техніко-економічне обґрунтування теми випускної кваліфікаційної роботи за другим (магістерським) рівнем вищої освіти

Після ознайомлення з підприємством та засобами розробки програмної продукції був розроблений план розробки програми. Був підрахований необхідний час для розробки та впровадження програми. Цей час склав 48 днів (два місяці). В магістерській роботі було проведене дослідження та виконана програмна реалізація хмарного сервісу для парсингу сайтів на основі технології Octoparse.

Розроблене програмне забезпечення має достатню надійність і задовольняє усім поставленим умовам, а саме:

а) невеликий розмір;

б) невеликі системні потреби;

в) незалежність від встановлених на комп’ютері баз даних чи ОС;

г) зручність у користуванні та надійність.

Таблиця 7.1 – Початкові дані

Показники	Позна-чення	Характеристика або величина
1	2	3
1. Кількість розроблених програм період, шт.	N	1
2. Кількість екземплярів програм, шт.	Ne	12
3. Запланований термін розробки, днів	Fpq	48 (2 місяці)
4. Група задачі підсистеми управління (1-6)	–	2
5. Ступінь новизни задачі (А, Б, В, Г)	–	В
6. Складність алгоритму (1, 2, 3)	–	2

Продовження таблиці 7.1

Показники	Позначення	Характеристика або величина
7. Кількість макетів вхідної інформації	–	4
8. Кількість форм вихідної інформації.	–	10-11
9. Мова програмування (1-6)	–	3
10. Попередній досвід (1-6)	–	4
11. Гнучкість проекту ПП (1-6)	–	5
12. Детальність проекту ПП (1-6)	–	2
13. Рівень спрацьованості колективу (1-6)	–	4
14. Ступінь вимірності процесів (1-6)	–	3
15. Необхідна надійність програмного забезпечення (1-6)	–	3
16. Розмір бази даних (порівняно з розміром програми) (1-6)	–	3
17. Складність кінцевого програмного продукту (1-6)	–	2
18. Необхідний рівень забезпечення повторного використання (1-6)	–	5
19. Документованість відповідно до планованого життєвого циклу (1-6)	–	4
20. Вимоги до швидкодії ПП (1-6)	–	4
21. Обмеження на розміри основного сховища даних (1-6)	–	3
22. Різноманітність використовуваних обчислювальних платформ (1-6)	–	2
23. Професійний рівень аналітиків (1-6)	–	2
24. Професійний рівень програмістів (1-6)	–	2
25. Постійність складу команди розробників (1-6)	–	2
26. Досвід розробки додатків (1-6)	–	2
27. Досвід роботи з обчислювальною платформою (1-6)	–	4

Продовження таблиці 7.1

Показники	Позначення	Характеристика або величина
28. Досвід роботи з мовою і інструментами середовища розробки (1-6)	–	1
29. Досвід роботи з програмними інструментами розробки (1-6)	–	1
30. Розробка ПЗ для декількох серверів одночасно (1-6)	–	2
31. Вимоги до дотримання встановленого графіка робіт (1-6)	–	3
32. Вартість ПЗ у розробника (НМА), грн.	–	95000
33. Норматив додаткової зарплати, % :	$Н_д$	10
34. Норматив відрахувань у соціальні фонди, %	Нс	22
35. Норматив загальногосподарських витрат, %	Нг	15
36. Норматив витрат на освоєння нових мов програмування, %	Нп	15
37. Рівень рентабельності програмної продукції, %	Ре	55
38. Ставка податку на додану вартість, %	Ндв	20

7.2 Розрахунок трудомісткості розробки програмної продукції

Значення трудомісткості розробки програмного забезпечення для стадій ТЗ, ЕК, ТП та ВП визначаємо по типовим нормам часу приведеним в додатках МВ. Стадія РП є тривалою і трудомісткою, впливаючи на інші стадії проекту.

Визначимо трудомісткість розробки ПЗ для стадії РП.

Обчислюємо номінальні трудовитрати, люд-міс.:

$$T_{ном} = A \, \text{Size}^B, \quad (7.1)$$

де: A – коефіцієнт Боема, A = 2,45;

Size – загальний об'єм відлагодженого програмного коду, тис. рядків;

B – показник ступеня, що визначається співвідношенням:

$$B = 1,01 + 0,001 \sum W_i , \quad (7.2)$$

де: *Wi* – сумарне значення п'яти показників (МВ, додаток 2), що відображають особливості розробки проекту програмного продукту (ПП) і колективу розробників.

$$B = 1,01+0,001(1,62+1,21+3,38+1,98+2,73) = 1,020.$$

$$T_{ном} = 2,45 \cdot 1,8^{1,020} = 4,46 \text{ люд-міс.}$$

Визначаємо уточнені (з урахуванням приведених в МВ додатку 3 сімнадцяти додаткових коефіцієнтів) трудовитрати, люд-міс.:

$$T_{уточн} = T_{ном}\, ПВ_j, \quad (7.3)$$

де: $ПВ_j$ – добуток сімнадцяти додаткових коефіцієнтів, приведених в МВ додатку 3.

$$T_{уточн} = 4,46 \cdot (1 \cdot 1 \cdot 0,88 \cdot 1,29 \cdot 1,06 \cdot 1,11 \cdot 1 \cdot 0,87 \cdot 1,22 \cdot 1,16 \cdot 1,1 \cdot 1,1 \cdot$$

$$\cdot 0,88 \cdot 1,22 \cdot 1,24 \cdot 1,1 \cdot 1) = 12,99 \text{ люд-міс.}$$

Ці коефіцієнти дозволяють диференційовано оцінювати результати роботи програмістів, беручи до уваги швидкодію програми, використання різноманітних обчислювальних платформ і інструментів розробки, взаємодію декількох серверів, вимоги до об'ємів баз даних і ін.

Визначаємо підсумкові трудовитрати по стадії робочий проект, люд-дні:

$$T_{РП} = 0,3\, CT_{уточн}^{\,0,33+0,2\,(B-1,01)} S , \quad (7.4)$$

де: *C* – визначений емпірично коефіцієнт, запропонований авторами методики, (МВ, додаток 4);

S – коефіцієнт стиснення (або подовження) графіка робіт %, що дозволяє коректувати терміни розробки ПЗ згідно встановленим вимогам. Вибираємо в межах (25…350)%.

$$T_{РП} = 0,3 \cdot 2,85 \cdot 12,99^{0,33+0,2(1,020-1,01)} \cdot 65 = 130 \text{ люд/день.}$$

Для зручності визначення загальної трудомісткості на розробку програмного забезпечення результати розрахунків по стадіям зводимо до таблиці 7.2.

Таблиця 7.2 – Визначення трудомісткості розробки програмного забезпечення

Стадії розробки	Трудомісткість за типовими нормами та розрахунками	
	Величина, люд/дні	Підстава
Технічне завдання	9	Д5
Ескізний проект	8	Д6
Технічний проект	16	Д7
Робочий проект	130	Ф 7.1-7.4
Впровадження	16	Д13
Всього	179	–

7.3 Визначення чисельності виконавців і планового фонду зарплати

Чисельність ставок інженерів-програмістів для розробки програмного забезпечення визначається за формулою:

$$Ч = \frac{T_{пз} N}{F_{рq} - H_{ев}}, \quad (7.5)$$

де: $F_{рq}$ – плановий фонд робочого часу одного спеціаліста, днів;

$T_{пз}$ – трудомісткість розробки програмного забезпечення люд-дні.

$$Ч = \frac{179 \cdot 1}{48 - 5} = 4{,}16 \text{ ставки.}$$

Чисельність інженерів-електронщиків для проведення технічного обслуговування та ремонту комп'ютерних мереж визначається в залежності від наявності технічних засобів і норм витрат часу на виконання профілактичних робіт на протязі року.

Визначаємо затрати часу на виконання профілактичних робіт по обслуговуванню обладнання за період розробки. Результати розрахунку зводимо до таблиці 7.3.

Таблиця 7.3 – Затрати часу на виконання профілактичних робіт по обслуговуванню обладнання за розрахунковий період

Найменування обладнання	Профілактичне обслуговування			
	Кількість хв. на один. обл.	Кількість облад-нання	Затрати часу в хв.	Затрати часу в год.
Системний блок ПК	120	5	600	10
Монітор	90	5	450	7,5
Клавіатура	70	5	350	
Маніпулятор «мишка»	30	5	150	2,5
Принтер матричний	60	0		0,0
Принтер лазерний	150	2	300	
Принтер струминний	60	1	60	1
Сканер	40	1	40	0,66
Концентратор-маршрутизатор	50	1	50	
Кабельні господарства ЛОМ на 1 м. п.	2,5	210	525	8,75
Копіювальний апарат	110	1	110	1,83
Усього за рік:			$З_ч$	43,87

Час на профілактику обладнання в загальному балансі робочого часу інженерів-електронщиків неповинен складати більше 10%.

Виходячи з цього фонд робочого часу інженерів-електронщиків складає:

$$Ф_{др}^{с} = \frac{З_ч \cdot n_{тіс}}{1,2}, \quad (7.6)$$

$$Ф_{др}^{с} = \frac{43,87 \cdot 2}{1,2} = 73,11 \text{ год.}$$

Визначаємо необхідну кількість ставок штатного персоналу сектора ТО:

$$Ч_{ел} = \frac{Ф_{др}^{с}}{F_{др} \cdot T_{зм}}, \quad (7.7)$$

$Ч_{ел} = 73,11/(48 \cdot 8) = 0,19$ ставки.

Для забезпечення нормального технічного обслуговування засобів ТО та мереж, необхідно прийняти найбільше ціле значення розрахункової чисельності інженерів-електронщиків.

Чисельність інженерів-системотехніків, адміністраторів мережі, дизайнерів WEB вузлів, системних програмістів (аналітиків), бухгалтерів-економістів визначається за потребою в залежності від функціональних обов'язків. Після визначення чисельності персоналу складається штатний розклад.

Таблиця 7.4 – Розрахунок чисельності штатного персоналу сектору системного та адміністративного обслуговування засобів ОТ та комп'ютерних мереж

Посада	Вид роботи	Час	К-ть штатних одиниць
Адміністратор загальної мережі, аналітик	Адміністрування локальної мережі, поштового та серверу DNS (ОС FreeBSD), маршрутизатора Cisco, доменного контролеру Windows Server, серверу доступу ADSL (ОС Linux), налаштування ADSL, VPN, PPPoE, Frame Relay, Wi-Fi	2	0,5
	Налаштування і конфігурування базової станції безпровідного зв'язку (CMTS)	0,5	
	Розробка та впровадження проектів з організації зв'язку між віддаленими об'єктами, ЛОМ	0,5	
	Забезпечення цілодобової роботи зв'язку клієнтів до мережі Інтернет	1	
Всього		4	

Продовження таблиці 7.4

Посада	Вид роботи	Час	К-ть штатних одиниць
Продакт-менеджер	Презентації нової продукції, пошук каналів збуту	1	0,25
	Підтримка постійних клієнтів	0,5	
	Оформлення договорів, ведення тендерів	0,25	
	Контроль взаєморозрахунків з постачальниками	0,25	
Всього		2	
Дизайнер WEB	Розробка концепції оформлення та інтерфейсу сайту, оптимізація дизайну існуючих, проектує їх структуру та навігацію	1	0,25
	Створення графічних і стилістичних елементів сайту	0,5	
	Оформлення банерів і промо-сторінок	0,25	
	Розміщення графіки і контенту на Інтернет сторінках	0,25	
Всього		2	
Інженер верстальник	Розробка та верстка макетів рекламної продукції та технічної документації	1	0,25
	Верстка друкованих видань	0,5	
	Додрукова підготовка макетів	0,25	
	Розміщення графіки і контенту на Інтернет сторінках	0,25	
Всього		2	

Складемо штатний розклад виконавців.

Таблиця 7.5 – Штатний розклад виконавців

Посада	Кількість ставок	Середньомісячний оклад, грн.	Всього за період розробки, грн.
Керівник (ІТ-менеджер)	1	22500	45000
Продакт-менеджер	0,25	17000	8500
Інженер-програміст	4,16	20000	166400
Інженер - електронщик	0,25	17000	8500
Інженер-системотехнік	0,25	17000	8500
Адміністратор мережі	0,5	15700	15700
Системний програміст	0,1	15000	3000
Дизайнер WEB	0,25	17300	8650
Інженер-верстальник	0,25	16000	8000
Бухгалтер-економіст	0,2	17000	6800
Всього за період розробки	$R_{сп} = 8,16$	-	$Ф_{роб} = 279050$

Розрахуємо середньоденну зарплату одного виконавця:

$$З_{сд} = \frac{Ф_{роб}}{R_{сп} F_{рд}}, \quad (7.8)$$

де: $Ф_{роб}$ – загальна сума зарплати за плановий період, грн.

$$З_{сд} = \frac{279050}{8,6 \cdot 48} = 675 \text{ грн}$$

7.4 Розрахунок капітальних вкладень та амортизаційних відрахувань у розробника

Балансова вартість будівель визначається з урахуванням кількості робочих місць виконавців, питомої площі на одне робоче місце, та вартості одного квадратного метра виробничої площі:

$$Б_{уд} = R^1_{сп} S_у Ц_{пл}, \quad (7.9)$$

де: $R_{сп}^{1}$ – кількість робочих місць виконавців, шт. Приймаємо 8 робочих місць;

S_y – питома площа на одне робоче місце, $м^2$;

$Ц_{пл}$ – вартість одного квадратного метра площі, грн.

Згідно даних інтернет ресурсу OLX.UA (https://olx.ua.com) ціна одного квадратного метра площі, на вул. Волкова, 3В, вік якої 7 років, складає 690 $у.о./м^2$. Враховуючи, що курс складає 1 у.о. = 38 грн. приймаємо для розрахунку вартість одного метра квадратного рівною 26200 грн./м^2. На кожне робоче місце у середньому потрібно 8 $м^2$. З урахуванням цього:

$$Б_{уд} = 6 \cdot 8 \cdot 29000 = 1392000 \text{ грн.}$$

Вартість передавальних пристроїв складає 10% від вартості будівель, і у даному випадку вона складе: 139200 грн.

Балансова вартість інвентарю розраховується за нормою 4000 грн. на одне робоче місце. Тобто:

$$I_{нв} = R_{сп}^{1} \cdot Ц_{м}, \quad (7.10)$$

де: $Ц_м$ – ціна меблів для одного робочого місця, грн.

$$I_{нв} = 6 \cdot 4000 = 24000 \text{ грн.}$$

Балансова вартість обчислювальної техніки визначається по оптовим цінам постачальника з врахуванням витрат на транспортування.

Специфікація на обчислювальну техніку наведена в таблиці 7.7.

Дані по оптовій ціні на обладнання та комплектуючі вибирались по прайсу фірми Brain за 07.11.22 – джерело http: //brain.com.ua.

Таблиця 7.6 – Специфікація

Найменування комплектуючої або обладнання	Тип	Оптова ціна
Персональний комп’ютер		20524
Системний блок		13887
Процесор	Intel Core i5(9400) 2.9-4.1 GHz	–
Системна плата	Asus Prime H510M-K (s1200, Intel H510, PCI-Ex16), Socket 1200	–
Відеокарта	Intel HD Graphics 630	–
Жорсткий диск	SSD 2.5" 512GB KINGSTON (SKC600/512G), SATA III (6Gb/s)	–
Оперативна пам’ять	DDR4 16GB (2X8GB) 3200MHZ FURY BEAST BLACK KINGSTON Б (KF432C16BBK2/16)	–
DVD-привод	DVD -RW/+RW , LG SATA SuperMulti Bulk 22x, black	–
Корпус	VINGA PILLAR WHITE Advanced A0079 (I5M16INT.A0079), black	–
Мишка	LOGITECH M185 SWIFT GREY (910-002238), black	699
Клавіатура	LOGITECH K120 UKR (920-002643)	369
Монітор	24" SAMSUNG LS24R350FZIXCI IPS 1920x1080 250/1000M:1 178/178 D-Sub+HDMI+DVI	5569
Принтер лазерний	XEROX PHASER 3020BI (WI-FI) (3020V_BI)	5400
Принтер струминний	EPSON L132 (C11CE58403)	7100
Копіювальний апарат	Canon i-SENSYS MF217W	6399

Для визначення необхідної кількості капітальних вкладень складемо таблицю 7.8.

Таблиця 7.7 – Балансова вартість обчислювальної техніки

Найменування обчислювальної техніки	Кількість, шт.	Ціна за одиницю, грн.	Витрати на транспортування, монтаж та випробовування.	Загальна вартість, грн.
Персональні комп'ютери	7	20524	12120	155768
Принтер лаз.	2	5400	500	5900
Принтер струм.	1	7100	500	7600
Копіюв. апарат	1	6399	500	6899
Всього	–	–	–	176167

Таблиця 7.8 – Вартість основних фондів та амортизаційні відрахування розробника

Групи та види основних фондів	Балансова вартість, грн.	Амортизація	
		Норма, %	Відрахування, грн.
1	2	3	4
Група 3			
1. Будівлі	1392000	–	–
2. Передавальні пристрої	139200	–	–
Всього по групі	1531200	5	76560
Група 4			
3. Обчислювальна техніка	176167	–	–
Всього по групі	176167	50	88083,5
4. Нематеріальні активи	55000	10	5500

Продовження таблиці 7.8

Група 5, 6			
5. Вимірювальні пристрої	8600	25	2150
6. Транспортні засоби	89 910	20	13000
7. Господарський інвентар	25000	25	6250
Всього по групі	98600	–	21400
Разом	$К_р$ = 1830810		$А_р$ = 191543,5

Примітка: вартість автомобіля Daewoo Lanos 2008 взята по даним з автосалону AUTO.RIA, джерело https://auto.ria.com/, складає 89 910 грн.

7.5 Визначення собівартості розробки та ціни програмної продукції

Визначимо основну зарплату виконавців:

$$З_о = \frac{З_{сд} \cdot Т_{пз}}{N_е}, \quad (7.11)$$

де: $N_е$ – кількість екземплярів програм, шт.

$$З_о = 675 \cdot 179/55 = 2196 \text{ грн.}$$

Визначимо додаткову зарплату (оплата відпусток, виконання державних та суспільних обов'язків) на рівні 10%:

$$З_д = З_о \cdot Н_q \cdot 0{,}01, \quad (7.12)$$

де: $Н_q$ – норматив додаткової зарплати, %.

$$З_д = 2196 \cdot 10 \cdot 0{,}01 = 219 \text{ грн.}$$

Відрахування на соціальні потреби за нормативом $Н_с = 22\%$ від суми основної та додаткової зарплати:

$$С_{оц} = 0{,}01 \cdot Н_с (З_о + З_д), \quad (7.13)$$

де: $Н_с$ – відрахування на соціальні потреби, %.

$$С_{оц} = 0{,}01 \cdot 22(2196+219) = 531 \text{ грн.}$$

Визначимо загальногосподарські витрати (електроенергію, ремонт і утримання приміщень і т.д) за нормативом $Н_г$ = 15% від основної зарплати:

$$Г_{осп} = З_о \cdot Н_г \cdot 0{,}01, \quad (7.14)$$

де: $Н_г$ – загальногосподарські витрати, %.

$$Г_{осп} = 2196 \cdot 15 \cdot 0{,}01 = 329 \text{ грн.}$$

Визначимо витрати на матеріали для розробки програмної продукції за нормами споживання та діючими цінами за одиницю виміру:

$$З_М = (З_{М1} + З_{М2} + З_{М3})/N_е, \quad (7.15)$$

де: $З_{М1}$ – вартість паперу, грн.;

$З_{М2}$ – вартість запам'ятовуючих пристроїв, грн.;

$З_{М3}$ – вартість фарби, картріджей, тонеру, грн.;

$N_е$ – кількість екземплярів програм, шт.

Згідно прийнятих норм на підприємстві $n_{вит}$ приймаємо 1,5 пачки паперу на період розробки. Тоді, враховуючи, що вартість пачки паперу складає $Ц_п$=218 грн., визначаємо вартість паперу за період розробки:

$$З_{М1} = Ц_п \cdot N_м. \quad (7.16)$$

$$З_{М1} = 218 \cdot 1{,}5 = 327 \text{ грн.}$$

Для деяких задач, можливо знадобляться CD диски. Їх кількість дорівнює 10:

$$З_{М2} = \sum Ц_д, \quad (7.17)$$

де: $Ц_д$ – вартість дисків CD/DVD: CDR box – 26,6 грн./шт..

$$З_{М2} = 26{,}6 \cdot 10 = 260 \text{ грн.}$$

Згідно норм одноразовій заправці підлягають усі друкуючі пристрої і становить:

$$З_{М3} = \sum Ц_з., \quad (7.18)$$

де: $Ц_з$ – вартість розхідних матеріалів друкуючих пристроїв: відновлення та заправка картриджу – 524 грн.; картридж – 340 грн.; відновлення картриджу – 537 грн.

$$З_{М3} = 524+340+537 = 1401 \text{ грн.}$$

$$З_М = (524+340+537)/99 = 25 \text{ грн.}$$

Визначимо витрати на освоєння нових мов програмування або операційних систем за нормативом ($H_п$ = 15%) від основної зарплати виконавців:

$$О_п = З_о \cdot Н_п \cdot 0{,}01, \quad (7.19)$$

де: $H_п$ – норматив витрат на освоєння нових мов програмування, %.

$$О_п = 2196 \cdot 15 \cdot 0{,}01 = 329 \text{ грн.}$$

Визначимо витрати на амортизацію основних фондів з урахуванням загальної річної суми амортизаційних відрахувань та кількості екземплярів програм ($N_е$ = 99 прим.):

$$А_м = \frac{А_р \cdot N_{міс}}{N_е \cdot 12}, \quad (7.20)$$

де: $A_р$ – загальна річна сума амортизаційних відрахувань, грн.

$$А_м = 191543 \cdot 2/(55 \cdot 12) = 580 \text{ грн.}$$

Повна собівартість ПЗ визначається як сума витрат за попередніми статтями калькуляції:

$$С_п = З_о + З_д + С_{оц} + Г_{осп} + З_м + О_п + А_м. \quad (7.21)$$

$$С_п = 2196+219+531+329+25+329+580 = 4209 \text{ грн.}$$

Визначимо плановий прибуток за рівнем рентабельності ($P_п$) програмної продукції, яка залежить від складності програми та ступеня новизни задачі.

Для даного програмного забезпечення рівень рентабельності складає 40%.

$$П_р = 0{,}01 \cdot Р_п \cdot С_п, \quad (7.22)$$

де: $P_п$ – рівень рентабельності, %.

$$П_р = 0{,}01 \cdot 40 \cdot 4209 = 1683 \text{ грн.}$$

Величини ціна підприємства, податок на додану вартість, відпускна ціна програмної продукції визначаються за формулами, приведеними в таблиці 7.9

Таблиця 7.9 – Нормативна калькуляція собівартості розробки програмного забезпечення задачі

Найменування статей витрат	Позначення	Величина, грн
1	2	3
1. Основна зарплата виконавців	$З_о$	2196
2. Додаткова зарплата виконавців	$З_д$	219
3. Відрахування на соціальні потреби	$С_{оц}$	531
4. Загальногосподарські витрати	$Г_{осп}$	329
5. Витрати на матеріали	$З_М$	25
6. Освоєння нових операційних систем, мов програмування	$О_п$	329
7. Амортизація основних фондів	$А_м$	580
8. Повна собівартість програмного забезпечення	$С_п$	4209
9. Плановий прибуток	$П_р$	1683
10. Ціна підприємства $Ц_п = С_п + П_р$	$Ц_п$	5892
11. Податок на додану вартість $ПДВ = 0.01 \cdot Н_{дв} \cdot Ц_п$	$ПДВ$	1178
12. Відпускна ціна програмної продукції $Ц = Ц_п + ПДВ$	$Ц$	7070

7.6 Визначення об’єму капітальних вкладень у споживача програмної продукції

Об’єм капітальних вкладень у споживача програмної продукції визначаємо на основі балансової вартості основних фондів, яка враховує ціну, транспортно-заготівельні витрати, вартість будівель, монтажних та пусконалагоджувальних робіт, а також витрати на випробування у виробничих умовах. Результати розрахунків зводимо у таблицю 7.10.

Таблиця 7.10 – Розрахунок об’єму капітальних вкладень у споживача програмної продукції

Найменування капітальних вкладень	Сума за варіантами, грн.	
	Базовий	Новий
Вартість програмної продукції	–	7070
Всього капітальних витрат	–	7070

7.7 Визначення експлуатаційних витрат

Експлуатаційні витрати у споживача програмної продукції визначаємо при умові роботи підсистеми на протязі року. Результати зводимо до таблиці 7.11.

$$З_о = Ч_е О_е М_е (1 + 0,01 Н_q)(1 + 0,01 Н_с), \quad (7.23)$$

де $Ч_е$ – чисельність працівників, чол.;

$О_е$ – заробітна плата, грн./год;

$М_е$ – час, що витрачається на нарахування ЗП.

Після купівлі нового програмного забезпечення час на нарахування зменшився з 882 годин на рік до 630 годин на рік, тому витрати на технічне обслуговування зменшилися з:

$$З_{обсв} = 1 \cdot 75 \cdot 882 \cdot 1,1 \cdot 1,22 = 88773 \ грн.$$

до:

$$З_{обсв} = 1 \cdot 75 \cdot 630 \cdot 1,1 \cdot 1,22 = 63409 \ грн.$$

Таблиця 7.11 – Розрахунок експлуатаційних витрат у споживача програмної продукції

Найменування статей витрат	Позначення	Сума витрат за варіантами, грн.	
		Базовий	Новий
1. Заробітна плата(основна, додаткова, відрахування в соціальні фонди)	$З_р$	88773	63409

Продовження таблиці 7.11

2. Витрати на електроенергію	$З_{ел}$	2306	2306
3. Витрати на амортизацію	$З_{ам}$	0	580
Всього витрат за рік	$І$	91079	66295

Витрати на електроенергію визначаються з урахуванням споживаємої потужності ($П_{ел}$) в кіловатах, часу експлуатації технічних засобів ($Т_р$) в годинах та ціни однієї кіловат-години ($Ц_{ел}$):

$$З_{ел} = П_{ел} \cdot Т_р \cdot Ц_{ел}. \quad (7.24)$$

$$З_{ел\ баз} = 0{,}54 \cdot 2304 \cdot 2{,}1 = 2306 \text{ грн.}$$

$$З_{ел\ нов} = 0{,}54 \cdot 2304 \cdot 2{,}1 = 2306 \text{ грн.}$$

Витрати по амортизації визначаються на основі норм амортизаційних відрахувань, вартості програмної продукції і основних фондів. Для розрахунку складаємо таблицю 7.12.

Таблиця 7.12 – Розрахунок амортизаційних відрахувань

Групи основних фондів	Норма амортизації %	Балансова вартість, грн., за варіантами		Сума відрахувань, грн., за варіантами	
		Базовий	Новий	Базовий	Новий
Програмна продукція	25	–	1675	–	580
Всього відрахувань	-	–	1675	–	580

7.8 Визначення економічної ефективності програмної продукції

Економічна ефективність програмного забезпечення визначається для виготовлювача і споживача за такими показниками.

Величина економічного ефекту при виготовленні програмної продукції, розраховуємо за формулою:

$$E_в = (Ц_п - C_п) \cdot N_е - \sum_{i=1}^{m} E_{p_m} \cdot K_{p_m}, \quad (7.25)$$

де: $K_р$ – балансова вартість основних фондів розробника, грн.; $E_р$ – розрахунковий коефіцієнт капіталовкладень.

$$E_в = (5892 - 4209) \cdot 48 - (0,05 \cdot 1531200 + 0,5 \cdot 176167 + 0,25 \cdot 63409 + 0,2 \cdot 89\,910 + 0,1 \cdot 55000) \cdot 2/12 = 46787 \text{ грн}$$

Визначимо період окупності додаткових капітальних вкладень у виробника програмної продукції:

$$T_в = \frac{K_p^*}{(Ц_п - C_п) \cdot N_е}, \quad (7.26)$$

де: K_p^* – балансова вартість основних фондів розробника без врахування вартості ОФ третьої групи, так як їх строк служби на порядок більший ніж період розробки ПЗ.

$$T_в = \frac{329767}{(5892 - 4209) \cdot 55 \cdot 12/2} = 0,59 \text{ року.}$$

Визначимо величину економічного ефекту у користувача програмної продукції за формулою:

$$E_{сп} = (I_б - I_н) - E_н(K_н - K_б), \quad (7.25)$$

де: $I_б$, $I_н$ – величина експлуатаційних витрат за базовим и новим варіантом відповідно;

$K_б$, $K_н$ – об'єм капітальних вкладень за варіантами, що порівнюються.

$$E_{сп} = (91079-66295)-0,25 \cdot 7070 = 23016 \text{грн.}$$

Визначимо період окупності додаткових капітальних вкладень у споживача програмної продукції за рахунок зниження експлуатаційних витрат:

$$T_{сп} = \frac{K_н - K_б}{I_б - I_н}, \quad (7.26)$$

$$T_{сп} = \frac{7070}{91079-66295} = 0,28 \text{ року.}$$

Показники економічної ефективності програмної продукції зводимо до таблиці 7.13.

Таблиця 7.13 – Показники економічної ефективності програмної продукції

Найменування показників	Одиниця виміру	Величина
1. Кількість екземплярів програми	Прим.	55
2. Повна собівартість розробленої програми	Грн.	4209
3. Ціна розробленої програми	Грн.	5892
4. Плановий прибуток від реалізації розробленої програми	Грн.	1683
5. Рентабельність програмної продукції	%	40
6. Об'єм додаткових капітальних вкладень у виробника програмної продукції	Грн.	169504
7. Загальний прибуток від реалізації програмної продукції	Грн.	176167
8. Величина економічного ефекту при виготовлені програмної продукції	Грн.	46787
9. Період окупності додаткових капітальних вкладень у виробника програмної продукції	Років	0,59
10. Об'єм додаткових капітальних вкладень у споживача програмної продукції	Грн.	7070
11. Величина економічного ефекту у користувача програмної продукції	Грн.	23016
12. Період окупності додаткових капітальних вкладень у користувача програмної продукції	Років	0,28

7.9 Висновки

Розроблена програма економічно вигідна. За рахунок впровадження програмного забезпечення досягається скорочення часу обробки інформації, підвищується культура праці, підвищення якості приймаючих управлінських рішень.

8 ЗАХОДИ З ОХОРОНИ ПРАЦІ ТА ТЕХНІКИ БЕЗПЕКИ

8.1 Вступ

У сучасному житті електронно-обчислювальна машина (ЕОМ) займає важливе місце. Нині ми можемо уявити свою професійну діяльність без такого обов'язкового елемента нашого побуту. Зараз за допомогою ЕОМ керуються життєво важливі процеси. Це і управління різними електростанціями, аеропортами, постачання, і документообігу, комерції та розробки, управління виробничими процесами тощо. За допомогою ЕОМ можна швидко створити документ, таблицю, розрахувати та побудувати графік, записати музику та багато іншого.

Але ЕОМ має і зворотний бік, а саме – небезпеку. Адже вона є складним електронним пристроєм, у якому є безліч процесів, які можуть стати джерелом небезпеки. З ЕОМ потрібно поводитися обережно та знати, як убезпечити себе від негативних наслідків його використання. Оскільки захворювання можуть бути спричинені надмірним фізичним або розумовим навантаженням, через велику нервово-емоційну напругу, або через виробниче середовище. Також програмісти у процесі роботи отримують негативний вплив на органи зору та руки (суглоби пальців та м'язи рук) при роботі з клавіатурою мають теж істотне навантаження. Ці шкідливі фактори можуть привести до професійних захворювань. В даному розділі магістерської роботи проведемо аналіз основних чинників при роботі програміста.

Законом України “Про охорону праці” [52] регламентуються загальні положення державної політики в галузі охорони праці, а також конкретизуються ці положення нормативно-правовими актами про охорону праці, зокрема Наказом Міністерства соціальної політики України 14.02.2018 № 207, який зареєстровано в Міністерстві юстиції України 25 квітня 2018 р. за №508/31960 «Про затвердження Вимог щодо безпеки та захисту здоров’я працівників під час

роботи з екранними пристроями» [53], яким затверджено нормативно-правовий акт з охорони праці НПАОП 0.00-7.15-18, «Правила охорони праці під час експлуатації електронно-обчислювальних машин», та «Державні санітарні правила і норми роботи з візуальними дисплейними терміналами електронно-обчислювальних машин» ДСанПіН 3.3.2-007-98.

8.2 Шкідливі і небезпечні фактори при роботі з комп'ютером

Електронно-обчислювальна машина (ЕОМ) та інше обладнання є джерелами небезпеки ураження електричним струмом. Так як робота програміста характеризується істотним зоровим навантаженням, то вимагає належного освітлення. У приміщенні, в якому працюють люди (у т.ч. програмісти) необхідно створити належний мікроклімат, праметри якого регламентуються, Державними санітарними правилами і нормами, зокрема ДСанПіН 3.3.2.007-98.

Розглянемо шкідливі чинники роботи програмістів керуючись наступними нормативно-правовими актами: «Державні санітарні правила і норми роботи з візуальними дисплейними терміналами електронно-обчислювальних машин» ДСанПіН 3.3.2-007-98 [53], та «Вимоги щодо безпеки та захисту здоров'я працівників під час роботи з екранними пристроями» НПАОП 0.00-7.15-18.

Умови праці програміста включають наступні фактори:

– вентиляція приміщення;
– освітлення приміщення;
– параметри повітряного середовища в приміщенні, тощо.

Щоб запропонувати заходи щодо зменшення негативного впливу комп'ютера на організм людини визначимо фактори, які можуть викликати професійне захворювання і впливають на працездатність програміста.

При роботі з використанням ЕОМ відзначають наступні небезпечні та шкідливі фактори:

– ризик виникнення надзвичайних ситуацій природного або штучного характеру на об'єкті або території.

– ризик виникнення пожежі;

– негативний вплив на органи зору людини;

– ризики ураження електричним струмом;

– недостатня, або надмірна освітленість робочого місця;

– електромагнітні (у т.ч. високочастотні) електромагнітні випромінювання (коливання);

– несприятливі мікрокліматичні умови;

– нервово-емоційна напруженість праці;

– інтелектуальні навантаження;

– монотонність праці;

– невідповідність ергономічних показників робочого місця діючим вимогам;

– шум;

– статичні навантаження на кістково-м'язовий апарат;

8.3 Аналіз санітарно-гігієнічних умов праці на робочому місці програміста

Розглянемо умови праці у приміщенні, в якому працюють програмісти. Геометричні розміри приміщення наведено у таблиці 8.1.

Таблиця 8.1 – Розміри приміщення

Найменування	Значення, м
Ширина	7
Довжина	9
Висота	2,8

У зазначеному приміщенні працюють 8 людей. За даними, які наведено у табл. 8.1, та табл. 8.2, можна зробити висновок, що площа та об'єм приміщення у розрахунку на одно робоче місце програміста відповідають нормативним вимогам ДСанПіН 3.3.2-007-98 «Державні санітарні правила і норми роботи з

візуальними дисплейними терміналами електронно-обчислювальних машин» [53]. Таним чином можна зробити висновок, що санітарно-гігієнічні умови праці на робочому місці програміста відповідають вимогам.

Таблиця 8.2 – Площа та обсяг приміщення, на одного працюючого*

Геометрична характеристика	Одиниця виміру	Нормативне значення*	Фактичне значення
Площа, S	$м^2$	не менше 6.0	7,8
Об'єм, V	$м^3$	не менше 20.0	22

* Згідно ДСанПіН 3.3.2.007-98 (Державні санітарні правила і норми роботи з візуальними дисплейними терміналами електронно-обчислювальних машин).

Температура повітря в приміщенні визначається впливом температури зовнішнього повітря і тепловою енергією, яка виділяється всередині приміщення. Джерелами виділення теплоти в даному приміщенні є електроустаткування, освітлювальні прилади, а також люди. У світлий час доби джерелом надлишкового тепла є сонячна радіація.

Згідно Постанови № 42 від 01.12.1999 Головного державного санітарного лікаря України, робота, виконувана в даному приміщенні, відноситься до категорії Іа. В цьому випадку людина витрачає енергії до 120 ккал у годину. Вологість повітря в приміщенні визначається впливом багатьох факторів, серед яких: вологість атмосферного повітря, виділення вологи людьми (при диханні та випарами з поверхні шкіри).

Мікроклімат повітряного середовища в приміщенні характеризується запиленістю та загазованістю повітря. Мікроклімат приміщення визначається діючим на організм людини поєднанням, вологості, температури, швидкості руху повітря та інтенсивності теплового випромінювання. Аналіз мікроклімату складається з визначення зазначених вище факторів і порівняння результатів із встановленими нормами.

У таблиці 8.3 наведено оптимальні та фактичні значення параметрів мікроклімату як для категорії ваги робіт Іа, так і розглянутого приміщення. У

приміщеннях, де встановлено ЕОМ, рекомендується застосування тільки оптимальних значень показників мікроклімату.

Таблиця 8.3 – Оптимальні і фактичні значення параметрів мікроклімату

Пора року	Оптимальні для Іа			Фактичні		
	Температура, °С	Воло-гість,%	Швидкість повітря, м/с	Температура, °С	Воло-гість%	Швидкість повітря, м/с
Холодна	22-24	40-60	0,1	22-24	40-55	0,12
Тепла	23-25	50-70	0,1	24-25	50-65	0,1

Проведений аналіз показує, що показники мікроклімату в приміщенні відповідають установленим нормам. Штучне опалення застосовується у холодний період року.

В літню пору застосовується кондиціонер.

Для боротьби з пилом робляться регулярні провітрювання та вологі прибирання приміщенні.

У приміщенні знаходяться наступні джерела шуму: принтер *Canon PIXMA G3411*, електродвигуни вентиляторів ЕОМ.

Одним з найважливіших факторів, які впливають на ефективність трудової діяльності людини, та попереджають травматизм і професійні захворювання програмістів є освітлення на робочому місці.

З 2019 року діють Державні будівельні норми України “Природне і штучне освітлення” – ДБН В.2.5-28:2018 [54], у яких прописані вимоги до використання всіх освітлювальних приладів, у т.ч. світлодіодних.

Працю працівника, який постійно працює за комп’ютером, згідно ДБН В.2.5-28:2018 [54], можна віднести до роботи з малою точністю (найменший розмір об'єкта розрізнення від 1 до 5 мм) V-го розряду зорової роботи, з великою контрастністю об'єкта розрізнення (символів на екрані дисплея), з темним тлом (під розряд зорової роботи В). Приміщення можна віднести до 1-ої групи приміщень, у яких проводиться розрізнення об'єктів зорової роботи при

фіксованому напрямку лінії зору того, що працює на робочу поверхню. Для такого типу приміщень і розряду зорової роботи нормоване значення коефіцієнта природної освітленості (КПО) робочої поверхні (при поєднаному, спільному освітленні), повинен становити не більше 1,5%, освітленість при штучному висвітленні повинна становити 300 Лк. [54].

Крім того все поле зору повинне бути освітлено достатньо рівномірно – це основна гігієнічна вимога. Так як яскраве світло на ділянці периферійного зору значно збільшує напруженість очей і, як наслідок, призводить до їх швидкої стомлюваності, ступінь освітлення приміщення і яскравість екрану комп'ютера повинні бути приблизно однаковими.

8.4. Розробка заходів з умов поліпшення охорони праці

Згідно аналізу умов праці в розглянутому приміщенні, ми одержали наступні результати:

– розміри приміщення, у розрахунку на одному працюючого, відповідають нормативам;

– мікроклімат відповідає нормативному значенню;

– акустичні умови роботи не перевищують нормативних значень;

Таким чином можна припустити, що основною причиною можливого зниження працездатності програміста є психофізіологічний фактор, тому основна пропозиція буде така: дотримання позитивної психологічної атмосфери в колективі та регламентованого режиму праці та відпочинку, організація робочого місця з урахуванням ергономічних вимог.

Рекомендовані заходи: регулярні періодичні наочні огляди персоналом шляхів для евакуації людей із приміщення, відповідно до плану евакуації (який повинен розташовуватись на видному місці у приміщенні), включення до колективного договору мінімально можливого вмісту аптечок з обов’язковою наявністю масок-клапанів, або іншого спорядження для штучного дихання.

Регулярна періодична перевірка параметрів заземлення та занулення (вимірювання опору ланцюга).

Регулярна наочне знайомство персоналу із шляхами для евакуації людей із приміщення відповідно до плану евакуації, забезпечення розподільних щитів спеціальними розетками з заземлюючими контактами; організація заземлення всіх приладів і пристроїв, які працюють при напрузі вище 36 В.

Так як при ураженні електричним струмом у людини може статися фібриляція шлуночків серця, в організації бажано мати дефібрилятор і підготовлений персонал для роботи з ним.

8.5 Розрахункова частина

Занулення, як основний засіб захисту, застосовується в електроустановках до 1 кВ з глухозаземленою нейтраллю трансформатора або генератора, або з глухозаземленим виводом джерела однофазного струму, а також з глухозаземленою середньою точкою в трьохпроводних мережах постійного струму.

Занулення електричних установок виконується навмисним з'єднанням корпусів електричних установок із захисним нульовим проводом за допомогою занулюючих провідників.

Занулення повинно відповідати вимогам [54,3].

Занулення електроустановок необхідно застосовувати :

– при номінальній напрузі 380 В і вище змінного струму і 440 В і вище постійного струму ;

– при номінальній напрузі вище 42 В змінного струму і вище 110В постійного струму тільки в приміщеннях з підвищеною небезпекою, особливо небезпечних і зовнішних установках;

– при встановлені електрообладнання у вибухонебезпечних зонах.

Ефективність роботи занулення визначається чітким і швидким відключенням пошкодженої ділянки електричної мережі при однофазному

замиканні на корпус електрообладнання. Швидкість спрацювання захисних пристроїв (запобіжники або автоматичні вимикачі) залежить від відношення струму короткого замикання у місці пошкодження до номінального струму уставки відключаючого пристрою.

Розрахунок занулення складається з трьох частин:

1. Розрахунок на відключаючу спроможність;

2. Визначення максимальної напруги на корпусі обладнання відносно землі при замиканні фази на корпус;

3. Розрахунок робочого і повторного заземлювачів.

Початкові данні:

1. Потужність електродвигуна, який підлягає зануленню : Р = 13кВт.

2. Кількість електродвигунів: m = 12.

3. Потужність освітлювальних приладів : Ро = 27 кВт.

4. Довжина магістрального кабеля: LM=100 м.

5. Довжина розгалудження (від розподільчого щита до електродвигуна) : l=25 м.

6. Материал провідників кабеля−алюміній.

7. Лінійна напруга U=380 В.

8. Фазна напруга Uф=220 В.

Розрахунок на відключаючу спроможність.

1.1. Визначаємо параметри розгалуження (дільниця 2).

1.1.1.Визначаемо силу номінального струму електроустановки (електродвигуна верстата, преса,

тощо) :

$$I_{ном} = I_{max} = \frac{P \cdot 1000}{\sqrt{3} \cdot U_{л} \cdot \cos\varphi} = \frac{13 \cdot 1000}{\sqrt{3} \cdot 380 \cdot 0{,}84} = 23{,}51 \qquad (1)$$

де Р − номінальна потужність електродвигуна, кВт.;

$U_{л}$ − лінійна напруга, В;

cos**φ**□ − коефіцієнт потужності, приймається в залежності від типу

електрообладнання в межах 0,8..0,87.

1.1.2. Визначаємо силу пускового струму електродвигуна, А :

$$I_{пус} = 5 \cdot I_{н} = 5 \cdot 23,51 = 117,55 \text{ А} \quad (2)$$

1.1.3. Визначаємо номінальну силу струму апарата захисту :

$$I_{Н} = \frac{I_{пус}}{\beta} = \frac{117,55}{2,7} = 43,5 \text{ А} \quad (3)$$

де **β** − коефіцієнт пуску електродвигуна, приймається :

− для легких умов пуску − 2,5..3.

З таблиці 8.4 вибираємо запобіжник ПН 2-100 з плавкою вставкою $I_{ном}$ =50 А.

Таблиця 8.4 – Технічні параметри запобіжників

тип	Номінальна Напруга	Номінальний струм. А		Граничний відключаючий струм, струм, кА, при напрузі (В):	
		Запобі жника	Вставка плавлення	220	380
ПН2-100	380 220	100	30,40,50 60,80,100	–	100
ПН2-250		250	80,100 120,150 200,250	–	100
ПТ2-100		400	200,250 300,400		40
ПТ2-600		600	300,400 500,600	–	25
ПТ17-39	380 440	1000	500,630 800,1000		110
ПТ18-33	660 440	160	50,63,80 100,125 160	–	–
ПТ18-34		250	125,160 200,250		
ПТ18-39		630	400,500 630		
ПТ18-41		1000	630,800 1000		
ПТ18-37		400	250 320 400		

Таблиця 8.5 – Технічні данні автоматичних вимикачів серії А3700.

1	2	3	4
Тип	Номінальний струм, А.	Межа регулювання Номінальногоструму	Розчеплювач, уставка струму трогання, А.
1	2	3	4
А 3714БА 3734БА 3744	Виконання струмообмежуюче з півпровідниковим і електро–магнітним розчеплювачем максимального струму.		
	160 400 630	80,100,125,160 250,320,400 400,500,6300	1600 4000 6300
А 3741Б	Виконання струмообмежуюче з електромагнітним розчеплювачем максимального струму		
	630		4000 5000 6300
А 3734СА 3744С	Виконання селективне з півпровідниковим розчеплювачем максимального струму.		
	250 250 400 630	160,200,250 160,200,250 250,320,400 400,500,630	Електромагнітного Розчеплювача нема
А 3725БА 3735БА 3745Б	Виконання з термобіметалевим і електромагнітним розчеплювачеммаксимального струму.		
	250 400 630	160,200,250 250,320,400 400,500,630	– 4000 6300

1.1.4 Визначаємо найменше допустиме по умовам спрацьовування захисту значення сили струму короткого замикання, А

$$I_{ктіп} = I_{Н} \cdot К = 50 \cdot 3 = 150 \text{ А} \quad (4)$$

де $I_Н$–номінальний струм апарата захисту (із табл. 8.4)

К – коефіцієнт надійності; значення коефіцієнта К приймається :

– при захисті автоматичним вимикачем зі зворотньою залежністю від струму характеристикою К=3 (в вибухонебезпечних приміщеннях К=6).

– при захисті плавкими запобіжниками : К=3 (в вибухонебезпечних приміщеннях К=4).

Апарат захисту вибираємо з таблиці 8.4 або 8.5.

1.1.5 Знаходимо переріз провода або кабеля розгалуження з умови допустимого нагрівання:

$$I_{доп} \geq I_{max}$$

$$I_{доп} \geq 23{,}51 \text{ А} \quad (5)$$

де $I_{доп}$ − тривало допустимий із умови нагрівання струм навантаження провідника, А.

Площа перерізу узгоджується з номінальним струмом плавкої вставки запобіжника із умови :

$$I_{доп} \geq \frac{I_{вст}}{\alpha} = \frac{50}{3} = 16{,}6 \text{ А} \quad (6)$$

де **α** − коефіцієнт відповідності, який залежить від умов прокладання і нагляду за мережею : **α** = 3 − для промисливих мереж.

Площа перерізу вибирається по більшому із розрахованих по формулам (5) і (6) (у нас 23,51 і 16,6 А.) з таблиці 8.5.

Вибираємо площу перерізу 6 мм2 (Sф) при числі проводів і = 4 розташований у повітрі.

1.2 Визначаємо параметри магістрального кабеля (дільниця 1).

1.2.1 Визначаємо максимальний робочий струм

$$I_{роб} = I_{max} = Ko\sum_{1}^{n}(K_з \cdot I_{ном}) = \left(K_з \frac{P \cdot 1000}{\sqrt{3} \cdot U_л \cdot \cos\varphi} \cdot m + K_з \frac{P_O \cdot 1000}{\sqrt{3} \cdot U_л \cdot \cos\varphi} \right) \quad (7)$$

$$I_{роб} = I_{max} = 0{,}77 \left(0{,}83 \frac{13 \cdot 1000}{1{,}73 \cdot 380 \cdot 0{,}84} \cdot 12 + \frac{27 \cdot 1000}{1{,}73 \cdot 380 \cdot 0{,}84} \right) = 282{,}98 \text{ А.}$$

де Іном − номінальний струм кожного електроприймача, А.;

Ко − коефіцієнт одночасності роботи групи електроприймачів;

Кз – коефіцієнт завантажених електродвигунів;

m=12 − кількість електроприймачів;

Ко = 0.7...0.8; Кз = 0.8...0.9;

Іном − визначається з (1)

Ро =27кВт − потужність освітлювальної мережі;

Таблиця 8.6 – Тривало допустимий струм Ідоп для проводів і кабелів на напругу до 1 кВ., з алюмінюевими жилами при напрузі навколишнього повітря 25 С.

Група провідників	Провода з гумовоюі пластмасовою ізоляцією					Кабелі і захищені провода з гумовою ізоляцією				Кабелі з паперовою пропитаною ізоляцією			
марка	АПР. АПРТО-АПРВ. АПВ					АВРГ- АНРГ-АВВГ- АВРБТ-АВВБ				ААГ - АСГ - ААБГ		ААБ-АСБ	
Спосіб прокладки	відкри-та	в стальних трубах				у повітрі		в землі		у повітрі		в землі	
Площа перерізу, мм^2.	*Ідоп*, А.	*Ідоп* при числі проводів:				*Ідоп* при числі проводів:							
	–	3	4	6	9	3	4	3	4	3	4	3	4
2,5	24	19	15	14	19	17	29	26	22	–	31	–	
4	32	28	23	22	21	27	24	38	35	22	–		–
6	39	32	30	26	24	32	29	46	42	35	3	31	4
10	60	47	39	39	35	42	38	70	63	46	5	55	6
16	75	60	55	48	45	60	54	90	81	60	45	75	65
25	105	80	70	65	60	75	68	115	104	80	60	90	90
35	130	95	85	75	70	90	81	140	126	95	75	1251	115
50	165	130	120	105	95	110	100	175	158	120	95	4518	135
70	210	165	140	130	125	140	126	210	190	155	110	0220	165
95	255	200	175	–	–	170	153	255	230	190	140	2603	200
120	295	240	200	–	–	200	190	295	266	220	165	0033	240
150	340	255	–	–	–	235	212	335	302	255	200	5380	270
185	390	–	–	–	–	270	243	385	347	290	230		305
											260		345

1.2.2 Визначається струм короткочасного перевантаження магістрального кабеля:

$$I_{ПЕР} = Ko \cdot \sum_{1}^{n-1} (K_З \cdot I_{ном}) + I_{ПУС} = Ko \left(K_З \frac{P \cdot 1000}{\sqrt{3} \cdot U_л \cdot \cos\varphi} \cdot n + K_З \frac{P_О \cdot 1000}{\sqrt{3} \cdot U_л \cdot \cos\varphi} \right) + I_{ПУС} \quad (8)$$

$$I_{ПЕР} = 0{,}77 \left(0{,}84 \frac{13 \cdot 1000}{1{,}73 \cdot 380 \cdot 0{,}84} \cdot 11 + 0{,}84 \frac{27 \cdot 1000}{1{,}73 \cdot 380 \cdot 0{,}84} \right) + 117{,}55 = 316{,}4 \text{ А.}$$

де $\sum_{1}^{n-1} (K_З \cdot I_{ном})$ – максимальний струм навантаження мережі від усіх електроприймачів за винятком струму навантаження того електродвигуна, який дає найбільший приріст пускового струму.

Іпус – визначається з (2).

1.2.3 Визначаємо струм спрацювання теплового або електромагнітного розчеплювалювача автоматичного вимикача:

$$I_{спр} \geq I_{роб} \geq 282{,}98 \text{ А.} \quad (9)$$

Струм спрацювання електромагнітного розчеплювача додатково перевіряємо помаксимальному струму перевантаження лінії:

$$I_{спр} \geq 1{,}25 \cdot I_{пер} = 1{,}25 \cdot 316{,}4 = 395{,}5 \text{ А.} \quad (10)$$

Струм спрацювання розчеплювача і автоматичний вимикач вибираємо з табл.8.6.

Приймаємо Іспр = 400 А. Вимикач : А3734Б.

Таблиця 8.7 – Технічні данні автоматичних вимикачів серії А3700.

1	2	3	4
Тип	Номінальнийструм, А.	Межа регулювання Номінальног оструму	Розчеплювач, уставка струму трогання, А.
1	2	3	4
	Виконання струмообмежуюче з півпровідниковим і електро–магнітним розчеплювачем максимального струму.		
А 3714Б А 3734Б А 3744	160 400 630	80,100,125,160 250,320,400 400,500,6300	1600 4000 6300
А 3741Б	Виконання струмообмежуюче з електромагнітним розчеплювачем максимального струму		
	630		4000 5000 6300

Продовження таблиці 8.7

Тип	Номін.струму, А.	Межа номін.струму	Розчеплювач установка. струму троган ня
А 3734СА 3744С	Виконання селективне з півпровідниковим розчеплювачеммаксимального струму.		
	250 250 400 630	160,200,250 160,200,250 250,320,400 400,500,630	Електромагнітного Розчеплювача нема
А 3725БА 3735БА 3745Б	Виконання з термобіметалевим і електромагнітним розчеплювачеммаксимального струму.		
	250 400 630	160,200,250 250,320,400 400,500,630	4000 6300

1.2.4 Вибираємо площу перерізу Sф магістрального кабеля (провідника) по Ідоп, із табл.8.5.

$$I_{доп} = I_{max} = 282{,}98 \text{ А.}$$

Sф = 120mm2 , − кабель АВВБ прокладений в землі, і=3 (число проводів).

Вибрану площу перерізу перевіряємо для автоматів з електромагнітним розчеплювачем:

$$I_{доп} \geq \frac{I_{спр}}{4{,}5} = \frac{395}{4{,}5} = 87{,}7 \text{ А.} \quad (11)$$

Проводимо узгодження з номінальним струмом автомата

$$I_{доп} = \frac{I_{спр}}{3} = \frac{395}{3} = 131{,}6 \text{ А.} \quad (12)$$

Значення 87,7 і 131,6 А. менше ніж I_{max}= 298,98 А., значить площа перерізу кабеля вибрана вірно.

1.3 Визначаємо потужність трансформатора:

$$N_{ТР} = \frac{K_{П} \cdot P_{ном}}{\cos\varphi} = \frac{0{,}7 \cdot 200}{0{,}8} = 175 \text{ } кВ\,А \quad (13)$$

де Рном − сумарна потужність електроприймачів, кВт.

$\cos^{\varphi}$ − середній коефіцієнт потужності електроприймачів (0,8);

Кп − коефіцієнт попиту (0,7).

Одержане значення потужності трансформатора округляємо до ближчого

стандартного значенняпо табл.8.7. По цим же таблицях визначають опір трансформатора ZT. Із табл.8.7 вибираємо трансформатор на 180 кВА (ZT =0,151 Ом).

Таблиця 8.8 – Приблизні розрахункові повні опори обмоток масляних трансформаторів

Потужність трансфор-матора КВ·А	Номінальна напруга обмоток вищої напруги, КВ	Z_T, Ом при схемі з'єднання обмоток	
		$Y/Y_н$ (не реком)	$Δ/Y_н$; $Y/Z_н$
25	6..10	3,110	0,906
40	6..10	1,949	0,562
63	6..10	1,237	0,360
	20..35	1,136	0,407
100	6..10	0,799	0,226
	20..35	0,781	0,327
160	6..10	0,487	0,141
	20..35	0,478	0,203
250	6..10	0,312	0,090
	20..35	0,305	0,130
400	6..10	0,195	0,056
	20..35	0,191	
630	6..10	0,129	0,042
	20..35	0,121	–
1000	6..10	0,054	0,017
	20..35	0,051	0,020

1.4 Визначаємо площу перерізу нульового захисного провідника із умов:

$$I_{KMIN} \geq K \cdot I_H \tag{14}$$

$$\frac{1}{Z_n} \geq \frac{0,5 \cdot 1}{Z_\phi} \tag{15}$$

де Z_H і $Z_Ф$– повні опори відповідно нульового і фазного провідників, із умов (14, 15) визначаємо орієнтовно площу перерізу провідника.

Для магістрального кабеля $S_{n1} \geq 0,5 \cdot S_\phi = 0,5 \cdot 120 = 60$ мм^2. Для розгалуження $S_{n1} \geq 0,5 \cdot 6 = 3$ мм2. По табл.8.6. округляємо ці значен ня до найближчих більших 50 мм2 (Sn1), і 4 мм2 (Sn2).

1.5 Визначаємо активний і індуктивний опір фазного і нульового захисного провідників на дільницях 1 і 2.

$$R_ф = \rho\frac{L_M}{S\varphi_1} + \rho\frac{L_M}{S\varphi_2} = 0{,}028\frac{100}{120} + 0{,}028\frac{25}{6} = 0{,}148 \ Ом \qquad (16)$$

$$R_ф = \rho\frac{L_M}{Sn_1} + \rho\frac{L_M}{Sn_2} = 0{,}028\frac{100}{50} + 0{,}028\frac{25}{4} = 0{,}231 \ Ом \qquad (17)$$

1.5.1 Визначаємо індуктивний опір.

Для окремо проложених нульових провідників иого приимають рівним 0,6 Ом/км. При прокладці кабелем, або в сталєвих трубах індуктивним опором нехтують.

1.6 Знаходимо дійсне значення (модуль) струма однофазного короткого замикання.

$$I_{KP} = \frac{U_ф}{\frac{Z_T}{3} + \sqrt{\left(R_ф + R_n\right)^2} + \sqrt{\left(X_ф + X_n + X_{'n}\right)^2}} = \frac{220}{\frac{0{,}151}{3} + \sqrt{(0{,}148+0{,}231)^2}} = 512{,}46 \text{ А.} \qquad (18)$$

Якщо значення Ікр перевищує значення наименшого допустимого по умовам спрацюваннння захисту Ікmin {залежність(4)}, то захисний провідник вибраний вірно. Якщо Ікр меньше Ікmin, то необхідно збільшити Sn1 (збільшуємо з 50 мм2 до 120 мм2) та Sn розгалуження (збільшуємо з 4 мм2 до 6 мм2) і знову зробити перерахунок по (17) (при Sn1 = 120 мм2, Sn2==6 мм2) і вибрати нові значення із табл.8.9., якщо нульовий провідник сталевий.

У нас Ікр > Ікmin, значить провідники вибрані вірно.

Таблиця 8.9 – Активний **r** та індуктивний ***X*** опори сталевих провідників при змінному струмі 50 Гц, Ом/км.

Розміри (діаметр) перерізу. мм².	Площа перерізу мм².	Густина струму, А/мм.							
		0,5		1,0		1,5		2,0	
		r	x	r	x	r	x	r	x
		Полоса прямокутного перерізу							
20x4	80	5,24	3,14	4,20	2,52	3,45	2,09	2,97	1,38
30x4	120	3,66	2,20	2,91	1,75	2,35	1,43	2,04	1,22
30x5	150	3,38	2,03	2,56	1,54	2,08	1,25	–	–
40x4	160	2,80	1,68	2,24	1,34	1,81	1,09	1,59	0,92
50x4	200	2,28	1,37	1,79	1,07	1,45	0,87	1,24	0,74
50x5	250	2,10	1,26	1,60	0,96	1,28	0,77	–	–
60x6	300	1,77	1,06	1,34	0,8	1,08	0,65	–	–
		Провідник круглого перерізу							
5	19,63	17,0	10,2	14,9	8,65	12,4	7,45	10,7	6.4
6	28,27	13,7	8,20	11,2	6,70	9,4	5,65	8,0	4,8
8	50,27	9,60	5,75	7,5	4,50	6,4	3,84	5,3	3,2
10	78,54	7,20	4,32	5,4	3,24	4,2	2,52	–	–
12	113,1	5,60	3,36	4,0	2,40	–	–	–	–
14	150,9	4,55	2,73	3,2	1,92	–	–	–	–
16	201,1	3,72	2,23	2,7	1,60	–	–	–	–

1.7 Визначення максимальної напруги на корпусі обладнання відносно землі при замиканні фази на корпус.

$$U_{kmax} = I_{кр} \cdot Z_{н} \le U_{дан.д} \tag{19}$$

У нас Zн = Rn.

$$U_{kmax} = 512{,}46 \cdot 0{,}231 = 118{,}37\ B > 36\ B \tag{20}$$

де Uдан.д. − це допустима напруга, що нормується по ГОСТ 12.1.038-82. При часі дії більше 1 с. приймається 36 В.

ZH− повний опір нульового провідника.

Умова не виконується. Необхіно збільшити перерізи Sn1 та Sn2 до Sф1 та Sф2 і зробити перерахунок починаючи з (17):

$$R_ф = \rho \frac{L_M}{S\varphi_1} + \rho \frac{L_M}{S\varphi_2} = 0{,}028 \frac{100}{120} + 0{,}028 \frac{25}{6} = 0{,}148 \ Ом \qquad (20)$$

$$I_{КР} = \frac{U_ф}{\frac{Z_T}{3} + \sqrt{\left(R_ф + R_п\right)^2} + \sqrt{\left(X_ф + X_п + X'_п\right)^2}} = \frac{220}{\frac{0{,}151}{3} + \sqrt{(0{,}148+0{,}148)^2}} = 767{,}62 \text{ А.} \qquad (21)$$

$U_{ктах} = I_{кр} \cdot Z_н \le U_{дан.д}$ (у нас Zн= Rn) $U_{ктах} = 767{,}62 \cdot 0{,}148 = 113{,}6$ В.

113,6 В > 36 В. Умова не виконується.

Якщо і після цього умова (19) не виконується, то необхідно або замінити запобіжник з плавкою вставкою на автоматичний вимикач із струмовим реле, що дає можливість зменшити час замикання на копус (див. додаток) і підвищити допустиму напругу на корпусі або застосувати повторне заземлення нульового захисного провідника. Застосовуємо повторне заземлення нульового захисного провідника.

Визначимо необхідний його опір:

$$R_п = \left(U_{дон} \cdot R_о\right) / \left(I_{кр} \cdot Z_н\right) - U_{дон} = 36 \cdot 4 / \left(\left(767{,}62 \cdot 0{,}148\right) - 36\right) = 1{,}85 \ Ом \qquad (22)$$

8.6 Висновки до розділу

Як ми з'ясували, робота за комп'ютером має деякі шкідливі фактори. З погляду санітарних норм роботу за комп'ютером можна зробити безпечною. Для цього потрібно забезпечити працівників достатнім робочим місцем та правильним заземленням.

З цих міркувань було здійснено аналіз умов праці, призначеного для праці програмістів, проведено розгляд небезпечних та шкідливих факторів, що негативно впливають на програмістів під час роботи. Виконано розрахунок робочого і повторного заземлення нульового привода, як одного з ключових факторів впливу на працездатність та здоров'я програміста. Розроблено заходи з умов поліпшення охорони праці.

9 ОСНОВНІ ВИСНОВКИ

Програмне забезпечення, створене в результаті виконання випускної кваліфікаційної роботи за другим (магістерським) рівнем вищої освіти, призначено для повноцінного парсингу даних з сайтів та подальшого збереження його в хмарному сховищі чи скачування.

У випускній кваліфікаційній роботі за другим (магістерським) рівнем вищої освіти наведені теоретичне узагальнення й рішення наукового завдання по дослідженню та програмній реалізація хмарного сервісу для парсингу сайтів на основі технології Octoparse.

Рішення даного завдання полягало у вирішенні наступних задач:

– Проведені огляду існуючих систем парсингу сайтів.

– Дослідженні системи хмарних сервісів.

– Створенні переваг над іншими системами парсинга.

Розроблене програмне забезпечення просте, що забезпечує легкість у освоєнні програмного продукту, зручність у використанні, і не потребує особливих спеціальних знань.

Проведено аналіз існуючих парсингових та хмарних сервісів, в ході якого були виявлені, як їх недоліки так і переваги; побудовано алгоритм і вибрано середовище розробки.

При створені програмного забезпечення було використано об’єктно-орієнтований підхід, та деякі паттерни проектування, що відповідає сучасним тенденціям у галузі розробки комерційних програмних систем.

Програма реалізована на мові JavaScript(NodeJS). Дана мова програмування дозволяє зручно працювати з мережею, з DOM вузлами та обробляти інформацію.

Програма може запускатися на будь-якій платформі, за умов встановленої платформи NodeJS.

В цілому створене програмне забезпечення підтверджує правильність

використаних проектних рішень та повністю відповідає вимогам технічного завдання. Створене програмне забезпечення має велику потенційну можливість для подальшого вдосконалення і застосування у різних напрямках

Розроблена програма має реальний економічний ефект від її впровадження у виробництво у сумі 46787 грн. З урахуванням вартості розробки програми та обладнання, строк окуплення становить 0,59 роки.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Ryan M., «Web Scraping with Python: Collecting Data from the Modern Web 2st Edition», Boston company HedgeServ, 2021, Pages 54-67 (Ebook).

2. Seppe v. B., Baesens B., «Practical Web Scraping for Data Science Best Practices and Examples with Python», ISBN-13 (electronic): 978-1-4842-3582-9, 2018, Pages 121-126 (Ebook).

3. Aydin O., «R Web Scraping Quick Start Guide Techniques and tools to crawl and scrape data from websites», Department of Statistics at Mimar Sinan University, 2018, Pages 34-38 (Ebook).

4. Degardin A., «Web Robots Boost your business with Robotic Process Automation and Web-Scraping on Web-marketing», Independently published (September 21, 2018), ISBN-10 : 1723909416, Pages 95-108 (Ebook).

5. M. J. Patel, «Getting Structured Data from the Internet Running Web Crawlers/Scrapers on a Big Data Production Scale», Published by data scientist of Specrom Analytics, ISBN-10 : 1484265750, November 2020, Pages 167-174, (Ebook).

6. Rafaels R. J., «Cloud Computing from Beginning to End», ISBN-10 : 1511404582, ISBN-13 : 978-1511404587, April 2015, Pages 75-83, (Ebook).

7. Hurwitz Judith S., R. Bloor, M. Kaufman, F. Halper, «Cloud Computing for Dummies», ISBN-10 : 1119546656, ISBN-13 : 978-1119546658, August, 2020, Pages 38-45, (Ebook).

8. Ronald O. J., «Cloud Computing», ISBN-10 : 1515401592, ISBN-13 : 978-1866404587, April 2016, Pages 65-83, (Ebook).

9. Judith S., R. Bloor, «Cloud Computing is easy», ISBN-10 : 1123546856, ISBN-13 : 978-1115532658, August, 2018, Pages 33-55, (Ebook).

10. M. Casciaro, L. Mammino, «Node.js Design Patterns», Format: Kindle Edition, ISBN-10: 1839214112, ISBN-13: 978-1839214110, July 2020, Pages 70-157, (Ebook).

11. J. Wexler, «Get Programming with Node.js», Format: Tankobon Softcover,

ISBN-10: 1617294748, ISBN-13: 978-1617294747, March 15, 2019, Pages 220-243 (Ebook).

12. A. Mead, «Advanced Node.js Development», Packt Publishing, ISBN-10 : 1788393937, ISBN-13 : 978-1788393935, March 30, 2018, Pages 98-108 (Ebook).

13. T. Hunter, «Distributed Systems with Node.Js», ISBN-10 : 1492077291, ISBN-13 : 978-1492077299, O'Reilly Media, 1st edition, November 24, 2020, Pages 134-156 (Ebook).

14. Астахова, І. Ф., «Комп'ютерні науки. Дерева, операційні системи», мережі/І.Ф. Астахова, І.К., Астанін та ін. – М.: Фізматліт, 2013. – 88 с.

15. Павлиш В. А., «Основи інформаційних технологій та систем»: Навчальний посібник. / Павліш В. А., Гліненко Л. К. – Львів: Видавництво Львівської політехніки, 2013. − 500 с.

16. Федотова Є. Л., «Комп’ютерні технології та системи»: навч. посібник/Є.Л. Федотова. - М.: ВД "ФОРУМ": ІНФРА-М, 2014. - 352 с.

17. Шостак І. В., «Підхід до автоматизації процесу формування нормативного профілю під час сертифікації програмних продуктів»// Системи обробки інформації. - Харків: ХУПС. - 2010. - № 8 (89). - С. 122-126.

18. Сікірда Ю. В., «Інформаційні системи і технології»: конспект лекцій / Ю. В. Сікірда, А. В. Залевський. – Кіровоград : Видавництво КЛА НАУ, 2013. – 177 с.

19. Соколов В. Ю., «Інформаційні системи і технології»: навч. посіб. / Соколов В.Ю. − К. : ДУІКТ, 2010.− 138 с.

20. Попов, І. І., «Комп'ютерні мережі»: Форум, 2004, – 336 с.

21. Лаврищева Є. М., «Методи програмування». Теорія, інженерія, практика. - К.: Наук. думка, 2006.-450с.

22. Node.js v17.9.1 documentation [Електрон. ресурс] − Режим доступу: https://nodejs.org/docs/latest-v17.x/api/.

23. Паттерни проектування [Електрон. ресурс]. − Режим доступу: https://refactoring.guru/ru/design-patterns.

24. Сучасний підручник JavaScript [Електрон. ресурс] − Режим доступу:

https://learn.javascript.ru/.

25. AWS Documentation [Електрон. ресурс] – Режим доступу: https://docs.aws.amazon.com/.

26. Аналізатор коду jslint [Електрон. ресурс] – Режим доступу: https://www.jslint.com/.

27. Збірач модулів Webpack [Електрон. ресурс] – Режим доступу: https://webpack.js.org/concepts/.

28. Система управління версіями git [Електрон. ресурс] – Режим доступу: https://git-scm.com/.

29. Лащевский Т., Арора К., Фарр Е., «»,Хмарні архітектури: розробка стійких та економічних хмарних програм, ISBN: 978-5-4461-1588-4, 2021, 320 с.

30. B. Piper, D. Clinton, «AWS Certified Cloud Practitioner Study Guide»: CLF-C01 Exam, ISBN-10: 1119490707, ISBN-13: 978-1119490708, July 2, 2019, Pages 450.

31. S. Shrivastava, N. Srivastav, R. Sheth, R. Karmarkar, «Solutions Architect's Handbook», ISBN-10: 1801816611, ISBN-13: 978-1801816618, January 17, 2022, Pages 590.

32. Буров Є., «Комп'ютерні мережі»: Наукове видання / Є. Буров. – 2-ге оновлене і доповн. Вид. Львів: БаК, 2003. – 584 с.: іл.

33. Капустін Д. А. «Інформаційно-обчислювальні мережі»: Навчальний посібник / Д. А. Капустін, В. Є. Дементьєв. - Ульяновськ: УлГТУ, 2011 - 141 с.

34. Клименко О. Ф. «Інформатика та комп'ютерна техніка», Навч.-метод. посібник / О. Ф. Клименко, Н. Р. Головко, О. Д. Шарапов; за заг. ред. О. Д. Шарапова. - К.: КНЕУ, 2002. - 534 с.

35. Колісніченко Д. Н. «JS та MySQL6. Розробка Web-додатків»,/ Д. Н. Колісніченко. - 2-ге вид., перероб. та дод, 2010. - 560 с.: Іл. + CD-ROM - (Професійне програмування).

36. Кудрявцева С. П. «Міжнародна інформація»: Навчальний посібник / С. П. Кудрявцева, В. В. Колос. - К.: Видавничий дім "Слово", 2005. - 400 с.

37. Кулаков Ю. О. «Комп'ютерні мережі»: Підручник / Ю. А. Кулаков, Г.

М. Луцький. – Київ: «Юніор», 2005. – 400 с., іл.

38. Кучинський В. Ф. «Мережеві технології обробки інформації»: Навч. посібник/В. Ф. Кучинський. Харків, Університет ІТМО, 2015. - 115 с.

39. Antony T.Velte, Toby J. Velte, Ph.D. Robert Elsenpeter «Cloud Computing: A Practical Approach», McGraw-Hill Companies, USA, 2010. – 334 p.

40. C. Baun, M. Kunze, J. Nimis, S. Tai, «Cloud Computing: Web-basierte dynamische IT-Services», Auflagen Christian Baun, Marcel Kunze, Jens Nimis, Stefan Tai. Springer-Verlag Berlin Heidelberg, German, 2011. – 172 p.

41. I. Loyd, «Own website the right way using HTML & CSS», SitePoint pty.Ltd, Australia, 2011. – 515 p.

42. I. Menken, «Cloud Computing – The Complete Cornerstone Guide to Cloud Computing Best Practices: Consepts, Terms, and Techniques for Successfully Planning, Implementing and Managing Enterprise IT Cloud Computing Technology», Australia, 2008. – 203 p.

43. Jon Duckett, «Beginning HTML, XHTML, CSS and JavaScript», Wiley Publishing, Canada, 2010. – 834 p.

44. M. Miller, «Cloud Computing Web-based Applications That Change the Way You Work and Collaborate Online. Que Publishing», USA, 2009. – 284 p.

45. R. Buyya, J. Broberg, A. Goscinski «Cloud computing Principles and Paradigms», John Wiley & Sons, Inc., New Jersey, Canada, 2011. – 637 p.

46. Ronald L. Krutz, Russell D. Vines «Cloud Security F Comprehensive Guide to Secure Cloud Computing», Wiley Publishing, Inc., USA, Canada, 2010. – 358 p.

47. Z. Matt, JS Objects, «Patterns, and Practice», Second Edition, Apress, USA, 2008. – 488 p.

48. Закон України «Про охорону праці» від 14.10.1992 р. № 2694-XII. - Режим доступу до ресурсу: https://zakon.rada.gov.ua/laws/show/2694-12 (дата звернення: 16.06.2022).

49. Наказ Міністерства соціальної політики України 14.02.2018 № 207 «Про затвердження Вимог щодо безпеки та захисту здоров'я працівників під час

роботи з екранними пристроями». - Режим доступу до ресурсу: https://zakon.rada.gov.ua/laws/show/z0508 (дата звернення: 16.06.2022).

50. Державні будівельні норми України: ДБН В.2.5-28:2018. - Режим доступу до ресурсу: https://goo.su/9AkQ).

Додаток А
(обов'язковий)

Технічне завдання

Зміст

1 Найменування та область застосування

Це технічне завдання розповсюджується на дослідження та програмну реалізацію хмарного сервісу для парсинга сайтів на основі технології Octoparse.

2 Підстава для розробки

Підставою для розробки служить завдання на випускну кваліфікаційну роботу за другим (магістерським) рівнем вищої освіти, видане на кафедрі кібербезпеки та програмного забезпечення (нак. № 18-13 від 17.08.2022 року).

3 Мета та призначення розробки

Метою випускної кваліфікаційної роботи за другим (магістерським) рівнем вищої освіти є дослідження та програмна реалізація хмарного сервісу для парсинга сайтів на основі технології Octoparse.

4 Джерела розробки

Джерелом цієї випускної кваліфікаційної роботи за другим (магістерським) рівнем вищої освіти є стосовна до теми література і існуючі аналоги.

5 Технічні вимоги

5.1 Склад продукції

Складниками розробки є:

– вибір і обґрунтування методів реалізації проекту;

– розробка програмної частин системи, а також розробка взаємодії системи з хмарним середовищем та з користувачем;

– техніко-економічне обґрунтування доцільності прийнятого до розробки програмного забезпечення;

– аналіз умов праці;

– розробка програми, що реалізує спроектовані алгоритми роботи системи.

5.2 Показники призначення

Система повинна забезпечувати:

– програмну реалізацію хмарного сервісу для парсинга сайтів на основі технології Octoparse;

– цілісність даних у процесі роботи та при зберіганні;

– простий, інтуїтивно зрозумілий інтерфейс.

5.3 Вимоги до функціональних характеристик

Розроблене програмне забезпечення не повинно мати обмежень на версію драйверів та операційної системи.

5.4 Вимоги до архітектури

Компонент, що розробляється повинен використовувати системні засоби та апаратні засоби, що на даному етапі розвитку обчислювальної техніки найбільше поширені.

5.5 Вимоги до надійності

Програмні модулі написані по всім правилам, які стосуються стандартних викликів процедур, функцій, методів і форм, визначених технічною документацією на середовище розробки.

5.6 Умови експлуатації

Робочі місця користувачів ПЗ повинні задовольняти наступним умовам експлуатації:

– температура повітря: 19-20 град. по Цельсію;

– відносна вологість повітря до 80%;

– атмосферний тиск 107 кПа.

5.7 Вимоги до складу та параметрів технічних засобів

Програмне забезпечення повинно бути реалізоване на ПЕОМ архітектури IBM PC, працювати в будь-якій ОС і з сумісними з цією платформою пристроями і прикладним програмним забезпечення.

5.8 Вимоги до інформаційної і програмної сумісності

Переносність програмного забезпечення повинна бути забезпечена за рахунок його реалізації стандартного інтерфейсу взаємодії з ОС, що працюють під управлінням ОС Windows 10/11.

5.8.1 Обладнання

Комп’ютер Intel® Celeron/8 Mb/1.2 Gb/SVGA 14” 1Mb або сумісні з ним.

5.8.2 Мова програмування

Платформа Node.js.

5.8.3 Вхідні дані

Опис алгоритму роботи запропонованої системи.

5.8.4 Вихідні дані

Робоча програма.

6 Вимоги до програмної документації

Програмна продукція повинна бути представлена у виді опису структури даних, схем та опису алгоритму, а також текстів вихідних модулів програмного забезпечення згідно ЄСПД .

7 Економічні вимоги

7.1 Для ПЗ необхідно виробити функціонально-вартісний аналіз варіантів розробки.

7.2 Виконати розрахунок витрат показників економічного ефекту з урахуванням цін на 15 вересня 2022 року.

8 Вимоги щодо охорони праці

В частині охорони праці випускної кваліфікаційної роботи за другим (магістерським) рівнем вищої освіти повинні бути розглянуті шкідливі і небезпечні фактори при роботі з комп’ютером.

9 Перелік документів, що розробляються

– Наукова новизна – 1 аркуш.
– Структурна схема системи – 1 аркуш.
– Функціональна схема системи – 1 аркуш.
– Діаграма процесів – 1 аркуш.
– Блок-схема алгоритму роботи програми – 2 аркуша.
– Показники економічної ефективності – 1 аркуш.
– Пояснювальна записка – 96 аркушів.

10 Етапи розробки

10.1 Збір і обробка інформації по темі випускної кваліфікаційної роботи за другим (магістерським) рівнем вищої освіти. Постановка задачі на виконання випускної кваліфікаційної роботи за другим (магістерським) рівнем вищої освіти (складання ТЗ).

10.2 Проведення досліджень або експериментальних робіт для уточнення основних положень випускної кваліфікаційної роботи за другим (магістерським) рівнем вищої освіти.

10.3 Розробка функціональних схем, блок схем алгоритмів роботи програмного забезпечення.

10.4 Побудова схем взаємодії даних.

10.5 Створення прототипу ПЗ.

10.6 Віднаходження ПЗ, аналіз отриманих результатів.

10.7 Робота над питанням охорони праці і техніки безпеки.

10.8 Розрахунок з техніко-економічного обґрунтування.

10.9 Оформлення пояснювальної записки і виконання робіт по графічній частині.

11 Порядок контролю та приймання

11.1 Подання випускної кваліфікаційної роботи за другим (магістерським) рівнем вищої освіти на попередній захист 10.12.2022 р.

11.2 Подання випускної кваліфікаційної роботи за другим (магістерським) рівнем вищої освіти на захист 22.12.2022 р.

Додаток Б

(обов’язковий)

Міністерство освіти і науки України

Центральноукраїнський національний технічний університет

ЗАТВЕРДЖУЮ

Керівник випускної кваліфікаційної роботи за

другим (магістерським) рівнем вищої освіти

_____ Смірнов О.А.

Дослідження та програмна реалізація хмарного сервісу для парсинга сайтів на основі технології Octoparse

Лістинг програми

Код документу 12

Носій: CD/DVD-диск / USB-флеш-накопичувач

Загальна кількість аркушів: *

Літера: РП

Кропивницький – 2022 року

Основна програма

package.json – Файл конфігурації проекту

```
{
  "name": "parsing-service",
  "version": "1.0.1",
  "description": "Parsing Service",
  "main": "server.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1"
  },
//вказуємо віддалений репозитарій
  "repository": {
    "type": "git",
    "url": "git+https://github.com/RostislavKalyuzhniy/parsing-service.git"
  },
  "author": "",
  "license": "ISC",
  "bugs": {
    "url": "https://github.com/RostislavKalyuzhniy/parsing-service/issues"
  },
  "homepage": "https://github.com/RostislavKalyuzhniy/parsing-service#readme",
//пакети які використовують в проекті
  "dependencies": {
    "@hapi/joi": "^17.1.1",
    "compression": "1.7.4",
    "cookie-parser": "1.4.6",
    "cors": "2.8.5",
    "express": "4.17.1",
    "body-parser": "^1.18.3",
    "helmet": "4.6.0",
    "mongoose": "6.0.14",
    "@aws-sdk/client-s3"
  },
//пакети які використовуються тільки на етапі розробки
  "devDependencies": {
    "eslint": "^8.3.0",
    "eslint-config-airbnb-base": "^15.0.0",
    "eslint-plugin-import": "^2.25.3"
  }
}
```

connect.js Підключення до AWS

```
// Імпорт AWS SDK клієнта і команд для Node.js.
import { PutObjectCommand, CreateBucketCommand } from "@aws-sdk/client-s3";
import { s3Client } from "./libs/s3Client.js";

// Налаштування параметрів
const params = {
  Bucket: "parsing_bucket_1", // Ім’я корзини.
  Key: "key.txt", // Вказуємо ключ.
  Body: "body.js", // Вказуємо тіло нашого запиту.
};

const run = async () => {
  // створення Amazon S3 корзини.
  try {
    const data = await s3Client.send(
        new CreateBucketCommand({ Bucket: params.Bucket })
    );
    console.log(data);
    console.log("Створено кошик ", data.Location);
    return data; // Для юніт тестів.
  } catch (err) {
    console.log("Error", err);
  }
  // Створення об’єкту і завантаження в Amazon S3 bucket корзину.
  try {
    const results = await s3Client.send(new PutObjectCommand(params));
    console.log(
        "Створено " +
        params.Key +
        " і завантажено " +
        params.Bucket +
        "/" +
        params.Key
    );
    return results;
  } catch (err) {
    console.log("Error", err);
  }
};
run();
```

server-ssesion.js Запуск сервера з сесіями

```
const http = require('node:http');

const Client = require('./client.js');
const Session = require('./session.js');
//прописуємо роути сервіса
const routing = {
  '/': async () => './index.html',
  '/start': async (client) => {
    Session.start(client);
    return `Session token is: ${client.token}`;
  },
  '/destroy': async (client) => {
    const result = `Session destroyed: ${client.token}`;
    Session.delete(client);
    return result;
  },
  '/api/parse': async (client) => {
    if (client.session) {
      client.session.set('method1', 'called');
      return { data: 'example result' };
    } else {
      return { data: 'access is denied' };
    }
  },
  '/api/ parse/': async (client) => ({
    url: client.req.url,
    headers: client.req.headers,
  }),
  '/api/data: async (client) => {
    if (client.session) {
      return [...client.session.entries()]
        .map(([key, value]) => `<b>${key}</b>: ${value}<br>`)
        .join();
    }
    return 'No session found'; //в разі невдалого запиту
  },
};
//визначаємо типи які будуть використовуватись
const types = {
  object: JSON.stringify,
  string: (s) => s,
  number: (n) => n.toString(),
  undefined: () => 'not found',
};
//створення сервера
http.createServer(async (req, res) => {
  const client = await Client.getInstance(req, res);
  const { method, url, headers } = req;
  console.log(`${method} ${url} ${headers.cookie}`);
  const handler = routing[url];
  res.on('finish', () => {
    if (client.session) client.session.save();
  });
  if (!handler) {
    res.statusCode = 404;
    res.end('Not found 404');
    return;
  }
//запускаємо перемикач
  handler(client).then((data) => {
    const type = typeof data;
    const serializer = types[type];
    const result = serializer(data);
    client.sendCookie();
    res.end(result);
  }, (err) => {
    res.statusCode = 500;
```

```
    res.end('Internal Server Error 500');
    console.log(err);
  });
}).listen(3000); //вказуємо порт на якому буде розміщений сервер
```

gracefullShutdown.js – інструкція в разі поломки сервера

```
'use strict';
//імпорт бібліотек
const cluster = require('node:cluster');
const http = require('node:http');
//список активних користувачі на даний момент
const connections = new Map();
let server = null;
let child = null;

const SERVER_PORT = 3000;
const LONG_RESPONSE = 60000; //мілісекунд, признака поломки
const SHUTDOWN_TIMEOUT = 5000;
const HTTP_REFRESH = {
  'Content-Type': 'text/html',
  'Refresh': '5',
};

const timeout = (msec) => new Promise((resolve) => {
  setTimeout(resolve, msec);
});
//запуск кластера
const start = () => {
  console.log('Запуск процесу');
  child = cluster.fork('./cluster.js');
  child.on('message', (message) => {
    if (message.status === 'restarted') {
      console.log('Рестарт');
      start();
    }
  });
};
//список з'єднань користувачів
const showConnections = () => {
  console.log('Connection:', [...connections.values()].length);
  for (const connection of connections.keys()) {
    const { remoteAddress, remotePort } = connection;
    console.log(`  ${remoteAddress}:${remotePort}`);
  }
};
//закриття з'єднань
const closeConnections = async () => {
  for (const [connection, res] of connections.entries()) {
    connections.delete(connection);
    res.writeHead(503, HTTP_REFRESH);
    res.end('Сервіс недоступний');
    connection.destroy();
  }
};

const freeResources = async () => {
  console.log('Free resources');
};
//процес виключення сервера
const gracefulShutdown = async () => {
  process.send({ status: 'restarted' });
  server.close((error) => {
    if (error) {
      console.log(error);
      process.exit(1);
    }
    process.exit(0);
  });
  await timeout(SHUTDOWN_TIMEOUT);
  await freeResources();
  await closeConnections();
};
```

```
if (cluster.isMaster) {
  start();

  process.on('SIGINT', async () => {
    child.send({ status: 'рестарт' });
  });
} else {
  server = http.createServer((req, res) => {
    console.log('Новий запит');
    connections.set(res.connection, res);
    setTimeout(() => {
      res.end();
    }, LONG_RESPONSE);
  });

  server.on('connection', (connection) => {
    console.log('Нове зёєднання');
    connection.on('close', () => {
      console.log('Close');
      connections.delete(connection);
    });
  });

  server.listen(SERVER_PORT);
  server.on('listening', () => {
    console.log(HTTP слухач);
  });

  process.on('message', async (message) => {
    if (message.status === 'restart') {
      console.log();
      console.log('Graceful shutdown');
      showConnections();
      await gracefulShutdown();
      showConnections();
      console.log(Воркер виключено');
    }
  });

  process.on('SIGINT', () => {}) виключення
}
```

workers.js – запуск воркерів

```
'use strict';

const fs = require('node:fs');
const http = require('node:http');
const path = require('node:path');

const PORT = 8000;

const STATIC_PATH = path.join(process.cwd(), './static');
const API_PATH = './api/';

const MIME_TYPES = {
  default: 'application/octet-stream',
  html: 'text/html; charset=UTF-8',
  js: 'application/javascript; charset=UTF-8',
  json: 'application/json',
  css: 'text/css',
  png: 'image/png',
  jpg: 'image/jpg',
  gif: 'image/gif',
  ico: 'image/x-icon',
  svg: 'image/svg+xml',
};

const toBool = [() => true, () => false];

const prepareFile = async (url) => {
  const paths = [STATIC_PATH, url];
  if (url.endsWith('/')) paths.push('index.html');
  const filePath = path.join(...paths);
  const pathTraversal = !filePath.startsWith(STATIC_PATH);
  const exists = await fs.promises.access(filePath).then(...toBool);
  const found = !pathTraversal && exists;
  const streamPath = found ? filePath : STATIC_PATH + '/404.html';
  const ext = path.extname(streamPath).substring(1).toLowerCase();
  const stream = fs.createReadStream(streamPath);
  return { found, ext, stream };
};

const api = new Map();

const receiveArgs = async (req) => {
  const buffers = [];
  for await (const chunk of req) buffers.push(chunk);
  const data = Buffer.concat(buffers).toString();
  return JSON.parse(data);
};

const cacheFile = (name) => {
  const filePath = API_PATH + name;
  const key = path.basename(filePath, '.js');
  try {
    const libPath = require.resolve(filePath);
    delete require.cache[libPath];
  } catch {
    return;
  }
  try {
    const method = require(filePath);
    api.set(key, method);
  } catch {
    api.delete(name);
  }
};

const cacheFolder = (path) => {
  fs.readdir(path, (err, files) => {
```

```
    if (!err) files.forEach(cacheFile);
  });
};

const watch = (path) => {
  fs.watch(path, (event, file) => {
    cacheFile(file);
  });
};

cacheFolder(API_PATH);
watch(API_PATH);

const httpError = (res, status, message) => {
  res.statusCode = status;
  res.end(`"${message}"`);
};

http.createServer(async (req, res) => {
  const [first, second] = req.url.substring(1).split('/');
  if (first === 'api') {
    const method = api.get(second);
    const args = await receiveArgs(req);
    try {
      const result = await method(...args);
      if (!result) {
        httpError(res, 500, 'Server error');
        return;
      }
      res.end(JSON.stringify(result));
    } catch (err) {
      console.dir({ err });
      httpError(res, 500, 'Server error');
    }
  } else {
    const file = await prepareFile(req.url);
    const statusCode = file.found ? 200 : 404;
    const mimeType = MIME_TYPES[file.ext] || MIME_TYPES.default;
    res.writeHead(statusCode, { 'Content-Type': mimeType });
    file.stream.pipe(res);
    console.log(`${req.method} ${req.url} ${statusCode}`);
  }
}).listen(PORT);
```

parsersing.js – запуск парсера

```
const { deepStrictEqual, throws } = require('assert');

//const { HTTPParser } = require('http-parser-js');
const { HTTPParser } = require('./http-parser.js');
// створення запиту до потрібного сайту
function parseRequest(input) {
  const parser = new HTTPParser(HTTPParser.REQUEST);
  let complete = false;
  let shouldKeepAlive;
  let upgrade;
  let method;
  let url;
  let versionMajor;
  let versionMinor;
  let headers = [];
  let trailers = [];
  let bodyChunks = [];
//вказуємо потрібні параметри
  parser[HTTPParser.kOnHeadersComplete] = function (req) {
    shouldKeepAlive = req.shouldKeepAlive;
    upgrade = req.upgrade;
    method = HTTPParser.methods[req.method];
    url = req.url;
    versionMajor = req.versionMajor;
    versionMinor = req.versionMinor;
    headers = req.headers;
  };

  parser[HTTPParser.kOnBody] = function (chunk, offset, length) {
    bodyChunks.push(chunk.slice(offset, offset + length));
  };

  // Це подія для трейлерів.
  parser[HTTPParser.kOnHeaders] = function (t) {
    trailers = t;
  };

  parser[HTTPParser.kOnMessageComplete] = function () {
    complete = true;
  };

  // Оскільки ми надсилаємо весь буфер одразу, усі зворотні виклики вище
відбуваються синхронно.
   // Синтаксичний аналізатор не робить _нічого_ асинхронного.
   // Однак, звичайно, ви можете викликати execute() кілька разів із кількома
фрагментами, наприклад. від струмка.
   // Але тоді вам доведеться змінити всю логіку, щоб вона була асинхронною
(наприклад, вирішити Promise у kOnMessageComplete та додати логіку очікування).
  parser.execute(input);
  parser.finish();

  if (!complete) {
    throw new Error('Could not parse request');
  }
//зліплюємо чанки раом
  let body = Buffer.concat(bodyChunks);

  return {
    shouldKeepAlive,
    upgrade,
    method,
    url,
    versionMajor,
    versionMinor,
    headers,
    body,
    trailers,
```

```
  };
}
// формуємо відповідь сервера
function parseResponse(input) {
  const parser = new HTTPParser(HTTPParser.RESPONSE);
  let complete = false;
  let shouldKeepAlive;
  let upgrade;
  let statusCode;
  let statusMessage;
  let versionMajor;
  let versionMinor;
  let headers = [];
  let trailers = [];
  let bodyChunks = [];
// вказуємо параметри
  parser[HTTPParser.kOnHeadersComplete] = function (res) {
    shouldKeepAlive = res.shouldKeepAlive;
    upgrade = res.upgrade;
    statusCode = res.statusCode;
    statusMessage = res.statusMessage;
    versionMajor = res.versionMajor;
    versionMinor = res.versionMinor;
    headers = res.headers;
  };

  parser[HTTPParser.kOnBody] = function (chunk, offset, length) {
    bodyChunks.push(chunk.slice(offset, offset + length));
  };

  parser[HTTPParser.kOnHeaders] = function (t) {
    trailers = t;
  };

  parser[HTTPParser.kOnMessageComplete] = function () {
    complete = true;
  };

  // Оскільки ми надсилаємо весь буфер одразу, усі зворотні виклики вище
відбуваються синхронно.
   // Синтаксичний аналізатор не робить _нічого_ асинхронного.
   // Однак, звичайно, ви можете викликати execute() кілька разів із кількома
фрагментами, наприклад. від струмка.
   // Але тоді вам доведеться змінити всю логіку, щоб вона була асинхронною
(наприклад, вирішити Promise у kOnMessageComplete та додати логіку очікування).
  parser.execute(input);
  parser.finish();
//перевіряємо на готовність
  if (!complete) {
    throw new Error('Could not parse');
  }

  let body = Buffer.concat(bodyChunks);

  return {
    shouldKeepAlive,
    upgrade,
    statusCode,
    statusMessage,
    versionMajor,
    versionMinor,
    headers,
    body,
    trailers,
  };
}

let parsed;
```

```
parsed = parseRequest(
  Buffer.from(`GET / HTTP/1.1
Host: input
`)
);

console.log(parsed);

deepStrictEqual(parsed.shouldKeepAlive, true);
deepStrictEqual(parsed.upgrade, false);
deepStrictEqual(parsed.method, 'GET');
deepStrictEqual(parsed.url, '/');
deepStrictEqual(parsed.versionMajor, 1);
deepStrictEqual(parsed.versionMinor, 1);
deepStrictEqual(parsed.headers, ['Host',input]);
deepStrictEqual(parsed.body.toString(), '');
deepStrictEqual(parsed.trailers, []);

parsed = parseRequest(
  Buffer.from(`POST /memes HTTP/1.1
Host: www.example.com
Content-Length: 7
Content-Type: text/plain
foo bar
`)
);

console.log(parsed);

parsed = parseResponse(
  Buffer.from(`HTTP/1.1 404 Not Found
Content-Type: text/html
Content-Length: 33
<strong>Computer says no</strong>
`)
);

console.log(parsed);

parsed = parseResponse(
  Buffer.from(`HTTP/1.1 200 OK
Content-Type: text/plain
Transfer-Encoding: chunked
Trailer: Expires 7Mozilla9Developer7Network
Expires: Wed, 21 Oct 2015 07:28:00 GMT
`)
);

// відловлення помилок
throws(function () {
  parseResponse(
    Buffer.from(`HTTP/1.1 200 OK
Content-Length: 1
`)
  );
});
```