DOI 10.34185/1562-9945-1-156-2025-13
УДК 004.7

В.В. Спірінцев, О.В. Спірінцева, О.С. Генчук

ДОСЛІДЖЕННЯ МОЖЛИВОСТЕЙ ПЛАТФОРМИ NODE.JS ПРИ РОЗРОБЦІ ПРОКСІ-СЕРВЕРІВ З УРАХУВАННЯМ СУЧАСНИХ ВИМОГ ДО ПРОДУКТИВНОСТІ ТА СТАБІЛЬНОСТІ

Анотація. В роботі досліджено можливості платформи Node.js до розробки проксі-сервера з використанням асинхронного підходу, описано процес створення власної реалізації проксі-сервера, включаючи архітектурні рішення та використані модулі. Здійснено тестування продуктивності в програмі FOGLDN Proxy Tester запропонованого рішення, результати якого порівняно із показниками інших популярних реалізацій Squid Proxy та Tinyproxy. На основі отриманих даних виконано аналіз ефективності запропонованого проксі-сервера. Для реалізації програмного пакету було використано мову програмування JavaScript, платформу Node.js, базу даних SQLite, ORM Sequelize, менеджер процесів PM2, бібліотеку React.
Ключові слова: проксі-сервер, JavaScript, Node.js, React, Squid Proxy, Tinyproxy .

Постановка проблеми. Проксі-сервери відіграють ключову роль у сучасних інформаційних системах, забезпечуючи ефективну, безпечну та масштабовану роботу мереж. Їх використання дозволяє оптимізувати трафік, підвищити продуктивність, забезпечити конфіденційність та стабільність у різноманітних середовищах – від корпоративних мереж до персональних користувачів. У сфері розробки проксі-серверів асинхронні платформи набули значної популярності та важливості. Вони дозволяють суттєво підвищити продуктивність, покращити масштабованість і забезпечити плавне обслуговування великої кількості одночасних з’єднань. Особливості використання асинхронних платформ можна простежити на прикладі технологій на зразок Node.js, які активно впроваджуються для реалізації проксі-серверів і суміжних мережевих рішень. Особливості використання асинхронних платформ у сфері розробки проксі-серверів полягають у неблокуючому підході до обробки трафіку [1], можливості ефективної обробки великої кількості одночасних запитів, підвищенні масштабованості та спрощенні архітектури. Це створює основу для розробки більш продуктивних, стабільних і масштабованих проксі-рішень, здатних успішно функціонувати в умовах сучасних динамічних інформаційних систем.

Аналіз останніх досліджень. Завдяки стрімкому розвитку технологій і підвищеним вимогам до продуктивності, сучасні підходи до розробки проксі-серверів акцентують увагу на масштабованості, швидкості роботи, надійності та підтримці новітніх мережевих протоколів [2]. Основні поточні тенденції:

– Використання асинхронних і подієво-орієнтованих архітектур. Один із ключових трендів у розробці проксі-серверів – перехід від традиційних блокуючих моделей обробки запитів до асинхронних, подієво-орієнтованих архітектур. Такі платформи, як Node.js, забезпечують неблокуючий ввід/вивід, що дозволяє обробляти тисячі одночасних з'єднань без суттєвого збільшення затримки [3], що не лише підвищує продуктивність, а й дає змогу ефективно масштабувати систему горизонтально.

– Мікросервісна архітектура. Сучасні підходи до розробки часто передбачають використання мікросервісної архітектури, де проксі-сервер виступає як один зі спеціалізованих сервісів або складових більшого комплексу. Завдяки поділу системи на невеликі, незалежні компоненти, кожен мікросервіс може бути оновлений, масштабований або замінений без впливу на всю систему. Ця гнучкість дозволяє швидко реагувати на зміни в навантаженні та запитах користувачів.

– Контейнеризація. Контейнеризація, наприклад за допомогою Docker, також стала стандартним інструментом для розгортання проксі-сервера. Контейнери дозволяють створювати відтворювані оточення розробки та швидко розгортати їх у будь-якій інфраструктурі – локально, у хмарі чи гібридних середовищах. Це спрощує процес оновлення, масштабування та підтримки рішень.

– Інтеграція з системами оркестрації та балансування навантаження. Сучасні проксі-сервісні рішення часто працюють у зв'язці з оркестраторами контейнерів на кшталт Kubernetes [4]. Оркестрація дозволяє автоматизувати процеси розгортання, масштабування та оновлення системи. Проксі-сервери інтегруються з сервісами балансування навантаження, які забезпечують ефективний розподіл трафіку між різними вузлами. Це сприяє підвищенню доступності та стійкості до збоїв.

– Оптимізація продуктивності за допомогою кешування та CDN. Інтеграція зі системами кешування допомагає суттєво скоротити час доступу до часто запитуваних ресурсів. Сучасні проксі-сервери можуть динамічно кешувати відповіді, що знижує навантаження на кінцеві сервери та прискорює доставку контенту кінцевому користувачу. Поєднання проксі-сервера з CDN розширює можливості оптимізації, дозволяючи обслуговувати користувачів із географічно близьких вузлів мережі.

– Підвищення безпеки та контроль доступу. Безпека є критично важливою складовою сучасних систем, і проксі-сервери відіграють тут провідну роль. Інтегровані механізми аутентифікації, авторизації, шифрування HTTPS-трафіку, а також захист від DDoS-атак дозволяють забезпечити надійну й безпечну роботу системи. Сучасні підходи включають використання протоколів OAuth, JWT-токенів та інших сучасних стандартів безпеки [5].

– Аналітика, моніторинг і спостережуваність. Сучасні підходи включають впровадження систем моніторингу, логування та трасування запитів. Це забезпечує спостережуваність і доступ до актуальних метрик у режимі реального часу, що допомагає оперативно реагувати на аномалії, оптимізувати продуктивність і покращувати стабільність проксі-сервера.

ISSN 1562-9945 (Print)
ISSN 2707-7977 (Online)

Мета роботи. Полягає у створенні універсального програмного пакету для побудови проксі-серверів з використанням асинхронного підходу і впровадженню незалежних підслужб на базі платформи Node.js, що забезпечує урахування сучасних вимог до продуктивності та стабільності.

Основна частина. Розроблений програмний пакет складається з чотирьох основних складових: головної служби, інтерфейсу налаштування, бази даних і множини генерованих підслужб для конкретних IP-адрес. Діаграма взаємодії компонентів наведена на рисунку 1.

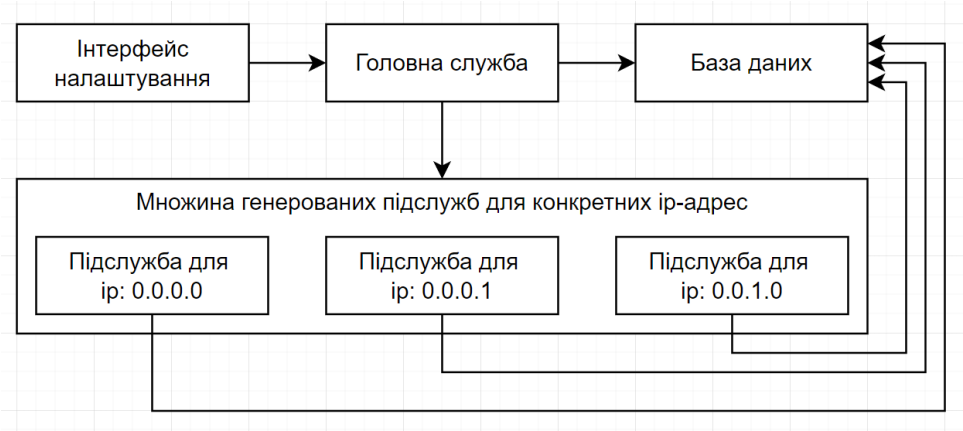

Рисунок 1 - Діаграма взаємодії компонентів програмного пакету

Головна служба відповідає за налаштування програмного пакету та виконує функції управління сутностями ip, user, auth та access, а також автоматично створює підслужби на основі екземплярів сутності ip. Це забезпечує централізоване управління всіма елементами системи. Структура головної служби розроблена для спрощення управління кодом та забезпечення модульності [6]. Вона включає наступні компоненти:

– файл app.js - основна вхідна точка служби, яка відповідає за запуск програми, підключення до бази даних, налаштування серверу та підключення маршрутизаторів;

– файли типу model - містять визначення моделей бази даних, таких як ip, user, auth та access. Вони відповідають за структуру таблиць у базі даних, а також за встановлення взаємозв’язків між ними;

– файли типу router - відповідають за маршрутизацію запитів. Кожна сутність має свій маршрутизатор, який визначає кінцеві точки для обробки CRUD-запитів та інших специфічних операцій, наприклад, маршрутизатор для сутності ip включає маршрути для активації, деактивації та отримання списку підслужб;

– файли типу controller - містять логіку обробки запитів, переданих маршрутизаторами. Контролери отримують дані з HTTP-запитів, взаємодіють із моделями бази даних, виконують необхідні операції та повертають відповідь клієнту. Для кожної сутності існує відповідний контролер.

Інтерфейс налаштування - це надбудова над API головної служби, для більш комфортної взаємодії. Це може буди CLI або сайт, що запускається разом з програмним пакетом. CLI працює постійно, повністю дублює можливості API і дозволяє вмикати і вимикати сайт. Сайт запускається на порту 13000, доступ до нього можна отримати че-

рез веб-браузер за адресою http://host:13000, де host це основна IP-адреса сервера де запущений програмний пакет. Окрім дублювання можливостей API сайт дозволяє масове завантаження нових IP-адрес і користувачів.

В базі даних (рисунок 2) зберігаються налаштування IP-адрес, дані користувачів, інформація про доступ користувачів до IP, а також записи логування. В якості бази даних було вибрано SQLite завдяки її простоті, легкості інтеграції та високій продуктивності в умовах невеликих і середніх проєктів. Розробка бази даних включала створення чотирьох таблиць: ip, user, auth і access, які зберігають всі необхідні дані для роботи програмного пакета. За допомогою ORM Sequelize забезпечено зручну інтеграцію бази даних із програмним кодом, автоматизацію створення таблиць та взаємозв'язків між ними. Це спрощує управління даними та забезпечує легкість у подальшій підтримці та розширенні функціональності.

Множина генерованих підслужб включає в себе по службі на кожну додану в програмний пакет IP-адресу. Для генерування підслужб був розроблений шаблон на основі бібліотеки Handlebars, який дозволяє створювати динамічні файли для кожної IP-адреси. Підслужби виконують функцію проксування, перевіряють авторизацію, логують запити та періодично синхронізують дані авторизації з головною службою.

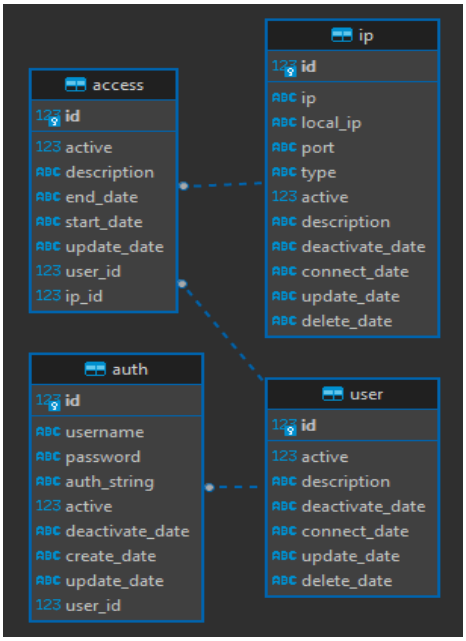

Рисунок 2 - Структура бази даних

Для підвищення швидкодії дані авторизації зберігаються в пам'яті підслужби, а не витягуються кожного разу з бази даних, для цього в підслужбі реалізована обробка запиту за фіксованою адресою із фіксованими даними авторизації. Завдяки використанню модулів http, net та url вдалося реалізувати високопродуктивний і надійний механізм обробки клієнтських запитів.

Для управління процесами програмного пакета використовується менеджер процесів PM2 - потужний інструмент для запуску, моніторингу та управління Node.js дода-

ISSN 1562-9945 (Print)
ISSN 2707-7977 (Online)

тками в реальному часі, що гарантує ефективне управління підслужбами, стабільну роботу та зручність у масштабуванні.

Розгортання програмного пакета включало налаштування виділеного серверу: встановлення Node.js, PM2 для управління процесами, Nginx для веб-інтерфейсу та маршрутизацію мережевих пакетів. Пакет легко інтегрується в інфраструктуру, забезпечуючи стабільну роботу.

На рисунках 3-6 наведено приклади взаємодії з головною службою через RESTful API. Для підвищення зручності користування в програмний пакет було включено web-інтерфейс розроблений за допомогою JavaSrcipt-бібліотеки React.

Web-інтерфейс надає доступ до повної функціональності основного API та включає 3 сторінки: /ip, /user і /access. Базовою сторінкою є /ip, де перелічені екземпляри сутності ip. Кожен екземпляр можна видалити, редагувати, активувати або деактивувати (рисунок 7). Для навігації по web-інтерфейсу використовується навігаційна панель.

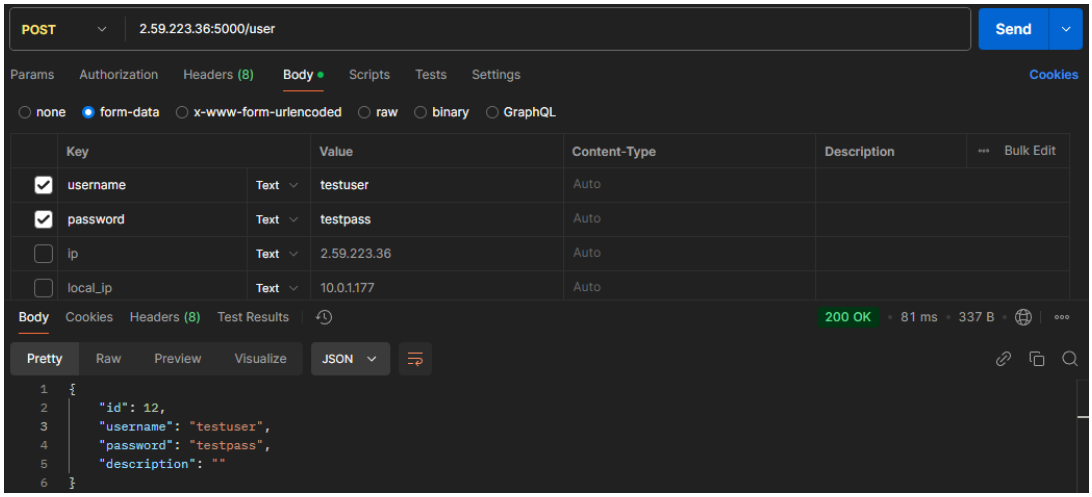

Рисунок 3 - Успішний запит на створення екземпляру сутності user

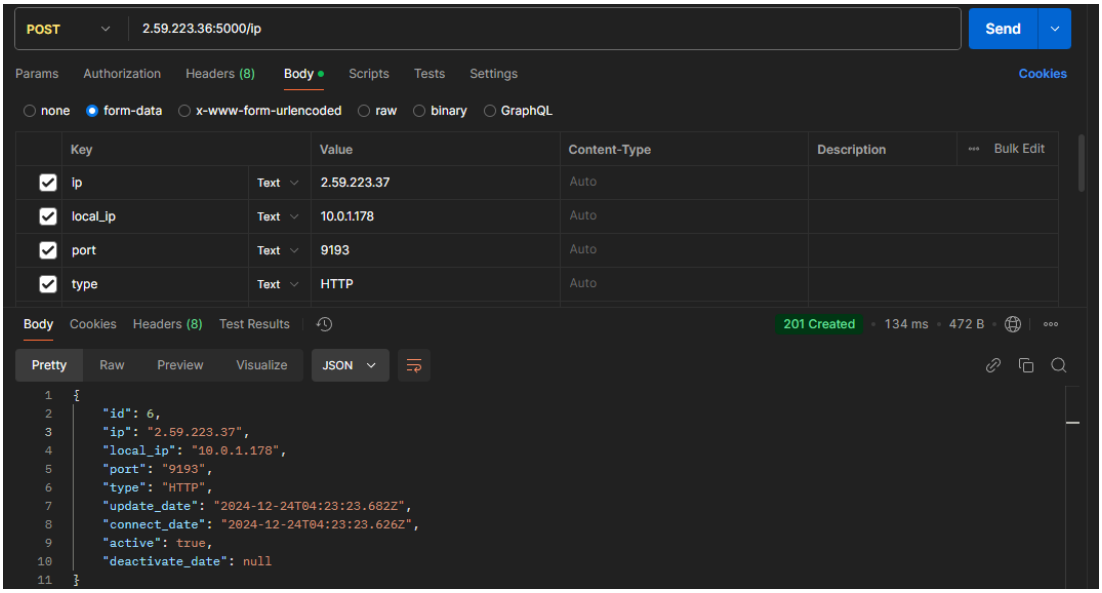

Рисунок 4 - Запит на створення екземпляру сутності ip

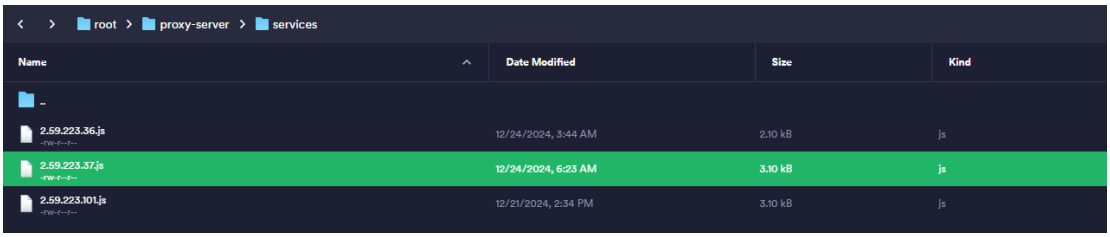

Рисунок 5 - Файл згенерований при створенні екземпляру сутності ip

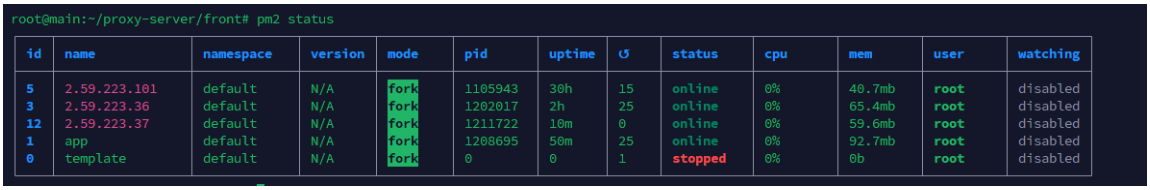

Рисунок 6 – Процес, запущений при створенні екземпляру сутності ip

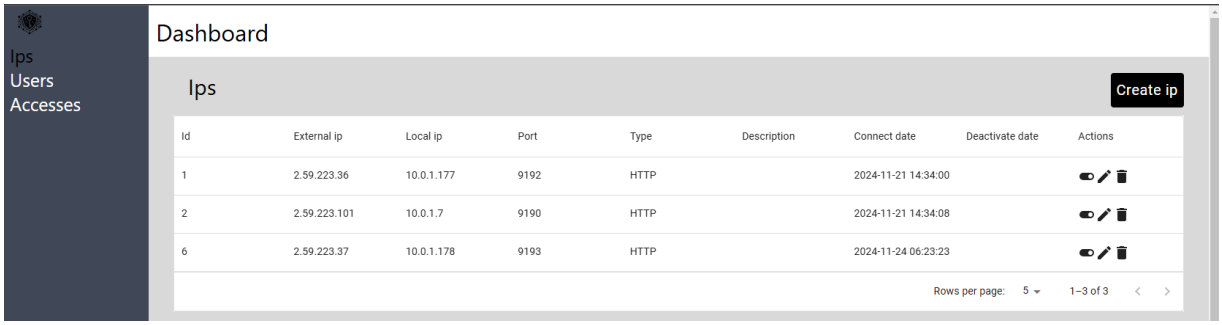

Рисунок 7 - Сторінка /ip

При натисканні на кнопку «Create ip» відкривається модальне вікно редагування (рисунок 8).

Рисунок 8 - Модальне вікно створення нового екземпляру сутності ip

Сторінка /user функціонально повторює сторінку /ip. Оскільки екземпляри сутності access деактивуються автоматично, на сторінці /access відсутні кнопки для маніпулюванням станом активності. Якщо доступ потрібно припинити або подовжити - необхідно через модальне вікно редагування змінити дату закінчення.

Існує велика кількість програмних рішень для створення проксі-серверів з урахуванням специфічних потреб, таких як швидкість обробка запитів, гнучкість конфігурації та рівень безпеки: Squid Proxy, Tinyproxy, Privoxy, HAProxy, Dante тощо. В цій робо-

ISSN 1562-9945 (Print)
ISSN 2707-7977 (Online)

ті аналізується продуктивність різних підходів до реалізації проксі-серверів (порівнюються проксі-сервер, створений за допомогою розробленого програмного пакету із популярними існуючими реалізаціями Squid Proxy та Tinyproxy), визначення їхніх переваг та недоліків у різних умовах. Особливу увагу приділено порівнянню часу обробки запитів до популярних веб-ресурсів, що надає змогу оцінити ефективність кожного рішення в реальних сценаріях використання.

Тестовим стендом є віртуальний виділений сервер орендований на сервісі onecloudplanet.com. Сервер розташований у місті Київ. Для тестування швидкодії використано програму FOGLDN Proxy Tester, запущену локально на персональному комп'ютері у місті Дніпро.

При тестуванні перевіряється підключення до сторінок: https://www.google.com/, https://www.youtube.com/, https://github.com/, https://x.com/home і https://stackoverflow.com/. До кожної сторінки зроблено 100 ітерацій запитів, по черзі через кожне із досліджуваних рішень. В результатах розглядається середня затримка в мілісекундах.

Таблиця 1

Результати тестування в однопоточному режимі

	Проксі-сервер на Node.js	Squid Proxy	Tinyproxy
https://www.google.com/	127	126	186
https://www.youtube.com/	211	206	270
https://github.com/	164	157	224
https://x.com/home	210	203	307
https://stackoverflow.com/	243	236	282

Таблиця 2

Результати тестування в багатопоточному режимі

	Проксі-сервер на Node.js	Squid Proxy	Tinyproxy
https://www.google.com/	147	169	253
https://www.youtube.com/	226	241	312
https://github.com/	184	199	267
https://x.com/home	224	216	355
https://stackoverflow.com/	251	260	288

В одпоточному режимі (таблиця 1) середній показник проксі-сервера на Node.js 191 мс, Squid Proxy 186 мс, а Tinyproxy 254 мс. В багатопоточному режимі (таблиця 2) середній показник проксі-сервера на Node.js 206 мс, Squid Proxy 217 мс, а Tinyproxy 295мс. Таким чином в однопоточному режимі проксі-сервера на Node.js на 2.69% повільніше ніж Squid Proxy і на 32.98% швидше ніж Tinyproxy, а в багатопоточному режимі на 5.34% швидше ніж Squid Proxy і на 43.20% ніж Tinyproxy.

Дослідження показало, що у режимі обробки послідовних запитів Squid Proxy демонструє найкращі результати завдяки своїй оптимізованій архітектурі, тоді як проксі-

сервер на Node.js лише трохи поступається йому за швидкістю, випереджаючи Tinyproxy із значним відривом.

У багатопоточному режимі проксі-сервер на Node.js виявився найбільш ефективним, випередивши Squid Proxy, що свідчить про його здатність обробляти високі навантаження та виконувати одночасні запити швидше. Tinyproxy, натомість, демонструє найнижчу продуктивність в обох режимах, що обмежує його застосування у сценаріях з підвищеними вимогами до швидкодії.

Аналіз показників також підкреслює перспективність застосування Node.js для створення проксі-серверів у сценаріях, де важливий баланс між продуктивністю та простотою реалізації.

Висновки. Удосконалений підхід до створення проксі-серверів, запропонований у рамках цієї роботи, сприяє значному зниженню витрат часу і матеріальних ресурсів завдяки впровадженню сучасних технологій. Використання платформи Node.js дозволяє забезпечити високу продуктивність та асинхронну обробку запитів, що робить розроблений програмний пакет ефективним і гнучким у застосуванні для різних сценаріїв. Крім того, вдосконалений підхід забезпечує підвищення якості створених проксі-серверів за рахунок модульної архітектури, що дозволяє легко масштабувати систему, адаптувати її під змінні навантаження та інтегрувати додаткові функції, такі як фільтрація трафіку, кешування або шифрування. Подальший розвиток запропонованих досліджень слід реалізовувати в напрямку розширення методів авторизації запитів, додавання можливості створення чорних і білих списків, більшої кількості способів обмеження доступу (наприклад обмеження по обсягу або IP-доступу).

ЛІТЕРАТУРА

1. Blocking vs Non-Blocking in NodeJS – What’s the Difference? [Електронний ресурс] – Режим доступу до ресурсу: https://cloudinfrastructureservices.co.uk/blocking-vs-non-blocking-in-nodejs/

2. Тренди у розвитку проксі-технологій та їхнє майбутнє [Електронний ресурс] – Режим доступу до ресурсу: https://go-proxy.com/ua/trendi-u-rozvitku-proksi-tehnologij-ta-yihnye-majbutnye

3. Exploring Event-Driven Architecture: A Beginner's Guide for Cloud Native Developers [Електронний ресурс] – Режим доступу до ресурсу: https://wso2.com/blogs/thesource/exploring-event-driven-architecture-a-beginners-guide-for-cloud-native-developers/

4. Service proxy, service mesh or API gateway – which do you need? [Електронний ресурс] – Режим доступу до ресурсу: https://tyk.io/blog/res-service-proxy-service-mesh-or-api-gateway-which-do-you-need/

5. The Role of Proxy Servers in Web Security [Електронний ресурс]. – Режим доступу: https://cloudinfrastructureservices.co.uk/the-role-of-proxy-servers-in-web-security/

6. NodeJS API-Part 5 / Model/Router/Controller structure [Електронний ресурс] – Режим доступу: https://soonsantos.medium.com/nodejs-api-part-5-model-router-controller-structure-c5b13c2660ae.

ISSN 1562-9945 (Print)
ISSN 2707-7977 (Online)

REFERENCES

1. Blocking vs Non-Blocking in NodeJS – What's the Difference? [Elektronnyi resurs] – Rezhym dostupu do resursu: https://cloudinfrastructureservices.co.uk/blocking-vs-non-blocking-in-nodejs/
2. Trendy u rozvytku proksi-tekhnolohii ta yikhnie maibutnie [Elektronnyi resurs] – Re-zhym dostupu do resursu: https://go-proxy.com/ua/trendi-u-rozvitku-proksi-tehnologij-ta-yihnye-majbutnye
3. Exploring Event-Driven Architecture: A Beginner's Guide for Cloud Native Developers [Elektronnyi resurs] – Rezhym dostupu do resursu: https://wso2.com/blogs/thesource/exploring-event-driven-architecture-a-beginners-guide-for-cloud-native-developers/
4. Service proxy, service mesh or API gateway – which do you need? [Elektronnyi resurs] – Rezhym dostupu do resursu: https://tyk.io/blog/res-service-proxy-service-mesh-or-api-gateway-which-do-you-need/
5. The Role of Proxy Servers in Web Security [Elektronnyi resurs]. – Rezhym dostupu: https://cloudinfrastructureservices.co.uk/the-role-of-proxy-servers-in-web-security/
6. NodeJS API-Part 5 / Model/Router/Controller structure [Elektronnyi resurs] – Rezhym dostupu: https://soonsantos.medium.com/nodejs-api-part-5-model-router-controller-structure-c5b13c2660ae.

Received 05.02.2025.
Accepted 07.02.2025.

Exploring the possibilities of the node.js platform in the development of proxy servers, taking into account modern requirements for performance and stability

Proxy servers play a key role in modern information systems, providing efficient, secure and scalable operation of networks. Their use allows you to optimize traffic, increase productivity, ensure confidentiality and stability in various environments from corporate networks to personal users. In the field of proxy server development, asynchronous platforms have gained significant popularity and importance. They can significantly increase productivity, improve scalability and ensure smooth servicing of a large number of simultaneous connections. The paper examines the capabilities of the Node.js platform for developing a proxy server using an asynchronous approach and the implementation of independent subservices, in particular, the process of creating your own proxy server implementation is described, including architectural solutions and the modules used. Performance testing was conducted in the FOGLDN Proxy Tester program for the proposed solution, the results of which were compared with the indicators of other popular implementations of Squid Proxy and Tinyproxy. Based on the obtained data, an analysis of the proposed proxy server efficiency was performed, which emphasized the prospects of using Node.js to create proxy servers in scenarios where the balance between performance and ease of implementation is important. The JavaScript programming language, Node.js platform, SQLite database, ORM Sequelize, PM2 process manager, and React library were used to implement the software package.

Спірінцев В'ячеслав Васильович - к.т.н., доцент, доцент кафедри програмного забезпечення комп'ютерних систем НТУ "Дніпровська політехніка".
Спірінцева Ольга Володимирівна – к.т.н., доцент, доцент кафедри електронних обчислювальних машин Дніпровського національного університету імені Олеся Гончара.
Генчук Олексій Станіславович – магістр, НТУ "Дніпровська політехніка".

Spirintsev Viacheslav Vasylovych - candidate of technical sciences, associate professor, associate professor of Computer System's Software Department of Dnipro University of Technology.
Spirintseva Olga Volodymyrivna - candidate of technical sciences, associate professor, associate professor of Computer Department of Oles Honchar Dnipro National University.
Henchuk Oleksii Stanislavovych – master, Dnipro University of Technology.

ISSN 1562-9945 (Print)
ISSN 2707-7977 (Online)