

Детектор Плагиата v. 2762 - Отчёт оригинальности: 20.01.2025 20:06:56

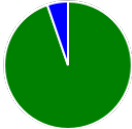
Проанализированный документ: **маг_робота_Острецов_Д.И._v.1.0.4.docx** Лицензия: ВОЛОДИМИР МАТИЄВСЬКИЙ

Тип поиска: Дословный поиск Язык: **UK**
Тип проверки: **Интернет**
ТЕЕ и кодировка: **DocX n/a**

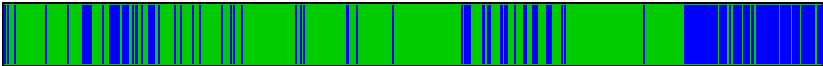
Детальный анализ тела документа:

Диаграмма соотношения частей:

Плагиат 0% Оригинал 94.75%
Кавычки 5.25% ИИ 0%



Граф распределения зон:



Источники плагиата: **3**

	→ 0,1%	21	1. https://www.education.ua/news/2024/10/25/inzheneriia-prohramnoho-zabezpechennia-use-pro-spetsialnist-121/
	→ 0,1%	11	2. https://vstup.tntu.edu.ua/speciality/121-inzheneriia-prohramnoho-zabezpechennia.html
	→ 0,1%	8	3. https://web-crawler.plagiarism-detector.com/get-doc-pt?did=9EiIP4y7-zZ8mg

Детали обработанных ресурсов: **135 - ОК / 1 - Ошибок**

Важные замечания:

Википедия:	Google Книги:	Сервисы платных работ:	Античит:
[не обнаружено]	[не обнаружено]	[не обнаружено]	Обнаружено сокрытие!

Античит-отчет UACE:

1. Статус: Анализатор Включен Нормализатор Включен сходство символов установлено на 100%
2. Обнаруженный процент загрязнения UniCode: 21,9% с лимитом: 4%
3. Процент нераспознанных символов после нормализации: 13,2%
4. Все подозрительные символы будут отмечены фиолетовым цветом: Abcd...
5. Найдены невидимые символы: 0
Рекомендации по оценке: Особое внимание следует уделить анализу этого отчета! Предполагается, что этот документ содержит значительное количество символов, чуждых языку документа. Это прямое указание на то, что автор документа использовал специальное программное обеспечение\онлайн-веб-сервис, чтобы эффективно скрыть текст в попытке избежать обнаружения потенциального плагиата. Настоятельно рекомендуется передать это дело на более высокий уровень! В случае сомнений обращайтесь: в службу поддержки Детектора плагиата!
Алфавитная статистика и анализ символов:

Активные ссылки (URL-адреса, извлеченные из документа):

URL не найдены

Исключённые ресурсы:

URL не найдены

Включённые ресурсы:

URL не найдены

Детальный анализ документа:	
МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ ДЕРЖАВНИЙ ЗАКЛАД	
Цитирования: 0,02%	id: 1
«ЛУГАНСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ ІМЕНІ ТАРАСА ШЕВЧЕНКА»	
Навчально-науковий інститут математики та інформаційних технологій Кафедра інформаційних технологій та систем Острецов Дмитро Іванович ДОСЛІДЖЕННЯ ТЕХНОЛОГІЙ РОЗРОБКИ ВЕБДОДАТКІВ ІЗ ЗАСТОСУВАННЯМ MEAN СТЕКУ кваліфікаційна робота здобувача вищої освіти другого (магістерського) рівня освітньої програми	
Цитирования: 0,01%	id: 2
«Мультимедійні системи»	
за спеціальністю 121	
Цитирования: 0,01%	id: 3
„Інженерія програмного забезпечення“	
Особистий підпис _____ Дмитро ОСТРЕЦОВ Науковий керівник _____ Світлана ПЕРЕЯСЛАВСЬКА, кандидат педагогічних наук, доцент кафедри інформаційних технологій та систем Завідувач кафедри _____ Микола СЕМЕНОВ, кандидат педагогічних наук, доцент кафедри інформаційних технологій та систем Полтава – 2025 АНОТАЦІЯ Острецов Д.І. Тема: Дослідження технологій розробки вебдодатків із застосуванням MEAN стеку. Спеціальність: 121	
Цитирования: 0,01%	id: 4
«Інженерія програмного забезпечення».	
Установка: ДЗ ЛНУ імені Тараса Шевченка, 2025р. Магістерська робота містить: 123 с., 35 рис., 4 табл., 1 додат., 59 джерел. Об'єкт дослідження – технології розробки вебдодатків. Предмет дослідження – швидкодія платформи Node у контексті розробки вебдодатків з використанням стеку технологій MEAN. Мета роботи – оцінити ефективність та продуктивність платформи Node при розробці вебдодатків з використанням технологій MEAN. Методи дослідження. Загальнонаукові: аналіз генези технологій веброботки, аналіз компонентів MEAN стеку, синтез інформації про генезу вебдодатків, синтез результатів порівняльного аналізу серверних платформ, експеримент, узагальнення емпіричних даних, отриманих в результаті експерименту, порівняння, моделювання тестового вебдодатку; методи інформаційного пошуку: бібліографічний пошук та аналіз наукових та спеціалізованих джерел інформації, реферування та цитування джерел. Результати роботи. Досліджено технології розробки вебдодатків MEAN стеку. Проаналізовано ключові особливості платформи Bun та її потенціал для розробки вебдодатків на MEAN стеку. Спроектовано методику тестування швидкодії вебдодатку на платформах Node та Bun у хмарі. Проведено експериментальне дослідження, яке довело, що вебдодаток стеку MEAN, перенесений на платформу Bun, демонструє значний приріст продуктивності порівняно з його роботою на платформі Node.js. Ключові слова: вебдодаток, MEAN, Node, Bun. ABSTRACT Ostretsov D.I. Topic: Research of web application development technologies using the MEAN stack. Specialty: 121	
Цитирования: 0,01%	id: 5
“Software Engineering”.	
Institution: Taras Shevchenko Luhansk National University, 2025. Master’s thesis contains: 123 pages, 35 figures, 4 tables, 1 appendix, 59 sources. Object of research: Web application development technologies. Subject of research: The performance of the Node platform in the context of web application development using the MEAN technology stack. Purpose of the work: To evaluate the efficiency and productivity of the Node platform in developing web applications using MEAN technologies. Research methods. General scientific: analysis of the genesis of web development technologies, analysis of the components of the MEAN stack, synthesis of information about the genesis of web applications, synthesis of the results of comparative analysis of server platforms, experiment, generalization of empirical data obtained as a result of the experiment, comparison, modeling of a test web application; information search methods: bibliographic search and analysis of scientific and specialized sources of information, referencing and citation of sources. Results of the work. The technologies of developing MEAN stack web applications were investigated. The key features of the Bun platform and its potential for developing web applications on the MEAN stack were analyzed. A methodology for testing the performance of a web application on Node and Bun platforms in the cloud was designed. An experimental study was conducted, which proved that the MEAN stack web application, ported to the Bun platform, demonstrates a significant increase in performance compared to its operation on the Node.js platform. Keywords: web application, MEAN, Node, Bun. ЗМІСТ ВСТУП 7 РОЗДІЛ 1. ТЕОРЕТИЧНІ ОСНОВИ ТЕХНОЛОГІЙ РОЗРОБКИ ВЕБДОДАТКІВ11 1.1. Поняття вебдодатків та їх роль у сучасному суспільстві13 1.2. Генеза технологій створення вебдодатків19 1.3. MEAN стек: MongoDB, Express.js, Angular, Node.js – загальний огляд технологій29 1.3.1. MongoDB: особливості зберігання даних та взаємодія з БД32 1.3.2. Express.js: створення сервера та обробка HTTP-запитів35 1.3.3. Angular: клієнтська частина вебдодатку38 1.3.4. Node.js: серверне середовище виконання JavaScript та його застосування у веброботці40 Висновки до розділу 144 РОЗДІЛ 2. ЕКСПЕРИМЕНТАЛЬНА ОЦІНКА ШВИДКОДІЇ СЕРВЕРНИХ ПЛАТФОРМ У КОНТЕКСТІ MEAN СТЕКУ46 2.1. Bun: ключові особливості та потенціал для MEAN стеку47 2.2. Методологія дослідження швидкодії платформ в контексті MEAN49 2.2.1. Інструменти тестування та критерії оцінювання швидкодії50 2.2.2. Конфігурація середовища експерименту51 2.3. Розробка тестового вебдодатку на MEAN стеку55 2.3.1. Архітектура та компоненти тестового додатку56 2.3.2. Програмна реалізація функціоналу додатку60 2.4. Експериментальне дослідження швидкодії65 2.4.1. Підготовка тестового сценарію та процедура його виконання65 2.4.2. Аналіз результатів дослідження швидкодії79 2.5. Bun в контексті MEAN стеку: перспективи та можливості80 Висновки до розділу 282 ВИСНОВКИ85 СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ87 ДОДАТКИ94 Додаток А. Вихідний код додатку94 ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, СКОРОЧЕНЬ БД – База даних AJAX – Asynchronous JavaScript and XML API – Application programming interface AWS – Amazon Web Services CERN – Conseil européen pour la Recherche nucléaire CGI – Common Gateway Interface CMS – Content management system GCP – Google Cloud Platform HTML – Hyper Text Markup Language HTTP – Hyper Text Transfer Protocol IE – Microsoft Internet Explorer IIS – Microsoft Internet Information Services JSON – JavaScript Object Notation MEAN – MongoDB, Express, Angular, and Node NCSA – National Center for Supercomputing Applications NPM – Node Package Manager OSI – The Open Systems Interconnection model PWA – Progressive Web Application REST – Representational State Transfer RPS – Requests Per Second SPA – Single Page Application SQL – Structured Query Language URL – Uniform Resource Locator VM – Virtual Machine W3C – World Wide Web Consortium ВСТУП Актуальність дослідження. Сучасний етап розвитку інформаційних технологій характеризується стрімким зростанням обсягу даних	

та вимог до швидкості доступу до інформації. В цьому контексті технології створення вебдодатків набувають особливої ваги, адже переважна більшість інформаційних систем реалізується саме у вигляді вебдодатків. Дослідження цих технологій безпосередньо корелює з пріоритетними напрямками наукових досліджень і науково-технічних розробок, визначеними постановою Кабінету Міністрів України № 476 від 30 квітня 2024 р., зокрема, такими напрямками як: технологічні засоби та сервіси програмного інжинірингу, інтелектуальні інтерактивні інформаційно-аналітичні системи та національні інформаційні ресурси. Одним із популярних наборів технологій для створення вебдодатків сьогодні є стек [MEAN](#), який надає комплексний підхід до розробки, забезпечуючи гнучкість, продуктивність та масштабованість. У вересні 2023 року світ побачила нова технологія під назвою [Bun](#), яка претендує на заміну платформи [Node.js](#), що широко використовується в сучасній веброзробці, в тому числі і в стеку [MEAN](#). Поява [Bun](#) ставить перед науковою спільнотою завдання детально дослідити перспективи її використання в контексті [MEAN](#) стеку. Дослідженню вебдодатків присвячені роботи таких вітчизняних науковців, як В. Зосімов, О. Войтюк, Д. Плечистий, О. Кошова, В. Федько, Б. Безуглий, Г. Півняк, С. Майстренко, С. Ужейко, Н. Матвеева, В. Нусс та ін. Вагомий також доробок західних дослідників, зокрема [D. Brooks](#), [L. Baresi](#), [A. Belfrage](#), [S. Kuras](#), [F. Ahmod](#), [A. Hoffman](#), [S. Fowler](#), [V. Stanwick](#), [E. Brown](#), [G. Kappel](#) та багатьох інших. Однак наявні в даний час дослідження, зважаючи на нещодавню появу [Bun](#), не заповнюють нестачу знань про особливості розробки вебдодатків із використанням нової технології. Виникла необхідність комплексного дослідження технологій розробки вебдодатків із застосуванням [MEAN](#) стеку в контексті перспектив використання у розробці додатків можливостей платформи [Bun](#). Актуальність обраної проблеми, недостатній рівень її теоретичного обґрунтування та практичної розробленості, недостатність комплексного наукового аналізу зумовили вибір теми магістерської роботи:

Цитування: 0,03%

id: 6

«Дослідження технологій розробки вебдодатків із застосуванням [MEAN](#) стеку».

Об'єктом дослідження є технології розробки вебдодатків. Предмет дослідження – швидкодія платформи [Node](#) у контексті розробки вебдодатків з використанням стеку технологій [MEAN](#). Мета – оцінити ефективність та продуктивність платформи [Node](#) при розробці вебдодатків з використанням технологій [MEAN](#). Поставлена мета передбачає вирішення наступних завдань: провести аналіз наукової літератури та узагальнити існуючу інформацію щодо еволюції вебтехнологій, архітектури вебдодатків, особливостей стеку технологій [MEAN](#) дослідити ключові особливості платформи [Bun](#) та її потенціал для розробки вебдодатків на [MEAN](#) стеку розробити тестовий вебдодаток на [MEAN](#) стеку, який дозволить провести експериментальне порівняння швидкодії платформ [Node.js](#) та [Bun](#) спроектувати методику тестування швидкодії вебдодатку на обох платформах провести експериментальне дослідження швидкодії вебдодатку на платформах [Node.js](#) та [Bun](#) проаналізувати отримані результати тестування, визначити переваги та недоліки кожної платформи сформулювати висновки про доцільність та перспективи застосування платформи [Bun](#) в контексті розробки вебдодатків на [MEAN](#) стеку. Методи дослідження. Наукові положення магістерської роботи базуються на сукупності фундаментальних загальнонаукових та спеціальних методів та підходів, необхідних для розкриття теми: аналіз генези технологій веброзробки, архітектури вебдодатків, компонентів [MEAN](#) стеку, аналіз результатів тестування; синтез інформації про генезу вебдодатків, результатів порівняльного аналізу [Node](#) та [Bun](#); узагальнення емпіричних даних, отриманих в результаті експерименту, для формулювання загальних висновків; на основі результатів тестування швидкодії [Bun](#) зроблені висновки про його перспективність для веброзробки; порівняння різних об'єктів, явищ, процесів для виявлення їх подібності та відмінності, зокрема [MariaDB](#) та [MongoDB](#), [Node.js](#) та [Bun](#); моделювання, тобто створення спрощеного образу реального об'єкту, зокрема розробка тестового вебдодатку для дослідження швидкодії платформи [Bun](#); бібліографічний пошук та аналіз наукових, спеціалізованих та популярних джерел інформації, що стосуються теми дослідження; реферування та цитування джерел інформації; тестування швидкодії з використанням спеціалізованого програмного забезпечення ([Bombardier](#)) для визначення продуктивності вебдодатку на різних платформах; аналіз даних; використання таблиць, діаграм та графіків для візуалізації та аналізу результатів експерименту. Наукова новизна дослідження полягає в комплексному вивченні перспектив технології [Bun](#) в контексті розробки вебдодатків [MEAN](#) стеку та експериментальній оцінці її можливостей. Вперше проведено порівняльний аналіз швидкодії вебдодатку [MEAN](#) стеку на платформах [Node.js](#) та [Bun](#), що дозволило не тільки виявити потенціал [Bun](#) для підвищення продуктивності, але й встановити наявність певних обмежень в сумісності. В рамках дослідження нами вперше запропоновано оригінальну модель зрілості вебдодатків, що відображає еволюцію технологій їх розробки, починаючи від статичних [HTML](#) сторінок до сучасних динамічних застосунків. Ця модель сприяє глибокому розумінню ключових етапів розвитку вебдодатків та дозволяє чіткіше окреслити відмінності між вебсайтом та вебдодатком, що досі є предметом дискусій в науковій спільноті. Практичним результатом дослідження є встановлення факту, що, незважаючи на заявлений розробниками [Bun](#) принцип повної зворотної сумісності з [Node.js](#), на практиці існують певні обмеження, які потребують додаткових зусиль для адаптації існуючих проєктів. Виявлені в процесі дослідження проблеми сумісності посприяють подальшому вдосконаленню та розширенню екосистеми платформи [Bun](#). Експериментальне підтвердження можливості суттєвого підвищення продуктивності вебдодатків [MEAN](#) стеку при використанні платформи [Bun](#) відкриває нові перспективи для розробників. Нова комбінація стеку [MEAB](#) ([MongoDB](#), [Express.js](#), [Angular](#), [Bun](#)) може стати привабливою альтернативою для створення високонавантажених вебдодатків, що потребують максимальної швидкодії та ефективності. Окремі результати магістерського дослідження обговорювались на всеукраїнських та міжнародних науково-практичних конференціях [7; 8; 9].

ТЕОРЕТИЧНІ ОСНОВИ ТЕХНОЛОГІЙ РОЗРОБКИ ВЕБДОДАТКІВ Всесвітня мережа, більш відома як мережа

Цитування: 0%

id: 7

«Інтернет»,

зростає впливаючи на кожен аспект людського існування. За більше ніж 30 років з моменту публічної появи інтернету, наше життя, побут та робота відчутно змінилися. Наразі інтернет – це домінуюча платформа для комунікації, розгортання бізнес-проєктів, впровадження соціальних програм та найрізноманітніших організаційно-інформаційних систем. Безліч підприємств розширили і продовжують розширювати обсяг і функціональні можливості своїх вебсайтів для величезної кількості користувачів, щоб задовольнити їх самі різні потреби. Станом на жовтень 2023 року у світі налічувалося 5,3 мільярда користувачів Інтернету, що становило 65,7 відсотка світового населення. З них 4,95 мільярда, або 61,4 відсотка населення світу, були користувачами соціальних мереж [41]. Перше місце серед регіонів світу за часткою населення, що користувалося інтернетом у 2023 році посіла Північна Європа. Станом на квітень 2023 року в Норвегії, Саудівській Аравії та Об'єднаних Арабських Еміратах 99 відсотків населення користувалося інтернетом. Північна Корея

навіпаки, живе практично без проникнення інтернету, посідаючи останнє місце у світі. Китай, Індія та США випереджають інші країни світу за кількістю користувачів Інтернету. За статистикою 92 відсотки населення країн з високим рівнем доходу користувалися Інтернетом, на відміну від лише 26 відсотків населення держав з низьким рівнем доходу. В Україні очікувалося, що рівень проникнення інтернету досягне 90% у 2022 і 2023 роках [48]. Показник постійно зростав і у 2021 році склав 86%. З початку повномасштабного вторгнення в Україні спостерігається стрімке зростання частки мобільного трафіку порівняно з використанням десктопних комп'ютерів. У 2023 році 80% трафіку в Україні припадало на смартфони. 94% українців мають досвід онлайн-шопінгу, а 14% зробили першу онлайн-покупку після повномасштабного вторгнення. Також в нашій країні, як і раніше спостерігається підвищена увага до новин: 98% людей читають новини щодня. Поєднуючи мільярди людей у всьому світі, інтернет наразі є основою сучасного інформаційного суспільства. Фактично інтернет став однією з найбільш важливих та найвпливовіших подій не лише в історії обчислювальної техніки але й в історії людства. Такі сайти, як [google.com](https://www.google.com), [amazon.com](https://www.amazon.com), [youtube.com](https://www.youtube.com) – змінили світ (рис 1.1). Найбільш відвідувані у 2023р. вебсайти [37] За даними журналу [forbes](https://www.forbes.com) у 2023 році в Інтернеті було близько 1,13 мільярда вебсайтів, але тільки 18% з них активно використовується та оновлюється [37]. Тим не менш кожні три секунди в світі створюється новий вебсайт. Наша залежність від мережі різко зросла. Швидкість, надійність, якість, зручність, доступність звичайних вебсторінок стали надзвичайно важливими, адже більшість вебсайтів сьогодні – це складні системи, тісно інтегровані з іншими інформаційними системами, базами даних, системами обробки транзакцій. Поняття вебдодатків та їх роль у сучасному суспільстві Алан Кей, творець ООП, казав, що

Цитування: 0,04% id: 8
«технологія – це те, чого не було, коли ви народилися».

Цікаво, що поняття

Цитування: 0% id: 9
«вебсайт»

не використовувалось в жодному акті законодавства України аж до 2017 року, а термін

Цитування: 0% id: 10
«вебдодаток»

не згадується там і на цей час. Наразі закон визначає

Цитування: 0% id: 11
«веб-сайт»,

як

Цитування: 0,12% id: 12
«сукупність даних, електронної (цифрової) інформації, об'єктів авторського права та (або) суміжних прав, пов'язаних між собою та структурованих у межах адреси веб-сайту, доступ до яких здійснюється через Інтернет» [10]

. В тому ж законі

Цитування: 0% id: 13
«веб-сторінка»

визначається як

Цитування: 0,04% id: 14
«складова частина веб-сайту, розташована за спеціальною адресою в мережі Інтернет».

В 2019 році нова редакція Українського правопису уточнила правила написання деяких слів серед яких і терміни, що починалися з компонента

Цитування: 0% id: 15
«веб»

– тепер такі слова як

Цитування: 0% id: 16
«вебсайт»,

Цитування: 0% id: 17
«вебресурс»,

Цитування: 0% id: 18
«вебзастосунок»,

Цитування: 0% id: 19
«вебсторінка»,

Цитування: 0% id: 20
«вебдодаток»

слід писати без дефісу [17]. Міністерства, хоча й повільно, проте все ж таки проводять узгодження існуючих законів та актів з новим правописом [11]. З цих причин в українському науковому полі спостерігаються тимчасові розбіжності, і, як наслідок цього, досліджуючи тему

Цитування: 0% id: 21
«веб»,

вітчизняні науковці стикаються з певними подвоєннями в українській термінології.

Міжнародна наукова спільнота, яка наразі є джерелом сучасних ідей та технологічним флагоманом, користується терміном

Цитування: 0,01% id: 22
«web application».

Слово













Цитування: 0% id: 23
«application»















зазвичай перекладалося українською як

Цитування: 0% id: 24
«програма»,

або ж використовувалось слово

Цитування: 0% id: 25
«додаток»

чи	
 Цитування: 0%	id: 26
«застосунок».	
Після появи терміну	
 Цитування: 0,01%	id: 27
« web application »	
виник і його український переклад –	
 Цитування: 0%	id: 28
«вебдодаток»,	
або	
 Цитування: 0,01%	id: 29
«вебзастосунок» [14]	
<p>Таким чином, досліджуючи вебдодатки, також необхідно брати до уваги весь спектр синонімів цього терміну, та їх попередню граматичну форму. В наш час дослідження вебдодатків все більше привертає увагу як закордонних так і вітчизняних науковців. Порівняльному аналізу продуктивності фреймворків та бібліотек для створення вебдодатків присвячені дослідження S. Skrzypiec, M. Plechawska-Wójcik [52], K. Bielak, B. Borek [26]. Архітектуру вебдодатків створених за допомогою фреймворку Angular детально розглянуто V. Garcia. Автор узагальнює свій практичний досвід розробника та пропонує цікаві рішення в процесі створення трьох повних програм [36]. Створенню прогресивних вебдодатків присвячені дослідження C. Rojas [51], Z. Tahir, A. Iham, M. Niswar, A. Fauzy [50], B. Susanto, B. Virginia, G. Proboyekti, U. Ester [49]. Робота M. Baker присвячена актуальній проблемі захисту вебдодатків від кібератак. Основна увага автора зосереджена на шляхах, якими хакери атакують вебдодатки та опису широкого арсеналу засобів захисту [20]. Д. Бартлетт – розробник програмного забезпечення та дослідник в своїй роботі приділяє увагу створенню вебдодатків на основі PHP та описує кілька хмарних рішень, що допоможуть прискорити роботу вебдодатку [22]. Впровадженню машинного навчання, для реалізації потенціалу штучного інтелекту присвячена робота V. Karunanidhi. Автор створив систему покрокових інструкцій процесу розгортання моделі машинного навчання у вебдодатку. В роботі дослідник зокрема використовує TensorFlow – бібліотеку JavaScript для розгортання моделей машинного навчання [45]. У статті М. Слабіноги та С. Чабан основна увага авторів акцентована на розробці рекомендацій щодо проектування вебдодатків з максимально швидким відображенням інтерфейсу користувача та розробці вебдодатку на основі цих рекомендацій. Дослідники проаналізували предметну область, та підходи до проектування вебдодатків, сформулювали критерії їх ефективності. Вдосконалені авторами методи проектування вебдодатків програмного забезпечення з урахуванням рекомендацій щодо підвищення продуктивності вебдодатку дозволяють досягти вищої швидкодії [13]. В. Зосімов досліджував методи видобування даних, моделі та методи створення мов обробки даних для корпоративних вебресурсів та інтеграції семантичної розмітки в HTML-код. Автор використовує термін</p>	
 Цитування: 0%	id: 30
«веб-ресурс»	
<p>в значенні вебсайту або вебдодатку. Після ретельного аналізу інформаційного вмісту та елементів навігації 1000 корпоративних вебресурсів українського сегменту інтернету дослідник створив онтологію семантичних класів та властивостей для опису їх інформаційного наповнення, розробив абстрактну модель системи комплексного оперування даними в мережі Інтернет та на її основі створив програмний комплекс – систему комплексного оперування даними в мережі Інтернет [3]. О. Войтюк та Д. Плечистий розкривають в своїй роботі процеси оптимізації промальовування вебзастосунків з використанням об'єктів з глибокою вкладеністю. Автори за допомогою браузерних інструментів дослідили проблеми ефективності при промальовуванні вебдодатка [2]. Стаття О. Кошової та ін. присвячена комплексному дослідженню особливостей розробки вебдодатків для системи дистанційного навчання з допомогою бібліотеки React. Авторкою висвітлено переваги та особливості використання бібліотеки React в розробці вебдодатків [6]. Проблема підбору конструктивних елементів для створення вебдодатків засобами CMS систем розглядали В. Федько та Б. Безуглий. Автори виявили основні недоліки технології використання CMS систем під час розробки та запропонували систему на основі колекції програмних хуків та віджетів, яка прискорює розробку, спрощує роботу з базою даних, допомагає швидко втілювати спільний функціонал [18]. Слід зазначити, що в українській науковій спільноті все ще дискусійним продовжує залишатися питання про визначення самого поняття</p>	
 Цитування: 0%	id: 31
«вебдодаток».	
Наприклад	
 Цитування: 0,02%	id: 32
«Тлумачний словник з інформатики»	
за авторством Г. Півняка та ін. не має визначень для	
 Цитування: 0%	id: 33
«вебдодатку»	
чи	
 Цитування: 0%	id: 34
«вебзастосунку»,	
проте він визначає поняття	
 Цитування: 0,1%	id: 35
«веб-застосування (web application) – Набір клієнтських і серверних програм, які працюють разом, щоб забезпечити рішення проблеми обробки контенту, що розташовується на веб-серверах...».	
<p>Іншим варіантом визначення цього терміну автори словника пропонують описувати вебзастосунок як набір вебсторінок, що відображаються браузером [15]. В виданні</p>	
 Цитування: 0,02%	id: 36
«Базові поняття і терміни веб-технологій»	
серед наведених основних термінів є	
 Цитування: 0%	id: 37
«веб-застосунок».	
Автори визначають його, як	

<p> Цитування: 0,13%</p> <p>«клієнт-серверний застосунок, в якому клієнтом виступає браузер, а сервером – веб-сервер. Логіка вебзастосунку розподілена між сервером і клієнтом, зберігання даних здійснюється, переважно, на сервері, обмін інформацією відбувається по мережі» [1]</p> <p>. С. Майстренко та ін. в статті присвяченій технології зберігання та візуалізації даних визначає вебдодаток, як</p>	id: 38
<p> Цитування: 0,05%</p> <p>«клієнт-серверний додаток, в якому клієнтом виступає браузер, а сервером – веб-сервер».</p> <p>Автори також приходять до висновку, що зазвичай вебдодатки мають три основні складові: сервер, клієнт та базу даних [4]. С. Ужейко вважає, що вебдодаток – це</p>	id: 39
<p> Цитування: 0,15%</p> <p>«клієнтська програма, що дає змогу вирішувати прикладні задачі користувача. На відміну від звичайного застосунку, вебзастосунок зазвичай містить серверну частину, що виконується на віддаленому вебсервері, і клієнтську частину, що виконується у браузері безпосередньо на пристрої користувача».</p> <p>Науковець акцентує увагу на незалежності вебдодатків від операційної системи та версій встановленого програмного забезпечення, що має зробити вебдодатки доступними з будь якого пристрою [16]. Н. Матвєєва та В. Нусс визначають вебдодаток, як</p>	id: 40
<p> Цитування: 0,1%</p> <p>«комплекс, який являє собою сукупність програмного забезпечення, серверу, а також системного коду. Завдяки цьому можливо розробити унікальну веб-сторінку для конкретної задачі» [5]</p> <p>. В. Щербань, описуючи складові вебтехнологій, визначає вебдодаток, як програмне забезпечення, що можна відкрити за допомогою будь-якого браузера. А відмінність вебдодатків від вебсайтів на думку науковця полягає в тому, що вони відображають не статичні дані, такі як тексти та зображення, а динамічні програми [19]. Проте, на наш погляд, вебдодатки створені на сучасних фреймворках в змозі відтворювати і лише статичні дані, і таким чином, згідно визначенню, вони будуть вебсайтами. Також в публічному полі іноді вебдодатки описуються як</p>	id: 41
<p> Цитування: 0,03%</p> <p>«динамічний вебсайт, що надає користувачам графічний інтерфейс...»,</p> <p>або</p>	id: 42
<p> Цитування: 0,06%</p> <p>«програма, що використовується як допоміжний засіб, націлений на виконання певних дій на вебсерверах»,</p> <p>або навіть</p>	id: 43
<p> Цитування: 0,08%</p> <p>«сукупність статичних і динамічних вебсторінок, тобто наперед створених сторінок і програм, що створюють вебсторінки у відповідь на звернення користувача».</p> <p>Часто вважається, що вебсайт і вебдодаток – це тотожні поняття. Інтернет-гігант, найбільша в світі платформа хмарних обчислень Amazon описує вебдодаток, як</p>	id: 44
<p> Цитування: 0,03%</p> <p>«програмне забезпечення, що працює у браузері».</p> <p>Пояснюючи природу вебдодатків Amazon зазначає, що після винайдення інтернету, вебсайти мали значно меншу функціональність, ніж сучасні вебдодатки. Вони були здатні доставляти користувачам лише статичну інформацію. Вебдодатки були створені, щоб подолати обмеження статичних сайтів. З того часу вебтехнології значно розвинулися і наразі більшість сучасних вебсайтів є складними вебдодатками за своєю архітектурою [59]. Стосовно поняття</p>	id: 45
<p> Цитування: 0%</p> <p>«вебдодаток»</p> <p>думки міжнародної спільноти також різняться. D. Brooks зазначав, що сьогодні термін</p>	id: 46
<p> Цитування: 0%</p> <p>«вебдодаток»</p> <p>став загальним поняттям, а важливою частиною більшості вебсайтів є інтерактивність, бо вона зробила їх складнішими, але й більш цікавими для розробки [28]. L. Baresi досліджуючи проблеми концептуального проектування вебдодатків, пропонує розглядати їх як розширення традиційної інформаційної системи, яка доповнена навігацією та складними інформаційними структурами. Або як розширення традиційних вебсайтів, що доповнені різними видами операцій [21]. A. Belfrage вважав, що вебдодаток – це вебсайт, який сфокусований на якусь задачу, наприклад онлайн-каталог, інтернет-банк, чи вебсистема навігації [23]. A. Hoffman аналізуючи архітектуру, безпеку та засоби захисту сучасних вебдодатків від хакерів визначає вебдодаток як програму, що завантажується та запускається в браузері. На думку автора вебдодатки відрізняються від традиційних вебсайтів тим, що мають багаторівневу систему дозволів, зберігають введені користувачем дані в базах даних і часто дозволяють користувачам обмінюватися контентом один з одним [38]. S. Fowler та V. Stanwick намагаючись у своїй книзі відповісти на питання</p>	id: 47
<p> Цитування: 0,04%</p> <p>«що ж таке вебдодаток і чим він відрізняється від сайту»</p> <p>приходять до висновку, що різниця між цими поняттями дуже розмита і слід сприймати вебсайти і вебдодатки як континуум без чітких ознак відмінностей [35]. В роботі, присвяченій веброботці на Node та Express, E. Brown зазначає, що</p>	id: 48
<p> Цитування: 0,03%</p> <p>«сайт – це вебдодаток і вебсторінка це вебдодаток».</p> <p>Автор вважає, що взагалі слово</p>	id: 49
<p> Цитування: 0%</p> <p>«додаток»</p> <p>використовується для позначення чогось, в чому є функціональність, чогось, що не просто статичний набір контенту, хоча і статичний контент автор називає дуже простим прикладом вебдодатків. Дослідник також прогнозує, що за кілька років зовсім не буде відмінностей між вебдодатком та вебсайтом [29]. Таким чином, проаналізувавши наявну наукову літературу можна констатувати, що термін</p>	id: 50
<p> Цитування: 0%</p> <p>«вебдодаток»</p>	id: 51

все ще не має остаточного визначення як в українській так і в міжнародній спільноті. На наш погляд авторкою найбільш вдалого, найбільш узагальнюючого формулювання цього поняття є професорка Інституту інформаційних систем Віденського технічного університету [G. Kappel](#). За її словами вебдодаток – це програмна система, створена на базі технологій і стандартів [World Wide Web Consortium \(W3C\)](#), яка надає через веббраузер специфічні вебресурси, такі як контент і сервіси [44]. З цим формулюванням важко сперечатися, бо саме [W3C](#) є першоджерелом всіх технологій та стандартів на яких базується інтернет. І саме ця організація забезпечує довгострокове зростання інтернету, шляхом розробки більш прогресивних протоколів і правил. Проте слід зазначити, що на сьогодні під це визначення підпадає і поняття

Цитирования: 0%id: 52

«вебсайт»,

адже це теж система, що заснована на технологіях і стандартах [W3C](#). З'ясування відмінностей вебсайту від вебдодатку – іноді нетривіальна задача. Наразі точно можна сказати, що це якщо не тотожні, то дуже близькі одне одному поняття. Генеза технологій створення вебдодатків На сучасному етапі технології створення вебдодатків займають центральне місце у світі інформаційних технологій, відіграючи ключову роль у сучасній цифровій трансформації. Починаючи від простих статичних вебсторінок і прогресуючи до високопродуктивних, інтерактивних додатків, генеза технологій створення вебдодатків – це складна мозаїка інновацій, стандартів та методологій. Шляхом аналізу ключових етапів і тенденцій цієї області спробуємо розкрити основні фактори, що визначають архітектурну парадигму сучасної веброзробки і передбачити можливі напрямки її майбутнього розвитку. Вебдодатків не було б без браузера, бо вони існують саме в його рамках, тому можна вважати початком історії вебдодатків створення першого браузера. В 1989 році у найбільшій у світі фізичній лабораторії [CERN](#) у Швейцарії англійський програміст і фізик Тім Бернерс-Лі подав керівництву дві пропозиції щодо того, що в майбутньому стане мережею Інтернет [24]. Жодна не була схвалена, проте він все одно продовжує працювати і до кінця 1990 року створює прототип

Цитирования: 0%id: 53

«[WorldWideWeb](#)».

Проект Тіма містив сервер, [HTML](#), [URL](#)-адреси та перший браузер – [Nexus](#) [27]. Цей браузер функціонує і як редактор, тобто текстовий процесор підключений до Інтернету, що відображає оригінальне авторське бачення того, що інтернет за задумом також має містити інструменти для розробки. Проте слід зазначити, що ідеям Т. Бернерса-Лі передували розробки інших дослідників, наприклад Т. Нельсона і Д. Енгельбарта, які ще у 1950-х роках запропонували комп'ютеризувати концепцію перехресних посилань, створивши посилання, які ми використовуємо в Інтернеті. Саме Нельсон назвав це

Цитирования: 0%id: 54

«гіперпосиланням»,

а комп'ютеризований текст –

Цитирования: 0,01%id: 55

«гіпертекстом» [58]

. Стосовно самої глобальної мережі – в США наприкінці 1970-х років почалося повільне зростання кількості персональних комп'ютерів, і разом з цим підключення їх до перших онлайн-сервісів на кшталт дошок оголошень – [BBS](#), [Usenet](#), та мереж обміну файлами – [BITNET](#). До 1990 року понад два мільйони жителів Північної Америки вже були онлайн для різноманітних дискусійних груп, покупок, новин, спілкування, електронної пошти тощо. У 1980 році Т. Бернерс-Лі у лабораторії [CERN](#) створює

Цитирования: 0%id: 56

«[Enquire](#)»

– мережеву гіпертекстову систему, яка використовується для управління проектами, але з набагато більшими амбіціями. Він прагнув класифікувати гіперпосилання таким чином, щоб їх могли читати як комп'ютери, так і люди. Пізніше він стверджував, що на той час не знав про ранні роботи Нельсона та Енгельбарта з гіпертекстом, тому це може бути незалежне переосмислення. В той час багато хто погоджувався з ідеєю розробки глобальної мережі. Проте не було згоди в тому як саме це зробити. До початку 1980-х років кілька різних національних і корпоративних мережевих протоколів конкурували між собою. Першим офіційним стандартом з міжнародною підтримкою став [OSI \(Open Systems Interconnect\)](#). [OSI](#) офіційно опубліковано в 1984 році. За допомогою цієї моделі різні пристрої змогли нарешті взаємодіяти один з одним через мережу. Саме в таких умовах і був створений проєкт Т. Бернерса-Лі. Його ідея полягала в тому, щоб мати можливість

Цитирования: 0%id: 57

«зв'язати»

різні типи дослідницьких документів воедино. Те, що створили Бернерс-Лі та інші, поклато розкрити новому протоколу передачі гіпертексту – [HTTP](#) [40], і новій мові гіпертекстової розмітки – [HTML](#) [55]. Сьогодні ми не бачимо [HTTP](#), якщо не хочемо цього навмисно. Цей протокол абстрагувався від нас, проте, він лежить в основі наших додатків. Дізнавшись про винахід Т. Бернерса-Лі інші представники вченої спільноти по всьому світі почали використовувати Інтернет для індексування своїх наукових документів. Проте редактор браузера Т. Бернерса-Лі працював лише на рідкісних комп'ютерах [Next](#), а [CERN](#) відмовився фінансувати розробку версій для інших платформ. Тому команда пише простий, лише текстовий, браузер для швидкого розповсюдження, надає код і просить волонтерів написати або адаптувати цей браузер для широкого кола комп'ютерів. В результаті з'являються кілька браузерів, серед яких [Viola](#), [Midas](#) та [Mosaic](#), який скоро стане [Netscape Navigator](#)-ом. З того часу Т. Бернерс-Лі вже не поверне контроль над своїм творінням. Слід зазначити, що виробники браузерів, намагаючись покращити свої продукти, почали додавати власні набори тегів в [HTML](#)-розмітку. Виникла мішанина з різних [HTML](#)-документів, доступних для перегляду то в одному, то в іншому браузері. Розробникам доводилося створювати кілька варіантів [HTML](#)-документів під кожну окрему програму для перегляду сторінок. В 1993 році більш розширений стандарт [HTML+](#) [39] додає нові функції, такі як малюнки, таблиці та форми, а в 1995 році – виходить новий стандарт [HTML 2.0](#) [25]. Найбільш популярним браузером на початку 1990-х став [Mosaic](#). Він дозволяв використовувати зображення в мові розмітки. Ідея включення зображень у мову розмітки поживала сухі сторінки, які були дуже перевантажені текстом. Раніше зображення позначалися як посилання і відкривалися у власному вікні після натискання на них. М. Андрессен на базі [Mosaic](#) створив перший комерційний браузер під назвою [Netscape Navigator](#). Цей продукт швидко отримав визнання, і його використання стрімко росло. Саме для [Netscape](#) в 1995 році [B. Eich](#) створив мову [JavaScript](#), без якої наразі складно уявити собі сучасний інтернет [34]. Але [Netscape](#) подарував світу ще одну необхідну технологію, а саме – [cookies](#), невеликі блоки даних, які сервер зберігає на комп'ютері користувача для того

щоб зберігати інформацію, або відстежувати зроблені дії. Наприклад в [cookies](#) можна зберегти товари, додані в кошик для покупок в онлайн-магазині і вони залишаться в кошику коли користувач повернеться на сайт знову. [Cookies](#) з'явилися в 1994 році і вже через рік їх використовували і інші браузери. Довгий час у [cookies](#) не було альтернативи, проте згодом браузері додали для збереження стану додатків такі [API](#) як [LocalStorage](#), [SessionStorage](#), [Web SQL](#), [IndexedDB](#) та ін. В той самий час, коли [Netscape](#) вже панував на ринку, на базі коду [Mosaic](#) і компанія [Microsoft](#) створила свій перший [Internet Explorer](#). Завдяки популярності ОС [Windows](#), в яку було вбудовано браузер від [Microsoft](#) до 1998 року популярність [Netscape](#) і [IE](#) спочатку зрівнялися, а потім почалося багаторічне домінування [Microsoft](#) на ринку браузерів. Суперництво браузерів призвело до появи нестандартних можливостей у кожного окремого браузера. Найбільші відмінності виникли при підтримці [JavaScript](#) – мови сценаріїв, що надає інтерактивність документам. У результаті багато документів були

Цитирования: 0%

id: 58

«оптимізовані»

для конкретного браузера і абсолютно не відтворювалися в іншому. [W3C](#) хоч і приймав безліч детально обговорених стандартів але відповідальність за дотримання цих стандартів все ж лягала на розробників браузерів. Користувачі бажали більше динаміки, а виробники браузерів намагалися забезпечити багатший клієнтський досвід. Першими з'явилися [Java](#)-аплети – це попередньо скомпільований байткод [Java](#), який завантажується в браузер, а потім виконується. Ця технологія випереджала свій час, але з різних причин пов'язаних з безпекою та продуктивністю не стала популярною. З 1996 року [Microsoft](#) розробляла [Active Server Pages](#) – власний механізм для динамічних вебсторінок. Технологія [ASP](#) використовує сценарії на сервері для створення контенту, який відправляється у веббраузер клієнта через [HTTP](#). Ці сценарії були написані з використанням мов [VBScript](#), [JScript](#) або [PerlScript](#), а для запитів на сервер використовувалися об'єкти [MSXML 2.ServerXMLHTTP](#) та [Microsoft.XMLHTTP](#). Потім цю концепцію було реалізовано в [Internet Explorer 5](#) через використання [ActiveXObject](#)(

Цитирования: 0,01%

id: 59

"[Mxml](#) 2. [XMLHTTP](#)"

). Інші браузери для асинхронних запитів на сервер використовували [JavaScript](#) клас [XMLHttpRequest](#), який з 2002 року підтримувався більшістю з них. В 2006 році його було додано в специфікацію, а з 2012 року [XMLHttpRequest](#) почав підтримувати і [Internet Explorer](#). До появи цієї технології [HTML](#)-форму потрібно було спочатку відправити на сервер, а потім повністю оновити сторінку браузера. Новий метод спілкування вебсторінки з сервером отримав назву [AJAX](#). Паралельно з цим, з 1996 року технологія [Flash](#) стала популярним методом створення на сайтах анімації та інтерактивності. За допомогою [Flash](#) створювалися рекламні банери, відео, інтерактивні дизайн-елементи і навіть повністю вебсайти. Технологія стрімко розвивалася, проте протягом багатьох років експерти наголошували на проблемах з безпекою у [Flash](#). У 2015 році виробники браузерів [Firefox](#), [Chrome](#) і [Safari](#) внесли [Flash Player](#) у чорний список. Одним із перших вебсерверів, розроблених після сервера Т. Бернєрса-Лі, був випущений у 1993 році сервер [NCSA](#). На 1995 рік [NCSA](#) обслуговував більшість усіх вебсерверів в Інтернеті, проте майже всі вони швидко перейшли на [Apache](#). У квітні 1996 року [Apache](#) випередив [NCSA](#) і був топовим сервером в Інтернеті до середини 2016 року. Наступні три роки лідерство було за [IIS](#) від компанії [Microsoft](#), але зрештою і він втратив позиції завдяки [Nginx](#) – високопродуктивному вебсерверу з відкритим кодом. Оскільки дизайн всесвітньої павутини за своєю суттю не був динамічним, ранній гіпертекст складався з закодованих вручну текстових файлів [HTML](#), які публікувалися на вебсерверах. У 1993 році для взаємодії сайтів із вебсерверами був представлений стандарт [Common Gateway Interface \(CGI\)](#). Він вперше забезпечив вебсторінкам динамічність даних [46]. Інтерфейс дозволив вебсторінкам викликати серверні програми написані на C, [Perl](#) та інших мовах, а також обробляти надіслані зі сторінки дані. Доволі швидко з'явилися альтернативи [CGI](#). Досі використовуються [FastCGI](#), [SCGI](#), модулі [Apache](#) та ін. Перші скрипти для динамічних сайтів були написані на мовах C та [Perl](#). Після найпростіших скриптів, що обробляли відправлену з вебсторінки форму з'явилися чати, форуми та інші динамічні сайти. В 1995 році з'явилася мова [PHP](#). Вона почалася коли її автор Р. Лердорф після появи інтерфейсу [CGI](#) написав кілька програм на C для підтримки форм на своєму персональному сайті. Він казав:

Цитирования: 0,09%

id: 60

«Я не знаю, як це зупинити. Ніколи не було наміру написати мову програмування. Я абсолютно не знаю, як писати мови програмування».

На лютий 2024 року 76,5% вебсайтів використовують саме [PHP](#) (рис. 1.2). Рис. 1.2. Відсоток сайтів, що використовують ту чи іншу серверну мову [57] У 1995 році на ринку з'явилося рішення від компанії [FileNet](#) для керування документами на сервері. Саме воно вважається першим справжнім рішенням для керування контентом. Після цього з'явилися і інші корпоративні [CMS](#), найвідоміші з яких [Interwoven](#) (1995), [Documentum](#) (1996), [FatWire](#) (1996), [Future Tense](#) (1996), [Inso](#) (1996), [EPIServer](#) (1997). [M. Kunze](#) в 1998 році продемонстрував спільноті набір безкоштовного програмного забезпечення з відкритим кодом, який може бути реальною альтернативою дорогим комерційним пакетам. В цей набір входили [Linux](#) – як операційна система, [Apache](#) – як [HTTP](#) сервер, [MySQL](#) – база даних, та [PHP](#), [Perl](#), або [Python](#) в якості мови програмування. Так виник один із перших стеків технологій розробки програмного забезпечення – [LAMP](#). Згодом ця модель була адаптована до використання і інших компонентів. Наприклад змінюючи операційну систему на [Windows](#) – отримуємо стек [WAMP](#), а якщо змінити ще й сервер на [IIS](#) – [WIMP](#). Варіацій наразі існує досить багато. Стек [LAMP](#) і сьогодні активно використовується у розробці вебдодатків. Для полегшення і прискорення веброзробки з початку 2000-х років почали з'являтися так звані

Цитирования: 0%

id: 61

«фреймворки»,

комплексні рішення, що забезпечували стандартний спосіб створення та розгортання сайту в інтернеті. Серед найбільш популярних: [Laravel](#), [Ruby on Rails](#), [Symfony](#), [Spring](#), [Yii](#), [Django](#), [Flask](#). Також з'явилися перші повноцінні [CMS](#): [Drupal](#), [Wordpress](#), [Joomla](#) та ін. В 2009 році [R. Dahl](#) представив світу [Node.js](#) – середовище виконання [JavaScript](#) з відкритим вихідним кодом, яке працює в [Windows](#), [Linux](#), [Unix](#), [macOS](#) та інших системах. Ця нова платформа дозволила розробникам використовувати просту мову [JavaScript](#) не тільки у браузері а й для створення сценаріїв на стороні сервера. Невдовзі [Node.js](#) отримала менеджер пакетів під назвою [NPM](#), який дозволив програмістам оприлюднювати та вільно ділитися програмними пакетами. Платформа [Node.js](#) виявилася вкрай продуктивною. Для цього в [Node](#) використовуються три концепції: події, асинхронний [API](#) та неблокуючий ввід/вивід – це означає, що програма може звернутись із запитом до зовнішнього ресурсу та зайнятися чимось іншим, а потім, коли зовнішню асинхронну операцію буде завершено, виконується функція зворотного виклику, яка обробляє результат. Завдяки появі [Node.js](#) з'явився ще один стек технологій – [MEAN](#). [MongoDB](#) – база даних, [Express.js](#) – сервер, [Angular](#) –

фреймворк для створення користувацького інтерфейсу та [Node.js](#) – як платформа). Всі компоненти стеку [MEAN](#) підтримують код, написаний на [JavaScript](#). Таким чином тепер можна однією мовою писати як для серверних, так і для клієнтських середовищ виконання. Як і у випадку зі стеком [LAMP](#) тут теж з'явилися варіанти і з іншими компонентами. Варіант, відомий як [MERN](#), замінює [Angular](#) бібліотекою [React.js](#), а інший під назвою [MEVN](#) використовує [Vue.js](#). Автор [Node.js](#) [R. Dahl](#) розробив ще одну платформу, наступницю [Node.js](#) з назвою [Deno](#). Проте наразі вона ще не отримала визнання і потребує доопрацювання. Інший кандидат на заміну [Node](#) – це нова розробка під назвою [Bun](#) [30]. Її автор [J. Sumner](#) обіцяє кращу швидкість та повну зворотну підтримку коду, що вже написаний під [Node](#). Проект активно розвивається і на вересень 2024 року вийшов стабільний реліз 1.1.29. З розвитком [JavaScript](#) для покращення візуальної складової вебсайтів виникло багато бібліотек. Однією з популярних стала бібліотека [jQuery](#), призначена для маніпулювання [DOM](#)-деревом вебсторінки, а також обробки подій, анімації і [AJAX](#). Це найпоширеніша бібліотека [JavaScript](#). На 2024 рік [jQuery](#) використовують 77% із 10 мільйонів найпопулярніших вебсайтів [56]. Бажання й надалі покращувати користувацький інтерфейс, прискорювати його роботу призвело до появи низки бібліотек та фреймворків що дозволяли створювати сайти за архітектурою [SPA](#) ([Single Page Application](#)). При такому підході веббраузер завантажує лише одну сторінку і потім динамічно перепишує її новими даними з сервера. При цьому повністю імітується поведінка звичайного вебсайту. За потреби, за технологією [AJAX](#) динамічно підвантажуються додаткові ресурси. Найпопулярнішими рішеннями в даному сегменті наразі є [React](#), [Angular](#), [View](#) та [Svelte](#). Підхід відтворення вебсторінок безпосередньо в браузері за допомогою [JavaScript](#) отримав назву [CSR](#), або [Client-Side Rendering](#). Логічним продовженням цієї технології була можливість збереження отриманих файлів на пристрої користувача, оскільки вони вже завантажені. Так з'явилися [PWA](#) – [Progressive Web Application](#). [PWA](#) дозволили користуватися додатком без підключення до інтернету, а з отриманням доступу до мережі автоматично синхронізуватися з сервером. [PWA](#) наразі стали популярним засобом відправлення так званих [Push](#)-повідомлень, які сайт може відправити користувачу у будь який час. Зростання популярності напрямку [SPA](#) виявило неочікуваний ефект. Справа в тому, що пошукові системи не можуть ефективно індексувати контент, що генерують односторінкові сайти, адже сервер віддає фактично пустий [HTML](#). Контент додається вже згодом за допомогою [JavaScript](#). Проте трафік з пошукових систем це дуже важлива складова сучасного інтернет-бізнесу. Як рішення цієї проблеми виникла низка технологій, що дозволяють частково чи повністю генерувати вебсторінки на сервері: [SSR](#), [SSG](#) та [ISR](#). У випадку з [Server-Side Rendering](#) ([SSR](#)) сервер на запит браузера надсилає практично готову сторінку. Далі застосовується такий метод як

» Цитування: 0%

id: 62

«гідратація»,

який додає на цю сторінку функції, що відслідковують події і надалі сторінка працює як звичайний [SPA](#). Очевидно, що цей метод вимагає більш потужного серверу ніж у випадку з [CSR](#), де всі обчислювання відбувалися на клієнтській машині. [Static Site Generation](#) ([SSG](#)) – рішення яке стало в нагоді в тих випадках, коли сайт досить невеликий чи доволі не часто оновлюваний. В такому випадку нам не обов'язково кожного разу генерувати на сервері сторінку. Ми можемо згенерувати певну кількість статичних [HTML](#)-файлів заздалегідь і відправляти їх браузеру. За рахунок цього клієнт максимально швидко отримає свій файл, бо нам вже нема потреби генерувати його. Відповідно й знижується навантаження на сервер. Недоліком є те, що в разі оновлення якихось даних вам потрібно провести перерендеринг всього сайту, що може стати проблемою якщо у сайту сотні тисяч сторінок. Додатковою перевагою є покращення безпеки вебсайту, адже клієнт отримує лише [HTML](#), а також економія на ресурсах вебхостингу. У випадку потреби якихось динамічних сервісів зазвичай використовують [Serverless](#) рішення від хмарних провайдерів, накшталт [AWS Lambda](#) або [Cloud Functions](#). [Incremental Static Regeneration](#) ([ISR](#)) – рішення, що поєднує методи [SSR](#) та [SSG](#). В цьому випадку частина вебсторінок що оновлюються часто – генерується динамічно, а частина тих, що практично не оновлюються – генеруються статично. З недоліків в цьому випадку можна відмітити лише складність реалізації. Проте існуючі фреймворки дуже з ними допомагають, адже кожен з лідерів розробки [SPA](#) наразі має і технології для [SSR](#): [Next](#) – для [React](#), [Angular Universal](#) – для [Angular](#) та [Nuxt](#) – для [Vue](#). Існують також і незалежні рішення. Розглядаючи сьогодні все існуюче розмаїття технологій для створення веборієнтованих рішень стає зрозумілим складність спроб описати якісь критерії, за якими б можна було однозначно визначити чи це сайт знаходиться перед вами чи вебдодаток. Тим більше, що і визначення терміну

» Цитування: 0%

id: 63

«вебдодаток»

ще остаточно не прийняте науковою спільнотою. Проте дослідження еволюції цих технологій, на наш погляд явно показало кілька етапів, або рівнів, які кожного разу давали поштовх новому розвитку веброзробки. Для опису цих рівнів дуже зручно використати модель зрілості (рис. 1.3), що йде від початкового рівня – перших статичних [HTML](#) сторінок, до сучасних складних програмних рішень. Рис. 1.3. Рівні зрілості вебдодатків На нульовому рівні знаходяться, як вже було зазначено, статичні сторінки. Тобто саме ті рішення, що створені за допомогою чистого, статичного [HTML](#) ми пропонуємо називати

» Цитування: 0%

id: 64

«вебсайтами»,

або вебдодатками нульового рівня зрілості. Перший рівень займають вебдодатки, що використовують або створені за допомогою лише серверних бекенд технологій. Ці вебдодатки вже можуть обробляти якісь запити з браузера, динамічно генерувати сторінки. Проте вони відправляють браузеру той самий статичний [HTML](#) код. Перші поштові форми, генератори статичних сторінок, та навіть деякі вебдодатки стеку [LAMP](#) – яскравий тому приклад. На другому рівні зрілості з'являються технології що вже мають певну динаміку на стороні веббраузера. Такі веб додатки вже можуть маніпулювати [DOM](#)-деревом, дозволяють з клієнтської сторони робити запити до сервера, використовуючи вбудовані рішення. На найвищому на сьогодні рівні розміщені технології і, відповідно, створені з їх допомогою вебдодатки, що тим чи іншим способом використовують в роботі браузера клас [XMLHttpRequest](#) або його спадкоємця – [Fetch API](#). Це всі рішення що використовують [AJAX](#), та сучасні бібліотеки та фреймворки, що допомагають в створенні вебдодатків. Відповідність конкретного вебдодатка рівню зрілості визначається за максимальним рівнем використаної в ньому технології. На наш погляд описана модель є зручним і доступним способом пояснити, важливі відмінності вебдодатків та різницю між вебдодатком та сайтом. Також наведена модель дозволяє послідовно зрозуміти основні етапи розвитку технологій веброзробки. [MEAN](#) стек: [MongoDB](#), [Express.js](#), [Angular](#), [Node.js](#) – загальний огляд технологій Натан Мірволд, [CTO Microsoft](#), казав:

» Цитування: 0,1%

id: 65

«Всі технології починаються з іскор в чийсь голові. Ідея чогось, чого раніше не існувало,

але одного разу буде винайдено, може змінити все».

Брендану Айку знадобилося лише десять днів, щоб створити першу версію [JavaScript](#). Жодна мова програмування не є ідеальною, але, якщо судити лише по її розповсюдженню, можливо, до ідеалу наближається [JavaScript](#), бо саме вона змінила все. Це мова, яку наразі можна розгорнути всюди: на серверах, у браузерях для настільних комп'ютерів, мобільних пристроях, навіть в мікроконтролерах [Arduino](#), тобто у роботах. [JavaScript](#) використовується інженерами програмного забезпечення на всіх рівнях досвіду та професійної підготовки (рис. 1.4). Якщо промислова революція була побудована за допомогою сталі, то Інтернет-революція – за допомогою мови [JavaScript](#). Коли з'явився [JavaScript](#), його основним завданням було обробити частину вхідних даних від користувача. Ця перевірка раніше проводилася на стороні сервера і можливість виконувати деякі базові перевірки на клієнті була захоплюючою новинкою в часи використання модемів для зв'язку з мережею. Рис. 1.4. Частка наймів працівників за спеціалізацією [12] З того часу [JavaScript](#) став важливою частиною кожного основного веббраузера. Завдяки йому тепер можна створити вебдодаток від початку і до кінця за допомогою лише однієї мови програмування. Такий набір технологій, що дозволяє повністю створити вебдодаток, тобто як клієнтську так і серверну його частини, отримав назву [full-stack](#) (повний стек). Стек [MEAN](#) – є саме таким набором технологій. Він включає в себе кращі примірники свого класу в цій області. В якості БД ми отримуємо [MongoDB](#), в якості серверного фреймворку вебдодатків – [Express](#), в якості клієнтського фреймворку – [Angular](#), а в якості серверної платформи – [Node](#). Раніше, якщо ви хотіли бути розробником [full-stack](#), вам потрібно було знати хоча б дві мови програмування: [JavaScript](#) для клієнтської частини і щось на кшталт [PHP](#) або [Ruby](#) для серверної. Сьогодні – достатньо лише [JavaScript](#). В наш час світова наукова спільнота приділяє багато уваги дослідженням вебдодатків на стеку [MEAN](#) та експериментам з їх використанням. А. [Dordevic](#) та ін в своїй роботі проаналізували технології [MEAN](#) стеку та довели, що розробка та використання вебдодатків на основі стеку [MEAN](#) забезпечує покращену можливість для розпізнавання шаблонів та виконання інших принципів контролю якості [43]. На конференції з тенденцій в електроніці та інформатиці [S. Shabu](#) та [S. Kumar](#) опублікували свої дослідження використання технологій [MEAN](#) стеку в купі з бібліотекою інструментів для створення монорепозиторіїв – [Nx](#). На думку дослідників використання цієї бібліотеки посилює можливості масштабування проєктів, та робить розробку вебдодатків на базі [MEAN](#) стеку у 3 рази швидше, ніж без цієї бібліотеки. Монорепозиторії дозволяють зберігати в одному репозиторії зв'язки з кількома проєктами, наприклад вебклієнт, мобільний клієнт і бекенд частину. Це дозволяє зокрема впровадити повторне використання коду, прискорити збірку проєкту [33]. [J. Cincovic](#) та [J. Delcev](#) в процесі дослідження архітектури вебдодатків на основі [Angular](#) виявили, що стек [MEAN](#) сьогодні це найкраща комбінація для зв'язку серверної частини вебдодатку з інтерфейсом. Науковці акцентували увагу на продуктивності і швидкості програми, а також широкій підтримці сторонніх бібліотек [31]. Незважаючи на значну кількість досліджень технологій стеку [MEAN](#), зумовлену їх постійним оновленням та появою нових розробок, дослідження [MEAN](#) стека залишається актуальним. Розширення можливостей кожного з компонентів стеку, а також їхнє інтегрування та взаємодія, вимагає постійного аналізу та вдосконалення. Такий підхід дає змогу ефективно використовувати потенціал [MEAN](#) стеку в розробці сучасних вебдодатків та забезпечує високу конкурентоспроможність на ринку програмного забезпечення. [MongoDB](#): особливості зберігання даних та взаємодія з БД [MongoDB](#) в 2007 році заснували [D. Merriman](#), [E. Horowitz](#) та [K. Ryan](#), які перед цим працювали з проєктом в інтернет-рекламі – [DoubleClick](#). Цей бізнес обслуговував 400 000 рекламних повідомлень на секунду, проте мав проблеми з масштабованістю та гнучкістю. Команда хотіла створити базу даних, яка б вирішила проблеми, з якими вона зіткнулася в [DoubleClick](#). Саме тоді й народилася [MongoDB](#). Сьогодні база даних [MongoDB](#) надає легкомасштабовані послуги та інструменти, необхідні для швидкого створення розподілених додатків відповідно до потреб користувачів. [MongoDB](#) має десятки тисяч клієнтів у понад 100 країнах. З 2007 року платформу бази даних [MongoDB](#) було завантажено сотні мільйонів разів, і мільйони розробників пройшли навчання на курсах університету [MongoDB](#). [MongoDB](#) – це документоорієнтована база даних, що створена для загального використання. Вона зберігає дані в оптимізованому форматі [JSON](#) під назвою [BSON](#) (двійковий [JSON](#)). Потім документи зберігаються в логічних групах, які називаються колекціями, а багато колекцій разом утворюють базу даних. Документна модель підтримує багато різних типів даних, включаючи рядки, дати, числа, масиви, вкладені об'єкти, геодані та двійкові дані. [MongoDB](#) використовує реплікацію та розділення даних для розподілу даних з метою високої доступності та масштабованості, що робить її стійкою до збоїв БД. Також [MongoDB](#) підтримує декілька методів розгортання, від запуску на різноманітних платформах і серверах, починаючи з вашого локального комп'ютера, до повномасштабного розгортання в хмарі за допомогою [MongoDB Atlas](#). Крім того, корпоративні пакети [MongoDB](#) пропонують такі функції як автентифікація, авторизація, а також наскрізне шифрування. Спробуємо порівняти можливості [MongoDB](#) та вже давно відомої розширеної версії БД [MySQL](#) – [MariaDB](#). Таблиця 1.1 Порівняльна таблиця [SQL](#) та [NoSQL](#) бази даних [MariaDB](#) [MongoDB](#) 1 2 3 Модель даних Зберігає дані в таблицях. Кожна таблиця містить рядки та стовпчики. Дані мають бути нормалізовані та поділені на різні логічні таблиці. Зберігає документи в оптимізованому форматі [BSON](#). Документи групуються в колекції та бази даних і повертаються як документи [JSON](#) із підтримкою великої кількості типів даних, включаючи: рядки, числа, геодані, дати, масиви, вкладені об'єкти, двійкові дані та ін. Індексація Дозволяє індексувати різні стовпчики. Індекси охоплюють лише стовпчики, в яких вони створені. Якщо потрібні спеціальні індекси, наприклад геоіндекси, до бази даних потрібно додавати зовнішні модулі. Підтримує багато типів індексів для різних випадків використання. Вторинні індекси для будь-якого поля доступні та підтримуються в різних типах: текстовий, географічний, [TTL](#) та ін. Мова запитів використовує діалект [SQL](#). Має широкі можливості надсилання запитів, які надає [Query API](#). Запити можуть використовувати складні оператори, а також структуру агрегації для розширеної обробки даних і аналітики. Транзакції Підтримує транзакції [ACID](#), проте без підтримки [snapshot isolation](#). Підтримує повністю сумісні з [ACID](#) транзакції, включно з сегментованими кластерами та [snapshot isolation](#). Багато-поточність Контроль багатопоточності в залежності від використаного механізму зберігання. Рівень ізоляції можна змінити на основі конкретних конфігурацій транзакцій. Дозволяє кільком користувачам бази даних одночасно отримувати доступ до тих самих даних, керуючи чітко визначеним контролем паралелізму. Використовує блокування на рівні документа, тому записи в один документ відбуваються або повністю, або не відбуваються взагалі, і клієнти завжди бачать послідовні дані. Разом із цими механізмами підтримує читання та запис у розподілених кластерах. Продовження таблиці 1.1 1 2 3 Безпека Автентифікація та авторизація через імена користувачів і паролі, [TLS/SSL](#), шифрування [x509](#), аудит. Механізми безпеки корпоративного рівня. Автентифікація та авторизація за допомогою вбудованого [SCRAM](#) або сертифікатів, [TLS/SSL](#), [x509](#), шифрування з можливістю запису та шифрування на рівні полів на стороні клієнта. Шифрування й аудит на стороні сервера. Інтеграції [LDAP](#) і [Kerberos](#). Сертифікати відповідності вимогам безпеки. Мобільна розробка Для мобільної розробки потрібно встановлювати додаткові сторонні модулі. Два ключових компоненти

для мобільної розробки: [Atlas Device SDKs](#) і [Atlas Device Sync](#) Хмарні пропозиції БД як послуга, частково доступна на [AWS](#) і [GCP](#). Деякі функції доступні лише в [GCP](#). Безкоштовний рівень недоступний. Вимагає попереднього визначення призначення кластера, наприклад транзакції чи аналітика. Обидва варіанти використання не можуть бути охоплені одним кластером. БД як послуга доступна у трьох основних хмарних постачальників – [AWS](#), [Azure](#) та [GCP](#) – починаючи від безкоштовного рівня до повномасштабного міжрегіонального та міжхмарного кластера. Послуги можна динамічно масштабувати, а хмарних провайдерів можна змінювати за потреби. Документація та навчання Пропонує документацію для версії з відкритим кодом і корпоративної версії. Також пропонує онлайн-курси та навчання. Пропонує документацію з прикладами та повними посібниками. Має широку спільноту та вебсайти центру розробників. Є онлайн-університет із безкоштовними курсами. Вбудовані інструменти візуалізації Не пропонує жодних власних інструментів візуалізації даних, проте інтегрується зі сторонніми програмами для візуалізації даних і бізнес-аналітики. [MongoDB Charts](#) забезпечує потужний спосіб візуалізації даних за допомогою [MongoDB Atlas](#). [MongoDB Compass](#) надає клієнт [GUI](#) для [MongoDB](#). Надає конектор для інтеграції з популярними сторонніми інструментами бізнес-аналітики. Як бачимо, [MariaDB](#) хоча й розширила та додала цінних функцій до БД [MySQL](#), проте вона створена на основі застарілих основних концепцій. Обмеження не дозволяють їй повністю адаптуватися до великих даних і хмарного характеру сучасної гнучкої розробки. [MongoDB](#) ж було створено для підтримки хмарних технологій та будь-якого обсягу даних. Вона є однаково продуктивною та гнучкою будь-де. Уніфікований [Query API](#) і потужні агрегації [MongoDB](#) у поєднанні з гнучкістю моделі документів зробили її найпопулярнішою базою даних документів загального призначення на ринку. Зазвичай кожна сучасна БД має так звану [ORM](#) – технологію, що зв'язує програму з базою і допомагає більш зручно з нею спілкуватися. В [MongoDB](#) ця технологія представлена бібліотекою [Mongoose](#) – [ODM](#)-бібліотекою ([Object Data Modelling](#)) для роботи з [MongoDB](#). [Mongoose](#) допомагає вирішувати наступні завдання: Організувати [CRUD](#) функціонал, причому виконувати навіть складні запити, наприклад шукати по [ID](#) і одразу видаляти Виконувати спеціалізовані запити Виконувати міграції, тобто додавати поля або перемішувати дані Валідацію даних ([MongoDB](#) не валідує дані і дозволяє певні варіації) Спростити запити до БД Для взаємодії з [MongoDB](#) через [Mongoose](#) спочатку ми маємо створити схему, в якій описати поля даних та їх особливості (значення за замовчуванням, обов'язковість, типи та ін.). Після цього на базі схеми створюється модель, і через неї вже відбувається робота з БД. [Express.js](#): створення сервера та обробка [HTTP](#)-запитів [Express](#) – це буква

Цитирования: 0% id: 66

в [MEAN](#). Згідно даним [Stack Overflow Express](#) викликає неабиякий інтерес у розробників і останні роки знаходиться на першому місці по запитам на цьому сайті (рис. 1.5). Оскільки [Node.js](#) – платформа, вона потребує певних налаштувань. [Express](#) – це фреймворк, спроектований саме для спрощення цих налаштувань. Саме в ньому ми визначаємо які вхідні запити буде прослуховувати і обробляти вебсервер, які відповіді надсилати, структуру каталогів та ін. Фактично наш сервер це [Node.js](#), а [Express](#) по суті – обгортка, допоміжний модуль. Рис. 1.5. Статистика трендів [Stack Overflow](#) по серверним технологіям [53] В [Node.js](#) є вбудований модуль під назвою [HTTP](#), який дозволяє передавати дані по протоколу передачі гіпертексту ([HTTP](#)). І якщо не використовувати [Express](#), то для створення найпростішої точки доступу з відповіддю

Цитирования: 0,01% id: 67

достатньо набрати кілька строк коду: Лістинг 1.1. Запуск серверу в [Node.js](#) `var http = require(`

Цитирования: 0% id: 68

`'http'`

`); http.createServer(function (req, res) { // створюємо об'єкт сервера res.writeHead(200, {`

Цитирования: 0% id: 69

`'Content-Type':`

Цитирования: 0% id: 70

`'text/html'`

`}); res.write(`

Цитирования: 0,01% id: 71

`'hello World!'`

`); // пишемо відповідь сервера res.end(); // кінець відповіді }) .listen(3000); // об'єкт сервера`

слухає порт 3000 Запуск такого ж сервера за допомогою [Express](#) вийде ще елегантнішим: Лістинг 1.2. Запуск серверу в [Express.js](#) `const express = require(`

Цитирования: 0% id: 72

`'express'`

`); const app = express(); app.get(`

Цитирования: 0% id: 73

`'/',`

`(req, res) = { res.send(`

Цитирования: 0,01% id: 74

`'hello World!'`

`); }); app.listen(3000); В якості альтернативи Express для роботи з сервером в Node`

використовуються також такі фреймворки як [Koa](#), [Sails](#), [Nest](#) та багато інших. Наприклад

[Sails](#) більше буде до вподоби тим розробникам, що працювали з [Ruby](#), а [Nest](#) тим, хто

знайомий з [Angular](#). Для роботи з [Express](#) потрібно визначити роути (маршрути) та функції,

що будуть спрацьовувати при зверненні клієнта за цими роутами (рис. 1.6). Рис. 1.6.

Визначення роута в [Express](#) На екземплярі сервера ми визиваємо метод [HTTP](#), для якого

застосовується функція-обробник. Першим аргументом вказуємо шлях (маршрут), який ми

будемо обробляти, Другим аргументом буде сама функція-обробник. Вона приймає

наступні аргументи: об'єкт [request](#), що містить всі заголовки, методи, [URL](#), тіло запиту;

об'єкт [response](#), в який ми додаємо те, що відправляється клієнту. Цей об'єкт також

містить кілька методів, що дозволяють генерувати різні відповіді в різних форматах;

необов'язковий параметр [next](#), для того щоб передати управління наступній функції-



обробнику, якщо їх буде декілька. В рамках функції-обробника ми виконуємо всі необхідні



дії, що має зробити сервер при виклику за цією [URL](#) адресою в комбінації з цим [HTTP](#)

методом. Фреймворк [Express](#) по суті – це набір таких роутів та обробників. На практиці

схема обробки запиту в [Express](#) може виглядати так як показано на рис. 1.7. Рис. 1.7.

Приклад проходження в [Express](#) запиту на зміну даних користувача [47] Клієнтський запит спочатку проходить кілька загальних [middleware](#), обробників що виконуються на всіх запитах. Кожен з [middleware](#) виконує якісь свої окремі операції на кшталт зчитування [cookies](#), логування, аутентифікація та передає цей запит через виклик функції [next\(\)](#) наступному обробнику. Далі запит обробляється вже конкретно призначеним для нього роутом: парсяться параметри запиту, виконується запит до БД, отримується відповідь від БД і вже фінальний результат операції надсилається назад клієнту [Angular](#): клієнтська частина вебдодатку [Angular](#) – це фреймворк для розробки вебдодатків з відкритим вихідним кодом. Його розробкою займається компанія [Google](#). [Angular](#) зазвичай використовується для створення масштабних, складних односторінкових вебдодатків ([SPA](#)). Фреймворк пропонує комплексний підхід до розробки, забезпечуючи надійність і масштабованість. Це серйозна платформа, побудована на [TypeScript](#). Як платформа [Angular](#) включає: Компонентну основу для створення масштабованих вебдодатків Набір добре інтегрованих бібліотек, які охоплюють широкий спектр функцій, включаючи маршрутизацію, керування формами, клієнт-серверний зв'язок та ін. Вбудований механізм [dependency injection](#) Набір інструментів розробника, які допоможуть вам розробляти, будувати, тестувати й оновлювати код. Однією з ключових особливостей [Angular](#) є вбудоване використання [TypeScript](#), мови, що додає до [JavaScript](#) статичну типізацію та інші розширені можливості. Це забезпечує більш надійну та безпечну розробку, дозволяє виявляти помилки на ранніх стадіях та краще організовувати код. Екосистема [Angular](#) наразі складається з понад 1,7 мільйона розробників, авторів бібліотек і контентмейкерів. Основним будівельним блоком для створення вебдодатків в [Angular](#) є компоненти. В компоненті окремі файли опікуються логікою, а інші – зовнішнім виглядом компоненту (представленням). Існують також такі сутності як: сервіси – основні постачальники даних та зв'язок між компонентами директиви – розширюють можливості певних компонентів; пайпи – фільтри, що зазвичай використовуються для форматування даних в [HTML](#) шаблоні; інтерсептори – перехоплювачі [HTTP](#) запитів, що за відповідних умов мають можливість маніпулювати запитом, додати чи прибрати якісь заголовки чи токени; гарди – об'єкти, що захищають певні роути (маршрути) вебдодатку від несанкціонованого доступу та багато інших. Компанія [Google](#) постійно працює над покращенням свого фреймворку. Зазвичай виходить по дві нові версії [Angular](#) на рік. За популярністю [Angular](#) займає другу сходинку поступаючись бібліотеці [React](#) (рис. 1.8). Рис. 1.8. Статистика трендів [SPA](#) на [Stack Overflow](#) [54] [Node.js](#): серверне середовище виконання [JavaScript](#) та його застосування у веброзробці [Node.js](#) – це кроссплатформене середовище для виконання коду [JavaScript](#). Платформа [Node.js](#) побудована на основі [JavaScript](#) рушія V8 від [Google](#), який використовується в браузері [Google Chrome](#). В основному вона використовується для створення вебсерверів, однак сфера її застосування цим не обмежується. Безкоштовний реєстр [Node Package Manager \(NPM\)](#) став центром обміну кодом [JavaScript](#) і, з більш ніж 2 млн. загальнодоступних і платних приватних програмних пакетів, став найбільшим реєстром програмного забезпечення у світі. [NPM](#) робить розробку [JavaScript](#) елегантною, продуктивною та безпечною. Однією з основних привабливих особливостей [Node.js](#) є швидкість. Код, що виконується в середовищі [Node.js](#), може бути в два рази швидше, ніж код, написаний на скомпільованих мовах, на кшталт [C](#) або [Java](#), а також на порядок швидше мов подібних [Python](#) або [Ruby](#). Причина цього – неблокуюча архітектура платформи. [Node.js](#) – це дуже швидко. У середовищі [Node.js](#) виконується код, написаний на [JavaScript](#). Це означає, що мільйони фронтенд-розробників, які вже використовують [JavaScript](#) в браузері, можуть писати і серверний, і клієнтський код на тій самій мові програмування без необхідності вивчати новий інструмент для переходу до серверної розробки. Платформа [Node.js](#) дуже проста в освоєнні та використанні. У браузері та на сервері використовуються однакові концепції мови. Крім того, в [Node.js](#) можна швидко перейти на використання нових стандартів [ECMAScript](#). Для цього не потрібно чекати, поки користувачі оновлять свої браузери, бо [Node.js](#) – це серверне середовище, яке повністю контролює розробник. У традиційних мовах програмування ([C](#), [Java](#), [Python](#), [PHP](#)) усі інструкції за замовчуванням, є блокуючими, якщо тільки розробник явним чином не буде опікуватись про асинхронне виконання коду. В результаті, якщо, наприклад, у такому середовищі, виконати мережевий запит для завантаження якогось [JSON](#), код, в потоці з якого зроблено запит, буде заблоковано поки не завершиться отримання і обробка відповіді. [JavaScript](#) значно полегшує написання асинхронного і неблокуючого коду при використанні єдиного потоку. Кожного разу, коли нам потрібно виконати важку операцію, ми передаємо відповідному механізму [callback](#) функцію, яку буде викликано одразу ж після завершення цієї операції. В результаті головний потік програми не блокується. Подібний механізм асинхронної роботи з'явився в браузерах. Браузер не може дозволити собі чекати, скажімо, закінчення виконання [AJAX](#)-запиту, і не мати при цьому можливості реагувати на дії користувача, наприклад, на кліки по кнопкам. Асинхронні механізми дозволяють [Node.js](#) одночасно обробляти тисячі підключень, не опікуючись управлінням потоками і організацією паралельного виконання коду. Більшість сучасних поширених вебсерверів, включно з [Apache](#) та [IIS](#) – багатопоточні. Це означає, що для кожного нового відвідувача (або сеанса) сервер створює новий потік і виділяє пов'язаний з цим потоком обсяг оперативної пам'яті, приблизно 8 Мб. Це створює неабияке навантаження на сервер в часи, коли на нього одночасно заходять кілька тисяч користувачів. Саме тому створюються сервери з великим запасом потужності та оперативної пам'яті, які 90% часу не використовуються на повну. На відміну від них сервер на [Node](#) – однопоточний. Тим не менш він здатний обробляти більше запитів з використанням менших ресурсів. Це працює завдяки асинхронним можливостям [JavaScript](#). Кожен запит не блокує потік, а делегується операційній системі (наприклад запит прочитати файл) і потік знову повертається в нескінченний цикл обробки запитів – [Event Loop](#) (рис. 1.9). Рис. 1.9. Схема роботи [Event Loop](#) [42] Після того як ОС виконає запит, вона повертає його результати через [callback](#) функцію. Таким чином, навіть якщо було 4 послідовні запита не гарантується що вони отримають відповіді в тому ж порядку. Відповіді відпрацюють асинхронно і надійдуть в тому порядку, в якому ОС завершить їх виконання. [Node.js](#) фактично є основою [JavaScript](#) веброзробки. Висновки до розділу 1 Дослідження інтернет-технологій та їх вплив на сучасне інформаційне суспільство, як виявлено в даному розділі, підкреслює надзвичайну вагомість Інтернету як основи для функціонування інформаційного суспільства. Його роль не обмежується лише етапом в історії обчислювальної техніки, але визначається як одна з ключових подій у великій історії людства. Це свідчить про необхідність глибокого розуміння та аналізу динаміки та впливу інтернет-простору на сучасний світ. Термін

 Цитування: 0%	id: 75
«вебдодаток»	
залишається предметом дискусій в українській та міжнародній спільноті. Найбільш узагальнюючий погляд на це поняття пропонує визначення, що базується на технологіях та стандартах World Wide Web Consortium , враховуючи значення W3C як першоджерела технологій і стандартів для інтернету. Наразі	
 Цитування: 0%	id: 76
«вебсайт»	

та	
 Цитування: 0%	id: 77
«вебдодаток»	
<p>– дуже близькі одне одному поняття. Запропоновано модель рівнів зрілості вебдодатків, яка виокремлює чотири рівні від статичних сторінок до вебдодатків з використанням сучасних технологій, та сприяє глибшому розумінню етапів розвитку веброзробки та визначенню різниці між різними формами вебресурсів. Стек технологій MEAN (MongoDB, Express.js, Angular, Node.js) є важливим і популярним вибором для сучасної веброзробки. Він об'єднує сучасні інструменти та надає розробникам зручність та продуктивність. MEAN використовує спільну мову програмування (JavaScript) на всіх рівнях додатка, що спрощує розробку та підтримку. Це особливо корисно для стартапів та проєктів з обмеженими ресурсами. Кожна складова MEAN має своє призначення: MongoDB – гнучка NoSQL база даних, яка дозволяє зберігати дані у форматі JSON-подібних документів. Express.js – фреймворк для побудови серверної частини додатка на Node.js. Angular – фреймворк для розробки клієнтської частини вебдодатків. Node.js – середовище виконання JavaScript на сервері. MEAN дозволяє розробникам створювати потужні та масштабовані вебдодатки, забезпечуючи гнучкість, продуктивність та спільноту підтримки. Загальний висновок полягає в тому, що технології створення вебдодатків стають центральним чинником в інформаційних технологіях, відіграючи ключову роль у сучасній цифровій трансформації. Розуміння їх розвитку не тільки сприяє адаптації до сучасних вимог, але й допомагає в ефективному використанні їх потенціалу для створення високопродуктивних та інноваційних рішень у сфері веброзробки.</p> <p>ЕКСПЕРИМЕНТАЛЬНА ОЦІНКА ШВИДКОДІЇ СЕРВЕРНИХ ПЛАТФОРМ У КОНТЕКСТІ MEAN СТЕКУ</p> <p>Вернер Гейзенберг, один з найвідоміших лідерів сучасної науки, сказав:</p>	
 Цитування: 0,05%	id: 78
«Науково-технічний прогрес – це лише спосіб зробити пекло більш комфортним для життя».	
<p>У випадку з Node.js сучасні прогресивні розробки пропонують кілька більш комфортних середовищ виконання коду JavaScript. Це і вже згадуваний раніше Deno, який робить акцент на безпеці, може використовувати TypeScript без встановлення додаткових пакетів, проте має обмежену кількість модулів порівняно з NPM. Інша платформа – Electron, використовується для створення десктопних додатків з використанням вебтехнологій. За допомогою Electron створено такі додатки як: Slack, Skype, Discord, VSCode, Atom, Postman та ін. Також нещодавно вийшла в реліз платформа Bun, націлена на повну сумісність з Node.js. Більшість пакетів NPM, призначених для середовища Node.js, мають працювати і з Bun, а його CLI містить сумісний з Node.js менеджер пакетів. Фактично Bun – позиціонується як новий, окремий інструмент, який за задумом працюватиме і у вже існуючих проєктах на Node. Bun – відносно нова технологія, її вважають експериментальною, а спільнота розробників продовжує працювати над вирішенням знайдених проблем та покращенням зручності її використання. Тим не менш релізна версія платформи, яка за заявами розробників готова до продакшн, була запропонована широкому загалу. Оскільки Bun має на меті замінити Node, то очевидним стає актуальність досліджень роботи нової технології в контексті MEAN стеку, тобто взаємодії Bun з MongoDB, Express та Angular. Також, враховуючи велику кількість вже існуючих, робочих проєктів створених на цьому стеку, актуальним є дослідження можливості міграції проєктів з Node на Bun. Тестування швидкодії вебдодатку на новій платформі дозволить зробити певні висновки щодо перспективності впровадження цієї розробки. Bun: ключові особливості та потенціал для MEAN стеку Bun – це новітнє середовище виконання JavaScript, яке набирає популярності. Воно призначене для швидкого виконання JavaScript, TypeScript, JSX, і завдяки використанню різноманітних оптимізацій та архітектурних рішень, обіцяє високу продуктивність. З наявної на офіційних сайтах документації спробуємо порівняти платформи Node та Bun. Таблиця 2.1 Порівняльна таблиця платформ Node та Bun. Node Мова реалізації В основному C++ Використовується мова Zig JS рушій V8 (від Google) JavaScriptCore (від WebKit) Призначення Високопродуктивний для чисельних операцій, великих масштабованих застосунків Підвищення продуктивності Node Сумісність з пакетами Повна підтримка NPM Пропонує підтримку пакетів Node, Web API, а також власний пакетний менеджер Екосистема Велика кількість існуючих модулів та розвинута спільнота Новіша платформа, активне співтовариство, але менша кількість доступних пакетів на даний момент Безпека Розширені можливості безпеки через додаткові модулі Фокус на безпеці з вбудованими функціями та меншою залежністю від сторонніх пакетів Популярність Широке використання у всіх напрямках веброзробки Викликає певну зацікавленість розробників Транспайлер Потребує додаткових модулів Вбудований транспайлер TypeScript Реалізація HTTP сервера На базі Express.js та інших фреймворків Пропонує власні механізми створення HTTP серверів Підтримка платформ Підтримує більшість операційних систем Триває робота над розширенням сумісності з різними платформами Інструментарій Широкий спектр інструментів для моніторингу та розробки Інструментарій постійно поповнюється Основні переваги Зрілість, стабільність, широка підтримка модулів Використання сучасних технологій, анонсується швидкість Завантаження пакетів Може бути певна інертність у завантаженні Повідомляється про прискорення завантаження Bun як і Node – дозволяє нам працювати з файловою системою, операційною системою, взаємодіяти з БД, відповідати на HTTP запити, створювати сервери, тощо. Проте Node можна порівняти з конструктором для роботи з яким нам кожного разу потрібно підключати якісь додаткові частини, HTTP-клієнти, модулі для роботи з вебсокетами, збиральники проєктів, інструменти для тестування, інструменти для відслідковування змін в коді, для зчитування змінних середовища. Завдяки цьому конструктору ми маємо велику гнучкість системи, але з іншого боку, кожен з цих додаткових модулів іноді виконує схожі дії і як результат ми втрачаємо швидкість, наприклад, при парсингу, збірці проєкту, транспіляції. Всі ці модулі недостатньо оптимізовані під спільну роботу, бо Node існує вже доволі давно і різні її додатки з'являлися в різний час. Крім того у Node доволі погана підтримка WEB-API, є свої особливості в типах даних, все це наслідок того що розвиток Node був доволі складним і певні рішення були виправдані лише задля збереження зворотної сумісності з попередніми її версіями. В свою чергу Bun – це, за заявами розробників, all-in-one інструмент. Коли ви починаєте писати на ньому проєкт, то з коробки отримуєте практично все необхідне: відслідковування змін коду, HTTP та вебсокет сервер, транспіляцію, збірку проєкту, роботу з пакетами, тестування. Окрім NPM надається власний інструмент, що за заявами розробників майже в 30 разів швидше ніж NPM чи Yarn і при цьому ці пакети оптимізовані під використання з Bun. Проте мабуть все це не мало б великого сенсу якби у Bun не було зворотної сумісності з Node. Але в Bun вона є, тобто мається на увазі що будь-який проєкт, що написаний на Node, можна запустити в Bun. Розробники запевняють, що проведені ними тести з React рендерингу показали в 5 разів швидшу роботу Bun в порівнянні з Node (66706 HTTP запитів на секунду проти 13967). А тестування роботи з БД SQLite показало перевагу Bun у 4 рази (81,37 запитів на секунду проти 21,29). Методологія дослідження швидкодії платформ в контексті MEAN Як ми вже визначали раніше, платформа Node в вебдодатках стеку MEAN</p>	

налаштовується за допомогою фреймворку [Express](#), та використовується в якості вебсервера, тобто обробляє [HTTP](#)-запити від клієнта, як показано на рис. 2.1. Рис. 2.1. Обмін даними між компонентами [MEAN](#) В реальних вебдодатках, створених за традиційною архітектурою з серверу [Node](#) за допомогою [ORM](#) чи [ODM](#) також відправляються запити до БД. Для того щоб зробити власні висновки, щодо швидкодії нової платформи, а також можливої заміни нею платформи [Node](#) ми маємо визначити та порівняти кількість [HTTP](#)-запитів на секунду, що спроможний обробити той самий вебдодаток на першій та на другій платформі. Також потрібно врахувати роботу додатку з базою даних та без неї. Для проведення експерименту потрібно розробити вебдодаток з використанням технологій [MEAN](#) стеку. В проєкті маємо використовувати стандартні, найбільш поширені бібліотеки і компоненти. Враховуючи те, що експеримент буде проходити на серверній частині додатку особливої уваги приділити саме серверній розробці. Клієнтська частина вебдодатку [Angular](#) – в експерименті участі практично не бере бо в реальних проєктах [HTML](#) та [JavaScript](#) частини [SPA](#)-додатку зазвичай віддає браузеру сервер [Nginx](#), який відповідає за роздачу статичного контенту. Інструменти тестування та критерії оцінювання швидкодії Для тестування роботи додатку на різних платформах обрано утиліту

Цитування: 0%

id: 79

«[Bombardier](#)».

[Bombardier](#) – є сучасним, популярним серед розробників інструментом для оцінки продуктивності [HTTP](#)-сервісів. Цей інструмент допомагає вимірювати швидкодію [HTTP](#)-ендпоінтів, що є критично важливим для оптимізації вебдодатків і забезпечення їх безперебійної роботи при високих навантаженнях. [Bombardier](#) написаний на мові програмування [Go](#) і використовує в роботі бібліотеку [fasthttp](#) замість стандартної бібліотеки [http Go](#). Це забезпечує високу швидкість і надійність в роботі, дозволяє значно зменшити витрати на обробку запитів порівняно зі стандартними інструментами, і таким чином підвищити точність. [Bombardier](#) здатний створювати значні навантаження на сервери, імітуючи багатокористувацькі запити для тестування різних аспектів [HTTP](#)-сервісів, такі як витривалість, стабільність та масштабованість. Через свою простоту та зручність (працює з командного рядка) ця утиліта стала вибором багатьох команд, що працюють у сфері розробки і тестування. До основних можливостей [Bombardier](#) можна віднести: Високопродуктивне тестування: здатність генерувати велику кількість запитів в короткий проміжок часу. Підтримка [HTTP](#) та [HTTPS](#): можливість тестувати обидва типи з'єднань. Налаштування параметрів: користувач може вказати кількість одночасних з'єднань, тривалість тесту та інші параметри. Автоматизація: [Bombardier](#) може бути інтегрований у скрипти для автоматизації процесів тестування. Виведення результатів: [Bombardier](#) забезпечує зручне виведення результатів, для подальшого аналізу. Кросплатформенність: в наявності є версії для різних операційних систем. Утиліта розповсюджується як відкрите програмне забезпечення, що дозволяє спільноті розробників вносити свій вклад у доробку, виправлення помилок та вдосконалення функціональності. Простота використання і можливість інтеграції в автоматизовані системи [CI/CD](#) робить [Bombardier](#) ідеальним вибором і дозволяє залишатися актуальним інструментом для проведення навантажувальних тестів на сервери різного рівня. Критерієм оцінювання буде середня кількість [HTTP](#)-запитів ([RPS](#)), що змогла обробити та чи інша платформа за виділений проміжок часу. Також буде контролюватися споживання ресурсів кожною платформою під час виконання, щоб уникнути обмежень з боку апаратного забезпечення. Для підвищення точності результатів експеримент буде проведено 10 разів. Для кожного експерименту буде виміряно кількість [RPS](#), і на базі цих даних розраховано середні значення за формулою: $RPS = \frac{1N}{1NRPSi}$ де N – кількість експериментів, $RPSi$ – кількість [RPS](#) в i -му експерименті. Конфігурація середовища експерименту Якщо для розробки та налагодження вебдодатка зазвичай цілком достатньо локальної машини, то для оцінки продуктивності доцільним буде використання хмарних сервісів для більш точного і реалістичного тестування. При необхідності тестувати на багатьох конфігураціях та масштабах, локальне тестування може вимагати великої кількості ресурсів та може стати витратним. Крім того локальне тестування не дає повної уяви про те, як вебдодаток буде поводити себе в реальному робочому середовищі. Адже локальне тестування не враховує можливі проблеми, пов'язані зі швидкістю мережі хостингу та іншими мережевими факторами, що можуть впливати на продуктивність вебдодатку. Використання хмарного сервера для проведення експериментів з тестуванням швидкості вебдодатків має кілька переконливих переваг: масштабованість – хмарні сервери надають можливість швидко масштабувати ресурси вгору або вниз в залежності від потреб. Це особливо важливо для експериментів, де потрібно тестувати додаток під високим навантаженням або з різних точок світу; гнучкість – хмарні платформи дозволяють налаштовувати характеристики серверів, такі як кількість [CPU](#), обсяг оперативної пам'яті та інші параметри, забезпечуючи гнучкість у виборі конфігурації для проведення експериментів; географічний фактор – хмарні провайдери дозволяють розгорнути сервери в різних географічних регіонах, що дозволяє проводити тестування з різних частин світу. Це особливо важливо, якщо вебдодаток має користувачів з різних країн; вартість – хмарні послуги забезпечують оплату лише за фактично використаними ресурсами. Це зменшує витрати на проведення експериментів, оскільки нам не потрібно купувати та утримувати велику кількість фізичних серверів; інструментарій – хмарні платформи часто надають різні служби та інструменти для моніторингу, логування та аналізу даних. Це спрощує процес тестування та аналізу продуктивності вебдодатку; швидкість розгортання – за допомогою хмарних серверів, ви можете швидко розгорнути нові віртуальні машини та експериментувати з різними конфігураціями, що збільшує ефективність тестування. В цілому, використання хмарного сервера для експериментів з вебдодатками дозволяє забезпечити зручність, ефективність та масштабованість процесу. Для проведення експерименту було обрано хмарний сервіс [Google Cloud Platform](#), [Google Cloud Platform](#) ([GCP](#)) – це хмарна обчислювальна платформа, надана компанією [Google](#). [GCP](#) пропонує різноманітні хмарні послуги, включаючи обчислювальні ресурси, зберігання даних, бази даних, машинне навчання, аналітику даних, мережеві послуги та багато іншого. [GCP](#) надає масштабовану та гнучку інфраструктуру, підтримує різноманітні технології та мови програмування, і пропонує високий рівень надійності та безпеки для додатків і даних. Можна використовувати ресурси [GCP](#) згідно зі своїми потребами, сплачуючи лише за ті ресурси, які реально використовуються. Одна з ключових послуг [GCP](#) це [Compute Engine](#), яка надає віртуальні машини ([VM](#)) для запуску різних видів обчислювальних завдань. Основні характеристики та переваги [Compute Engine](#) включають: Масштабованість: [Compute Engine](#) дозволяє легко масштабувати ресурси Різноманітність конфігурацій: [Compute Engine](#) пропонує різноманітні типи і розміри віртуальних машин, що відповідають різним потребам і вимогам робочих завдань Попередньо налаштовані образи: [Compute Engine](#) надає попередньо налаштовані образи віртуальних машин для різних операційних систем, включаючи різні версії [Linux](#) та [Windows](#). Це спрощує процес розгортання та конфігурації серверів Мережеві можливості: ми маємо повний контроль над налаштуваннями мережі для наших віртуальних машин, включаючи можливість створення власних віртуальних приватних хмарних мереж ([VPC](#)) для ізоляції ресурсів Безпека та надійність: [Compute Engine](#) використовує технології віртуалізації для ізоляції віртуальних

машин, що забезпечує безпеку та стабільність обчислювального середовища Інтеграція з іншими послугами [GCP: Compute Engine](#) легко інтегрується з іншими сервісами [GCP](#), такими як [Cloud Storage](#), [Cloud SQL](#), а також із сервісами моніторингу та керування ресурсами. В цілому, обираючи [Compute Engine](#), ми отримуємо гнучкість, ефективність і високий рівень контролю над своїм обчислювальним середовищем у хмарі. Додатковою перевагою [GCP](#) є те, що цей сервіс пропонує безкоштовний рівень ([Free Tier](#)), який дозволяє користувачам випробувати та використовувати певні ресурси та послуги [GCP](#) без витрат в межах обмеженого обсягу. [Free Tier](#) надається для нових клієнтів [GCP](#) та надає можливість ознайомитися з хмарними послугами, експериментувати та розвивати невеликі проекти без фінансових витрат. Основні можливості [Free Tier GCP](#) включають [Compute Engine](#), що закриває потреби нашого експерименту. Проте потрібно враховувати, що [Free Tier](#) надається на обмежений час та що після завершення цього періоду буде розраховано вартість використаних ресурсів, якщо їх обсяг перевищує ліміти. В якості операційної системи для проведення експерименту обрано [Debian 12 \(bookworm\)](#). Ця ОС пропонується серед інших як один із можливих образів для [VM](#) у [GCP](#). Використання саме [Debian 12](#) має свої переваги та може бути привабливим в ряді сценаріїв. [Debian](#) відома своєю високою стабільністю та надійністю і разом з цим своєю ефективністю та низьким споживанням ресурсів. Вона має одну з найдавніших та найбільших спільнот розробників відкритого програмного забезпечення, а це означає широкий спектр документації, форумів та ресурсів, що можуть бути корисними. [Debian](#) базується на принципах відкритого програмного забезпечення, під неї розроблено велику кількість програм. Система в активній розробці, тобто постійно оновлюється та доповнюється новим функціоналом. В підсумку ми маємо створити наступне середовище для проведення експерименту: Таблиця 2.2 Компоненти середовища для проведення експерименту

Найменування	Конфігурація / Версія
Сервер	GCP Belgium (europe-west1) , Тип VM : c2-highcpu-8 (Intel Xeon 2.2 ГГц 4 ядра 8 потоків) , 8 Гб RAM , SSD 10 Gb , ОС Debian 12 Bun 1.1.29 Node 20.17.0 (LTS) Bombardier 1.2.6 MongoDB GCP Belgium (europe-west1) , M0 Sandbox (Shared RAM, 512 MB Storage)

Розробка тестового вебдодатку на [MEAN](#) стеку Вебдодаток в даному дослідженні використовуються у допоміжній ролі. Головними вимогами до нього є розробка на стеку технологій [MEAN](#) та використання стандартних, найбільш поширених бібліотек. Зважаючи на це в розробці будуть використані наступні інструменти: [MongoDB Compass](#) – графічний інтерфейс для адміністрування та візуалізації даних БД [Mongoose](#) – бібліотека для об'єктно-документного відображення ([ODM](#)), що допомагає взаємодіяти з [MongoDB NPM \(Node Package Manager\)](#) – для управління залежностями та пакетами [Angular CLI](#) – командний інтерфейс для швидкого створення, розгортання та управління [Angular](#) додатками [RxJS](#) – бібліотека для роботи з асинхронними подіями та потоками даних в [Angular Visual Studio Code v.1.86.2 - IDE](#), редактор коду Архітектура та компоненти тестового додатку Наш додаток за легендою – прототип сервісу, схожого на [UBER](#) для вантажних автомобілів. Цей сервіс має допомагати людям перевезти в межах міста свої речі, а водіям вантажівок – знайти замовника і заробити. Додаток містить дві ролі, водія та вантажовідправника. На рис. 2.2 наведено діаграму варіантів використання розроблюваної системи. Рис. 2.2. Діаграма варіантів використання розроблюваної системи Водій і відправник можуть зареєструватися в системі, увійти в систему, переглядати інформацію свого профілю. Водій – може додавати в свій профіль різні моделі вантажівок, переглянути створені їм вантажівки, закріпити за собою певну вантажівку, може видаляти не закріплені за ним вантажівки, може взаємодіяти з призначеним йому вантажем (мінати статус вантажу). Відправник – може створювати вантажі в системі, переглянути створені їм грузи, видаляти грузи що ще не відправлені, розмістити вантаж, переглядати інформацію про статус доставки. Система автоматично підбирає водія для вантажу. Проектування моделі даних у [NoSQL](#) базі даних, такий як [MongoDB](#), відрізняється від традиційного підходу до реляційних баз даних через її документ-орієнтовану структуру (рис. 2.3). Рис. 2.3. Структура документів бази даних вебдодатка [MongoDB](#) використовує [BSON \(Binary JSON\)](#) для представлення даних у вигляді документів та об'єктів, які можуть бути вкладені один в одного. При проектуванні моделі даних у [MongoDB](#) слід розглянути сценарії використання системи та визначити, які запити виникають найчастіше. Дизайн моделі даних повинен враховувати ефективність виконання цих запитів. Модель даних у [MongoDB](#) має відповідати конкретним потребам вебдодатка. В нашому випадку БД представлена трьома документами [User](#), [Truck](#) та [Load](#). [MongoDB](#) для досліджень та навчання пропонує всім бажаним безкоштовний пакет M0, який дозволяє розгорнути цю БД у хмарі [AWS](#), [GCP](#) чи [Azure](#) (рис. 2.4). Після реєстрації виберемо хмарного провайдера [Google Cloud](#) та region [Belgium \(europe-west1\)](#). Далі треба буде додати власну [IP](#)-адресу до списку дозволених, створити логін та пароль адміністратора і отримати строку з параметрами для з'єднання з базою. Рис. 2.4. Створення БД [MongoDB](#) у хмарі Протестувати з'єднання та роботу з [MongoDB](#) можна вже на цьому етапі за допомогою програми [MongoDB Compass](#). Це потужний графічний інтерфейс для цієї БД. Програма безкоштовна, має простий у використанні інтерфейс, який дозволяє легко працювати з даними. Крім цього [Compass](#) має вбудований [shell](#), який дозволяє виконувати команди [MongoDB](#) безпосередньо з нього. Доступні версії для [macOS](#), [Windows](#) та [Linux](#). Фізичне розташування компонентів вебдодатку зручно розглянути на діаграмі розгортання (рис. 2.5). [Angular](#)-клієнт взаємодіє з [Express.js](#)-сервером, який в свою чергу спілкується з базою даних [MongoDB](#). Рис. 2.5. Діаграма розгортання розроблюваної системи На основі діаграми варіантів використання системи, було спроектовано [REST API](#) вебдодатку. Нижче наведено список кількох методів даного [API](#) та їх короткий опис. Кожен метод при виникненні несподіваної помилки в процесі своєї роботи повертає [HTTP 500](#) з повідомленням про помилку. [POST /api/auth/register](#) Реєструє в системі нового користувача (відправника чи водія). [POST /api/auth/login](#) Вхід у систему. Ролі визначаються автоматично. Перевіряє логін та пароль і у разі успіху надсилає у відповідь авторизаційний [jwt](#)-токен. [GET /api/users/me](#) Отримання інформації про профіль користувача. [POST /api/trucks](#) Додає автомобіль у список авто водія. Доступно лише водію. [GET /api/trucks](#) Отримання списку вантажівок для авторизованого користувача (доступно лише для ролі водія). [DELETE /api/trucks/{id}](#) Видаляє вантажівку користувача (доступно лише для ролі водія). [POST /api/trucks/{id}/assign](#) Призначити вантажівку користувачу (доступно лише для ролі водія) [GET /api/loads](#) Отримання списку вантажів для авторизованого користувача, повернення списку завершених і активних вантажів для водія або списку всіх доступних вантажів для відправника. [POST /api/loads](#) Додати вантаж користувача (доступно лише для ролі відправника). [DELETE /api/loads/{id}](#) Видалити вантаж користувача (доступно лише для ролі відправника). [PATCH /api/loads/active/state](#) Перейти до наступного стану доставки вантажу (доступно лише для водія). [GET /test](#) Додаткова точка доступу для тестування вебдодатку без звернення до БД та бізнес-логіки. Метод одразу повертає код 200 та строку

Цитування: **0,01%**

id: **80**

«[hell](#)!».

Програмна реалізація функціоналу додатку Серверну частину вебдодатку розроблено на мові [JavaScript](#) з використанням [Node.js](#) та фреймворку [Express.js](#). Основним інструментом розробки додатка в [Express](#) є використання так званих [middleware](#) – проміжних функцій, що обробляють клієнтські запити. Код розподілений між трьома директоріями: [routers](#) – де описані роути і [HTTP](#)-методи (Лістинг 2.1), [models](#) – в якій описані моделі даних для

MongoDB (Лістинг 2.2), а також controllers – в якій реалізовано бізнес-логіку вебдодатку (Лістинг 2.3). Лістинг 2.1. Роути, що відповідають за реєстрацію користувача. const express = require (id: 81
Цитирования: 0%	
‘express’	
); const router = new express.Router (); const ash = require (id: 82
Цитирования: 0%	
‘express-async-handler’	
); const { register, login, } = require (id: 83
Цитирования: 0,01%	
‘../controllers/authController’	
); router.post (id: 84
Цитирования: 0%	
‘/auth/register’,	
ash(register)); router.post (id: 85
Цитирования: 0%	
‘/auth/login’,	
ash(login)); module.exports = router ; Лістинг 2.2. Створення схеми та моделі даних для MongoDB const mongoose = require (id: 86
Цитирования: 0%	
‘mongoose’	
); const { Schema } = mongoose ; module.exports = mongoose.model (id: 87
Цитирования: 0%	
‘user’,	
new Schema ({ email: { type: String, required: true, unique: true }, password: { type: String, required: true }, ... Лістинг 2.3. Код контролера відправки профайла користувача. const User = require (id: 88
Цитирования: 0,01%	
‘../models/userModel’	
); const getProfile = async (req, res) = { const currentUser = await User.findOne ({ _id: req.user_id }, { _id: 1, email: 1, created_date: 1 }); if (!currentUser) { return res.status (400).json({ message:	id: 89
Цитирования: 0,02%	
‘User not found!’	
}); return res.status (200).json({ user: currentUser}); }); Точка входу в серверну частину вебдодатку – файл index.js . В ньому за допомогою бібліотеки dotenv підтягуються параметри вебдодатку, такі як порт, секрети, параметри з’єднання з БД. Також підключаються загальні для всіх запитів middleware такі як cors – механізм безпеки, який контролює запити браузера до сервера, json – відповідає за форматування даних, morgan – відповідає за логування. Підключаються роутери з директорій, та запускається вебсервер. Аутентифікація користувачів відбувається із використанням популярної бібліотеки JSONWebToken . Це бібліотека для створення (підпису) і перевірки JWT токенів. Перевіривши логін і пароль користувача, контролер генерує унікальний токен, підписує його і надсилає клієнту. В подальшому з кожним запитом клієнт має надсилати цей токен на сервер у заголовку запиту, тим самим автоматично ідентифікуючи себе. У разі відсутності токена або його недейсності сервер буде відповідати помилкою 400, за це відповідає підключений до всіх захищених роутів authMiddleware . Клієнтську частину вебдодатку розроблено на Angular 14 із використанням стилів бібліотеки Angular Material . Для користувача доступна об’єднана форма реєстрації та аутентифікації. На етапі реєстрації користувач визначає свою роль в системі. Після входу в систему користувачу в залежності від його ролі надаються окремі можливості. Водій зможе додати або видалити автівки, переглянути їх список та поточний статус, закріпити за собою автівку, керувати статусом виконання замовлення. Відправник в свою чергу зі свого інтерфейсу має можливість створити або видалити вантаж, переглянути список своїх вантажів, ініціювати пошук виконавця замовлення. В проєкті використовуються модулі, компоненти, сервіси, пайпи, інтерфейси, директиви та інші можливості, що пропонує фреймворк Angular . Реалізовано сервіс спінеру, який сигналізує користувачу про поточну недоступність системи в період звернення до серверу. Спінер є важливим елементом інтерфейсу вебдодатків. Він використовується для показу стану завантаження. Коли користувач виконує дію, яка вимагає певного часу на обробку (наприклад, завантаження даних з сервера), спінер відображається, щоб показати, що процес все ще виконується. Це поліпшує користувацький досвід, оскільки користувачі розуміють, що додаток активно обробляє їхній запит. Без спінера користувач може подумати, що додаток завис або не працює. Бібліотека ngx-toastr задіяна для виведення зручних повідомлень користувачу про успішні результати його дій або помилку. Цей компонент значно покращує користувацький досвід бо дозволяє швидко інформувати користувачів про результати їхніх дій. Наприклад, після натискання кнопки	
Цитирования: 0%	id: 90
«Створити»	
можна відобразити так званий	
Цитирования: 0%	id: 91
«ТОСТ»,	
тобто повідомлення з текстом	
Цитирования: 0,01%	id: 92
«Дані успішно створено».	
Тости з’являються і зникають автоматично, не перериваючи роботу користувача. На відміну від модальних вікон або спливаючих повідомлень вони не вимагають від користувача жодних дій для їх закриття. При роботі з HTTP -запитами в додатку активно використані можливості бібліотеки RxJS (Лістинг 2.4). Ця бібліотека дозволяє керувати асинхронними операціями на більш вищому рівні, надає можливість описати складні асинхронні процеси більш декларативно, що робить код легшим для розуміння та подальшої підтримки. За допомогою таких операторів як map , mergeMap , switchMap можна керувати потоками даних, що дозволяє легко виконувати наприклад послідовні та паралельні HTTP -запити. Лістинг 2.4. Метод видалення автівки водієм. deleteTruck (truckId: string) { this.dialogService.openConfirmationDialog (
Цитирования: 0,03%	id: 93

‘Ви впевнені, що бажаєте видалити цю машину?’,	
Цитирования: 0,01%	id: 94
‘Підтвердження видалення машини’	
<pre>) .pipe(first(), filter((confirmed) = confirmed), switchMap(() = { return this.truckService.deleteTruck(truckId) .pipe(map((answer) = { this.toastr.info(answer.message); this.getTrucksList(); }), catchError(error) = { this.toastr.badRequestError(error); return EMPTY; }) })) .subscribe(); } Також в проєкті використовується об’єкт веббраузера LocalStorage, який дозволяє зберігати пари</pre>	
Цитирования: 0%	id: 95
«ключ-значення»	
<p>в браузері. Дані зберігаються навіть після закриття браузера або перезавантаження комп’ютера. В нашому випадку в LocalStorage зберігається JWT-токен, який клієнт отримує від сервера після аутентифікації. Завдання по використанню JWT-токену в процесі спілкування з сервером покладено на такий клас фреймворку, як інтерсептор. Інтерсептори в Angular – це потужний інструмент, що дозволяє перехоплювати та маніпулювати HTTP-запитами та відповідями. Вони інтегрується в процеси звернення до сервера, дозволяючи модифікувати та інспектувати їх перед тим, як вони досягнуть сервера або клієнта. Інтерсептори можуть виконувати різноманітні завдання. В нашому випадку інтерсептор бере з LocalStorage токен, та формує з нього заголовок, який додається у запити до сервера. Лістинг 2.5. Фрагмент коду інтерсептора вебдодатка.</p> <pre>@injectable() export class JwtTokenInterceptor implements HttpInterceptor { constructor(private readonly localStorageService: LocalStorageService) {} intercept(request: HttpRequest unknown , next: HttpHandler): Observable HttpEvent unknown { const token = this.localStorageService.getToken(); if (token) { const header = ‘Bearer ‘ + token; let headers = request.headers.set(‘Authorization’, header); request = request.clone({ headers }); } return next.handle(request); } }</pre> <p>Незважаючи на те, що клієнтська частина вебдодатку не має істотного значення для запланованого експерименту, практичний досвід розробки на Angular продемонстрував потужність цього фреймворку. Він надає набір інструментів, які суттєво полегшують розробку складних проєктів, що вимагають високої продуктивності, масштабованості та гнучкості. Розроблений проєкт (клієнтську і серверну частини) було завантажено в репозиторій GitHub. Експериментальне дослідження швидкодії На цьому етапі в нас вже є працююча БД MongoDB, яку ми створили і підключили в процесі розробки тестового вебдодатку. Тому почнемо зі створення віртуального серверу на платформі GCP. Як вже зазначалося, GCP пропонує безкоштовний термін користування своїми сервісами</p>	
Цитирования: 0,01%	id: 96
« Free Tier ».	
<p>GCP надає ці послуги безкоштовно, щоб допомогти користувачам досліджувати платформу та запропонувати безпечний спосіб експериментувати з хмарними обчисленнями. В рамках Free Tier GCP пропонує широкий спектр послуг яких буде цілком достатньо для проведення запланованого нами експерименту. Підготовка тестового сценарію та процедура його виконання Після проходження процедури реєстрації, в консолі GCP необхідно перейти в меню</p>	
Цитирования: 0,01%	id: 97
« Compute Engine »	
для створення за допомогою кнопки	
Цитирования: 0,01%	id: 98
« Create Instance »	
<p>нового екземпляру серверу. Враховуючи що БД нашого проєкту розташована в зоні europe-west1 саме в цій зоні і створюємо віртуальну машину. Серед наявних пресетів обираємо конфігурацію High CPU - e2-highcpu-8, яка представляє собою 4 ядерний 8 поточний процесор з 8 Гб оперативної пам’яті (рис. 2.6). В розділі</p>	
Цитирования: 0,01%	id: 99
« Boot disk »	
за допомогою кнопки	
Цитирования: 0%	id: 100
« Change »	
змінюємо образ операційної системи за замовчуванням на образ	
Цитирования: 0,02%	id: 101
« Debian GNU/Linux 12 (bookworm) ».	
В розділі	
Цитирования: 0%	id: 102
« Firewall »	
пропонуємо чекбокси, дозволяючи всі види трафіку і за допомогою кнопки	
Цитирования: 0%	id: 103
« Create »	
<p>запускаємо процес створення віртуальної машини. Рис. 2.6. Створення екземпляру віртуальної машини в GCP Про успішне створення VM система GCP сигналізує за допомогою повідомлень та відображаючи зелену позначку поряд з назвою нового серверу. Підключитися до віртуальної машини зручно за допомогою вбудованого SSH-клієнта. Для цього біля кнопки</p>	
Цитирования: 0%	id: 104
« SSH »	
обираємо	
Цитирования: 0,02%	id: 105
« Open in browser window »	
<p>та у вікні що відкривається підтверджуємо під’єднання до серверу (рис. 2.7). Рис. 2.7. Підключення до VM по SSH Отримавши доступ до консолі сервера необхідно виконати певні оновлення списків доступних пакетів програмного забезпечення з офіційних репозиторіїв, а потім встановити ці оновлення. Для цього послідовно виконуємо в консолі команди: sudo apt update sudo apt upgrade Оновлення системи зазвичай займає доволі довгий час, приблизно 15 хвилин. Після успішного оновлення необхідно встановити наступні програми та утиліти, що допоможуть у проведенні експерименту: mc - Midnight Commander, файловий менеджер із текстовим інтерфейсом схожий на Norton Commander для UNIX-подібних операційних систем git – система менеджменту версій ПЗ curl – популярна службова програма, що дозволяє взаємодіяти з серверами, наприклад створювати запити</p>	

unzip – архіватор nvm – Node Version Manager, утиліта, що дозволяє встановлювати, переключати та видаляти різні версії Node.js. Дає можливість тримати на одній машині будь-яку кількість версій Node Node.js – програмна платформа Для встановлення цих програм послідовно виконуємо в консолі наступні команди: sudo apt install mc -y sudo apt install git -y sudo apt install curl -y sudo apt install unzip -y curl https://raw.githubusercontent.com/creationix/nvm/master/install.sh | bash source ~/.bashrc nvm install 20.17.0 Наступним етапом встановлюємо платформу Bun: curl -fsSL https://bun.sh/install | bash -s

Цитирования: 0,01%

id: 106

"bun-v1.1.29"

source ~/.bashrc Перевіряємо роботу встановлених платформ за допомогою команд node -v та bun -v (рис. 2.8). Рис. 2.8. Перевірка встановлених версій платформ Node та Bun Також послідовно виконуємо наступні інсталяції: завантажуюмо архів з мовою програмування Go wget https://dl.google.com/go/go1.22.0.linux-amd64.tar.gz розпаковуємо цей архів та додаємо шлях до інтерпретатора Go у змінні середовища: sudo tar -C /usr/local -xzf go1.22.0.linux-amd64.tar.gz export PATH=\$PATH:/usr/local/go/bin В результаті по команді go version маємо отримати інформацію про версію встановленого інтерпретатора мови Go (рис. 2.9). Рис. 2.9. Перевірка встановленого інтерпретатора Go Цей інтерпретатор потрібен в системі для використання тестової утиліти Bombardier, останню версію якої необхідно встановити командою: go install github.com/codesenberg/bombardier@latest Для зручного запуску утиліти Bombardier можна додати шлях до неї у змінні середовища командою: export PATH=\$PATH:/home/ ваша домашня директорія /go/bin На цьому етапі підготовку середовища експерименту можна вважати завершеною. Тепер ми маємо встановити на сервер тестовий вебдодаток та перевірити його правильну роботу. Вихідний код вебдодатку завантажуюмо з серверу GitHub, використовуючи команду git clone https://github.com/dmytro-ost/mean.git Перейдемо у відповідні каталоги з фронтенд та бекенд частинами вебдодатку та встановимо необхідні залежності (рис. 2.10): cd mean/frontend npm ci cd ../backend npm ci Рис. 2.10. Встановлення залежностей Створюємо командою touch .env файл з налаштуваннями порту доступу, параметрів з'єднання з БД MongoDB, та константою JWT_SECRET (рис. 2.11). Це має бути індивідуальна конфіденційна інформація наступного формату: PORT=8080 DB_CONNECT_STRING=mongodb+srv://dmytro строка від MongoDB JWT_SECRET=

Цитирования: 0%

id: 107

'JustAnyWhere'

Рис. 2.11. Налаштування серверу за допомогою файлу .env В процесі розробки вебдодатку ми використовували локальний сервер localhost. В реальних умовах, для того щоб Angular робив запити до бекенд частини додатку, дані про наш працюючий сервер необхідно ввести в конфігураційний файл фронтенд частини вебдодатку. Для цього переходимо в директорію командою cd ../frontend/src/config та в файлі default.json змінюємо localhost на зовнішню IP адресу нашої віртуальної машини (рис. 2.12), в нашому випадку це

Цитирования: 0,01%

id: 108

"localhost":

"http://34.140.154.75:8080". Рис. 2.12. Налаштування клієнту за допомогою файлу default.json На цьому налаштування фронтенд частини вебдодатку можна вважати завершеними. Тепер знаходячись в директорії frontend ми маємо запустити процес створення білду проєкту, тобто загальної зборки фронтенд частини (рис. 2.13). Це необхідно для транспіляції та перетворення всіх вихідних файлів, стилів, бібліотек у оптимізовані html, js та css файли. Запускаємо цей процес командою npm run build. Рис. 2.13. Зборка Angular – клієнтської частини вебдодатку Після завершення процесу зборки в директорії frontend/dist/frontend знаходяться готові файли. Перенесемо їх всі без виключення разом із піддиректоріями в серверну частину вебдодатку (рис. 2.14), а саме в директорію backend/public. Рис. 2.14. Перенос клієнтських файлів в робочу директорію сервера Express Перед запуском вебдодатку залишилось виправити ще одну деталь. Справа в тому, що вебсервер Express згідно з нашими налаштуваннями прослуховує порт 8080. Для того щоб до цього порту був доступ із зовнішніх мереж ми маємо налаштувати вбудований в GCP файрвол, а саме надати дозвіл на використання цього порту. В головному меню консолі GCP треба вибрати розділ

Цитирования: 0,01%

id: 109

«VPC network»

підрозділ

Цитирования: 0%

id: 110

«Firewall».

Налаштування файрволу надзвичайно важлива для безпеки і тому відповідальна задача, яка потребує певного досвіду і навичок. Проте для проведення запланованого експерименту нам достатньо лише створити правило, яке б дозволяло з'єднання за протоколом TCP по порту 8080. Нове правило створюється кнопкою

Цитирования: 0,02%

id: 111

«Create a firewall rule»

(рис. 2.15). Треба вказати ім'я правила, наприклад

Цитирования: 0%

id: 112

«allow-8080».

В полі

Цитирования: 0%

id: 113

«Targets»

вибираємо

Цитирования: 0,02%

id: 114

«All instances in the network»,

в полі

Цитирования: 0,01%

id: 115

«Source IP: 4 ranges»

пишемо 0.0.0.0/0. В розділі

Цитирования: 0,01%

id: 116

«Protocols and ports»

вибираємо TCP та порт 8080. Після чого натискаємо кнопку

Цитирования: 0%

id: 117

«[route](#)».

Рис. 2.15. Налаштування правил [firewall](#) Запуск вебдодатку відбувається з [SSH](#)-консолі віртуальної машини, з директорії [mean/backend](#) командою [node index.js](#) або у випадку використання замість [Node.js](#) платформи [Bun](#) – [bun index.js](#). Після чого в браузері власного комп’ютера можна перейти на [IP](#)-адресу нашої віртуальної машини, наприклад [http://34.140.154.75:8080](#). В результаті ми маємо побачити працюючий вебдодаток. При першому запуску вебдодатку виявилось, що він добре працює на платформі [Node.js](#), проте виникає помилка при спробі запуску того самого вебдодатку на платформі [Bun](#) (рис. 2.16).

Рис. 2.16. Помилка при запуску вебдодатку на платформі [Bun](#) Цей факт вже свідчить про відсутність заявленої розробниками повної сумісності проєктів, написаних і працюючих під [Node](#) з новою платформою. Проте для перевірки швидкодії [Bun](#) спробуємо змінити код вебдодатку так, щоб його можна було запустити під обома платформами. В інформаційному повідомленні про помилку йдеться про те, що система не може знайти модуль [mongodb-extjson](#). Проте як виявилось після встановлення цього модуля ланцюгом додалася ще низка помилок які після кількох спроб так і не вдалося виправити. З повідомлення про помилку також зрозуміло, що проблема пов’язана з модулем [mongoose-morgan](#), який використовується для запису журналу дій в БД [MongoDB](#). Цей функціонал є скоріше додатковою можливістю вебдодатку, тому спробуємо його відключити і запустити вебдодаток без несумісного з [Bun](#) модуля (рис. 2.17). Для цього закоментуємо кілька строк коду, що відповідає за імпорт і застосування модулю [mongoose-morgan](#) в файлі [/backend/index.js](#). Рис. 2.17. Виключення з коду сервера модуля [mongoose-morgan](#) Видалення несумісного з [Bun](#) модулю привело до відновлення працездатності вебдодатку. Тепер запуск сервера як в середовищі [Node.js](#) так і на платформі [Bun](#) відбувається без помилок (рис. 2.18). Рис. 2.18. Перевірка запуску сервера в середовищах [Node](#) та [Bun](#) Зареєструємо в нашому вебдодатку нового користувача, наприклад в якості перевізника. Для цього перейдемо на головну сторінку за [IP](#)-адресою, виберемо відповідну вкладку, заповнимо форму та натиснемо кнопку

Цитирования: 0% id: 118

«Зареєструватися»

(рис. 2.19). Рис. 2.19. Реєстрація нового користувача у вебдодатку Після реєстрації додаємо в панель перевізника вебдодатку кілька автомобілів різного формату (рис. 2.20). Рис. 2.20. Додавання нового автомобіля Після кожного успішного додавання авто маємо отримати повідомлення від системи. В результаті в розділі

Цитирования: 0,01% id: 119

«Мої машини»

буде відображено список автомобілів, що знаходяться у власності користувача і які він пропонує для здійснення вантажних перевезень (рис. 2.21). Рис. 2.21. Список dodаних авто користувача Натиснувши на кнопку [F12](#) запустимо інструменти розробника [Chrome](#). Вкладки

Цитирования: 0% id: 120

«Application»

цієї панелі містить розділ

Цитирования: 0% id: 121

«Storage».

В ньому, в частині

Цитирования: 0,01% id: 122

«Local storage»,

зберігається [JWT](#)-токен за допомогою якого відбувається авторизація в нашому вебдодатку. Його можна легко знайти за допомогою ключа

Цитирования: 0% id: 123

«Access token»

(рис. 2.22), який ми визначили при розробці [Angular](#) додатку. Скопіюємо цей унікальний код для подальшого використання у тестах. Рис. 2.22. Отримання [JWT](#)-токену Для проведення тестів потрібно створити ще одну [SSH](#)-консоль із доступом до серверу. В одному терміналі буде запускатися сервер вебдодатку, а інший потрібен для старту утиліти [Bombardier](#), за допомогою якої буде відбуватися тестування. Подальший алгоритм дій буде наступний: В першому терміналі запускаємо сервер вебдодатку командою [node index.js](#) В другому терміналі за допомогою утиліти [curl](#) створюємо тестовий запит до ендпоінту [localhost:8080/api/trucks/](#). Передаємо в авторизаційному заголовку отриманий із браузера [JWT](#)-токен (рис. 2.23) [curl --header "Authorization: Bearer eyJhbGciOiJIUzI1NiJ9.eyJfaWQiOiI2NWUxOGU4YmM3ZmRlNDQ5YmQ4OGQ5NDQiLCJyb2xkljoifRFlJkVksVSIiwZW1haWwiOiJkbXl0cm9AZ2114pyvmAwIxsxv2DKqIzOkQ8r5dlkG41WeYag" localhost:8080/api/trucks/](#) Рис. 2.23. Запуск серверної частини вебдодатку та тест ендпоінту Переконавшись, що сервер відповідає на запит без помилок (надсилає [JSON](#) повідомлення із списком автомашин користувача), запускаємо утиліту [Bombardier](#) (рис. 2.24). В параметрах утиліти вказуємо, що тестування має проходити 180 секунд, з використанням 125 одночасних з’єднань, з максимальним таймаутом 5 секунд. Також в параметрах вказуються [JWT](#)-токен, метод запиту та адреса на яку будуть відправлятися запити: [bombardier -c 125 -d 180s --method=GET --timeout=5s -header=" Authorization: Bearer eyJhbGciOiJIUzI1NiJ9.eyJfaWQiOiI2NWUxOGU4YmM3ZmRlNDQ5YmQ4OGQ5NDQiLCJyb2xkljoifRFlJkVksVSIiwZW1haWwiOiJkbXl0cm9AZ2114pyvmAwIxsxv2DKqIzOkQ8r5dlkG41WeYag" localhost:8080/api/trucks/](#) Рис. 2.24. Робота утиліти [Bombardier](#) Після завершення роботи утиліти [Bombardier](#) фіксуємо середню кількість [Requests per second \(RPS\)](#), тобто запитів на секунду та вносимо результати до таблиці. Формуємо наступний запит до серверу, без використання токену і [MongoDB](#). Тестуємо відповідь серверу за допомогою утиліти [curl curl localhost:8080/test](#) В разі отримання успішної відповіді зі строкою

Цитирования: 0,01% id: 124

«Hello!»

запускаємо тест за допомогою утиліти [Bombardier bombardier -c 125 -d 180s --method=GET -timeout=5s localhost:8080/test](#) Після завершення роботи утиліти [Bombardier](#) вносимо результати до таблиці. Зупиняємо роботу серверу в першому терміналі. Запускаємо роботу серверу в першому терміналі проте тепер вже в середовищі [Bun](#) командою [bun index.js](#). Повторюємо пункти 2–8. В результаті отримуємо таблицю, в якій вказано середню кількість запитів на секунду для платформи [Node](#) що працює з [MongoDB](#),

Цитирования: 0% id: 125

«чистої»

[Node](#) без запитів до БД, платформи [Bun](#) в зв’язці з [MongoDB](#) (рис. 2.25) та

Цитирования: 0% id: 126

«чистої»

Bun. Рис. 2.25. Тестування утилітою **Bombardier** платформи **Bun** з метою отримання більш точних результатів було проведено 10 циклів тестування [32]. Результати тестування швидкодії наведено у табл. 2.3. Таблиця 2.3 Результати тестування швидкодії платформ утилітою **Bombardier** Спроба № **Node** + **MongoDB**, **RPS Node**, **RPS Bun** + **MongoDB**, **RPS Bun**, **RPS** 1 411,28 3718,61 796,50 8006,35 2 425,27 3750,95 840,50 7988,08 3 433,59 3672,13 841,35 8131,81 4 434,39 3816,12 826,36 8047,98 5 375,09 3700,46 840,89 7942,80 6 397,03 3761,59 812,23 8023,33 7 435,31 3761,15 831,14 8064,75 8 426,16 3851,21 807,65 7995,45 9 431,12 3821,45 838,64 8101,45 10 412,19 3677,75 821,17 8075,46 В середньому, **RPS**: 418,14 3753,14 825,64 8037,75 Аналіз результатів дослідження швидкодії Як видно з таблиці результатів (табл. 2.3) і наведеної діаграми (рис. 2.26) експеримент на вебдодатку, що використовує в процесі своєї роботи базу даних **MongoDB**, продемонстрував доволі низьку кількість оброблених запитів на обох досліджуваних платформах. Рис. 2.26. Кількість оброблених запитів на секунду на експериментальних платформах Це пояснюється вузьким місцем системи, а саме БД **MongoDB**. Незважаючи на те, що база даних, як і віртуальна машина, розміщувалася в **GCP** і дослідження проводилося в одній зоні **europa-west1**, безкоштовний пакет від **MongoDB** хоч і дозволяє повноцінну роботу з базою, проте не передбачає можливості тестування системи під навантаженням. Саме ці обмеження і демонструють результати тестування. Фактично **MongoDB** в безкоштовному режимі дозволила надсилати біля 825 запитів на секунду, що доволі непоганий результат і цілком підходить як для тестування так і для роботи не надто навантажених вебдодатків. Проте навіть за умов цих обмежень платформа **Bun** обробила в 1,97 раз більше запитів ніж **Node**. Виключення з **Express** логіки, що робила запити до **MongoDB**, дозволило виокремити роботу саме платформ і відмежуватися від БД. Як видно з наведеної діаграми в такій конфігурації вебдодатку платформа **Bun** продемонструвала ще більше підвищення продуктивності в 2,14 раз у порівнянні з **Node**, обробляючи в середньому 8038 запитів на секунду. Це доволі вражаючий результат, незважаючи на те, що ми не підтвердили анонсоване розробниками платформи **Bun** підвищення продуктивності у 4,8 рази. Зазвичай розробники створюють особливі умови, що демонструють їх рішення в більш вигідних позиціях. Навпаки, наш експеримент проводився в наближених до реальної роботи умовах, без певних оптимізацій. Слід також зазначити, що анонсована розробниками **Bun** повна підтримка додатків що працюють під **Node** в ході дослідження виявила певні обмеження. Вебдодаток, що працював під **Node** не запустився в **Bun**, з повідомленням про помилку в модулі і почав працювати лише після виключення цього модулю з коду. Тобто платформа **Bun** все ще знаходиться на шляху вдосконалення і виправлення помилок. **Bun** в контексті **MEAN** стеку: перспективи та можливості Використання платформи **Bun** в розробці вебдодатків може суттєво підвищити їх продуктивність. **Bun** демонструє значно кращі результати навіть на неоптимізованому під нього проєкті. Це свідчить про величезний потенціал **Bun** особливо для проєктів, де велике навантаження є критичним аспектом. Той факт, що підтримка в **Bun** коду розробленого для **Node** має на даному етапі певні обмеження вказує на те, що платформа все ще знаходиться в процесі вдосконалення. Існування таких обмежень може призвести до додаткових витрат часу на коригування та тестування коду для розробників, що мають бажання перейти з **Node** на **Bun**. Проте, з огляду на значне підвищення продуктивності системи, тимчасові незручності можна вважати виправданим компромісом. Результати, отримані в ході дослідження, ясно показують, що **Bun** має значний потенціал для майбутнього веброботи. Зі зростанням навантаження на мережеві ресурси та збільшенням потреб у швидкісному доступі до даних, платформи, здатні обробляти велику кількість запитів за одиницю часу, стануть незамінними. У той же час, розвиток та вдосконалення **Bun** мають продовжуватися, зокрема, шляхом поліпшення сумісності з розробленими під **Node** додатками та виправленням існуючих обмежень. На цей час можна сказати, що **Bun** розкриває нові горизонти в розробці вебдодатків, пропонуючи рішення, яке може значно збільшити їх продуктивність та ефективність. Висновки до розділу 2 Результати дослідження свідчать про наявність важливих відмінностей в роботі додатків на різних платформах, які можуть впливати на вибір стеку технологій для розробки. Встановлено, що вебдодаток, розроблений на **MEAN** стеку, виявив себе непрацездатним на платформі **Bun**, вимагаючи додаткових зусиль для його адаптації. Це може свідчити про те, що інтеоперабельність між платформами **Node** та **Bun** може бути проблемою та вимагати детальної перевірки та налаштувань для безперебійної роботи. Другий важливий висновок полягає в тому, що в певних умовах вебдодаток, розроблений на **MEAN** стеку і запущений на платформі **Bun**, продемонстрував вражаючий приріст продуктивності порівняно з варіантом на **Node**. Швидкісний тест за допомогою утиліти **Bombardier** показав, що працюючи на **Bun** вебдодаток обробив в 2,14 раз більше запитів ніж його копія, що працює на **Node**. Це може свідчити про ефективність та оптимізацію, які забезпечує платформа **Bun**, що робить її привабливим варіантом для високонавантажених вебдодатків. Тести з використанням **MongoDB** показали менш вражаючі результати: **Bun** обробив в 1.97 рази більше запитів на секунду, ніж **Node.js**. Така різниця в результатах пояснюється тим, що сама база даних **MongoDB** стала вузьким місцем системи. Безкоштовний пакет **MongoDB**, який використовувався в експерименті, хоч і дозволяє повноцінну роботу з базою, але має обмеження щодо навантаження. В результаті **MongoDB** дозволила надсилати біля 825 запитів на секунду, що обмежило можливості як **Node.js**, так і **Bun**. Фактично, **Bun** в тестах з **MongoDB** працював на рівні пропускної здатності бази даних, а не на повну свою потужність. Обмеження безкоштовного пакету **MongoDB** діляти в обох випадках, проте **Bun** + **MongoDB** все ж таки показав кращий результат ніж **Node.js** + **MongoDB**. Це свідчить про те, що **Bun**, навіть працюючи на рівні пропускної здатності бази даних, все ж таки ефективніше взаємодіє з нею ніж **Node.js**. Ймовірно, це пов'язано з архітектурними особливостями **Bun** та його оптимізаціями, які дозволяють швидше обробляти запити та ефективніше використовувати ресурси системи. Наприклад, **Bun** має вбудований **JavaScriptCore** (від **WebKit**), який може бути швидшим за **V8** (від **Google**), що використовується в **Node.js**, для певних типів завдань, зокрема пов'язаних з мережевими операціями. Крім того, **Bun** написаний на мові **Zig**, яка відома своєю продуктивністю та ефективністю в керуванні пам'яттю. Це може призводити до менших накладних витрат при обробці запитів та взаємодії з **MongoDB**. Не слід також виключати, що **Bun**, як нова платформа, може мати більш сучасну та оптимізовану реалізацію драйвера для **MongoDB**, що дозволяє йому ефективніше взаємодіяти з базою даних. Враховуючи результати експерименту, є підстави розглядати платформу **Bun** як перспективну альтернативу **Node**, зокрема для проєктів, де вимоги до продуктивності відіграють ключову роль. Таким чином, нова комбінація стеку **MEAB** (**MongoDB**, **Express.js**, **Angular**, **Bun**) – може виявитися привабливою для розробників, які шукають оптимізовану та ефективну альтернативу у своїх проєктах, не зважаючи на поточні труднощі та ризики. Під час створення конкретної реалізації вебдодатку на стеку **MEAN** було враховано всі ключові елементи стеку, а саме використано **MongoDB** для бази даних, **Express.js** для серверної частини, **Angular** для клієнтського інтерфейсу та **Node.js** як середовище виконання. Детально описано функціонал додатку, його основні можливості та взаємодію з користувачем. Зазначимо, що обране технологічне рішення дозволило ефективно реалізувати необхідний функціонал

вебдодатку, а також забезпечити зручний та високопродуктивний користувацький інтерфейс. Важливим є той факт, що стек [MEAN](#) дозволяє створювати вебдодатки, які легко масштабуються та підтримуються, що є ключовим аспектом для розвитку та управління проектами в сучасному середовищі веброзробки. До того ж для розробки достатньо знання лише однієї мови програмування – [JavaScript](#). Необхідно врахувати, що результати експерименту є специфічними для даного вебдодатку та його конкретного використання. При виборі технологій для веброзробки слід враховувати потреби та характеристики конкретного проекту, а також здатність розробників адаптувати та оптимізувати код під конкретну платформу. Загалом, отримані результати стимулюють подальший аналіз та дослідження для кращого розуміння причин виявлених різниць у продуктивності та можливості оптимізації додатків для різних платформ в межах стеку [MEAN](#).

ВИСНОВКИ В магістерському дослідженні представлена генеза технологій створення вебдодатків, проведено аналіз наукових, спеціалізованих та популярних джерел, що стосуються вебдодатків, веброзробки та стеку технологій [MEAN](#). Робота висвітлює важливість інтернет-технологій та їх вплив на сучасне суспільство. Підкреслено роль Інтернету як основи для функціонування інформаційного суспільства та важливість глибшого розуміння, аналізу динаміки і впливу інтернет-простору на сучасний світ. На основі аналізу генези технологій створення вебдодатків нами вперше запропоновано модель рівнів зрілості вебдодатків, яка виокремлює чотири рівні від статичних сторінок до вебдодатків з використанням сучасних технологій, та сприяє глибшому розумінню етапів розвитку веброзробки та визначенню різниці між різними формами вебресурсів. Досліджено технології створення вебдодатків. Стек [MEAN](#) ([MongoDB](#), [Express.js](#), [Angular](#), [Node.js](#)), визначено як важливий і популярний вибір для сучасної веброзробки. Цей стек технологій дозволяє створювати вебдодатки, які легко масштабуються та підтримуються, що є ключовим аспектом для розвитку та управління проектами в сучасному середовищі веброзробки. Експериментально доведено, що вебдодаток стеку [MEAN](#), перенесений на платформу [Bun](#), може продемонструвати значний приріст продуктивності порівняно з його роботою на [Node.js](#). Це може свідчити про ефективність та оптимізацію, які забезпечує платформа [Bun](#), що робить її привабливим варіантом для розробки високонавантажених вебдодатків. Результати експерименту є специфічними для даної реалізації вебдодатку. При виборі стеку технологій для веброзробки слід враховувати потреби та характеристики конкретного проекту, а також здатність розробників адаптувати та оптимізувати код під конкретну платформу. Платформу [Bun](#) можна розглядати як перспективну альтернативу [Node](#), зокрема для проектів, де вимоги до продуктивності відіграють ключову роль. Таким чином, нова комбінація стеку [MEAB](#) ([MongoDB](#), [Express.js](#), [Angular](#), [Bun](#)) – може виявитися привабливою для розробників, які шукають оптимізовану та ефективну альтернативу у своїх проектах, не зважаючи на можливі труднощі та ризики. Дослідження важливе для розуміння сучасних тенденцій в області веброзробки та може бути корисним для розробників, які шукають оптимальні стеки технологій для своїх проектів. Результати роботи є підґрунтям для подальших пошуків у вивченні особливостей технологій стеку [MEAN](#) та їх перспектив, підкреслюють важливість подальшого дослідження та аналізу для кращого розуміння причин виявлених різниць у продуктивності та можливості оптимізації вебдодатків для різних платформ в межах стеку [MEAN](#).

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ Базові поняття і терміни веб-технологій / [А. В. Кільченко, О. І. Поповський, О-р В. Тебенко, О-й. В. Тебенко, Н. М. Матросова]; Упорядник: Кільченко А. В. – К. : ІІТЗН НАПН України, 2014. – 49 с. Войтюк О. В., Плечистий Д. Д. Оптимізація промальовування вебзастосунку на основі об'єктів з глибокою вкладеністю та багатозалежними зв'язками. Технічна інженерія. 2023. № 1(91). С. 140–145. URL: [https://doi.org/10.26642/ten-2023-1\(91\)-140-145](https://doi.org/10.26642/ten-2023-1(91)-140-145) (дата звернення: 05.03.2024). Зосімов В. В. Моделі та засоби інтелектуальної обробки даних корпоративних веб-ресурсів : автореф. дис. д-ра техн. наук. Київ, 2019. 43 с. Інформаційна технологія зберігання та візуалізації даних гідрометеорологічного прогнозування на основі [WRF](#)-Україна / С. Я. Майстренко, Т. О. Донцов-Загреба, К. В. Хурцилава, М. І. Харчук, С. В. Грибков, І. В. Ковалець // Математичні машини і системи. — Київ : ІПММС НАНУ, 2019. — № 1. — С. 56–67. Матвеева Н., Нусс В. Використання фреймворку [nuxt.js](#) 3 для розробки веб-додатку. [Grail of science](#). 2023. № 25. С. 198–202. URL: <https://doi.org/10.36074/grail-of-science.17.03.2023.031> (дата звернення: 05.03.2024). Особливості розробки [web](#)-застосунків для системи дистанційного навчання з допомогою бібліотеки [react](#) / О. П. Кошова та ін. [Systems and technologies](#). 2023. Т. 65, № 1. С. 20–31. URL: <https://doi.org/10.32782/2521-6643-2023.1-65.3> (дата звернення: 05.03.2024). Острецов Д. І. Модель рівнів зрілості вебдодатків: від статичних сторінок до сучасних фреймворків. Стан, досягнення та перспективи інформаційних систем і технологій : Матеріали XXIV Всеукр. науково-техн. конф. молодих вчен., аспірантів та студентів, м. Одеса, 18–19 квіт. 2024 р. Одеса, 2024. С. 258–259. URL: https://www.ontu.edu.ua/download/konfi/2024/Conference_abstract-IT-2024.pdf. Острецов Д. Модель рівнів зрілості вебдодатків: ключові етапи еволюції. Автоматизація та комп'ютерно-інтегровані технології у виробництві та освіті: стан, досягнення, перспективи розвитку : Матеріали Всеукр. науково-практ. [Internet](#)-конф., м. Черкаси, 11–17 берез. 2024 р. Черкаси, 2024. С. 315–317. URL: https://conference.ikto.net/pub/akit_2024_11-17_march_1.pdf. Острецов Д., Переславська С. Етапи еволюції вебдодатків: модель рівнів зрілості. Актуальні аспекти розвитку [STEAM](#)-освіти в умовах євроінтеграції : зб. матеріалів II Міжнар. науково-практ. інтернет-конф., м. Кропивницький, 26 квіт. 2024 р. : ДонДУВС, 2024. С. 56–58. URL: https://dnuvs.ukr.education/wp-content/uploads/2024/06/zbirnyk_tez_konferencziya_steam_26_04_2024.pdf.

Про авторське право і суміжні права : Закон України від 01.12.2022 р. № 2811-IX : станом на 15 квіт. 2023 р. URL: <https://zakon.rada.gov.ua/laws/show/2811-20#Text> (дата звернення: 05.03.2024). Про врахування висловлених органом державної реєстрації зауважень до наказу Міністерства цифрової трансформації України від 23 червня 2022 року № 57 : Наказ М-ва цифр. трансформації України від 21.07.2022 р. № 67. URL: <https://zakon.rada.gov.ua/laws/show/20828-22#Text> (дата звернення: 05.03.2024). Руденко О. Аналітика [djinni](#) за січень 2024. Лампа, блог [Djinni](#). URL: <https://blog.djinni.co/post/january-2024-analytics> (дата звернення: 05.03.2024). Слабінога М. О., Чабан С. В. Розробка веб-додатків в контексті оптимізації їх швидкодії. Таврійський науковий вісник. Серія: технічні науки. 2022. № 3. С. 63–69. URL: <https://doi.org/10.32851/tnv-tech.2022.3.7> (дата звернення: 05.03.2024). Словник української мови : у 20 т. / НАН України, Укр. мовно-інформ. фонд. — Київ : Наук. думка, 2010-. — (Словники України - Державна програма розвитку Національної словникової бази України). Т. 5 : 3 – Зв'язати / уклад.: І. В. Шевченко та ін. ; наук. ред. О. О. Тараненко], 2014. — 991 с. : табл. — Бібліогр. у підрядк. прим. — Покажч. словосполучень: с. 834–933. Тлумачний словник з інформатики / Г.Г. Півняк, Б.С. Бусигін, М.М. Дівізюк та ін. – Д., Нац. гірнич. ун-т, 2010. – 600 с. Ужейко С. О. Способи відображення показників вітрової та сонячної енергії / С. О. Ужейко // Відновлювана енергетика. - 2017. - № 2. - С. 34–40. - Режим доступу: http://nbuv.gov.ua/UJRN/vien_2017_2_6. Український правопис / ред.: Є. І. Мазніченко та ін. Наук. думка, 2019. 390 с. Федько В. В., Безуглий Д. Є. Побудова веб застосунків на основі конструктивних елементів. Системи обробки інформації. 2020. № 1(160),. С. 123–127. URL: <https://doi.org/10.30748/soi.2020.160.16> (дата звернення: 05.03.2024). Щербань В.Ю. Методи представлення, збереження та аналізу даних інформаційних систем / В.Ю. Щербань, С.М. Краснитський, Т.І. Астістова, В.М. Яхно. – К.: ТОВ

» Цитування: 0,01%	id: 127
"Фастбінд Україна",	
<p>2023. – 472 с. Baker M. Secure web application development: a hands on guide with python and django. Apress L. P., 2023. Baresi L., Garzotto F., Paolini P. From web sites to web applications: new issues for conceptual modeling. Conceptual Modeling for E-Business and the Web. Berlin, Heidelberg, 2000. С. 89–100. URL: https://doi.org/10.1007/3-540-45394-6_9 (дата звернення: 05.03.2024). Bartlett J. Creating a simple web app. Building scalable PHP web applications using the cloud. Berkeley, CA, 2019. С. 43–55. URL: https://doi.org/10.1007/978-1-4842-5212-3_4 (дата звернення: 05.03.2024). Belfrage A. Garden history and the web: dipping, dabbling, and diving. Australian garden history. 2011. 1 жовт. С. 13–16. URL: https://www.jstor.org/stable/24918777 (дата звернення: 05.03.2024). Berners-Lee T. The original proposal of the WWW, HTMLized. W3C. URL: https://www.w3.org/History/1989/proposal.html (дата звернення: 05.03.2024). Berners-Lee T., Connolly D. Hypertext markup language - 2.0. RFC Editor, 1995. URL: https://doi.org/10.17487/rfc1866 (дата звернення: 05.03.2024). Bielak K., Borek B., Plechawska-Wójcik M. Web application performance analysis using Angular, React and Vue.js frameworks. Journal of computer sciences institute. 2022. Т. 23. С. 77–83. URL: https://doi.org/10.35784/jcsi.2827 (дата звернення: 05.03.2024). Boulton J. The nexus browser digital archaeology. Digital Archaeology Documenting the formative years of digital culture, raising the profile of web archiving, @jim_boulton. URL: https://digital-archaeology.org/the-nexus-browser/ (дата звернення: 05.03.2024). Brooks D. W. Web-teaching: A guide for designing interactive teaching for the World Wide Web. 2-ге вид. New York : Kluwer Academic/Plenum Publishers, 2001. 331 с. Brown E. Web development with node and express. O'Reilly Media, Incorporated, 2014. Bun. Bun – A fast all-in-one JavaScript runtime. URL: https://bun.sh/ (дата звернення: 22.09.2024). Cincović, J., Delčev, S., Drašković, D. Architecture of web applications based on Angular Framework: A Case Study. In: Konjović, Z., Zdravković, M., Trajanović, M. (Eds.) ICIST 2019 Proceedings, pp.254-259, 2019. D.Ostretsov. Experiment Test Bun vs Node, 2024. YouTube. URL: https://www.youtube.com/watch?v=57wQpglceic (дата звернення: 14.10.2024). Development of an E-Commerce System using MEAN Stack with NX Monorepo / S. L. J. Shabu та ін. 2023 7th international conference on trends in electronics and informatics (ICOEI), м. Tirunelveli, India, 11–13 квіт. 2023 р. 2023. URL: https://doi.org/10.1109/icoei56765.2023.10125948 (дата звернення: 05.03.2024). Flanagan D. JavaScript: the definitive guide / ред. P. Ferguson. 4-те вид. Sebastopol, CA : O'Reilly, 2002. 916 с. Fowler S. L. Web application design handbook: best practices for web-based software. Amsterdam : Morgan Kaufmann Publishers, 2004. 658 с. Garcia V. H. Introduction to angular framework. Getting started with angular. Berkeley, CA, 2023. С. 1–11. URL: https://doi.org/10.1007/978-1-4842-9206-8_1 (дата звернення: 05.03.2024). Haan K. Top website statistics for 2023. Forbes Advisor. URL: https://www.forbes.com/advisor/business/software/website-statistics/ (дата звернення: 05.03.2024). Hoffman A. Web application security: exploitation and countermeasures for modern web applications. O'Reilly Media, 2020. 330 с. HTML+ (Hypertext markup format). W3C. URL: https://www.w3.org/MarkUp/HTMLPlus/htmlplus_1.html (дата звернення: 05.03.2024). Hypertext transfer protocol (HTTP/1.1): message syntax and routing / ред.: R. Fielding, J. Reschke. RFC Editor, 2014. URL: https://doi.org/10.17487/rfc7230 (дата звернення: 05.03.2024). Internet and social media users in the world 2024 Statista. Statista. URL: https://www.statista.com/statistics/617136/digital-population-worldwide (дата звернення: 05.03.2024). Java vs Nodejs: How to Choose the Right Technology. Belitsoft. URL: https://belitsoft.com/java-development-services/java-vs-nodejs (дата звернення: 04.04.2024). JavaScript MEAN stack application approach for real-time nonconformity management in SMEs as a quality control aspect within Industry 4.0 concept / A. Đorđević та ін. International journal of computer integrated manufacturing. 2023. С. 1–22. URL: https://doi.org/10.1080/0951192x.2023.2228274 (дата звернення: 05.03.2024). Kappel G., Retschitzegger W., Schwinger W. Modeling customizable Web applications - a requirement's perspective. 2000 kyoto international conference on digital libraries: research and practice, м. Kyoto, Japan. URL: https://doi.org/10.1109/dlpr.2000.942171 (дата звернення: 05.03.2024). Karunanidhi V. Deploying tensorflow models to a web application. Berkeley, CA : Apress, 2020. URL: https://doi.org/10.1007/978-1-4842-6699-1 (дата звернення: 05.03.2024). McCool R. WWW-Talk oct-dec 1993: server scripts. The World Wide Web History Project. URL: http://1997.webhistory.org/www.lists/www-talk.1993.4/0485.html (дата звернення: 05.03.2024). Mejia A. Creating RESTful APIs with NodeJS and MongoDB Tutorial (Part II). Adrian Mejia Blog. URL: https://adrianmejia.com/creating-a-restful-api-tutorial-with-nodejs-and-mongodb/ (дата звернення: 04.04.2024). Newage. research 2023: the impact of war on users and media consumption trends. Newage. - Digital Advertising Agency. URL: https://newage.agency/blog/newage-research-2023-the-impact-of-war-on-users-and-media-consumption-trends/ (дата звернення: 05.03.2024). Progressive web app implementation in omah wayang klaten website / B. Susanto та ін. Mobile computing and sustainable informatics. Singapore, 2023. С. 333–348. URL: https://doi.org/10.1007/978-981-99-0835-6_24 (дата звернення: 05.03.2024). Progressive web apps development and analysis with angular framework and service worker for e-commerce system / Z. Tahir та ін. 2021 IEEE international conference on computing (ICOCO), м. Kuala Lumpur, Malaysia, 17–19 листоп. 2021 р. 2021. URL: https://doi.org/10.1109/coco53166.2021.9673557 (дата звернення: 05.03.2024). Rojas C. Making your first progressive web app. Building progressive web applications with vue.js. Berkeley, CA, 2019. С. 1–46. URL: https://doi.org/10.1007/978-1-4842-5334-2_1 (дата звернення: 05.03.2024). Skrzypiec S., Plechawska-Wójcik M. Comparative analysis of Angular and React development frameworks. Journal of computer sciences institute. 2023. Т. 28. С. 256–263. URL: https://doi.org/10.35784/jcsi.3724 (дата звернення: 05.03.2024). Stack overflow. Stack Overflow Insights - Developer Hiring, Marketing, and User Research. URL: https://insights.stackoverflow.com/trends?tags=js,apache,express,nginx (дата звернення: 05.03.2024). Stack overflow. Stack Overflow Insights - Developer Hiring, Marketing, and User Research. URL: https://insights.stackoverflow.com/trends?tags=reactjs,angular,vue.js,vuejs3,svelte (дата звернення: 05.03.2024). Tags used in HTML. http://info.cern.ch. URL: https://info.cern.ch/hypertext/WWW/MarkUp/Tags.html (дата звернення: 05.03.2024). Usage statistics and market share of javascript libraries for websites, march 2024. W3Techs - extensive and reliable web technology surveys. URL: https://w3techs.com/technologies/overview/javascript_library (дата звернення: 05.03.2024). Usage statistics and market share of server-side programming languages for websites, march 2024. W3Techs - extensive and reliable web technology surveys. URL: https://w3techs.com/technologies/overview/programming_language (дата звернення: 05.03.2024). Welcome timeline of computer history computer history museum. Home - CHM. URL: https://www.computerhistory.org/timeline/ (дата звернення: 05.03.2024). What is a web app? - web application explained - AWS. Amazon Web Services, Inc. URL: https://aws.amazon.com/what-is/web-application/ (дата звернення: 05.03.2024). ДОДАТКИ Додаток А. Вихідний код додатку // <code>backend/index.js const path = require</code></p>	
» Цитування: 0%	id: 128

'path'	
); require(
Цитирования: 0%	id: 129
'dotenv'	
).config(); const express = require(
Цитирования: 0%	id: 130
'express'	
); const mongoose = require(
Цитирования: 0%	id: 131
'mongoose'	
); const morgan = require(
Цитирования: 0%	id: 132
'mongoose-morgan'	
); const cors = require(
Цитирования: 0%	id: 133
'cors'	
); const app = express(); const PORT = process.env.PORT 8080; app.use(cors());	
app.use(express.json()); app.use(morgan({ connectionString: process.env.DB_CONNECT_STRING	
}, {});	
Цитирования: 0%	id: 134
'combined'	
); app.use(
Цитирования: 0%	id: 135
'/',	
express.static(path.join(__dirname,	
Цитирования: 0%	id: 136
'public'	
))); app.get(
Цитирования: 0%	id: 137
'/api',	
(req, res) = res.status(200).json(
Цитирования: 0,01%	id: 138
'hello!'	
)); const userRouter = require(
Цитирования: 0,01%	id: 139
'./routes/userRouter'	
); const authRouter = require(
Цитирования: 0,01%	id: 140
'./routes/authRouter'	
); const truckRouter = require(
Цитирования: 0,01%	id: 141
'./routes/truckRouter'	
); const loadRouter = require(
Цитирования: 0,01%	id: 142
'./routes/loadRouter'	
); app.use(
Цитирования: 0%	id: 143
'/api',	
userRouter); app.use(
Цитирования: 0%	id: 144
'/api',	
authRouter); app.use(
Цитирования: 0%	id: 145
'/api',	
truckRouter); app.use(
Цитирования: 0%	id: 146
'/api',	
loadRouter); app.all(
Цитирования: 0%	id: 147
'*/*',	
(req, res) = { res.status(400).json({ message:	
Цитирования: 0,01%	id: 148
'Bad request.'	
}); }); app.use((err, req, res, next) = { res.status(500).json({ message: err.message }); }); const	
start = async () = { try { await mongoose.connect(process.env.DB_CONNECT_STRING);	
app.listen(PORT, () = { console.log('Server works at port \${PORT}!'); }); } catch (error) {	
console.log(error); } }; start(); // backend/routes/middlewares/authMiddleware.js const jwt =	
require(
Цитирования: 0%	id: 149
'jsonwebtoken'	
); const JWT_SECRET = process.env.JWT_SECRET	
Цитирования: 0%	id: 150
'justAnyWords'	
; module.exports.authMiddleware = (req, res, next) = { const header = req.headers.authorization;	
if (!header) { return res.status(400).json({ message:	

» Цитирования: 0,03%	id: 151
'No Authorization http header found!'	
}); } const [tokenType, token] = header.split(
» Цитирования: 0,17%	id: 152
''); if (!tokenType) { return res.status(400).json({ message: 'Token type (JWT) is not exist at begin of header!' }); } if (!token) { return res.status(400).json({ message: 'No JWT token found!' }); } try { req.user = jwt.verify(token, JWT_SECRET); return next(); } catch (err) { return res.status(400).json({ message: 'Token is not valid. \${err}' }); } }; // backend/routers/authRouter.js const express = require(
» Цитирования: 0%	id: 153
'express'	
); const router = new express.Router(); const ash = require(
» Цитирования: 0%	id: 154
'express-async-handler'	
); const { register, login, forgotPassword } = require(
» Цитирования: 0,01%	id: 155
'../controllers/authController'	
); router.post(
» Цитирования: 0%	id: 156
'/auth/register',	
ash(register)); router.post(
» Цитирования: 0%	id: 157
'/auth/login',	
ash(login)); router.post(
» Цитирования: 0%	id: 158
'/auth/forgot_password',	
ash(forgotPassword)); module.exports = router; // backend/routers/loadRouter.js const express = require(
» Цитирования: 0%	id: 159
'express'	
); const router = new express.Router(); const ash = require(
» Цитирования: 0%	id: 160
'express-async-handler'	
); const { authMiddleware } = require(
» Цитирования: 0,01%	id: 161
'../middlewares/authMiddleware'	
); const { getLoads, addLoad, activeLoad, iterateState, getLoadById, updateLoadById, deleteLoadById, postLoadById, shippingInfo } = require(
» Цитирования: 0,01%	id: 162
'../controllers/loadController'	
); router.get(
» Цитирования: 0%	id: 163
'/loads',	
authMiddleware, ash(getLoads)); router.post(
» Цитирования: 0%	id: 164
'/loads',	
authMiddleware, ash(addLoad)); router.get(
» Цитирования: 0%	id: 165
'/loads/active',	
authMiddleware, ash(activeLoad)); router.patch(
» Цитирования: 0%	id: 166
'/loads/active/state',	
authMiddleware, ash(iterateState)); router.get(
» Цитирования: 0%	id: 167
'/loads/:loadid',	
authMiddleware, ash(getLoadById)); router.put(
» Цитирования: 0%	id: 168
'/loads/:loadid',	
authMiddleware, ash(updateLoadById)); router.delete(
» Цитирования: 0%	id: 169
'/loads/:loadid',	
authMiddleware, ash(deleteLoadById)); router.post(
» Цитирования: 0%	id: 170
'/loads/:loadid/post',	
authMiddleware, ash(postLoadById)); router.get(
» Цитирования: 0%	id: 171
'/loads/:loadid/shipping_info',	
authMiddleware, ash(shippingInfo)); module.exports = router; // backend/routers/truckRouter.js const express = require(
» Цитирования: 0%	id: 172
'express'	
); const router = new express.Router(); const ash = require(
» Цитирования: 0%	id: 173
'express-async-handler'	
); const { authMiddleware } = require(
»	

Цитирования: 0,01%	id: 174
); const { getTrucks, addTruck, truckDetailsById, updateTruckById, deleteTruckById, assignTruckById } = require(
Цитирования: 0,01%	id: 175
'../controllers/truckController'	
); router.get(
Цитирования: 0%	id: 176
'/trucks',	
authMiddleware, ash(getTrucks)); router.post(
Цитирования: 0%	id: 177
'/trucks',	
authMiddleware, ash(addTruck)); router.get(
Цитирования: 0%	id: 178
'/truck/:truckId',	
authMiddleware, ash(truckDetailsById)); router.put(
Цитирования: 0%	id: 179
'/truck/:truckId',	
authMiddleware, ash(updateTruckById)); router.delete(
Цитирования: 0%	id: 180
'/truck/:truckId',	
authMiddleware, ash(deleteTruckById)); router.post(
Цитирования: 0%	id: 181
'/truck/:truckId/assign',	
authMiddleware, ash(assignTruckById)); module.exports = router // backend/routers/userRouter.js	
const express = require(
Цитирования: 0%	id: 182
'express'	
); const router = new express.Router(); const ash = require(
Цитирования: 0%	id: 183
'express-async-handler'	
); const { authMiddleware } = require(
Цитирования: 0,01%	id: 184
'../middlewares/authMiddleware'	
); const { validOldNewPass } = require(
Цитирования: 0,01%	id: 185
'../middlewares/changePassMiddleware'	
); const { getProfile, deleteProfile, changePassword } = require(
Цитирования: 0,01%	id: 186
'../controllers/profileController'	
); router.get(
Цитирования: 0%	id: 187
'/users/me',	
authMiddleware, ash(getProfile)); router.delete(
Цитирования: 0%	id: 188
'/users/me',	
authMiddleware, ash(deleteProfile)); router.patch(
Цитирования: 0%	id: 189
'/users/me/password',	
authMiddleware, ash(validOldNewPass), ash(changePassword)); module.exports = router; // backend/controllers/authController.js	
const bcrypt = require(
Цитирования: 0%	id: 190
'bcrypt'	
); const Joi = require(
Цитирования: 0%	id: 191
'joi'	
); const jwt = require(
Цитирования: 0%	id: 192
'jsonwebtoken'	
); const User = require(
Цитирования: 0,01%	id: 193
'../models/userModel'	
); const JWT_SECRET = process.env.JWT_SECRET	
Цитирования: 0%	id: 194
'anythingWords'	
; const registerCredentialsSchema = Joi.object({ email: Joi.string().email().required(), password:	
Joi.string().required(), role: Joi.string().valid(
Цитирования: 0%	id: 195
'DRIVER',	
Цитирования: 0%	id: 196
'SHIPPER'	
).required() }); const loginCredentialsSchema = Joi.object({ email: Joi.string().email().required(),	
password: Joi.string().required() }); const forgotPasswordSchema = Joi.object({ email:	
Joi.string().email().required() }); const register = async (req, res) = { try { // joi try/catch const {	
email, password, role } = Joi.attempt(req.body, registerCredentialsSchema); const currentUser =	
await User.findOne({ email }); if (currentUser) { return res.status(400).json({ message:	

Цитирования: 0,02%	id: 197
Такий користувач вже зареєстрований	
}); } const user = new User({ email, password: await bcrypt.hash(password, 10), role }); await user.save(); return res.status(200).json({ message:	
Цитирования: 0,01%	id: 198
Профайл зареєстровано	
}); } catch (error) { return res.status(400).json({ message: `Bad request. Check input fields. \${error}` }); }; }; const login = async (req, res) = { try { // Joi try/catch const { email, password } = Joi.attempt(req.body, loginCredentialsSchema); const user = await User.findOne({ email }); }.exec(); if (!user) { return res.status(400).json({ message:	
Цитирования: 0,02%	id: 199
User not found!	
}); } const currentUser = { _id: user._id, role: user.role, email: user.email, created_date: user.created_date }; const isPasswordOK = await bcrypt.compare(password, user.password); if (!isPasswordOK) { return res.status(400).json({ message:	
Цитирования: 0,01%	id: 200
Wrong password	
}); } return res.json({ jwt_token: jwt.sign(JSON.stringify(currentUser), JWT_SECRET) }); } catch (error) { return res.status(400).json({ message: `Bad request. Check input fields. \${error}` }); }; }; const forgotPassword = async (req, res) = { try { // Joi try/catch const { email } = Joi.attempt(req.body, forgotPasswordSchema); const user = await User.findOne({ email }); }.exec(); if (!user) { return res.status(400).json({ message:	
Цитирования: 0,02%	id: 201
User not found!	
Цитирования: 0,03%	id: 202
New password sent to your email address	
}); } catch (error) { return res.status(400).json({ message: `Bad request. Check input fields. \${error}` }); }; }; module.exports = { register, login, forgotPassword }; // backend/controllers/loadController.js const Joi = require(
Цитирования: 0%	id: 203
Joi	
); const { Types } = require(
Цитирования: 0%	id: 204
mongoose	
); const User = require(
Цитирования: 0,01%	id: 205
../model/userModel	
); const Truck = require(
Цитирования: 0,01%	id: 206
../model/truckModel	
); const Load = require(
Цитирования: 0,01%	id: 207
../model/loadModel	
); const SIZE = { SPRINTER_MAX: { payload: 1700, width: 300, length: 250, height: 170 }, SMALL_MAX: { payload: 2500, width: 500, length: 250, height: 170 }, LARGE_MAX: { payload: 4000, width: 700, length: 350, height: 200 } }; const searchSchema = Joi.object({ status: Joi.string().allow(null,	
Цитирования: 0,04%	id: 208
).empty(['', null]).default(null).valid('NEW', 'POSTED',	
Цитирования: 0%	id: 209
'ASSIGNED',	
Цитирования: 0%	id: 210
'SHIPPED'	
), limit: // numbers from 0 to 50 allowed Joi.string().allow(
Цитирования: 0,1%	id: 211
null).empty(['', null]).default('10').pattern(/^(?:[0-9]{1-4}[0-9]{50})\$/m), offset: Joi.string().allow('', null).empty(['', null]).default('0'	
).pattern(/^\d*\$/m) }); const addLoadSchema = Joi.object({ name: Joi.string().allow(null,	
Цитирования: 0,13%	id: 212
).empty(['', null]).default('Name not specified'), payload: Joi.number().integer().min(0).default(0), pickup_address: Joi.string().allow(null, '').empty(['', null]).default('Pickup address not specified'	
), delivery_address: Joi.string().allow(null, '').empty(['', null]).default('Delivery address not specified'), dimensions: Joi.object({ width: Joi.number().integer().min(0).default(0), length: Joi.number().integer().min(0).default(0), height: Joi.number().integer().min(0).default(0)	
}).allow(null, '').default({ width: 0, length: 0, height: 0 }); }; const getLoads = async (req, res) = { try { let { status, limit, offset } = Joi.attempt(req.query, searchSchema); limit = parseInt(limit, 10); offset = parseInt(offset, 10); const currentUser = await User.findOne({ _id: req.user._id }); if (!currentUser) { return res.status(400).json({ message: 'User not found!' }); }; if (currentUser.role ===	
Цитирования: 0%	id: 213
SHIPPER	
) { const query = {}; query.created_by = currentUser._id; if (status) { query.status = status; } // status ? query.status = status : null; const loads = await Load.find(query).skip(offset).limit(limit); return res.json({ loads }); }; } const query = {}; query.assigned_to = currentUser._id; if (status) { query.status = status; } const loads = await Load.find(query).skip(offset).limit(limit); return res.json({ loads }); }; } catch (error) { return res.status(400).json({ message: `Bad request. Check input fields. \${error}` }); }; }; const addLoad = async (req, res) = { try { const { name, payload, pickup_address, delivery_address, dimensions } = Joi.attempt(req.body, addLoadSchema); const currentUser = await User.findOne({ _id: req.user._id }); if (!currentUser) { return	

```

res.status(400).json({ message:
  "Цитирования: 0,02%" id: 214
  'User not found!'
}); } if (currentUser.role !==
  "Цитирования: 0%" id: 215
  'SHIPPER'
) { return res.status(400).json({ message:
  "Цитирования: 0,03%" id: 216
  'You have no right to do this'
}); } const newLoad = new Load({ name, payload, pickup_address, delivery_address,
dimensions, created_by: req.user_id, created_date: Date.now(), status:
  "Цитирования: 0%" id: 217
  'NEW'
}); await newLoad.save(); return res.json({ message:
  "Цитирования: 0,01%" id: 218
  'Load created successfully'
}); } catch (error) { return res.status(400).json({ message: `Bad request. Check input fields.
${error}` }); } }; const activeLoad = async (req, res) = { const currentUser = await
User.findOne({ _id: req.user_id }); if (!currentUser) { return res.status(400).json({ message:
  "Цитирования: 0,02%" id: 219
  'User not found!'
}); } if (currentUser.role !==
  "Цитирования: 0%" id: 220
  'DRIVER'
) { return res.status(400).json({ message:
  "Цитирования: 0,03%" id: 221
  'You have no right to do this'
}); } const load = await Load.findOne({ status:
  "Цитирования: 0%" id: 222
  'ASSIGNED',
assigned_to: req.user_id }); if (!load) { return res.status(400).json({ message:
  "Цитирования: 0,01%" id: 223
  'Load not found'
}); } return res.json({ load }); }; const iterateState = async (req, res) = { const currentUser =
await User.findOne({ _id: req.user_id }); if (!currentUser) { return res.status(400).json({
message:
  "Цитирования: 0,02%" id: 224
  'User not found!'
}); } if (currentUser.role !==
  "Цитирования: 0%" id: 225
  'DRIVER'
) { return res.status(400).json({ message:
  "Цитирования: 0,03%" id: 226
  'You have no right to do this'
}); } const load = await Load.findOne({ status:
  "Цитирования: 0%" id: 227
  'ASSIGNED',
assigned_to: req.user_id }); if (!load) { return res.status(400).json({ message:
  "Цитирования: 0,01%" id: 228
  'Load not found'
}); } if (load.assigned_to !== req.user_id) { return res.status(400).json({ message:
  "Цитирования: 0,03%" id: 229
  'You have no right to do this'
}); } const _enumArray = load.schema.path(
  "Цитирования: 0%" id: 230
  'state'
).enumValues; let curIndex = _enumArray.findIndex((str) = str === load.state); curIndex =
curIndex _enumArray.length - 1 ? curIndex += 1 : _enumArray.length - 1; await
Load.findOneAndUpdate({ _id: load_id }, { state: _enumArray[curIndex], status: curIndex ===
_enumArray.length - 1 ?
  "Цитирования: 0%" id: 231
  'SHIPPED'
:
  "Цитирования: 0%" id: 232
  'ASSIGNED',
$push: { logs: { message: `Load state changed to
  "Цитирования: 0%" id: 233
  `${_enumArray[curIndex]}`
} } }); if (curIndex === _enumArray.length - 1) { await Truck.findOneAndUpdate({ assigned_to:
load.assigned_to, status:
  "Цитирования: 0%" id: 234
  'OK'
}, { status:
  "Цитирования: 0%" id: 235
  'S'
}); } return res.json({ message: `Load state changed to
  "

```

Цитирования: 0%	id: 236
` \${_id: req.user._id} { message:	
` } }); const getLoadById = async (req, res) => { const { loadId } = req.params; const currentUser = await User.findOne({ _id: req.user._id }); if (!currentUser) { return res.status(400).json({ message:	
Цитирования: 0,02%	id: 237
'User not found!'	
}); } if (!Types.ObjectId.isValid(loadId)) { return res.status(400).json({ message:	
Цитирования: 0,01%	id: 238
'Invalid Load ID'	
}); } const load = await Load.findById(loadId); if (!load) { return res.status(400).json({ message:	
Цитирования: 0,01%	id: 239
'Load not found'	
}); } if (currentUser.role ===	
Цитирования: 0%	id: 240
'SHIPPER'	
) { if (load.created_by !== req.user._id) { return res.status(400).json({ message:	
Цитирования: 0,03%	id: 241
'You have no right to do this'	
}); } return res.json({ load }); } if (load.assigned_to !== req.user._id) { return res.status(400).json({ message:	
Цитирования: 0,03%	id: 242
'You have no right to do this'	
}); } return res.json({ load }); }; const updateLoadById = async (req, res) => { try { const { loadId } = req.params; const { name, payload, pickup_address, delivery_address, dimensions } = joi.attempt(req.body, addLoadSchema); const currentUser = await User.findOne({ _id: req.user._id }); if (!currentUser) { return res.status(400).json({ message:	
Цитирования: 0,02%	id: 243
'User not found!'	
}); } if (currentUser.role !==	
Цитирования: 0%	id: 244
'SHIPPER'	
) { return res.status(400).json({ message:	
Цитирования: 0,03%	id: 245
'You have no right to do this'	
}); } if (!Types.ObjectId.isValid(loadId)) { return res.status(400).json({ message:	
Цитирования: 0,01%	id: 246
'Invalid Load ID'	
}); } const load = await Load.findById(loadId); if (!load) { return res.status(400).json({ message:	
Цитирования: 0,01%	id: 247
'Load not found'	
}); } if (load.created_by !== req.user._id) { return res.status(400).json({ message:	
Цитирования: 0,03%	id: 248
'You have no right to do this'	
}); } if (load.status !==	
Цитирования: 0%	id: 249
'NEW'	
) { return res.status(400).json({ message:	
Цитирования: 0,02%	id: 250
'Load is already in progress'	
}); } if (name) { await Load.findOneAndUpdate({ _id: load._id }, { name }); } if (payload) { await Load.findOneAndUpdate({ _id: load._id }, { payload }); } if (pickup_address) { await Load.findOneAndUpdate({ _id: load._id }, { pickup_address }); } if (delivery_address) { await Load.findOneAndUpdate({ _id: load._id }, { delivery_address }); } if (dimensions) { await Load.findOneAndUpdate({ _id: load._id }, { dimensions }); } return res.json({ message:	
Цитирования: 0,01%	id: 251
'Load updated successfully'	
}); } catch (error) { return res.status(400).json({ message: `Bad request. Check input fields. \$\${error}` }); } }; const deleteLoadById = async (req, res) => { const { loadId } = req.params; const currentUser = await User.findOne({ _id: req.user._id }); if (!currentUser) { return res.status(400).json({ message:	
Цитирования: 0,02%	id: 252
'User not found!'	
}); } if (currentUser.role !==	
Цитирования: 0%	id: 253
'SHIPPER'	
) { return res.status(400).json({ message:	
Цитирования: 0,03%	id: 254
'You have no right to do this'	
}); } if (!Types.ObjectId.isValid(loadId)) { return res.status(400).json({ message:	
Цитирования: 0,01%	id: 255
'Invalid load ID'	
}); } const load = await Load.findById(loadId); if (!load) { return res.status(400).json({ message:	
Цитирования: 0,01%	id: 256
'Load not found'	
}); } if (load.created_by.toString() !== req.user._id) { return res.status(400).json({ message:	
Цитирования: 0,03%	id: 257

'You have no right to do this'	
}); } if (load.status !==	
Цитирования: 0%	id: 258
'NEW'	
) { return res.status(400).json({ message:	
Цитирования: 0,02%	id: 259
'Load is already in progress'	
}); } await Load.findByIdAndDelete(loadId); return res.json({ message:	
Цитирования: 0,01%	id: 260
'Load deleted successfully'	
}); }; const postLoadById = async (req, res) = { const { loadId } = req.params; const currentUser = await User.findOne({ _id: req.user._id }); if (!currentUser) { return res.status(400).json({ message:	
Цитирования: 0,02%	id: 261
'User not found!'	
}); } if (currentUser.role !==	
Цитирования: 0%	id: 262
'SHIPPER'	
) { return res.status(400).json({ message:	
Цитирования: 0,03%	id: 263
'You have no right to do this'	
}); } if (!Types.ObjectId.isValid(loadId)) { return res.status(400).json({ message:	
Цитирования: 0,01%	id: 264
'invalid Load id'	
}); } const load = await Load.findById(loadId); if (!load) { return res.status(400).json({ message:	
Цитирования: 0,01%	id: 265
'Load not found'	
}); } if (load.status !==	
Цитирования: 0%	id: 266
'NEW'	
) { return res.status(400).json({ message:	
Цитирования: 0,02%	id: 267
'Load is already in progress'	
}); } const trucks = await Truck.find({ status:	
Цитирования: 0%	id: 268
'S',	
assigned_to: { \$ne: null // truck assigned to driver } }); if (trucks.length === 0) { await Load.findByIdAndUpdate(load_id, { \$push: { logs: { message:	
Цитирования: 0,02%	id: 269
'No trucks in service'	
} } }); return res.json({ message:	
Цитирования: 0,02%	id: 270
'No trucks in service',	
driver_found: false }); } (async (trucks) = { for (const truck of trucks) { const truckType = `\${truck.type.split(
Цитирования: 0,21%	id: 271
'')[0]_MAX'; if (load.maxLoad = SIZE[truckType].maxLoad && load.dimensions.width = SIZE[truckType].width && load.dimensions.length = SIZE[truckType].length && load.dimensions.height = SIZE[truckType].height) { await Truck.findByIdAndUpdate(truck_id, { status: 'OK' }); await Load.findByIdAndUpdate(load_id, { status: 'ASSIGNED',	
state:	
Цитирования: 0,02%	id: 272
'En route to Pick Up',	
assigned_to: truck.assigned_to, \$push: { logs: { \$each: [{ message: 'Load assigned to driver with id \${truck.assigned_to} }, { message:	
Цитирования: 0,02%	id: 273
'Load state changed to \'	
En route to Pick Up\" } } } }); return res.json({ message:	
Цитирования: 0,01%	id: 274
'Load posted successfully',	
driver_found: true }); } } await Load.findByIdAndUpdate(load_id, { \$push: { logs: { message:	
Цитирования: 0,05%	id: 275
'There is no truck corresponding to the size of the load'	
} } }); return res.json({ message:	
Цитирования: 0,05%	id: 276
'There is no truck corresponding to the size of the load',	
driver_found: false }); } (trucks); }; const shippingInfo = async (req, res) = { const { loadId } = req.params; const currentUser = await User.findOne({ _id: req.user._id }); if (!currentUser) { return res.status(400).json({ message:	
Цитирования: 0,02%	id: 277
'User not found!'	
}); } if (currentUser.role !==	
Цитирования: 0%	id: 278
'SHIPPER'	
) { return res.status(400).json({ message:	
Цитирования: 0,03%	id: 279

'You have no right to do this'	
}); } if (!Types.ObjectId.isValid(loadId)) { return res.status(400).json({ message:	
Цитирования: 0,01%	id: 280
'invalid load id'	
}); } const load = await Load.findById(loadId); if (!load) { return res.status(400).json({ message:	
Цитирования: 0,01%	id: 281
'Load not found'	
}); } if (load.status ===	
Цитирования: 0%	id: 282
'NEW'	
) { return res.status(400).json({ message:	
Цитирования: 0,02%	id: 283
'No shipment for this load'	
}); } const truck = await Truck.findOne({ assigned_to: load.assigned_to }); if (!truck) { return	
Цитирования: 0,01%	id: 284
'Truck not found'	
}); } return res.json({ load, truck }); }; module.exports = { getLoads, addLoad, activeLoad,	
iterateState, getLoadById, updateLoadById, deleteLoadById, postLoadById, shippingInfo }; //	
backend/controllers/profileController.js const bcrypt = require(
Цитирования: 0%	id: 285
'bcrypt'	
); const User = require(
Цитирования: 0,01%	id: 286
'../models/userModel'	
); const getProfile = async (req, res) = { const currentUser = await User.findOne({ _id:	
Цитирования: 0,02%	id: 287
'User not found!'	
}); } return res.status(200).json({ user: currentUser }); }; const deleteProfile = async (req, res) =	
Цитирования: 0,02%	id: 288
'User not found!'	
}); } if (currentUser.role ===	
Цитирования: 0%	id: 289
'DRIVER'	
) { return res.status(400).json({ message:	
Цитирования: 0,02%	id: 290
'Driver can not delete profile'	
}); } User.deleteOne({ _id: req.user._id }, (err, result) = { if (err) { return res.status(400).json({	
Цитирования: 0,01%	id: 291
'Profile deleted successfully'	
}); }); const changePassword = async (req, res) = { const currentUser = await User.findOne({	
Цитирования: 0,02%	id: 292
'User not found!'	
}); } if (!(await bcrypt.compare(req.user.oldPassword, currentUser.password))) { return	
Цитирования: 0,01%	id: 293
'Wrong password!'	
}); } User.updateOne({ _id: req.user._id }, { password: await bcrypt.hash(req.user.newPassword,	
Цитирования: 0%	id: 294
'Success'	
}); }); }; module.exports = { getProfile, deleteProfile, changePassword }; //	
Цитирования: 0%	id: 295
'id'	
); const { Types } = require(
Цитирования: 0%	id: 296
'mongoose'	
); const User = require(
Цитирования: 0,01%	id: 297
'../models/userModel'	
); const Truck = require(
Цитирования: 0,01%	id: 298
'../models/truckModel'	
); const createTruckSchema = Joi.object({ type: Joi.string().valid(
Цитирования: 0%	id: 299
'SPRINTER',	
Цитирования: 0,01%	id: 300
'SMALL STRAIGHT',	

Цитирования: 0,01%	id: 301
'LARGE STRAIGHT'	
) .required() }); const getTrucks = async (req, res) => { const currentUser = await User.findOne({ id: req.user_id }); if (!currentUser) { return res.status(400).json({ message:	
Цитирования: 0,02%	id: 302
'User not found!'	
}); } if (currentUser.role !==	
Цитирования: 0%	id: 303
'DRIVER'	
) { return res.status(400).json({ message:	
Цитирования: 0,03%	id: 304
'You have no right to do this'	
}); } const trucks = await Truck.find({ created_by: req.user_id }); return res.status(200).json({ trucks }); }; const addTruck = async (req, res) => { try { const { type } = req.body, createTruckSchema); const currentUser = await User.findOne({ id: req.user_id }); if (!currentUser) { return res.status(400).json({ message:	
Цитирования: 0,02%	id: 305
'User not found!'	
}); } if (currentUser.role !==	
Цитирования: 0%	id: 306
'DRIVER'	
) { return res.status(400).json({ message:	
Цитирования: 0,03%	id: 307
'You have no right to do this'	
}); } const newTruck = new Truck({ type, status:	
Цитирования: 0%	id: 308
'',	
created_by: req.user_id, created_date: Date.now() }); await newTruck.save(); return res.status(200).json({ message:	
Цитирования: 0,01%	id: 309
'Truck created successfully'	
}); } catch (error) { return res.status(400).json({ message: `Bad request. Check input fields. \$ {error}` }); }; }; const truckDetailsById = async (req, res) => { const { truckId } = req.params; const currentUser = await User.findOne({ id: req.user_id }); if (!currentUser) { return res.status(400).json({ message:	
Цитирования: 0,02%	id: 310
'User not found!'	
}); } if (currentUser.role !==	
Цитирования: 0%	id: 311
'DRIVER'	
) { return res.status(400).json({ message:	
Цитирования: 0,03%	id: 312
'You have no right to do this'	
}); } if (!Types.ObjectId.isValid(truckId)) { return res.status(400).json({ message:	
Цитирования: 0,01%	id: 313
'invalid Truck ID'	
}); } const truck = await Truck.findById(truckId); if (!truck) { return res.status(400).json({ message:	
Цитирования: 0,01%	id: 314
'Truck not found'	
}); } if (truck.created_by !== req.user_id) { return res.status(400).json({ message:	
Цитирования: 0,03%	id: 315
'You have no right to do this'	
}); } return res.status(200).json({ truck }); }; const updateTruckById = async (req, res) => { try { const { truckId } = req.params; const { type } = req.body, createTruckSchema); const currentUser = await User.findOne({ id: req.user_id }); if (!currentUser) { return res.status(400).json({ message:	
Цитирования: 0,02%	id: 316
'User not found!'	
}); } if (currentUser.role !==	
Цитирования: 0%	id: 317
'DRIVER'	
) { return res.status(400).json({ message:	
Цитирования: 0,03%	id: 318
'You have no right to do this'	
}); } if (!Types.ObjectId.isValid(truckId)) { return res.status(400).json({ message:	
Цитирования: 0,01%	id: 319
'invalid Truck ID'	
}); } const truck = await Truck.findById(truckId); if (!truck) { return res.status(400).json({ message:	
Цитирования: 0,01%	id: 320
'Truck not found'	
}); } if (truck.created_by !== req.user_id) { return res.status(400).json({ message:	
Цитирования: 0,03%	id: 321
'You have no right to do this'	
}); } if (truck.status ===	
Цитирования: 0%	id: 322

'OK'	
) { // on load and assigned to this driver return res.status(400).json({ message:	
Цитирования: 0,02%	id: 323
'Truck is on load'	
}); } await Truck.findByIdAndUpdate(truckId, { type }); return res.status(200).json({ message:	
Цитирования: 0,02%	id: 324
'Truck details changed successfully'	
}); } catch (error) { return res.status(400).json({ message: `Bad request. Check input fields.	
\${error} }); } }; const deleteTruckById = async (req, res) = { const { truckId } = req.params;	
const currentUser = await User.findOne({ _id: req.user._id }); if (!currentUser) { return	
res.status(400).json({ message:	
Цитирования: 0,02%	id: 325
'User not found!'	
}); } if (currentUser.role !==	
Цитирования: 0%	id: 326
'DRIVER'	
) { return res.status(400).json({ message:	
Цитирования: 0,03%	id: 327
'You have no right to do this'	
}); } if (!Types.ObjectId.isValid(truckId)) { return res.status(400).json({ message:	
Цитирования: 0,01%	id: 328
'invalid Truck ID'	
}); } const truck = await Truck.findById(truckId); if (!truck) { return res.status(400).json({	
message:	
Цитирования: 0,01%	id: 329
'Truck not found'	
}); } if (truck.created_by !== req.user._id) { return res.status(400).json({ message:	
Цитирования: 0,03%	id: 330
'You have no right to do this'	
}); } if (truck.status ===	
Цитирования: 0%	id: 331
'OK'	
) { return res.status(400).json({ message:	
Цитирования: 0,02%	id: 332
'Truck is on load'	
}); } if (truck.assigned_to !== null) { return res.status(400).json({ message:	
Цитирования: 0,03%	id: 333
'Can not remove Truck assigned to Driver'	
}); } await Truck.findByIdAndDelete(truckId); return res.status(200).json({ message:	
Цитирования: 0,01%	id: 334
'Truck deleted successfully'	
}); } const assignTruckById = async (req, res) = { const { truckId } = req.params; const	
currentUser = await User.findOne({ _id: req.user._id }); if (!currentUser) { return	
res.status(400).json({ message:	
Цитирования: 0,02%	id: 335
'User not found!'	
}); } if (currentUser.role !==	
Цитирования: 0%	id: 336
'DRIVER'	
) { return res.status(400).json({ message:	
Цитирования: 0,03%	id: 337
'You have no right to do this'	
}); } const isDriverHasTruckLoaded = await Truck.find({ created_by: req.user._id, status:	
Цитирования: 0%	id: 338
'OK'	
}); if (isDriverHasTruckLoaded.length 0) { return res.status(400).json({ message:	
Цитирования: 0,02%	id: 339
'Driver has a load'	
}); } if (!Types.ObjectId.isValid(truckId)) { return res.status(400).json({ message:	
Цитирования: 0,01%	id: 340
'invalid Truck ID'	
}); } const truck = await Truck.findById(truckId); if (!truck) { return res.status(400).json({	
message:	
Цитирования: 0,01%	id: 341
'Truck not found'	
}); } if (truck.created_by !== req.user._id) { return res.status(400).json({ message:	
Цитирования: 0,03%	id: 342
'You have no right to do this'	
}); } if (truck.status ===	
Цитирования: 0%	id: 343
'OK'	
) { return res.status(400).json({ message:	
Цитирования: 0,02%	id: 344
'Truck is on load'	
}); } await Truck.updateMany({ created_by: req.user._id, status:	
Цитирования: 0%	id: 345

' g '	
}, { assigned_to: null }); await Truck.findByIdAndUpdate(truckId, { assigned_to: req.user_id }); return res.status(200).json({ message:	
Цитирования: 0,01%	id: 346
'Truck assigned successfully'	
}); module.exports = { getTrucks, addTruck, truckDetailsById, updateTruckById, deleteTruckById, assignTruckById }; // backend/models/loadModel.js const mongoose = require(
Цитирования: 0%	id: 347
'mongoose'	
); const { Schema } = mongoose; const { Types } = mongoose; module.exports = mongoose.model(
Цитирования: 0%	id: 348
'load',	
new Schema({ created_by: { type: Types.ObjectId, ref:	
Цитирования: 0%	id: 349
'User',	
required: true }, assigned_to: { type: Types.ObjectId, ref:	
Цитирования: 0%	id: 350
'User'	
}, status: { type: String, required: true, enum: [
Цитирования: 0%	id: 351
'NEW',	
Цитирования: 0%	id: 352
'POSTED',	
Цитирования: 0%	id: 353
'ASSIGNED',	
Цитирования: 0%	id: 354
'SHIPPED'	
]], state: { type: String, enum: [
Цитирования: 0,02%	id: 355
'En route to Pick Up',	
Цитирования: 0,02%	id: 356
'Arrived to Pick Up',	
Цитирования: 0,02%	id: 357
'En route to delivery',	
Цитирования: 0,01%	id: 358
'Arrived to delivery'	
]], name: { type: String, required: true }, payload: { type: Number, required: true }, pickup_address: { type: String, required: true }, delivery_address: { type: String, required: true }, dimensions: { width: { type: Number, required: true }, length: { type: Number, required: true }, height: { type: Number, required: true } }, logs: [{ message: { type: String, required: true }, time: { type: Date, default: Date.now() } }], created_date: { type: Date, default: Date.now() } }, { versionKey: false // hide __v field })); // backend/models/truckModel.js const mongoose = require(
Цитирования: 0%	id: 359
'mongoose'	
); const { Schema } = mongoose; const { Types } = mongoose; module.exports = mongoose.model(
Цитирования: 0%	id: 360
'Truck',	
new Schema({ created_by: { type: Types.ObjectId, ref:	
Цитирования: 0%	id: 361
'User',	
required: true }, assigned_to: { type: Types.ObjectId, ref:	
Цитирования: 0%	id: 362
'User',	
default: null }, type: { type: String, required: true, enum: [
Цитирования: 0%	id: 363
'SPRINTER',	
Цитирования: 0,01%	id: 364
'SMALL STRAIGHT',	
Цитирования: 0,01%	id: 365
'LARGE STRAIGHT'	
]], status: { type: String, required: true, enum: [
Цитирования: 0%	id: 366
'OK',	
Цитирования: 0%	id: 367
' g '	
// ON LOAD, IN SERVICE }, created_date: { type: Date, default: Date.now() } }, { versionKey: false // hide __v field })); // backend/models/userModel.js const mongoose = require(
Цитирования: 0%	id: 368
'mongoose'	
); const { Schema } = mongoose; module.exports = mongoose.model(
Цитирования: 0%	id: 369
'User',	

```
new Schema({ email: { type: String, required: true, unique: true }, password: { type: String,
required: true }, role: { type: String, enum: [
  " Цитирования: 0% id: 370
  'DRIVER',
  " Цитирования: 0% id: 371
  'SHIPPER'
], required: true }, created_date: { type: Date, default: Date.now() } }, { versionKey: false // hide
_v field }));
```

Заявление об ограничении ответственности:

Этот отчет должен быть правильно истолкован и проанализирован квалифицированным специалистом, который несет ответственность за оценку!

Любая информация, представленная в этом отчете, не является окончательной и подлежит ручному просмотру и анализу. Пожалуйста, следуйте инструкциям: [Рекомендации по оценке](#)