

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «ЗАПОРІЗЬКА ПОЛІТЕХНІКА»

Факультет комп'ютерних наук та технологій
Кафедра комп'ютерних систем та мереж

Пояснювальна записка

до дипломного проєкту (роботи)

магістра

(ступінь вищої освіти)

на тему **ВЕБСИСТЕМА З ВИКОРИСТАННЯМ СИСТЕМНОГО ПІДХОДУ ДО
ВИБОРУ ТЕХНОЛОГІЙ ВЕБРОЗРОБКИ**

Виконав: студент 2 курсу, групи КНТ-514м
спеціальності _____

123 Комп'ютерна інженерія

(код і найменування спеціальності)

Освітня програма (спеціалізація)

Комп'ютерні системи та мережі

РЮМІН О. О.

(ПРИЗВИЩЕ та ініціали)

Керівник СКРУПСЬКИЙ С. Ю.

(ПРИЗВИЩЕ та ініціали)

Рецензент ХАРИТОНОВ О. Б.

(ПРИЗВИЩЕ та ініціали)

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний університет «Запорізька політехніка»

Факультет Комп'ютерних наук і технологій
Кафедра «Комп'ютерні системи та мережі»
Ступінь вищої освіти магістерський
Спеціальність 123 Комп'ютерна інженерія
(код і найменування)
Освітня програма (спеціалізація) Комп'ютерні системи та мережі
(назва освітньої програми (спеціалізації))

ЗАТВЕРДЖУЮ
Зав. кафедри Кудерметов Р.К.

“ 15 ” Жовтня 2025 року

З А В Д А Н Н Я
НА ДИПЛОМНИЙ ПРОЄКТ (РОБОТУ) СТУДЕНТА

РЮМІНА Олександра Олександровича

(ПРИЗВИЩЕ, ім'я, по батькові)

1. Тема проєкту (роботи) Вебсистема з використанням системного підходу до вибору технологій веброзробки

керівник проєкту (роботи) к.т.н., доцент, СКРУПСЬКИЙ Степан Юріювич

(науковий ступінь, вчене звання, ПРИЗВИЩЕ, ім'я, по батькові)

затверджені наказом вищого навчального закладу від “ 30 ” жовтня 2025 року № 491

2. Строк подання студентом проєкту (роботи) 23 грудня 2025 року

3. Вихідні дані до проєкту (роботи) методики збору технічних і освітніх метрик, визначення ваг, розробка авторського інструменту, побудова дерева прийняття рішень, порівняння з аналогами

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити) 1) Теоретичні основи вибору технологій веброзробки;

2) Аналіз існуючих підходів та аналогів у виборі технологічного стеку;

3) Проєктування методів та інструменту вибору стеку;

4) Реалізація та дослідження системи;

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

Листи презентації

6. Консультанти розділів проєкту (роботи)

Розділ	ПРИЗВИЩЕ, ініціали та посада консультанта	Підпис, дата	
		завдання видав	прийняв виконане завдання
1-4	СКРУПСЬКИЙ С. Ю., доцент		
нормоконтроль	ЩЕРБАК Н.В., ст. викл.		

7. Дата видачі завдання 01.10.2025 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів дипломного проєкту (роботи)	Строк виконання етапів проєкту (роботи)	Примітка
1	Теоретичні основи вибору технологій веброзробки	04.10.2025 р.	
2	Аналіз існуючих підходів та аналогів у виборі технологічного стеку	10.10.2025 р.	
3	Проектування методів та інструменту вибору стеку	20.10.2025 р.	
4	Реалізація та дослідження системи	15.11.2025 р.	
5	Оформлення графічного матеріалу	11.12.2025 р.	
6	Оформлення допоміжного матеріалу	11.12.2025 р.	

Студент Олександр РІОМІН
(підпис) (ім'я, ПРИЗВИЩЕ)

Керівник проєкту (роботи) Степан СКРУПСЬКИЙ
(підпис) (ім'я, ПРИЗВИЩЕ)

РЕФЕРАТ

ПЗ: 94 с., 18 табл., 28 рис., 1 додаток, 31 джерело.

АНАЛІЗ, АРХІТЕКТУРА, ВЕБЗАСТОСУНОК, ВИПРОБУВАННЯ,
ДЕРЕВО, МЕТРИКИ, МОВА ПРОГРАМУВАННЯ, ФРЕЙМВОРК

Об'єкт дослідження – процеси проєктування і реалізації системи вибору стеку технологій веброзробки на основі технічних та освітніх критеріїв.

Предмет дослідження – методика збору, нормалізації та агрегації технічних і освітніх метрик і логіка побудови дерева прийняття рішень для вибору стеку технологій веброзробки.

Мета роботи – це підвищення ефективності визначення стеку технологій веброзробки для новачків.

Методи дослідження – аналіз предметної області та літератури, аналіз наявних аналогів, розробка авторського інструмента для автоматичного збору технічних метрик, збір освітніх метрик на платформах надавачів освіти, вакансій, документації і спільнот допомоги програмістам та формування дерева прийняття рішень.

Результат дослідження – реалізований прототип вебінструмента та документовані методики збору й обробки метрик, модель нормалізації і присвоєння ваг, початкові калібрувальні дані й логіка дерева прийняття рішень для ранжування стеків, придатні для подальших масштабних прогонів і інтеграції в навчальні та дослідні середовища.

ABSTRACT

Explanatory note: 94 p., 18 tab., 28 fig., 1 addition, 31 sources.

ANALYSIS, ARCHITECTURE, FRAMEWORK, METRICS, PROGRAMMING
LANGUAGE, TESTING, TREE, WEB APPLICATION

The object of the study is the process of designing and implementing a system for selecting a web development technology stack based on technical and educational criteria.

The subject of the study is the methods of collecting, normalizing and aggregating technical and educational metrics and the logic of building a decision tree for selecting a web development technology stack.

The purpose of the work is to increase the efficiency of determining the stack of webdevelopment technologies for beginners.

Research methods are analysis of the subject area and literature, analysis of existing analogues, development of an author's tool for automatic collection of technical metrics, collection of educational metrics on platforms of education providers, vacancies, documentation and communities of assistance to programmers and formation of a decision tree.

The result of the research is an implemented prototype of a web tool and documented methods for collecting and processing metrics, a normalization and weighting model, initial calibration data, and decision tree logic for stack ranking, suitable for further large-scale runs and integration into educational and research environments.

ЗМІСТ

Скорочення та умовні позначки	8
Вступ.....	9
1 Теоретичні основи вибору технологій веброзробки.....	11
1.1 Аналіз предметної області та особливостей веброзробки	11
1.2 Аналіз архітектур вебсистем.....	13
1.3 Аналіз мов програмування	16
1.4 Аналіз фреймворків.....	18
1.5 Важливість освітніх критеріїв при підборі стеку розробки.....	20
2. Аналіз існуючих підходів та аналогів у виборі технологічного стеку	21
2.1 Пошук аналогів.....	21
2.2 Порівняння аналогів.....	25
2.3 Висновки за результатами аналізу аналогів	28
3. Проєктування методів та інструменту вибору стеку.....	29
3.1 Складові дослідження й обґрунтування подальших кроків	29
3.2 Відбір критеріїв оцінювання	30
3.3 Методика збору технічних метрик	32
3.4 Розробка інструмента для збору технічних метрик.....	35
3.5 Визначення технічних вимог до розроблюваної вебсистеми	41
3.6 Методика аналізу метрик за освітніми критеріями	43
3.7 Проєктування дерева прийняття рішень.....	44
3.8 Висновки проведення проєктування	45
4. Реалізація та дослідження системи	45
4.1 Реалізація скриптів випробувань та логіка збору технічних метрик.....	45
4.2 Визначення освітніх метрик.....	50
4.3 Реалізація методів і алгоритмів присвоєння ваг	59
4.4 Отримання та представлення результатів дослідження.....	75
4.5 Порівняння витрат часу на визначення метрики	78

Висновки	81
Перелік джерел посилання	83
Додаток А	86

СКОРОЧЕННЯ ТА УМОВНІ ПОЗНАКИ

SOLID – Single responsibility, Open-closed, Liskov substitution, Interface segregation і Dependency inversion;

DRY – Don't Repeat Yourself;

API – Application Programming Interface;

SEO – Search Engine Optimization;

SSR – Server-Side Rendering;

SSG – Static Site Generation;

MVP – Minimum Viable Product;

MEAN – MongoDB, Express, Angular, Node;

MERN – MongoDB, Express, React, Node;

LAMP – Linux, Apache, MySQL, PHP/Perl/Python;

RPS – Requests Per Second;

AHP – Analytic Hierarchy Process;

HTML – HyperText Markup Language;

CSS – Cascading Style Sheets;

DOM – Document Object Model;

HTTP – HyperText Transfer Protocol;

ПК – Персональний комп'ютер;

TLS – Transport Layer Security;

ASGI – Asynchronous Server Gateway Interface;

WSGI – Web Server Gateway Interface;

UX – User Interface;

SLA – Service Level Agreement;

CRUD – Create, Read, Update, Delete;

REST – Representational State Transfer.

ВСТУП

Сфера інформаційних технологій є однією з найдинамічніших галузей сучасної економіки. Вона визначає темпи розвитку суспільства та забезпечує впровадження інновацій у промисловості, освіті, науці й управлінні. ІТ-індустрія охоплює широкий спектр напрямів – від програмної розробки та адміністрування мереж до машинного навчання, аналітики даних, веб- та мобільної розробки. Така різноманітність відкриває значні можливості для професійного зростання, але ускладнює вибір оптимального напрямку.

Однією з важливих проблем для студентів і початківців є визначення доцільних технологій для створення програмних систем. Особливо це стосується веброзробки – сфери, що часто стає першим етапом професійного становлення. Вебсистеми сьогодні є основою більшості інформаційних сервісів, тому виникає питання вибору мови програмування, фреймворку чи архітектури серед великої кількості доступних рішень.

Актуальність роботи полягає у необхідності створення системного підходу до вибору технологій веброзробки на основі порівняння за технічними та освітніми критеріями. Технічні параметри включають швидкість виконання коду, споживання пам'яті, асинхронність, стабільність та масштабованість. Освітні критерії охоплюють складність освоєння, вартість та час на навчання, актуальність, популярність технології та доступність документації. Урахування цих показників дозволяє обґрунтовано обирати стек для створення вебсистем.

Метою дипломної роботи є підвищення ефективності підходу обрання оптимального стеку технологій веброзробки студентами і початківцями за допомогою дерева прийняття рішень.

Завданням даної роботи є комплексне вирішення поставленої мети шляхом поетапного виконання наступних завдань: провести огляд і систематизацію сучасних архітектур, мов програмування та фреймворків, що використовуються у веброзробці; сформулювати набір технічних та освітніх критеріїв оцінки

технологічних альтернатив; розробити математично обґрунтовану модель для порівняння і ранжування варіантів; виконати емпіричну перевірку технічних показників за допомогою авторського інструменту для бенчмарків; провести аналіз доступних навчальних ресурсів, вакансій та освітніх пропозицій для оцінки освітніх критеріїв. На основі отриманих результатів побудувати дерево прийняття рішень, яке дозволить студентам і початківцям обирати оптимальний стек розробки.

1 ТЕОРЕТИЧНІ ОСНОВИ ВИБОРУ ТЕХНОЛОГІЙ ВЕБРОЗРОБКИ

1.1 Аналіз предметної області та особливостей веброзробки

Веброзробка – це галузь програмування, яка охоплює проектування, створення, розгортання та підтримку вебсистем, що можуть включати як фронтенд (інтерфейс користувача), так і бекенд (серверна логіка, база даних), а також інтеграцію, конфігурацію серверів і забезпечення безпеки. Сучасні вебсистеми використовуються повсюдно в бізнесі, освіті, державному секторі, електронній комерції й медіа.

Ключові етапи веброзробки:

- програмування веб-ресурсу, що включає збирання вимог замовника, розробку ТЗ та проекту інтерфейсів;
- створення загальної концепції проекту з урахуванням креативу;
- аналіз дизайнерської концепції та втілення її в життя за допомогою розробки макета, але це завдання може виконувати і дизайнер;
- створення мультимедійних та FLASH-елементів;
- верстання веб-сторінок;
- робота з ПЗ, що включає розробку функціональних елементів або інтеграцію певних інструментів у вже наявну систему керування вмістом;
- перенесення проекту на платформу та оптимізація текстового контенту;
- перевірка веб-ресурсу за допомогою тестів та додавання коригувань, якщо в цьому є потреба;
- запуск вебпроекту на сервері;
- обслуговування проекту загалом або його програмної складової.

Залежно від особливостей того чи іншого проекту, деякі етапи розробки веб-сайтів і програм можуть бути взаємопов'язані між собою [1].

Так виглядають загальні етапи веброзробки, але під час цього можуть виникати різні питання або проблеми. Одним із таких питань є – як саме має

виглядати структура вебзастосунку, тобто яка її архітектура. Архітектурні рішення безпосередньо впливають не лише на якість продукту, але й на швидкість розробки. Чітка структура кодової бази, модульність і можливість повторного використання компонентів значно зменшують час, необхідний для написання коду. Крім того, спільне розуміння архітектури в команді спрощує комунікацію та прискорює прийняття рішень.

Архітектура також відіграє ключову роль у мінімізації ймовірності помилок. Розділення відповідальності, висока тестованість та ефективна обробка помилок дозволяють розробникам створювати більш надійне ПЗ. Наявність чітких стандартів кодування, які надає архітектура, також зменшує хаотичність коду, а відтак і кількість помилок. Застосування принципів програмування, таких як SOLID і DRY, унеможливорює дублювання коду та сприяє його зрозумілості, що знижує ризик появи багів [2].

Наступним є обрання мови програмування для реалізації проєкту. Мова програмування є найбільш важливим інструментом для розробників програмного забезпечення, саме тому серед розробників і програмістів-початківців часто постає важливе питання щодо того, які мови програмування є затребуваними та які слід обрати для вивчення, щоб побудувати успішну кар'єру та бути затребуваним фахівцем крізь роки [3].

Також слід звернути увагу на те, що різні частини вебзастосунку, а саме: фронтенд і бекенд, можна писати різними мовами програмування. Для цього потрібно вивчати додаткову «серверну» мову програмування або фрейворки, які здатні виконувати потрібні функції. Фреймворк – це програмна платформа, яка надає розробнику структуру проєкту, архітектурні шаблони, вбудовані бібліотеки та API для типових завдань: маршрутизація, валідація, робота з формами, підключення до бази даних, аутентифікація та багато іншого [4].

Отже, веброзробка є складною та багатокомпонентною сферою, де ключову роль відіграють архітектура, мови програмування та фреймворки. Для новачків важливо розуміти критерії вибору технологій, що враховують технічні, практичні

та перспективні аспекти. Саме це підкреслює необхідність системних підходів оцінки і вибору засобів веброзробки.

1.2 Аналіз архітектур вебсистем

Архітектура вебсистем визначає високорівневу структуру застосунку, яка охоплює компоненти, їхню взаємодію та принципи організації. Вона виступає своєрідним «планом» системи – описує, як фронтенд (інтерфейс користувача), бекенд (сервіси, логіка, обробка даних) та бази даних сполучаються між собою, як розподіляється навантаження, забезпечується масштабованість і надійність [5]. Для вебзастосунку, який має бути сучасним, ефективним та конкурентоспроможним, правильний вибір архітектури є одним із основних етапів – саме він задає рамки для подальшого вибору мови програмування, фреймворку, інструментів та організації процесу розробки.

Вебсистеми характеризуються підвищеними вимогами до доступності, інтерактивності, продуктивності, масштабування і підтримки. У сучасному цифровому середовищі це означає, що архітектура має передбачати: ефективне оброблення запитів, можливість інтеграції з API та сторонніми сервісами, зручність оновлень, а також відмовостійкість і безпеку [5]. Водночас, як зауважують експерти-практики, неправильно обрана або недооцінена архітектура може стати джерелом багатьох проблем – від ускладненого розгортання та масштабування до високих витрат на підтримку. Наприклад, відсутність розділення обов'язків компонентів призводить до хаотичності кодової бази, робить складнішим впровадження нових функцій і знижує продуктивність команди [2].

У веброзробці застосовуються різні типи архітектурних рішень, кожне з яких має свої особливості, переваги та обмеження. Класична трирівнева архітектура (Presentation – Application – Data) передбачає чітке розділення системи на рівні: інтерфейс користувача, серверна логіка та база даних. Такий підхід дозволяє

підтримувати структурованість проєкту, спрощує управління кодом і забезпечує зрозумілу взаємодію між компонентами, що особливо важливо для командної роботи та підтримки середніх і великих проєктів.

Односторінкові застосунки (SPA) орієнтовані на підвищену інтерактивність і швидкий відгук користувача, оскільки більшість логіки обробляється на клієнтській стороні. Водночас такий підхід вимагає ретельного планування структури застосунку, оптимізації SEO та забезпечення сумісності з різними браузерами, що робить їх більш складними для реалізації та підтримки [5].

Серед архітектурних рішень у веброзробці також поширена монолітна архітектура, коли всі компоненти застосунку – інтерфейс, логіка та база даних – інтегровані в єдину систему. Такий підхід спрощує початкову розробку і розгортання, дозволяє швидко запускати невеликі або середні проєкти, однак у міру зростання складності та навантаження підтримка та масштабування стають проблематичними [2].

Популярним серед розробників є клієнт-серверна архітектура, яка розділяє відповідальність між фронтендом (клієнтом) і бекендом (сервером). Такий підхід є базовим для більшості вебзастосунків: клієнт відповідає за інтерфейс і локальну логіку, сервер – за бізнес-логіку, зберігання даних і безпеку. Клієнт-серверна модель забезпечує гнучкість у виборі технологій на кожному рівні, полегшує версіонування API та робить можливим розподілення навантаження. Водночас вона підвищує значущість якісного проєктування API, контролю версій та тестування сумісності компонентів. Профільні матеріали з тестування й навчання вказують на те, що для проєктів середнього масштабу клієнт-серверна модель часто є «золотою серединою» між простотою моноліту і складністю мікросервісів [6].

Особливе місце займає мікросервісна архітектура, яка передбачає розподіл системи на незалежні, слабозв'язані сервіси. Кожен сервіс відповідає за окремий функціонал і може розгортатись, масштабуватись і тестуватись автономно. Це забезпечує високу гнучкість і дозволяє швидко адаптувати систему до змінних вимог, що особливо корисно у великих та динамічних проєктах. Водночас

мікросервіси підвищують складність управління, потребують більшої уваги до інтеграції, моніторингу та координації команд.

Ще одним сучасним підходом є безсерверна (serverless) архітектура, де розробник концентрується на функціоналі застосунку, а всі серверні ресурси, масштабування та управління інфраструктурою обслуговуються провайдером (наприклад, AWS Lambda, Azure Functions). Такий підхід дозволяє значно скоротити витрати на інфраструктуру та швидко запускати прототипи або невеликі сервіси. Водночас він накладає обмеження на контроль над середовищем виконання, вимагає оптимізації коду для коротких обчислювальних сесій та залежить від сторонніх провайдерів [5].

Архітектура відіграє роль орієнтира. Інакше кажучи: архітектура задає рамки, в межах яких доцільно обирати мову програмування, фреймворк або іншу технологію. Наприклад, якщо проєкт передбачає високі навантаження і потребує гнучкого масштабування, мікросервісна архітектура може бути кращою ніж моноліт. Відповідно, це визначає вимоги до вибору інструментів. І навпаки, для невеликого вебзастосунку з обмеженими ресурсами може бути доцільним простіший підхід, що зменшує витрати. У цьому контексті оцінка архітектури повинна стати частиною системи критеріїв: наскільки обрана архітектура відповідає вимогам продуктивності, масштабованості, безпеки, витрат на підтримку, простоти впровадження.

Отже, аналіз архітектури вебсистем показує, що вона не є лише «технічною деталлю» – це фундамент рішення, який задає весь подальший стек технологій, структуру команди, процес розробки і витрати. Саме тому у рамках дипломної роботи з розробки вебсистеми з використанням системного підходу до вибору технологій архітектура повинна стати одним із головних етапів оцінювання – до вибору мови програмування чи фреймворку має бути чітке усвідомлення обраної архітектури системи.

1.3 Аналіз мов програмування

Мова програмування – це формальна система для опису алгоритмів і створення програмного забезпечення. У контексті веброзробки вона є основним інструментом, який дозволяє реалізовувати логіку застосунку, обробляти дані, взаємодіяти з користувачем та іншими сервісами, а також забезпечувати ефективну і безпечну роботу вебсистеми. Вибір мови програмування визначає не лише швидкість розробки, але й масштабованість проєкту, його продуктивність та можливість інтеграції з іншими технологіями. Для вебзастосунків правильний вибір мови програмування є критично важливим, оскільки він задає фундамент для всієї подальшої архітектури та технологічного стеку [7].

За даними останніх рейтингів популярності мов програмування, найбільшим попитом користуються JavaScript, Python, Java, C# та TypeScript. JavaScript залишається основною мовою для фронтенд-розробки, забезпечуючи інтерактивність і динамічність вебсторінок. Python популярний завдяки простоті синтаксису, широкому набору бібліотек та застосуванню у бекенд-розробці, аналітиці даних і машинному навчанні. Java та C# традиційно використовуються для створення корпоративних систем і великих вебзастосунків з високим навантаженням, а TypeScript забезпечує типобезпечність і покращену підтримку великих фронтенд-проєктів [7]. Вибір мови програмування безпосередньо впливає на технологічні можливості проєкту та визначає, які фреймворки і бібліотеки можна використовувати.

Згідно з оглядом SoftServe, на сьогодні для початківців і професійних веброзробників актуальними є мови, що дозволяють швидко реалізовувати функціонал і підтримують сучасні підходи, такі як SPA та мікросервіси. На першому місці за популярністю залишаються JavaScript і Python, далі йдуть Java, C#, PHP і Go, причому вибір часто залежить від спеціалізації та типу проєкту [8]. Це означає, що під час розробки вебзастосунку необхідно оцінювати не лише поточну популярність мови, але й її перспективи розвитку, наявність активної

спільноти, стабільність інструментів і бібліотек, що безпосередньо впливає на довгострокову підтримку системи.

За даними порталу Robot Dreams, для вебзастосунків найбільш затребуваними залишаються JavaScript, Python, TypeScript, PHP та Ruby, що відображає сучасні тренди фронтенд- і бекенд-розробки. JavaScript залишається універсальною мовою для клієнтської сторони, Python ефективно використовується для серверної логіки і роботи з базами даних, а TypeScript дозволяє підтримувати масштабованість та структурованість великих вебпроектів. PHP продовжує активно використовуватися у системах управління контентом, таких як WordPress, а Ruby зберігає популярність завдяки простоті і фокусі на швидку розробку [9].

Для початківців особливо важливим є баланс між простотою вивчення та практичною користю мови. Ресурс GOIT рекомендує новачкам розпочинати з мов, які швидко дозволяють отримати перший функціональний результат, мають активну спільноту та широкую документацію, адже це сприяє більш ефективному входженню у веброзробку. У цьому контексті JavaScript та Python є оптимальними стартовими мовами, оскільки вони дозволяють відразу працювати як з фронтендом, так і з бекендом, і формують базу для подальшого освоєння фреймворків та бібліотек [10].

Хоча класичні рейтинги популярності найчастіше відзначають JavaScript, TypeScript, Python, Java, C# та PHP, сучасні огляди демонструють помітне зростання інтересу до мови Go (Golang), зокрема в контексті cloud-native та мікросервісних рішень. У TIOBE Go піднімалась у топ-10 індексу, досягнувши рекордних позицій за останні роки, що свідчить про зростаючий інтерес до мови серед розробницької спільноти. У зв'язку з цим у роботі до розгляду включені чотири мови – JavaScript, TypeScript, Python та Go – оскільки вони найповніше відповідають сучасним вимогам вебархітектури, зокрема для SPA та мікросервісних парадигм [11].

Таким чином, аналіз популярних мов програмування показує, що вибір конкретної мови для вебзастосунку має базуватися на поєднанні кількох факторів:

популярності серед розробників, перспектив розвитку, простоти освоєння для початківців, вимог до проєкту і сумісності з обраною архітектурою. Це безпосередньо пов'язано з темою дипломної роботи, оскільки системний підхід до вибору технологій веброзробки має враховувати ці критерії для оптимізації процесу розробки та підвищення якості кінцевої вебсистеми.

1.4 Аналіз фреймворків

У сучасній веброзробці фреймворки відіграють ключову роль, оскільки вони задають структуру застосунку, визначають підхід до організації коду та суттєво впливають на швидкість створення продукту й масштабованість архітектури.

Сучасні огляди та аналітичні звіти однозначно констатують домінування React у фронтенд-екосистемі: опитування розробницьких спільнот і ринкові звіти свідчать про найвищу поширеність та попит на React серед технологій для побудови інтерфейсів, при цьому мета-фреймворки на його основі (зокрема Next.js) закріплюють позицію React як пріоритетного рішення для проєктів, де важливі SEO, SSR/SSG та ефективний перехід від прототипу до продуктивного розгортання. Дані опитувань (State of JS, Stack Overflow) відображають стабільно високі показники використання та «бажаності» React серед розробників; Vue демонструє поступове зростання як прогресивна альтернатива з більш пологою кривою навчання, а Angular зберігає значущі позиції в корпоративних проєктах завдяки суворій структурі й орієнтації на TypeScript [12].

Профільні аналітичні статті та технічні блоги деталізують зазначену ситуацію: React характеризують як універсальний інструмент із розгалуженою екосистемою пакетів і великою кількістю вакансій; Vue – як баланс між простотою впровадження та можливістю масштабування (особливо після впровадження Composition API та покращеної підтримки TypeScript); Angular – як платформу з жорсткими конвенціями, що підходить для великих команд і довготривалих

корпоративних проєктів. Порівняння за метриками GitHub, npm-завантаженнями та іншими показниками також демонструють аналогічну ієрархію, хоча в окремих регіональних та нішевих контекстах співвідношення може варіюватися. [13].

На бекендному рівні аналітичні матеріали вказують на помітне зміщення інтересу в бік легких, асинхронних та типізованих рішень. Зокрема, FastAPI за останні роки зафіксував значне зростання популярності та завантажень у репозиторіях пакетів, що поєднується з високою продуктивністю на основі Starlette/Uvicorn і зручністю автоматичної валідації через Pydantic; огляди та бенчмарки відзначають переваги FastAPI для побудови API-орієнтованих мікросервісів порівняно зі звичними WSGI-фреймворками. Одночасно Django зберігає позицію повнофункціональної платформи з набором інструментів «з коробки», що робить його гостро релевантним для класичних CRUD-додатків і швидкого створення MVP; Flask лишається мінімалістичним рішенням, яке застосовується для простих сервісів або як легка «обгортка» над існуючою логікою. У підсумку сучасні огляди відводять FastAPI роль найшвидше зростаючого інструмента в сегменті Python-API [14].

Огляди та технічні статті щодо Go-екосистеми вказують, що вибір між Gin, Fiber та Echo зазвичай обумовлений вимогами до API-стилю, продуктивності та сумісності з існуючими бібліотеками: Gin часто розглядають як лідера за популярністю серед Go-фреймворків (за метриками GitHub), Fiber приваблює розробників завдяки знайомому Express-подібному API і оптимізації на базі fasthttp, а Echo оцінюють як легкий і розширюваний фреймворк з продуманим middleware-підходом. У матеріалах, що присвячені високонавантаженим сервісам, Go-фреймворки рекомендують там, де критичною є низька латентність і висока пропускна здатність [15].

У більш загальному плані численні огляди підкреслюють, що вибір фреймворку слід розглядати як компроміс між продуктивністю (латентністю, RPS), швидкістю розробки (наявністю готових рішень у фреймворку), зрілістю екосистеми та рівнем компетенцій команди. Рецензії рекомендують у поєднанні з якісною оцінкою також використовувати кількісні метрики (PyPI/npm-

завантаження, GitHub-метрики, синтетичні бенчмарки, наприклад TechEmpower) та ілюструвати вибір практичними експериментами.

Проведений аналіз показує, що сучасний стан фронтенд- і бекенд-фреймворків характеризується значною різноманітністю та швидкою еволюцією технологій, що суттєво ускладнює процес їхнього вибору. React домінує на ринку завдяки зрілій екосистемі та широкій підтримці спільноти, Vue демонструє стабільне зростання як гнучка альтернатива, а Angular зберігає актуальність у великих корпоративних проєктах. У Python-середовищі FastAPI стрімко набирає популярність через високу продуктивність та асинхронну модель, тоді як Django й Flask продовжують виконувати нішеві ролі повнофункціонального та мінімалістичного підходів відповідно. У Go-екосистемі фреймворки Gin, Fiber та Echo є ключовими інструментами для створення високонавантажених сервісів.

З огляду на різні архітектурні підходи, продуктивність, криву навчання та специфіку застосування кожного фреймворку, процес їхнього вибору потребує ретельного аналізу і не може ґрунтуватися лише на популярності чи інтуїтивних уподобаннях. Оптимальний вибір можливий лише за умови врахування цільових вимог проєкту, технічних обмежень, досвіду команди та практичних метрик ефективності, що підкреслює важливість системного підходу до прийняття технологічних рішень.

1.5 Важливість освітніх критеріїв при підборі стеку розробки

Проведений теоретичний аналіз підтверджує наявність гострої проблеми: значна різноманітність архітектур, мов і фреймворків у поєднанні з різною якістю джерел і розрізненими підходами до оцінки технологій ускладнює прийняття виважених рішень, особливо для студентів та початківців. Існуючі публікації й огляди здебільшого фрагментарні – вони або детально аналізують технічні метрики, або описують ринкову популярність і доступність освітніх ресурсів, але

рідко поєднують ці класи критеріїв у єдині відтворювані методи. Це породжує практичні ризики: вибір технологій, обумовлений лише трендами або суб'єктивними уподобаннями, збільшує ймовірність технічного боргу, затримок у реалізації та додаткових витрат на навчання.

Підсумком дослідження має стати інтеграція результатів, отриманих за технічними та освітніми критеріями. Технічні показники, такі як: продуктивність, масштабованість, тощо слід поєднати з освітніми показниками: складність опанування, вартість навчання, попит на ринку праці, тощо. Така комбінація забезпечить збалансований метод до оцінки технологій. На основі зведених оцінок буде побудовано дерево прийняття рішень, яке формалізує логіку вибору – послідовність критеріїв і граничні значення, що визначають рекомендацію для конкретного типу користувача або проєкту. Запропоноване дерево має надати студентам і початківцям чіткі, практично застосовні інструкції для вибору стеку з урахуванням їхніх ресурсів, завдань та компетенцій, що суттєво знизить ризики помилкового вибору і спростить адаптацію в навчальному процесі.

2. АНАЛІЗ ІСНУЮЧИХ ПІДХОДІВ ТА АНАЛОГІВ У ВИБОРІ ТЕХНОЛОГІЧНОГО СТЕКУ

2.1 Пошук аналогів

Пошук аналогічних рішень та джерел, які прямо або опосередковано вирішують завдання обґрунтованого вибору стеку технологій, виконано з позиції стремління знайти як науково-методичні підходи, так і практичні інструменти та огляди ринку, котрі можуть бути адаптовані до задачі дипломної роботи. Головною метою цього етапу є ідентифікація наявних підходів, оцінка їхньої релевантності щодо поєднання технічних і освітніх критеріїв, а також виділення методологічних і практичних прогалин, які виправдовують розробку власного рішення у вигляді дерева прийняття рішень.

Пошукова стратегія передбачала комбіноване використання академічних баз даних та індустріальних джерел. До першої групи належали Google Scholar, ResearchGate та arXiv – для виявлення методологічних і теоретичних робіт щодо формалізації мультикритеріального вибору технологій, математичних моделей оцінювання та застосування нечітких/вагових методів; до другої – профільні технічні ресурси (TechEmpower), щорічні огляди і опитування спільнот (State of JS, Stack Overflow Developer Survey), а також освітні каталоги (Coursera, Udemy). Такий підхід забезпечив широку картину: від строго формалізованих академічних моделей до практичних чек-листів і синтетичних бенчмарків, які застосовуються інженерами на ринку [16-20].

Критерії відбору джерел включали:

- релевантність до теми;
- наявність чіткої методології або емпіричної перевірки;
- репутація джерела (наукові публікації, визнані галузеві бенчмарки, великі опитування спільнот);
- застосовність матеріалу для формування компонентів майбутнього дерева прийняття рішень (наприклад, набір критеріїв, правило ранжування, вимірювальні сценарії).

За такими критеріями було відібрано кілька класів аналогів: академічні моделі мультикритеріальної оцінки, систематичні бенчмарки продуктивності, ринково-аналітичні огляди та практичні гіді/чек-листи. При цьому джерела, які обмежувалися лише однією площею (наприклад – тільки популярністю у вакансіях без технічних метрик), розглядалися як часткові аналоги з обмеженою корисністю для комплексного рішення.

Академічні роботи, що пропонують формалізовані моделі, виявилися корисними як джерело ідей щодо математичного оформлення задачі та процедур отримання ваг для критеріїв. Зокрема, дослідження з використанням нечіткої логіки та інших багатовимірних моделей демонструють можливість формування шкал оцінювання і агрегування неоднорідних показників у єдиний індикатор ефективності стеку. Такі підходи корисні для формального опису операцій з

перетворення різнорідних результатів (наприклад, RPS та «кривої навчання») у порівнянні бали, але часто вимагають експертної інформації для калібрування ваг і нечітких функцій, що обмежує їх практичність для новачків без доступу до широкої експертної бази [21, 22].

Синтетичні бенчмарки, зокрема набір тестів TechEmpower, надають стандартизовані сценарії для вимірювання продуктивності веб-фреймворків (RPS, латентність, споживання пам'яті). Вони є незамінним джерелом при формуванні технічної частини оцінки, проте їхні результати слід інтерпретувати обережно: синтетичні умови не завжди відображають реальні робочі навантаження, що включають інтеграцію з базами даних, мережеві затримки, зовнішні сервіси та особливості бізнес-логіки. Тому у якості аналога TechEmpower та подібні огляди корисні як еталонні точки порівняння для низькорівневих метрик, але потребують доповнення сценаріями, ближчими до реальних кейсів, що заплановано у практичній частині дослідження [16, 23].

Ринкові огляди і великі опитування (State of JS, Stack Overflow) оперують іншими, але не менш важливими метриками: популярністю, «бажаністю» технології, частотою використання та тенденціями на ринку праці. Ці джерела є безпосереднім входом у вимір освітніх критеріїв: вони показують, які технології мають велику кількість курсів, форумів, вакансій і, відповідно, кращу можливість для навчання і працевлаштування. Аналіз освітніх пропозицій доповнює картину, дозволяючи оцінити доступність і якість навчальних матеріалів, а також їхню вартість, що важливо для аудиторії студентів і новачків [17-20].

Практичні гайди та чек-листи від консалтингових і продуктово-інженерних команд (наприклад, публікації типу «How to Choose Right Technology Stack» або «5 Steps to Choosing the Best Tech Stack») корисні як зразки поетапних процедур, які можна адаптувати у вигляді фільтрів або вузлів у дереві прийняття рішень. Такі матеріали зазвичай враховують організаційні аспекти: наявність кадрів, часові обмеження проєкту, ризики технічного боргу і необхідність довготривалої підтримки, що відповідає критеріям економічної доцільності і сумісності з навчальними пріоритетами. Водночас вони часто не мають чіткої математичної

бази або емпіричного підкріплення, тому вимагають формалізації та перевірки на практичних сценаріях, що і буде зроблено в межах проєкту [24, 25].

На етапі пошуку були також виявлені приклади комплексних оглядів стеків (MEAN, MERN, LAMP), які поєднують архітектурні методи, витрати та освітню складову. Такі матеріали корисні для формування базового переліку альтернатив і розуміння профілювання стеків, а також для ілюстрації типових компромісів між простотою навчання і продуктивністю. Проте академічні порівняння зазвичай не враховують сучасних асинхронних фреймворків (наприклад, FastAPI) або Go-стеків у повній мірі, що вказує на додаткову нішу для оновленого дослідження [26].

Результатом ієрархізації знайдених аналогів стала класифікація джерел за їх внеском у майбутню методику:

- формалізовані моделі – як джерело ідей для агрегування неоднорідних метрик;
- бенчмарки – як основа для технічної перевірки;
- ринкові опитування та каталоги курсів – як джерело освітніх метрик;
- практичні гайди – як джерело процедурних кроків для побудови логіки дерева рішень.

Наведені джерела мають і обмеження, які необхідно врахувати. По-перше, жодне окреме джерело не поєднує в собі повністю технічні й освітні критерії в єдиній відтворюваній процедурі; по-друге, синтетичні бенчмарки часто упереджені під час інтерпретації щодо конкретних робочих навантажень; по-третє, ринкові опитування відображають настрої спільнот і попит на момент проведення опитування, що в умовах швидкої еволюції технологій може давати застарілі сигнали. Ці недоліки вказують на необхідність комбінувати дані з різних джерел і застосовувати чутливий аналіз результатів при побудові рекомендацій.

2.2 Порівняння аналогів

Дослідження, в якому запропоновано модель вибору технологічного стеку на основі нечіткої логіки, застосовує формалізовану методику до агрегування неоднорідних критеріїв: кількісні метрики продуктивності й кількісно-якісні оцінки з інтерпретативною природою переводяться в нечіткі множини і, через набір правил та функцій належності, зводяться в інтегральний індекс ефективності. Методика демонструє можливість задавати нечіткі пороги між категоріями «прийнятно / допустимо / неприйнятно», що робить її корисною при роботі з неточними даними й суб'єктивними оцінками. Водночас практична реалізація цієї моделі вимагає значної експертної участі для калібрування функцій належності й правил агрегації, що ускладнює її застосування в контексті студентських проєктів без додаткових автоматизованих процедур чи спрощень; у порівнянні з процедурними методами нечітка логіка дає вищу точність у формалізації непевних оцінок, зате поступається їм у простоті і прозорості для початківців [21].

Університетські керівництва та дипломні методики, які формалізують алгоритм вибору технологій, орієнтуються на процедурну послідовність: збір вимог, застосування жорстких обмежень для початкового відсію, після чого використання простих рангових або бальних систем для остаточного порівняння. Така методика відрізняється зрозумілістю та практичною застосовністю – вона легко документується й може бути швидко реалізована у навчальному середовищі – але в разі множинної взаємозалежності критеріїв не забезпечує достатньої точності агрегації результатів. Порівняно з нечіткими та статистично-алгоритмічними моделями, університетські процедури мають перевагу у доступності для студентів, проте потребують доповнення формалізованими правилами для коректної обробки складних компромісів між критеріями [22].

Набір синтетичних бенчмарків, що представлений у TechEmpower, пропонує строгу методику до порівняння продуктивності фреймворків в умовах стандартизованого тестування: єдині сценарії виконуються на уніфікованих

конфігураціях, що дозволяє отримувати зіставні значення RPS, p99-латентності та споживання ресурсів. Ця методика дає об'єктивні технічні входи, корисні для калібрування технічного блоку будь-якої моделі вибору, проте її результати слід інтерпретувати у зв'язку з реальними робочими навантаженнями, які включають інтеграції з базами даних, мережеві фактори й особливості бізнес-логіки; таким чином, TechEmpower є необхідним, але не достатнім елементом технічної оцінки, і його слід поєднувати з експериментами на репрезентативних стендах [16].

Практичні інженерні огляди, які порівнюють, наприклад, FastAPI і Flask, використовують змішану методику: поєднання локальних емпіричних тестів із якісним аналізом зручності розробки, ергономіки API і доступності екосистеми. Такі матеріали зазвичай орієнтовані на практиків і відрізняються тим, що прагнуть контекстуалізувати цифри бенчмарків у реальних сценаріях застосування, описати компроміси у розробці та надати рекомендації щодо практичних виборів. У порівнянні зі строгими бенчмарками, ці огляди пропонують більш прикладну перспективу, а порівняно з університетськими методиками – більше емпіричної конкретики, але їхні висновки часто залежать від конфігурації тестів і суб'єктивної інтерпретації автора [23].

Щорічні опитування та огляди спільнот, у тому числі State of JS і Stack Overflow Developer Survey, базуються на репрезентативних для своєї вибірки анкетних даних і забезпечують інформацію про популярність, «бажаність» та тенденції у використанні технологій. Методологія таких досліджень дозволяє оцінити імовірність легкого доступу до наставництва, вакансій і навчальних матеріалів для конкретної технології, що робить їхні дані цінною складовою освітнього блоку оцінки. Порівнюючи їх з каталогами курсів, можна зіставити соціально-ринкові сигнали з реальною доступністю навчального контенту; у порівнянні з технічними джерелами ці опитування дають зовсім інший тип входу – ринковий і соціальний, який необхідно враховувати при рекомендаціях для студентів [17, 18].

Каталоги масових онлайн-курсів, такі як Coursera і Udemy, надають емпіричні дані щодо доступності та вартості навчання, кількості курсів, рейтингів

та відгуків, що дозволяє оцінити, наскільки легко студенту знайти структурований навчальний матеріал та яку мінімальну інвестицію часу й коштів необхідно вкласти. Методологічно ці платформи не замінюють опитування спільнот, але доповнюють їх кількісними показниками доступності освіти, а саме: складність освоєння, ціна та час на навчання, порівняно з формалізованими моделями каталогів курсів дають практичні індикатори витрат на навчання, проте їх слід фільтрувати за якістю матеріалів і актуальністю контенту [19, 20].

Інженерні гайди і бізнес-чеки, що публікують компанії на кшталт TechMagic або Leanware, пропонують прості, але практично орієнтовані процедури: аналіз вимог, оцінка ризиків, визначення часових обмежень і пріоритезація критеріїв. Методологія таких матеріалів робить їх природним джерелом правил «якщо-то» для вузлів у дереві прийняття рішень, оскільки вони відображають поширені корпоративні практики управління ризиками і ресурсами. Порівняно з академічними моделями, ці гайди цінні своєю практичною доступністю, але потребують формалізації і емпіричної валідації перед тим, як їх можна буде надати студентам як частину офіційних рекомендацій [16, 25].

Академічні порівняння традиційних стеків, які аналізують MEAN, MERN та LAMP, застосовують методологію комплексного порівняння архітектур, враховуючи витрати, криву навчання і підтримку спільноти. Ці роботи створюють корисний історичний контекст і дають початковий набір критеріїв для відсіювання очевидно невідповідних альтернатив, але зазвичай потребують оновлення, щоби адекватно охопити сучасні асинхронні та високопродуктивні технології, які нині набирають популярності. У зіставленні з сучасними бенчмарками й оглядами вони є відправною точкою для розуміння фундаментальних компромісів між простотою та продуктивністю [26].

Дослідження, що розглядають застосування штучного інтелекту для автоматизації вибору стеку, пропонують методологію, основу на навчанні моделі на історичних кейсах та метаданих про проєкти, з можливістю динамічного коригування ваг критеріїв на підставі накопиченої статистики. Хоча і методологія відкриває перспективи для майбутньої автоматизації і підвищення адаптивності

рекомендацій, вони наразі вимагають значних даних і ретельної валідації, а отже розглядаються поки що як довгострокова перспектива, а не як безпосередньо застосовний інструмент для студентських сценаріїв [27].

Після опису кожного із зазначених підходів варто відзначити ключові співставлення: нечіткі та алгоритмічні моделі забезпечують математичну строгість, але потребують калібрування й експертних ресурсів; синтетичні бенчмарки дають об'єктивні технічні входи, проте їх треба доповнювати реалістичними тестами; опитування спільнот і каталоги курсів дають освітні та ринкові індикатори, що є незамінними при рекомендаціях для студентів; практичні гайди корисні для формалізації дискретних правил відсіву в дереві прийняття рішень.

2.3 Висновки за результатами аналізу аналогів

Проведений аналіз наявних підходів і практичних рішень підтверджує, що завдання обґрунтованого вибору стеку технологій є мультидисциплінарним і вимагає поєднання різнорідних типів даних. Існуючі джерела забезпечують корисні, але часткові інструменти: формалізовані моделі (нечітка логіка, АНР) надають математичні засоби для агрегування неоднорідних метрик; синтетичні бенчмарки (TechEmpower) дають об'єктивні технічні входи; ринкові опитування і каталоги курсів (State of JS, Stack Overflow, Coursera, Udemy) постачають інформацію про освітню доступність і попит; практичні гайди – прості процедурні правила для відсіву альтернатив. Жоден із розглянутих підходів сам по собі не покриває повністю потреби студентської аудиторії: математична строгість часто супроводжується складністю калібрування, синтетичні тести виводять лише частину технічної картини, а ринкові сигнали можуть бути часово зміщеними.

На основі порівняльного аналізу доцільно прийняти комбінований підхід: технічні ваги базуватимуться на результатах стандартизованих бенчмарків та додаткових експериментах на репрезентативних стендах (у тому числі із

застосуванням авторського інструменту), освітні ваги – на статистиці опитувань і доступності курсів, а процедурні правила для дерева прийняття рішень – на адаптованих практичних гайдах. Також необхідно передбачити процедуру чутливісного аналізу та калібрування ваг, щоб забезпечити стійкість рекомендацій у різних сценаріях застосування. Врахування цих положень дозволить сформулювати адаптивний і зрозумілий для студентів інструмент, який знизить ризики помилкового вибору технологій і підвищить практичну цінність рекомендацій.

3. ПРОЄКТУВАННЯ МЕТОДІВ ТА ІНСТРУМЕНТУ ВИБОРУ СТЕКУ

3.1 Складові дослідження й обґрунтування подальших кроків

Для досягнення мети дослідження – створення надійного, відтворюваного та практично корисного алгоритму прийняття рішень, який допоможе студентам і початківцям обрати стек розробки – необхідно реалізувати кілька взаємопов'язаних дослідницьких етапів. Першим і найбільш фундаментальним кроком є чітке визначення критеріїв оцінювання: технічні та освітні показники повинні бути сформульовані не як абстрактні поняття, а як репрезентативні індикатори з описом методик їхнього вимірювання, умов проведення спостережень і правил нормалізації. Формалізація критеріїв забезпечує однорідність вхідних даних і є передумовою для коректної агрегації результатів та отримання відтворюваних висновків.

Другим етапом є проведення емпіричних тестів і збір кількісних даних у контрольованих умовах. Цей крок обґрунтований тим, що лише на основі систематично зібраних і документованих експериментальних результатів можна однозначно визначити технічні властивості розглянутих рішень і оцінити їхню поведінку в реальних умовах виконання. Експериментальний блок повинен включати розробку набору репрезентативних сценаріїв, процедури повторюваних

прогонів, фіксацію умов стенду й методу обробки сирих даних, що гарантуватиме коректну нормалізацію та порівняння.

Третім необхідним складником є систематичний аналіз надавачів освітніх ресурсів і ринкових сигналів. Цей блок передбачає збір даних про доступність і якість навчальних матеріалів, структуру курсів, наявність офіційної документації, а також аналіз попиту на ринку праці. Поєднання кількісної інформації з якісною оцінкою дозволить адекватно вагувати освітню складову при ухваленні рішення, що особливо важливо для цільової аудиторії – студентів і новачків.

Четвертий етап полягає в інтеграції зібраних технічних та освітніх даних у статичний алгоритм прийняття рішень, призначений для рекомендацій у контексті повноцінних проєктів. На відміну від інтерактивного чи адаптивного дерева, цей алгоритм реалізує заздалегідь визначену послідовність перевірок і правил ранжування, які застосовуються до нормалізованих оцінок і дозволяють отримати однозначний висновок щодо придатності стеку для поставлених задач. Використання статичного алгоритму спрощує відтворюваність результатів, зменшує потребу в експертних інтервенціях у процесі роботи і забезпечує прозорі правила прийняття рішень у межах предметної області дослідження.

Кожен із перелічених етапів є невідмінними складовими дослідження. Виконання цих кроків у зазначеній послідовності забезпечує логіку дослідження – від формування репрезентативних входів до генерації зрозумілих, кількісно підкріплених рекомендацій.

3.2 Відбір критеріїв оцінювання

У межах підрозділу формалізується набір критеріїв, які будуть застосовані для кількісної та якісної оцінки розглянутих фреймворків і стеків при виконанні практичної частини дослідження. Відбір критеріїв базується на попередньому теоретичному аналізі і спрямований на забезпечення повноти, відтворюваності та

практичної корисності результатів для цільової аудиторії – студентів і початківців. Критерії згруповано за двома великими класами – технічні та освітні, причому, кожна група охоплює набір взаємодоповнювальних показників, що разом дозволяють всебічно оцінити придатність технології для реальної розробки.

До технічних критеріїв віднесено показники, які відображають продуктивність і ресурсну ефективність реалізацій, стійкість поведінки під навантаженням та мережеві характеристики застосунків. У рамках продуктивності розглядаються часові показники, що формують загальне уявлення про швидкість виконання робіт (зокрема середні значення для математично-інтенсивних і алгоритмічних ділянок), оскільки вони визначають прийнятний час реакції і пропускну здатність сервісу. Параметри використання пам'яті включають окремі індикатори для роботи з масивами і для об'єктної моделі, а також їх сумарні значення. Така деталізація дозволяє відрізнити оптимізовані під великі блоки даних реалізації від тих, що передбачені для обробки великої кількості об'єктів. Оцінка стабільності спрямована на виявлення варіацій у поведінці системи: середні значення і їх відхилення дають інформацію про детермінованість виконання, індикатори витоку пам'яті і тренди дозволяють прогнозувати деградацію у часі, а побудовані індекси стабільності служать узагальненими показниками для порівняння реалізацій. Масштабованість оцінюється через зміну часових характеристик при збільшенні навантаження та коефіцієнт деградації, що показує, наскільки продуктивність знижується в реалістичних сценаріях. Окрему групу становлять метрики, що описують поведінку асинхронних сценаріїв: середній час однієї асинхронної задачі, зміна часу при збільшенні кількості паралельних задач і сумарний час пакетної обробки – вони визначають придатність технології до асинхронних моделей і конкурентних навантажень. Нарешті, мережеві показники, серед яких: затримка, джиттер, втрата пакетів, пропускну здатність вводяться як коригувальні фактори, що впливають на інтерпретацію часових метрик у розгорнутих системах з розподіленою архітектурою.

Освітні критерії обрані з огляду на потреби цільової аудиторії та практичну придатність рекомендацій. Складність опанування і вартість навчання визначають

реальні інвестиції часу і ресурсів, яких потребуватиме студент для досягнення продуктивності з обраною технологією; попит на ринку праці відображає перспективи працевлаштування та затребуваність навичок; розвиненість спільноти та доступність документації і навчальних ресурсів впливають на можливість самостійного навчання і пошуку підтримки та актуальність, яка є вихідною з попиту на ринку праці, але має характер прогнозу. Дані критерії дозволяють сформулювати збалансований освітній профіль кожного стеку.

Вибрані критерії не розглядаються ізольовано: у методиці передбачено логіку їхньої взаємодії і пріоритизації. Пріоритети між технічними і освітніми блоками визначатимуться за замовчуванням для повноцінних проєктів, тобто алгоритм прийняття рішення використовуватиме набір ваг і правил без диференціації за типами проєктів на цьому етапі. Надалі можливе виносне перенастроювання ваг для специфічних випадків, але базова модель повинна забезпечувати репрезентативні рекомендації для типових завдань.

Обґрунтування вибору зазначених складових полягає у прагненні забезпечити двовимірну релевантність результатів: по-перше, технічна повнота дає змогу відокремити рішення, які формально задовольняють вимоги продуктивності та стабільності; по-друге, освітня складова гарантує, що рекомендації є практично здійсненними для студентів, які обмежені ресурсами і часом на навчання. У підсумку відбір критеріїв сформує набір репрезентативних входів для подальшої обробки; ці входи ляжуть в основу статичного алгоритму прийняття рішень і забезпечать його обґрунтованість, відтворюваність та прикладну корисність.

3.3 Методика збору технічних метрик

Методика збору технічних метрик полягає в серії прогонів допустимих варіантів стеків технологій за допомогою авторського інструмента. Потреба у ньому обґрунтована кількома факторами. По-перше, лише централізована система

дозволяє створити єдині стандарти запуску експериментів: однакові сценарії, фіксовані параметри стенду, ідентичні процедури прогонів та збереження повних метаданих (версії ПЗ, конфігурації, час запуску). По-друге, автоматизація знижує людський фактор і ймовірність помилок при багаторазових прогонах. По-третє, інструмент забезпечує зручну агрегацію й довготривале зберігання сирих даних, що прискорює подальшу нормалізацію, аналіз чутливості та побудову рекомендацій. З огляду на ці вимоги вебзастосунок виступає оптимальною формою реалізації: його інтерфейс забезпечує просте задавання експериментів, а серверна частина – надійне середовище запусків і збір даних.

Інструмент проектується як вебзастосунок з трьома основними підсистемами: інтерфейс користувача для формування конфігурацій експериментів і перегляду результатів, модуль виконавця (сервер), що запускає тестові сценарії і збирає метрики, та сховище результатів (база даних). Користувач обирає набір (стек) технологій, що включає архітектуру, мову програмування та фреймворк, або обирає готові тестові сценарії. Сервер запускає сценарій випробувань до заданого стеку, зберігає дані у базу даних та надсилає результати до користувацького інтерфейсу, який показує їх користувачу. У разі потреби можна подивитися на увесь список проведених раніше випробувань. Даний список виглядатиме як таблиці, в яких наведені параметри випробування (стек технологій), час проведення та всі отримані результати. Алгоритм роботи вебзастосунку наведений на рисунку 3.1.

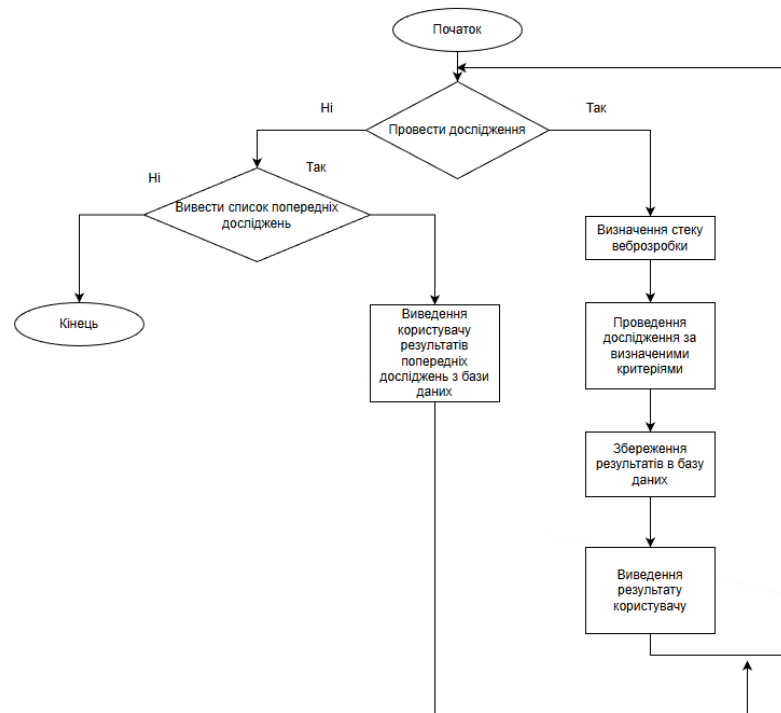


Рисунок 3.1 – Алгоритм роботи вебзастосунку

Прогнозовано, що кількість запусків випробувань (прогонів) для кожного стеку буде три рази, а це означає, що отримані результати потрібно усереднити і для їх відображення треба використати таблицю. Причиною збільшення вибірки результатів обраний через отримання більш ймовірних результатів і зменшення похибки в обчисленні.

Описана методика збору й аналізу технічних метрик через спеціалізований вебінструмент забезпечує ключові переваги експериментальної частини дослідження: стандартизацію сценаріїв, автоматизацію прогонів, централізацію даних та їхню прозору підготовку до подальшого використання. Запропонована архітектура й алгоритми гарантують відтворюваність і дозволяють одержати коректні технічні входи для статичного алгоритму прийняття рішень, який буде застосований у практичній частині роботи. Деталі будуть наведені у розділі, присвяченому реалізації інструмента і практичним експериментам.

3.4 Розробка інструмента для збору технічних метрик

Як було сказано у підрозділі 3.3, даний інструмент для збору технічних метрик являє собою клієнт-серверний вебзастосунок, який спрямований для запуску випробовувань для стеків технологій веброзробки з метою отримання числових даних згідно технічним критеріям та збереження результатів випробовувань.

Відповідно до технічного плану інструмент реалізується як вебзастосунок з клієнтською частиною на чистому TypeScript (без застосування фронтенд-фреймворків), серверною частиною на TypeScript із Node.js та Express.js і простим JSON-сховищем (json-server) для проміжного зберігання та підготовки даних. Такий вибір дозволяє швидко прототипувати інтерфейс і API, забезпечує легкість розгортання і простоту формату збереження.

Основні вимоги до клієнтської частини включають відображення користувачу основних функцій та можливостей, а саме вибір стеку технологій що можуть бути взяті з шаблону, або вручну обрані за допомогою конструктора і відправка їх на сервер для подальшої логіки та отримання результатів випробовування, включно з можливістю переглянути попередні. Згідно цього, сервер має отримувати дані з клієнта, запускати логіку випробовувань та отримувати результати, зберігати їх і відправляти назад до клієнта. Окрім цього сервер має взаємодіяти з базою даних з метою читання та запису (збереження) даних.

Клієнтська частина інструмента побудована як невеликий, але функціонально завершений веб-інтерфейс на чистому HTML/CSS і TypeScript. Весь інтерфейс формується динамічно за допомогою функцій, що створюють елементи й одразу навішують обробники подій. При завантаженні сторінки запускається функція отримання шаблонів, яка звертається до серверу та виводить перелік готових стеків. Кожен шаблон рендериться як окремий елемент списку із заголовком, набором ілюстрацій та списком специфікацій, і до нього додається кнопка «Протестувати». Такий метод дає змогу тримати інтерфейс максимально

простим і водночас гнучким – зміна моделі шаблону на сервері автоматично відобразиться у клієнті. На рисунку 3.2 зображено початковий екран вебсторінки.

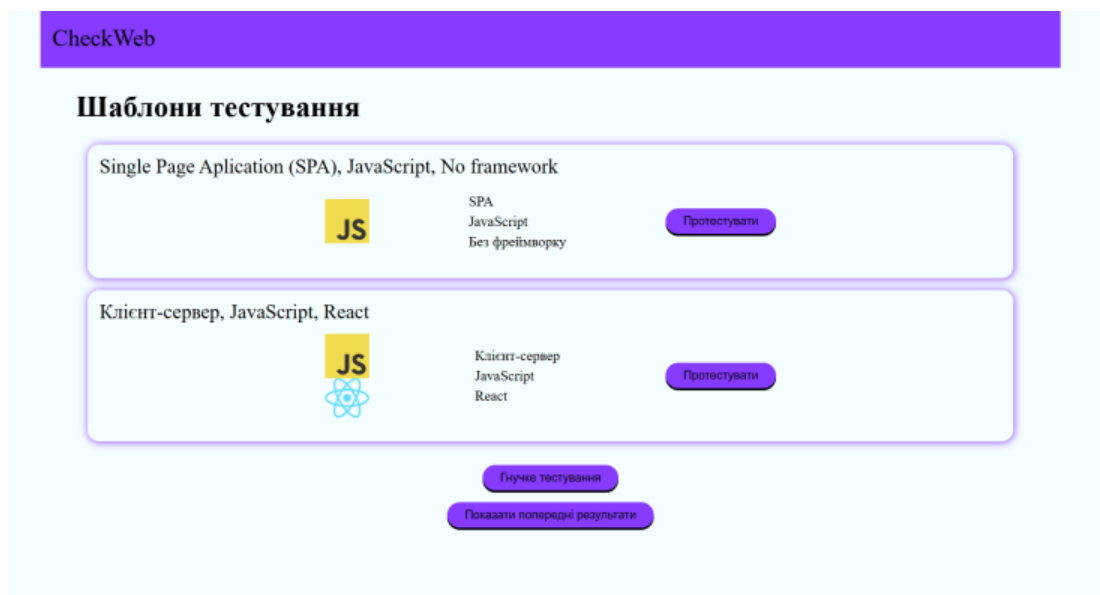


Рисунок 3.2 – Початковий екран зі списком шаблонів

Запуск тесту здійснюється через простий ланцюжок: натискання кнопки генерує POST-запит до ендпоінту «/runTest» з ідентифікатором шаблону, після чого інтерфейс відкриває напівпрозорий модальний шар і виводить анімований текст «Йде завантаження». Механізм індикації стану реалізований локально: створюється «чорний екран», на ньому з'являється абзац із циклічно змінним рядком крапок – доки сервер не поверне відповідь. Коли відповідь надходить, вміст модального вікна замінюється на табличне відображення результату – функція, що створює цю таблицю, розбирає структуру отриманого JSON і формує читабельну таблицю з підписами «Параметр» і «Значення», а також окремо виводить метадані прогона (параметри запуску та час). Такий дизайн забезпечує просту, передбачувану взаємодію: користувач бачить, що запит передано, і після завершення отримує повний набір метрик у зрозумілому вигляді. На рисунках 3.3 і 3.4 зображено стан прогону (ще завантажуються) та отриманий результат у вигляді таблиці відповідно.

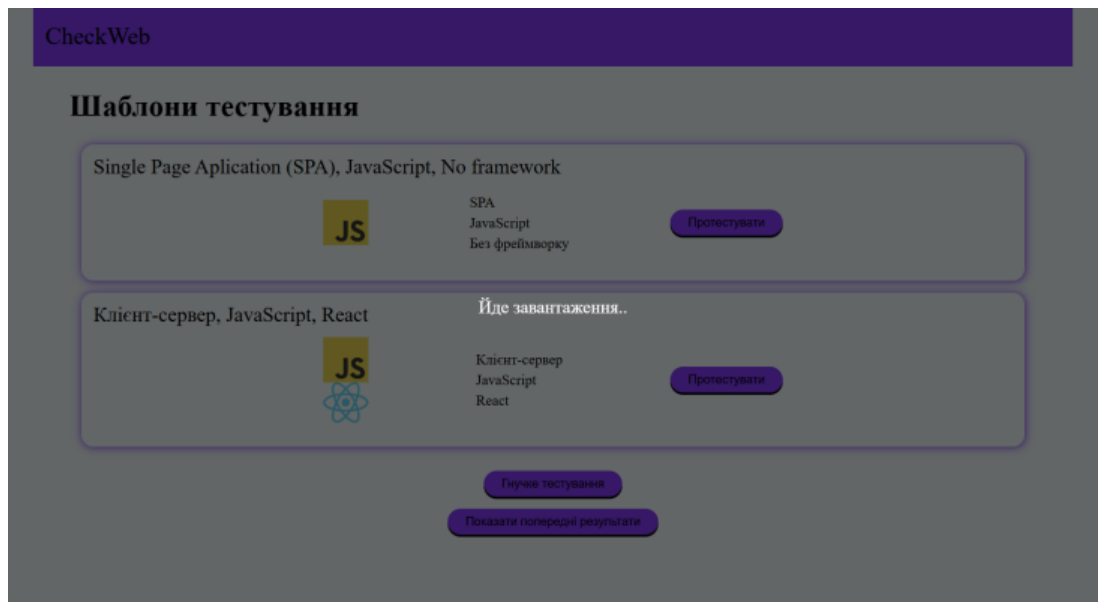


Рисунок 3.3 – Модальне вікно завантаження

CheckWeb

Шаблиони тестування

Single Page Application (SPA), JavaScript, No framework

Клієнт-сервер, JavaScript, React

Параметри запуску: SPA, JavaScript, Без фреймворку	
Час проведення запуску випробування: 18:26	
Результати тестувань	
Параметр	Значення
Середній час математичного навантаження (ms)	16.671
Середній час алгоритмічного навантаження (ms)	1.073
Середній загальний час виконання коду (ms)	17.763
Середнє споживання пам'яті масивами (MB)	3.503
Середнє споживання пам'яті об'єктами (MB)	3.719
Середнє загальне споживання пам'яті (MB)	7.219
Середній час виконання (ms)	56.989
Відхилення часу (ms)	5.695
Середнє використання пам'яті (MB)	442.676
Відхилення пам'яті (MB)	156.482
Середній витік пам'яті (MB)	22.636
Тренд витоку пам'яті (MB)	515.311
Індекс стабільності часу	0.9
Індекс стабільності пам'яті	-0.164
Середній час обробки запиту (ms)	247.474
Середній час на запит (ms)	1.627
Середній коефіцієнт деградації	2.012
Середній час виконання 1 асинхронної задачі (ms)	0.348
Середній коефіцієнт зміни часу при зростанні кількості задач	1.489
Середній загальний час виконання пакетів задач (ms)	35.213
Середня затримка (ms)	35.93
Середній джиттер (ms)	0.916
Середня втрата пакетів	0
Середня пропускна здатність (KB/ms)	177.831

Рисунок 3.4 – Екран результату з таблицею

Поряд із шаблонами реалізовано можливість формувати власну конфігурацію тесту через невеликий конструктор стеку. При активації цього режиму клієнт запитує у сервера перелік доступних архітектур, мов і фреймворків, після чого динамічно створює групи опцій. Логіка взаємозв'язків реалізована на стороні інтерфейсу: вибір мови може активувати або деактивувати відповідні фреймворки (за ознакою «tiedTo»), а візуальне відображення вибору здійснюється

через додавання/видалення класів «selected» та «disabled». Після вибору всіх трьох компонентів користувач може ініціювати індивідуальний тест, який викликає інший ендпоінт «/runCustomTest» і використовує ту ж саму схему відображення результатів. Конструктор виконаний так, щоб забезпечити інтуїтивну роботу та запобігти некоректним поєднанням технологій. На рисунках 3.5 та 3.6 наведено конструктор у його початковому стані та у процесі вибору стеку, де спрацьовує функція запобігання некоректних поєднань технологій відповідно.

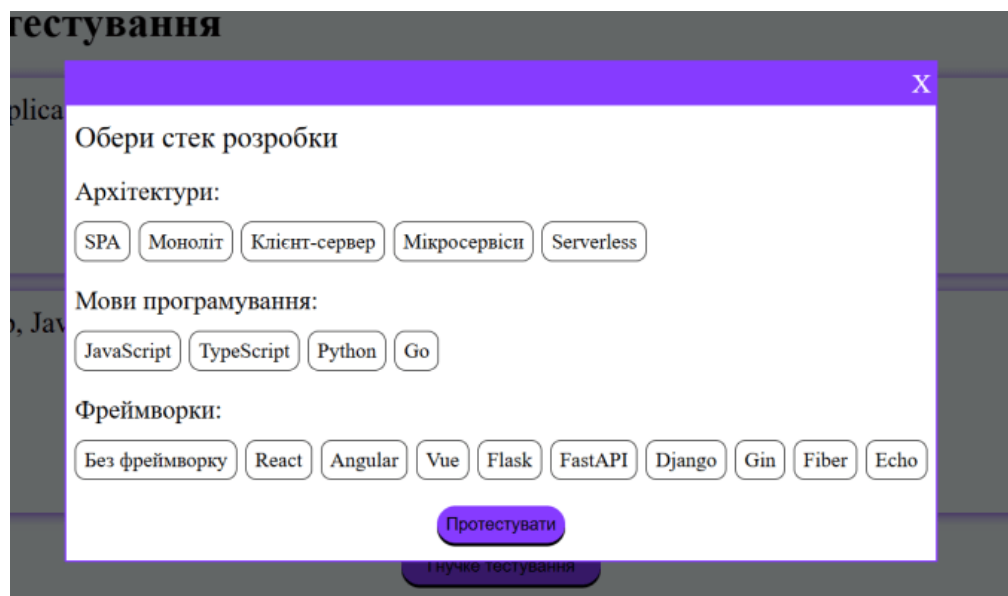


Рисунок 3.5 – Конструктор обрання стеку

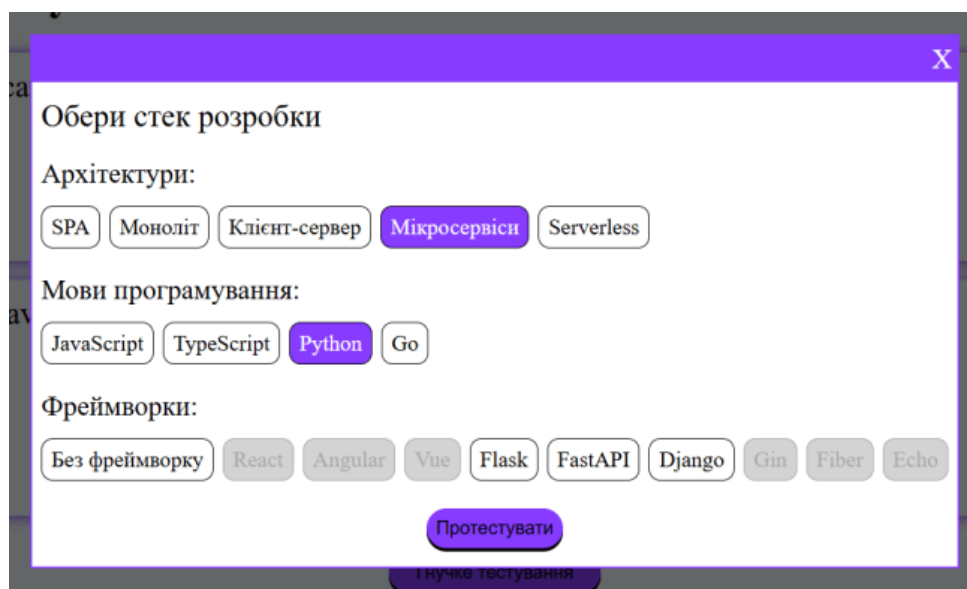


Рисунок 3.6 – Запобігання некоректних поєднань технологій

Окремо варто відзначити спосіб, у який клієнт працює з історією: запит до «/getResults» повертає масив записів, які клієнт перетворює на послідовність таблиць – кожна таблиця відповідає одному прогону з усіма метриками й метаданими. Це дозволяє швидко переглянути накопичену базу результатів прямо в інтерфейсі без додаткових інструментів. Формат очікуваних даних у коді має декілька простих припущень (наприклад, перші два ключі об’єкта можуть бути «Параметри запуску» та «Час проведення запуску»), що полегшує читабельність таблиці, але потребує узгодженості формату на стороні серверу.

Технічно, клієнт побудований на базових сучасних веб-технологіях: асинхронна взаємодія здійснюється через `fetch` із Promise-ланцюжками, стан інтерфейсу зберігається простими об’єктами та відображається маніпуляцією DOM, а валідація введення і логіка взаємозв’язків здійснюються прямими перевітками і зміною CSS-класів. У коді присутні утиліти для створення закритого модального шару з хрестиком для закриття, для формування таблиць результатів і для покрокового відображення стану завантаження; всі ці елементи роблять інтерфейс самодостатнім і не залежать від сторонніх бібліотек.

Серверна частина інструмента виконана як невеликий, але функціонально завершений HTTP-сервіс на базі Express і призначена для координації випробувань, агрегації результатів і їхнього збереження. При ініціалізації додатка підключено базові проміжні програми – CORS для організації крос-доменної взаємодії та «`express.json()`» для парсингу тіла запитів – що дозволяє клієнту надсилати конфігурації тестів у вигляді JSON. Конфігураційні константи вказують на дві зовнішні точки: колекцію шаблонів і колекцію результатів у простому JSON-сховищі (наприклад, `json-server`), які використовуються відповідно для зчитування набору тестових шаблонів і для збереження отриманих прогонів.

Логіка сервера структурована навколо набору REST-ендпоінтів, кожен із яких виконує чітку роль у робочому процесі. Ендпоінти для отримання списку шаблонів і факторів сценаріїв слугують джерелом даних для клієнтського каталогу і конструктора конфігурацій. Ендпоінт отримання результатів повертає накопичену базу експериментів для перегляду в інтерфейсі.

Ключовим функціоналом є обробники запуску тестів: ендпоінти, які приймають або ідентифікатор шаблону, або довільну специфікацію від клієнта, передають отримані параметри у центральну функцію «runTest». Ця функція створює екземпляри набору модульних тестових класів (CodePerformanceTest, MemoryPerformanceTest, StabilityPerformanceTest, ScalabilityPerformanceTest, AsyncPerformanceTest, NetworkPerformanceTest), послідовно викликає їхні методи запуску та збирає повернені об'єкти. У результаті формується єдиний агрегований JSON-об'єкт з метаданими (параметри запуску, час) і повним набором технічних метрик, який відсилається клієнту і фіксується у сховищі результатів через окрему функцію збереження.

Збереження результатів організовано як простий POST-виклик до кінцевої точки сховища. Після отримання сукупного результату сервер передає дані на запис, а відповідь із агрегатом повертається клієнту. Така структура робить сервер координатором експериментів: він приймає конфігурацію, делегує виконання спеціалізованим модульним класам, агрегує результати і гарантує, що вихідні дані будуть повними й структурованими для подальшої обробки.

Окрему частину реалізації становлять власні скрипти і модулі випробувань, які викликаються з «runTest». Детальний опис їхньої реалізації, логіки генерації навантаження та алгоритмів збору окремих технічних метрик наведено в наступному розділі.

Як сховище шаблонів і результатів випробувань використано просту REST-сервісну імплементацію на базі json-server. У рамках прототипу json-server виконує роль легкої локальної бази даних: він надає HTTP-інтерфейс до файлу з даними і обробляє стандартні запити (GET, POST тощо) на кінцеві точки, що використовуються сервером інструмента. Практично це виражено у двох колекціях – Templates (шаблони стеків) і Results (збережені прогони). Сервер звертається до відповідних адрес для читання переліку шаблонів, запису та отримання агрегованих результатів.

Описаний інструмент є невеликою, але чіткою ілюстрацією принципу «простого прототипування»: він не прагне стати повнофункціональною панеллю з

усіма можливими функціями, натомість робить акцент на надійному зборі й відображенні даних, прозорій логіці взаємодії та простоті подальшої інтеграції, причому реалізовано його з розрахунком на локальне використання без потреби в хостингу.

3.5 Визначення технічних вимог до розроблюваної вебсистеми

Для розроблюваної вебсистеми було визначено такі технічні вимоги, що наведені у таблицях 3.1 і 3.2 . Таблиці містять мінімальні та рекомендовані характеристики для клієнтської та серверної частин системи, а також короткі примітки щодо застосування та призначення кожного параметра.

Таблиця 3.1 – Технічні вимоги для клієнтської частини

Компонент	Мінімальні вимоги	Рекомендовані вимоги	Примітка
Процесор (CPU)	Двоядерний 2.0 GHz (x86_64 або arm64)	Чотириядерний 2.5 GHz або вище	Забезпечує плавну роботу UI та одночасних вкладок
Оперативна пам'ять (RAM)	4 GB	8 GB і більше	Для зручної роботи з браузером та devtools
Постійна пам'ять (ROM)	64 GB (SSD бажано)	128 GB SSD	SSD пришвидшує запуск ОС та кешування ресурсів
Мережа (інтернет)	3 Mbps (завантаження)	10 Mbps	Стабільне з'єднання для API-запитів та завантажень

Продовження таблиці 3.1

Компонент	Мінімальні вимоги	Рекомендовані вимоги	Примітка
Монітор (розмір в пікселях)	1024×576	1920×1080	Для зручного відображення інтерфейсу й таблиць метрик
Пристрої	ПК, ноутбук	ПК, ноутбук	Система не адаптована під мобільні пристрої
Операційні системи	Windows 10, Linux, macOS	Windows 10, Linux, macOS	Підтримка актуальних релізів підвищує сумісність
Браузери	Chrome/Edge/Firefox/Safari (останні 2 версії)	Останні версії Chrome/Firefox/Edge/Safari	Підтримка ES6+, HTTPS, TLS

Таблиця 3.2 Технічні вимоги до серверної частини

Компонент	Мінімальні вимоги	Рекомендовані вимоги	Примітка
Процесор	Двоядерний 2.0 GHz (x86_64 або arm64)	Чотириядерний 2.5 GHz або вище	Для запуску тестів і паралельної роботи сервера
Оперативна пам'ять	4 GB	8 GB і більше	Пам'ять для Node.js процесу, тестових симуляцій і кешу
Постійна пам'ять	40 GB (SSD бажано)	80-120 GB SSD	Місце для логів, тимчасових файлів і результатів тестів
Мережа	10-50 Mbps	100 Mbps	Достатньо для локального збереження/завантаження, інтеграцій з локальною БД чи сервісами
Середовище виконання	Node.js 16	Node.js 18+	Використовувати LTS-версії

Запропоновані технічні вимоги формують чітку базу для розгортання, тестування та експлуатації розробленої вебсистеми – інструмента для визначення технічних метрик. Для робочих місць користувачів достатньо мінімальних конфігурацій, але для стабільної роботи рекомендується використовувати вищі (рекомендовані) характеристики, що передбачають контейнеризацію, балансування навантаження, кешування та реплікацію бази даних.

3.6 Методика аналізу метрик за освітніми критеріями

У межах методики аналізу метрик за освітніми критеріями передбачається ручне опрацювання постачальників навчальних ресурсів, документацій, актуальної платформи спільноти розробників та вакансій. Ручний огляд дозволяє відокремити релевантні пропозиції від рекламних або неподібних за змістом курсів, зафіксувати якісні особливості програм та витягти порівнянні показники – зокрема орієнтовну ціну навчання, тривалість курсу, наявність практичних прикладів, популярність на ринку праці.

Результатом цього етапу є структурована таблиця результатів, що містить для кожної технології освітні показники. Далі формується друга таблична структура – агрегована таблиця середніх значень по джерелах, яка має допомогти у подальшій процедурі зважування та ранжування.

Таким чином, отримавши усереднені результати за освітніми критеріями, отримуємо другу, і останню, частину для подальшого поєднання результатів з метою отримання дерева прийняття рішення.

3.7 Проєктування дерева прийняття рішень

Метою проєктування дерева прийняття рішень є перетворення набору гетерогенних результатів випробувань та аналітичних показників у зрозумілу, відтворювану й обґрунтовану процедуру вибору технологічного стеку. Ключова ідея полягає у поєднанні двох компонентів: нормалізованих результатів експериментів та ваг, що визначають відносну важливість кожного критерію у загальній оцінці. Дерево реалізує цю ідею як послідовність відсівів і ранжувальних кроків: фільтри для критичних вимог і агреговані зважені оцінки для порівняння альтернатив.

Визначення ваг здійснюється у два етапи. Перший – формування початкових (базових) ваг за допомогою метода «мін-макс». Ідея тут прагматична: створити прозору і відтворювану стартову конфігурацію, де групи критеріїв отримують попередні частки, а всередині групи ваги розподіляються за вибраним правилом (наприклад, рівномірно по групах важливості). Метод «мін-макс» у цьому контексті забезпечує просту процедуру приведення сирих показників і служить опорою для першого розподілу впливу критеріїв без залучення складних експертних ваг.

Другий етап – калібрування отриманих первинних ваг за допомогою парних порівнянь. Це якісно-калібрувальна процедура, у якій критерії зіставляються попарно між собою. Попарні рішення базуються на логіці застосування, оглядах/бенчмарках та емпіричних спостереженнях. Їхня мета – уточнити первинну конфігурацію, врахувавши практичну релевантність різних показників. Результатом є відкоригований вектор ваг, який поєднує формальну стартову схему і аргументовані якісні уточнення.

Поєднання ваг і результатів виконується через послідовну агрегацію: спочатку показники комбінуються у внутрішні індикатори блоків з урахуванням їхніх фінальних ваг; далі блокові індикатори поєднуються відповідно до групових

часток, формуючи інтегральну оцінку для кожної технології. Саме ця інтегральна оцінка стає основою для ранжування у відповідних вузлах дерева.

Результатом є структура дерева, що поєднує прозорий метод присвоєння ваг з чітким алгоритмом агрегації нормалізованих результатів і зручною моделлю фільтрів/вузлів. Така архітектура гарантує, що рекомендації будуть одночасно емпірично обґрунтованими, інтерпретованими та готовими до подальшої валідації і адаптації під різні сценарії застосування.

3.8 Висновки проведення проєктування

У результаті проєктування сформовано методичну основу, яка охоплює відбір технічних і освітніх критеріїв, процедури їхнього збору та обробки, двошарову модель збереження експериментальних даних та методи до нормалізації й призначення ваг. Запропоновані процедури гарантують відтворюваність і прозорість отриманих вимірів і готують коректні входи для статичного алгоритму ранжування. Оскільки проєктувальна частина опрацьована в повному обсязі та враховує ключові методологічні вимоги, на її підставі можна впевнено переходити до розділу реалізації, де буде здійснено практичну імплементацію інструмента й виконано експериментальні прогони для отримання і калібрування реальних результатів.

4 РЕАЛІЗАЦІЯ ТА ДОСЛІДЖЕННЯ СИСТЕМИ

4.1 Реалізація скриптів випробувань та логіка збору технічних метрик

Окрему частину реалізації становлять власні скрипти і модулі випробувань. Кожен модуль реалізує окремий клас тестів. Загальна логіка їхньої роботи

однорідна: модуль приймає специфікацію експерименту, проводить серію фіксованих вимірів, збирає сирі часові та/або ресурсні показники, застосовує правила попередньої обробки (усічені середні або інші агрегатні функції) і повертає стандартизований JSON-об'єкт з метриками та метаданими (конфігурація, час запуску, ідентифікатори прогона). Така уніфікована структура тестових модулів забезпечує відтворюваність експериментів і дає готові до подальшої нормалізації вихідні дані для інтеграції в загальну процедуру оцінки стеків.

Перший із модулів – тест продуктивності коду – фокусує вимірювання на двох типах навантаження: математично-інтенсивних обчисленнях та алгоритмічних операціях над колекціями, а також на їхній сумарній часовій витраті. Реалізація включає коротку фазу «розігріву» (кілька попередніх викликів) і основну серію прогонів (за замовчуванням 200 вимірів). У кожному прогоні фіксується час виконання математичної частини і час виконання алгоритмічної частини, а їхня сума розглядається як загальний час прогона. Для зменшення впливу викидів застосовано процедуру усіченої (обрізаної) середньої: відкидаються по 5 % найменших і найбільших значень, після чого обчислюється середнє по залишку. Для малих вибірок використовується звичайне середнє. Модуль повертає стандартизований об'єкт з трьома ключовими метриками у мілісекундах: середній час математичного навантаження, середній час алгоритмічного навантаження і середній загальний час виконання коду.

Модуль оцінки пам'яті збирає показники, які дозволяють роздільно оцінити вплив реалізації на використання пам'яті масивами та об'єктною моделлю, а також їхню сумарну величину. Перед безпосереднім вимірюванням виконується кілька знімків стану купи з примусовим викликом збирача сміття та невеликою паузою для стабілізації. У циклі прогонів модуль послідовно ініціює створення великих масивів і наборів об'єктів, фіксує приріст використаної купи після кожної операції й перетворює його в мегабайти. Для усунення випадкових аномалій у вибірці застосовується метод усіченої середньої, при цьому для занадто малих вибірок модуль повертає визначену поведінку, що запобігає хибним оцінкам. На виході формується стандартизований об'єкт з трьома полями: середнє споживання пам'яті

масивами, середнє споживання пам'яті об'єктами та середнє загальне споживання пам'яті.

Модуль оцінки стабільності покликаній виявляти варіативність часових показників та поведінки пам'яті під навантаженням, зокрема потенційні витoki і тренди деградації. Реалізація виконує велику кількість прогонів, у кожному з яких фіксується час виконання та поточне використання купи (пам'ять, у якій JS розміщує динамічні структури даних). Одночасно ведеться облік позитивних приростів пам'яті як індикаторів можливих витоків. Після серії вимірів модуль обчислює сукупні статистики: середній час виконання, стандартне відхилення часу, середнє використання пам'яті та його відхилення, середній витік і тренд витoku та формує індекси стабільності часу та пам'яті, які служать узагальненими індикаторами детермінованості поведінки.

Модуль оцінки масштабованості імітує навантаження з різною кількістю паралельних запитів і вимірює реакцію системи при збільшенні числа одночасних з'єднань. У межах тесту формується внутрішня симуляція «запиту», після чого послідовно запускаються батчі (набір транзакцій, накопичений за певний період часу) паралельних задач на декількох рівнях навантаження. Для кожного рівня фіксується загальний час виконання всіх паралельних запитів, обчислюється середній час на один запит і коефіцієнт деградації порівняно з попереднім рівнем. Експеримент повторюється кілька разів, після чого повертаються усереднені значення трьох ключових показників: середній час обробки запиту, середній час на запит і середній коефіцієнт деградації.

Модуль оцінки асинхронної поведінки проводить пакетні прогони з різною кількістю асинхронних задач, фіксуючи час виконання кожного батча і середній час виконання однієї задачі. Тест виконує серії пакетів із набором рівнів навантаження і для кожного рівня вимірює загальний час пакета, час на одну задачу та коефіцієнт «насичення» (відношення часу поточного рівня до попереднього). Після повторення серії експериментів модуль повертає усереднені показники: середній час виконання однієї асинхронної задачі, середній коефіцієнт зміни часу при зростанні кількості задач і середній загальний час виконання пакетів.

Модуль мережевих характеристик імітує мережеві умови та вимірює їхній вплив на затримку й пропускну здатність реалізації: він генерує затримки з випадковим джиттером, емулює передачу даних для оцінки часу передачі та моделює можливі втрати пакетів. У кожному прогоні фіксуються значення затримки, спостережуваний джиттер, індикатор втрати пакета та розрахована пропускну здатність (KB/ms) за емульованого обсягу даних. Після серії прогонів, модуль повертає усереднені значення: середню затримку, середній джиттер, середню втрату пакетів і середню пропускну здатність. Отримані мережеві показники використовуються як коригувальні фактори при інтерпретації часових метрик у реальному розгортанні і дозволяють порівнювати стеки в умовах різних мережевих обмежень.

Ідея, що об'єднує всі модулі випробувань, полягає в уніфікованій і контролюваній імітації навантаження за допомогою множника. Кожен тестовий модуль приймає на вхід специфікацію стеку і зіставлені для неї множники, яких у коді агреговано в один робочий множник – усереднений коефіцієнт для поточного експерименту. Цей множник служить параметром масштабування: він змінює інтенсивність генерації навантаження всередині модулів – збільшує або зменшує число ітерацій і розміри структур у обчислювальних тестах, змінює кількість паралельних запитів у тестах масштабованості, коригує число одночасних асинхронних задач у пакетних прогонах, впливає на тривалості затримок і ймовірності втрат у мережевому модулі тощо. Завдяки такому підходу один і той самий набір сценаріїв може симулювати різні «вагові» профілі реалізацій – від «легких» до «важких» – без зміни базової логіки тестів.

Операційно, множники зберігаються у конфігураційному файлі і при запуску кожного тесту модулі отримують його як вхідний параметр. Це дозволяє фіксувати версії факторів, відтворювати прогони і документувати зміни. Використання множників дає кілька важливих переваг: воно забезпечує порівнянність результатів між стек-налаштуваннями, дозволяє здійснювати контрольовані експерименти для калібрування первинних ваг і провести аналіз чутливості (як сильно зміни ваг впливають на підсумкове ранжування). Після серій прогонів множники піддаються

валідації. Всі зміни фіксуються в метаданих експериментів, щоб зберегти відтворюваність і прозорість дослідження.

Нижче наведено обрані множники для кожної технології:

- SPA – множник 0.95;
- моноліт – множник 1.00;
- клієнт-сервер (класична) – множник 1.05;
- мікросервіси – множник 1.15;
- Serverless – множник 1.20;
- JavaScript – множник 1.00;
- TypeScript – множник 1.02;
- Python – множник 1.60;
- Go – множник 0.70;
- без фреймворку (чистий JS/TS) – множник 1.00;
- React – множник 1.08;
- Angular – множник 1.15;
- Vue – множник 1.04;
- Flask – множник 1.30;
- FastAPI – множник 0.95;
- Django – множник 1.20;
- Gin – множник 0.90;
- Fiber – множник 0.88;
- Echo – множник 0.92.

Множники виступають орієнтовними індикаторами відносних накладних витрат у синтетичних сценаріях: для архітектур моноліт обрано за базу 1.00 як еталон традиційної серверної обробки, SPA – 0.95 через перенесення рендерингу на клієнт, класична клієнт-серверна модель – 1.05 через додаткові мережеві цикли, мікросервіси – 1.15 через збільшення зовнішніх викликів і синхронних/асинхронних затрат, а Serverless – 1.20 через короткоживучі функції і холодні старту; для мов JavaScript взято 1.00 як орієнтир, TypeScript – 1.02 через інструментальні та організаційні додаткові витрати, Python – 1.60 у типовій WSGI-

моделі через інтерпретований рантайм і вищі «сирі» часові затрати, а Go – 0.70 як результат компільованості і ефективної моделі конкурентності; фреймворки оцінено як надбудова над мовою/архітектурою – варіант без фреймворку (чистий JS/TS) встановлено 1.00 як мінімум додаткових витрат, React – 1.08 через віртуальний DOM і керування станом, Vue – 1.04 як більш легкий варіант у простих сценаріях, Angular – 1.15 через більший набір внутрішніх механізмів, Flask – 1.30 і Django – 1.20 відображають синхронні та повнофункціональні оверхеда відповідно, FastAPI – 0.95 демонструє переваги асинхронного ASGI-підходу, а Go-двигуни Fiber (0.88), Gin (0.90) і Echo (0.92) показують нижчі множники завдяки легкій і швидкій обробці запитів [16, 17, 18, 23, 28, 29].

Результати експериментальних прогонів, виконаних з використанням наведених множників із-за свого великого розміру неможливо додати у дану пояснювальну записку, тому дані були збережені локально і застосовані при побудові дерева прийняття рішень.

4.2 Визначення освітніх метрик

Даний етап передбачає проведення дослідження з метою визначення середніх значень освітніх метрик технологій веброзробки. В даному аналізі не враховуються архітектури веброзробки, оскільки вибір визначається в залежності від його технічного завдання або цілей проєкту. Тому, метрики збираються для мов програмування та фреймворків.

Кандидатами для визначення метрик стали відомі та перевірені джерела, оскільки це є важливим критерієм не тільки для дослідження, а й для початківців.

Як було визначено на етапі підбору матеріалів для визначення метрик, освітні курси можуть охоплювати як окремо технології, так і повноцінний стек технологій, що може включати в себе як одну чи дві мови програмування, так і один чи два фреймворки. Через це неможливо оцінити якість курсів, та визначити ціну

конкретно для одної технології з набору. Тому, було прийнято рішення не використовувати дані джерела.

Для знаходження метрик до таких критеріїв як: складність освоєння, вартість та час на навчання було використано джерела Coursera та Udemu. Для визначення попиту на ринку праці було використано джерела work.ua, Dou (jobs/dou.ua) і LinkedIn (LinkedIn.com/jobs). Аби визначити ступінь розвиненості спільноти використовується Stack Overflow.

Для визначення складності освоєння був використаний такий алгоритм в два кроки:

а) визначити складність освоєння за дискретною шкалою:

- 1) початковий – 0.1;
- 2) середній (треба знати деякі технології або концепції) – 0.5;
- 3) просунутий (треба знати багато про різні технології та концепції,

або передбачено складність самого матеріалу) – 1;

б) відняти 0.05, якщо рейтинг даного курсу ≥ 4.8 .

Рішення, яке було прийняте при визначенні метрики (етап б), полягало в тому, щоб для курсів із рейтингом ≥ 4.8 застосовувати корекцію – це навмисне відображення висновку, що більшість курсів із рейтингами до 4.7 вважаються середніми за якістю, тоді як курси з рейтингом 4.8 і вище систематично відзначаються користувачами як найзрозуміліші й найефективніші для освоєння через структуру тестів і домашніх завдань, тож зниження величини метрики на 0.05 служить простим кількісним відображенням цієї підвищеної доступності освоєння.

Для визначення вартості навчання було узятю запропоновану ціну на підписку (Coursera) або виділену окремо на сам курс (Udemu). Час на навчання був порахований як сума годин, які потрібні на огляд матеріалу. Оскільки неможливо визначити час на освоєння матеріалу на виконання домашніх завдань людиною, цей час опускається. Наведені критерії додаються до таблиці відповідної технології як середнє значення за 10 подібних курсів.

Попит на ринку праці визначається кількістю вакансій, в яких наведено відповідну технологію в списку необхідних навичок від кандидата за місяць.

Розвиненість спільноти визначається як кількість статей обговорень та допомоги на відповідну технологію у відсотках. Доступність документації визначається як число від 0 до 2, де 0 – є неперекладена офіційна документація, 1 – офіційна документація є перекладеною та/або перефразованою, 2 – офіційна документація перекладена та/або перефразована зрозуміліше (достатнє для будь-якого читача) та містить приклади (інтерактивні чи статичні).

Актуальність визначається як зважена сума нормалізованого ринкового попиту та розвиненості спільноти і визначається за формулою (4.1):

$$A = \alpha \times V_{norm} + (1 - \alpha) \times k_{community}, \quad (4.1)$$

де α – це коефіцієнт пріоритету попиту працедавців над розвиненістю спільноти при виборі стеку для навчання чи застосування в проєкті і дорівнює 0.8, оскільки популярність на ринку праці і характеризує кількість обговорені і статей, $k_{community}$ – коефіцієнт розвиненості спільноти, V_{norm} це min-max-нормалізована середня кількість вакансій, яка знаходиться за формулою (4.2):

$$V_{norm} = \frac{avrVac - \min(avrVac)}{\max(avrVac) - \min(avrVac)}, \quad (4.2)$$

де $avrVac$ – це середня кількість вакансій для технології, що підібрані за допомогою визначених джерел. $\min(avrVac)$ і $\max(avrVac)$ є значеннями мінімальної та максимальної середньої кількості вакансій серед визначених технологій відповідно.

Під час збору метрик, який відбувався п'ятого грудня 2025 року, було отримано кількість питань у Stack Overflow для кожної технології, які були перетворено у діаграму для визначення відповідних відсотків. Дана діаграма наведена на рисунку 4.1.

Таблиця 4.5 – Освітні метрики для фреймворку React

Критерій/ Платформа	Coursera	Udemy	Work.ua	Dou	LinkedIn	Stack Overflow	Документація	Актуальність
Складність освоєння	0.33	0.43	-	-	-	-	-	-
Вартість навчання (\$)	43.75	74	-	-	-	-	-	-
Час на навчання (годин)	38	42.25	-	-	-	-	-	-
Попит на ринку праці (за місяць)	-	-	25	158	89	-	-	-
Розвиненість спільноти (%)	-	-	-	-	-	7.549	-	-
Доступність документації	-	-	-	-	-	-	2	-
Актуальність	-	-	-	-	-	-	-	0.4101

Таблиця 4.6 – Освітні метрики для фреймворку Angular

Критерій/ Платформа	Coursera	Udemy	Work.ua	Dou	LinkedIn	Stack Overflow	Документація	Актуальність
Складність освоєння	0.29	0.49	-	-	-	-	-	-
Вартість навчання (\$)	48	70	-	-	-	-	-	-
Час на навчання (годин)	23	20.4	-	-	-	-	-	-
Попит на ринку праці (за місяць)	-	-	7	32	68	-	-	-
Розвиненість спільноти (%)	-	-	-	-	-	4.856	-	-
Доступність документації	-	-	-	-	-	-	2	-
Актуальність	-	-	-	-	-	-	-	0.3857

Таблиця 4.7 – Освітні метрики для фреймворку Vue

Критерій/ Платформа	Coursera	Udemy	Work.ua	Dou	LinkedIn	Stack Overflow	Документація	Актуальність
Складність освоєння	0.1	0.49	-	-	-	-	-	-
Вартість навчання (\$)	35	62.5	-	-	-	-	-	-
Час на навчання (годин)	16	12.45	-	-	-	-	-	-
Попит на ринку праці (за місяць)	-	-	9	46	53	-	-	-
Розвиненість спільноти (%)	-	-	-	-	-	1.711	-	-
Доступність документації	-	-	-	-	-	-	2	-
Актуальність	-	-	-	-	-	-	-	0.4943

Усі зібрані метрики об'єднані в підсумкову таблицю. Дана таблиця з остаточними значеннями для всіх технологій наведена у таблиці 4.14.

Таблиця 4.14 – Усереднені освітні критерії технологій веброзробки

Назва критерію/Назва технології	JavaScript	Typescript	Python	Go	React	Angular	Vue	Flask	FastAPI	Django	Gin	Fiber	Echo
Складність	0.16	0.43	0.072	0.4	0.38	0.39	0.295	0.365	0.39	0.48	0.38	0.5	0.48
Вартість	72	68.3	70.35	46	58.9	59	48.75	49.5	38	55.15	42.14	20	27.5
Час на	35.3	13.7	44.65	18.55	40.12	21.7	14.25	15.12	12.55	25.5	8.35	5.25	4.4
Попит на	60.3	34	143	13	90.6	35.6	36	22	2	22	33	0	0
Розвиненіс	39.9	3.749	34.96	1.179	7.549	4.856	1.711	0.873	0.117	4.931	0.001	0.005	0.0015
Доступніс	2	2	2	2	2	2	2	2	2	2	1	1	1
Актуальніс	0.46	0.387	0.424	0.286	0.410	0.385	0.494	0.272	0.533	0.295	0.266	0.0000	0.00018

Дані зібрані освітні метрики готові для внесення вкладу у розробку дерева прийняття рішень.

4.3 Реалізація методів присвоєння ваг

У цій частині використовуємо двоетапну методику: на першому етапі методом «мін-макс» виставляються первинні (базові) ваги у прозорій і відтворюваній схемі, а на другому – на підставі перших експериментальних спостережень і логіки застосування виконується якісне калібрування шляхом порівняння критеріїв між собою. Така послідовність дозволяє поєднати простоту стартової конфігурації й можливість її обґрунтованого коригування після отримання реальних даних.

Для практичної реалізації обрано баланс 0,60 / 0,40 на користь технічного блоку критеріїв. Це рішення впливає із завдань проєкту: при оцінці стеку в умовах розробки і розгортання продуктивність, стабільність і експлуатаційні характеристики найчастіше визначають придатність технології для реальних систем. Водночас освітні критерії – попит, документація, доступність курсів – мають істотне значення при виборі стеку студентами та початківцями, тому їм залишено значну частку ваги, щоб рекомендація враховувала і кар'єрні перспективи, і простоту входу.

На першому етапі метод «мін-макс» застосовується спочатку для приведення сирих показників до спільної шкали перед обчисленням оцінок, а потім у сенсі початкового розподілу ваг. Всередині блоків критеріїв згруповано за трьома рівнями важливості: висока, середня і низька. Групові частки технічного блоку розподілено так, щоб ключові експлуатаційні метрики отримали найпомітніший вплив (приблизно половина технічної ваги), проміжні характеристики – значущу, але меншу частку, а вузькоорієнтовані показники асинхронності та мережі – найменшу. Усередині кожної групи первинні ваги розподілено рівномірно – це робоче правило, яке забезпечує простоту відтворення і дає чітку базу для подальшого калібрування. Результатом цього етапу наведено в таблиці 4.15.

Таблиця 4.15 – Результати визначення початкових вагів коефіцієнтів методом «мін-макс»

Назви критеріїв	Важливість	Вага
Технічні критерії		
Середній загальний час виконання коду, середній час виконання, індекс стабільності часу, індекс стабільності пам'яті, середнє загальне споживання пам'яті, середній витік пам'яті	Висока	0.05
Середній час математичного навантаження, середній час алгоритмічного навантаження, середнє, споживання пам'яті масивами, середнє споживання пам'яті об'єктами, відхилення часу, середнє використання пам'яті, відхилення пам'яті, тренд витоку пам'яті, середній час обробки запиту, середній час на запит	Середня	0.018
Середній коефіцієнт деградації, середній час виконання 1 асинхронної задачі, середній коефіцієнт зміни часу при зростанні кількості задач, середній загальний час виконання пакетів задач, середня затримка, середній джиттер, середня втрата пакетів, середня пропускна здатність	Низька	0.015
Освітні критерії		
Попит на ринку праці, актуальність	Висока	0.1
Розвиненість спільноти, доступність документації	Середня	0.06
Складність освоєння, вартість навчання, час на навчання	Низька	0.027

Показники загальної продуктивності й стабільності отримали пріоритет, оскільки саме вони найпряміше визначають придатність технології для довготривалої експлуатації. Швидкість виконання та детермінованість часу впливають не лише на користувацький досвід, а й на розміри необхідної інфраструктури й операційні витрати на підтримку сервісу. Тому, метрики типу середнього загального часу виконання, середнього часу виконання та індексів стабільності виділено як ключові. Їхні значення формують як миттєві показники

якості обслуговування, так і накопичувальні наслідки (наприклад, необхідність частих перезапусків або масштабування), що робить ці метрики критичними при виборі стеку для реальних систем.

Показники, що описують витрати пам'яті – як сумарні, так і пов'язані з витокami – мають подібний ранг важливості через ризик накопичувальних дефектів, які згодом призводять до деградації сервісу. Саме тому сумарне споживання пам'яті, поведінка масивів і об'єктів, а також індикатори витоків і тренди отримали підвищену увагу: вони інформують про довгострокові операційні ризики і дозволяють прогнозувати витрати на підтримку та апаратну інфраструктуру. Параметри, що описують алгоритмічні витрати і деталі розподілу пам'яті, отримали середню пріоритетність – вони важливі для розуміння масштабованості та оптимізації, але самі по собі, за відсутності проблем на рівні загальної повільності або нестабільності, зазвичай не є вирішальними.

Група показників з низькою початковою вагою обґрунтована їхньою прикладною спрямованістю: метрики асинхронності (середній час однієї асинхронної задачі, коефіцієнти зміни часу при зростанні числа задач, час пакетної обробки) та мережеві характеристики (затримка, джиттер, втрата пакетів, пропускна здатність) мають високу значущість у спеціалізованих розподілених або реального часу системах, але не є універсальними для широкого набору типових вебпроектів. Через це на старті їм відведено меншу питому вагу, але їхню роль передбачено можливість підвищити під час калібрування для конкретних класів задач.

У освітньому блоці пріоритет віддано ринковим сигналам: попит на ринку праці та актуальність технології найкраще відображають практичну цінність для кар'єрних перспектив і швидкої адаптації студентів. Розвиненість спільноти й якість документації сприяють прискореному навчанню, полегшують вирішення практичних завдань і підвищують автономність розробника, тому їм надано середню пріоритетність. Бар'єри входу – складність опанування, вартість і тривалість навчання – важливі, але зазвичай другорядні при остаточному виборі стека, особливо якщо ринкові перспективи є визначальними.

Після отримання початкових ваг постає мета в їх калібруванні методом порівняння. Для цього використовується порівняння між собою критеріїв серед групи за важливістю які були визначені на попередньому етапі. Таким чином визначається чи критерій А важливіший за В, рівноцінний або менш важливий. З залежності визначеного рішення критерії отримують бали за простим способом: переможець отримує 1 бал, якщо нічия – 0.5, якщо програвший – отримує 0. Для кожного критерію підсумовується бали. Отримані бали будуть поділені на кількість порівнянь з метою нормалізації всередині групи. Результати будуть помножені на загальну суму ваг критеріїв всередині групи і це буде нова відкалібрована вага.

У межах групи технічних критеріїв високої важливості виділено такі показники: середній загальний час виконання коду (А), середній час виконання окремих операцій (В), індекс стабільності часу (С), індекс стабільності пам'яті (D), середнє загальне споживання пам'яті (Е) та середній витік пам'яті (F). Проведений аналіз попарних порівнянь дозволяє сформулювати пріоритети між цими показниками з поясненням причин кожного рішення.

При зіставленні сумарного часу виконання (А) і середнього часу окремих операцій (В) встановлено перевагу А над В ($A > B$): агрегована латентність запиту є більш релевантним показником для кінцевого користувача та інфраструктурних витрат, ніж окремі деталізовані частини часу виконання. Порівняння А і середнього загального споживання пам'яті (С) показало перевагу стабільності часу (С) над середнім часом ($C > A$), оскільки передбачуваність часу (низька варіативність і відсутність піків) безпосередньо впливає на SLA та кількість інцидентів, незначні програші в середніх значеннях можуть бути допустимими за умови низької варіативності. У співвідношенні А і індексу стабільності пам'яті (D) перевагу віддано А ($A > D$): у типовому CRUD/веб-сценарії повільна відповідь для користувача має більш помітний і прямий бізнес-вплив, тоді як коливання пам'яті частіше компенсуються інфраструктурно. Аналогічно у порівнянні А і середнього загального споживання пам'яті (Е) перевага на боці часу ($A > E$), оскільки стабільно більший об'єм пам'яті зазвичай можна вирішити масштабуванням, тоді як повільний відгук сервісу безпосередньо погіршує користувацький досвід.

Водночас при порівнянні A і середніх витоків пам'яті (F) виявлено F ($F > A$): системні витки – накопичувальна проблема, що призводить до деградації і простоїв, і навіть краща початкова латентність не компенсує непридатності стеку в довгостроковій експлуатації.

Для показників, пов'язаних із середнім часом окремих операцій (B), проведені додаткові порівняння підтверджують пріоритет стабільності часу (C) над B ($C > B$) – відсутність флуктуацій надає виробничу передбачуваність, яка важливіша за точкову оптимізацію окремих підфункцій. У зіставленні B і індексу стабільності пам'яті (D) пріоритет віддано B ($B > D$): повільне виконання базових операцій швидше позначається на якості сервісу, тоді як флуктуації пам'яті частіше виявляються пізніше і можуть бути виявлені моніторингом. При порівнянні B і середнього загального споживання пам'яті E також віддана перевага B ($B > E$): для первісної діагностики та вибору технології важливіші тимчасові вузькі місця, що прямо впливають на реактивність системи, а не абсолютний об'єм пам'яті. Однак у співвідношенні B і середнього витоку пам'яті (F) переважає B ($F > B$) через ризик ескалації повторюваних витоків у серйозні інциденти та втрату працездатності системи.

Порівняння індексу стабільності часу (C) з іншими показниками показало загальну його перевагу над індексом стабільності пам'яті (D) ($C > D$) і над середнім споживанням пам'яті (E) ($C > E$): нестабільність часу відповіді частіше призводить до проблем зі SLA і до явних звернень користувачів, тоді як коливання пам'яті або високий, але прогнозований об'єм пам'яті мають більш відстрочений або керований характер прояву. У співвідношенні C і середнього витоку пам'яті (F) за загальною практикою стабільність часу має невелику перевагу через негайний вплив на UX ($C > F$); однак за наявності або зафіксованих сильних витоків пам'яті пріоритет може змінитися на користь F – тобто остаточне рішення залежить від контексту експлуатації та інтенсивності витоків.

Що стосується порівняння внутрішніх характеристик пам'яті, встановлено, що індекс нестабільності пам'яті (D) має перевагу над стабільно великим, але прогнозованим споживанням пам'яті (E) ($D > E$), оскільки флуктуації створюють

більше оперативної невизначеності і складнощів в управлінні. Водночас систематичні витоки пам'яті (F) мають перевагу над індексом нестабільності пам'яті ($F > D$) і над стабільним споживанням пам'яті ($F < E$): витоки є прямим шляхом до накопичувальної деградації та потенційних простоїв, що робить їх критичним фактором при довгостроковій оцінці придатності стеку [16, 21, 22, 23].

Провівши дане порівняння, підраховано таку кількість балів:

- середній загальний час виконання коду – 3;
- середній час виконання – 2;
- індекс стабільності часу – 5;
- індекс стабільності пам'яті – 1;
- середнє загальне споживання пам'яті – 0;
- середній витік пам'яті – 4.

Нормалізуємо ваги всередині групи:

- середній загальний час виконання коду: $3 \div 15 \approx 0.2$;
- середній час виконання: $2 \div 15 \approx 0.13$;
- індекс стабільності часу: $5 \div 15 \approx 0.333$;
- індекс стабільності пам'яті: $1 \div 15 \approx 0.06$;
- середнє загальне споживання пам'яті: $0 \div 15 = 0$;
- середній витік пам'яті: $4 \div 15 \approx 0.26$.

Визначення відкаліброваних вагів:

- середній загальний час виконання коду: $0.2 \times 0.30 = 0.06$;
- середній час виконання: $0.13 \times 0.30 = 0.04$;
- індекс стабільності часу: $0.333 \times 0.30 = 0.1$;
- індекс стабільності пам'яті: $0.06 \times 0.30 = 0.02$;
- середнє загальне споживання пам'яті: $0 \times 0.30 = 0$;
- середній витік пам'яті: $0.26 \times 0.30 = 0.08$.

Оскільки при визначенні нових ваг один критерій отримав 0 балів, а отже треба додати мінімальну (абсолютну) вагу, яка має бути масштабована пропорційно, щоб сума ваг залишалась незмінною. Конкретно для цієї групи даною

вагою буде значення 0.015 (половина від ваги критерію з одним балом), а це означає що, коефіцієнт масштабування дорівнюватиме 0.95 $((0.30 - 0.015) \div 0.30)$.

Згідно цього, здійснено перекалібрування ваг:

- середній загальний час виконання коду: $0.06 \times 0.95 = 0.057$;
- середній час виконання: $0.04 \times 0.95 = 0.038$;
- індекс стабільності часу: $0.1 \times 0.95 = 0.095$;
- індекс стабільності пам'яті: $0.02 \times 0.95 = 0.019$;
- середнє загальне споживання пам'яті: 0.015;
- середній витік пам'яті: $0.08 \times 0.95 = 0.076$.

Перевіримо загальну суму критеріїв:

$$\begin{aligned} \text{Сума критеріїв} &= 0.057 + 0.038 + 0.095 + 0.019 + \\ &+ 0.015 + 0.076 = 0.30. \end{aligned}$$

У групі технічних показників середньої важливості виділені такі метрики: середній час математичного навантаження (А), середній час алгоритмічного навантаження (В), середнє споживання пам'яті масивами (С), середнє споживання пам'яті об'єктами (D), відхилення часу (Е), середнє використання пам'яті (F), відхилення пам'яті (G), тренд витоку пам'яті (H), середній час обробки запиту (I) та середній час на запит (J). Нижче наведено результати попарних зіставлень і аргументацію для кожного випадку.

У зіставленнях, де приймається до уваги співвідношення середнього часу математичного навантаження (А) з іншими показниками, отримано такі висновки: при порівнянні А і середнім часом алгоритмічного навантаження (В) перевага віддана В ($B > A$), оскільки алгоритмічні витрати визначають порядок зростання часу при масштабуванні і суттєвіше впливають на поведінку системи під великим навантаженням; у співвідношенні А і середнього споживання пам'яті масивами (С) пріоритет віддано А ($A > C$), бо час виконання математичних блоків додає затримку, тоді як споживання пам'яті масивами часто можна компенсувати ресурсами; аналогічна логіка застосовується при порівнянні А і середнього

споживання пам'яті об'єктами (D) – A має перевагу над D ($A > D$), оскільки вплив на час відповіді важливіший на ранніх етапах оцінки; у випадку A і відхиленням часу (E) пріоритет на боці варіативності часу ($E > A$), бо велика варіативність породжує інциденти і ускладнює виконання SLA; при зіставленні A і середнього використання пам'яті (F) час переважає над середнім споживанням пам'яті ($A > F$), оскільки користувачі відчують відгук негайно, а питання пам'яті частіше вирішується інфраструктурно; при співставленні A і відхиленням пам'яті (G) пріоритет також на боці часу ($A > G$), оскільки коливання часу помітніші для користувача, ніж статистичні відхилення пам'яті; водночас тренд витоків пам'яті (H) переважає над часом математичного навантаження ($H > A$), оскільки виток є накопичувальною загрозою, яка може призвести до деградації та простоїв; у зіставленнях з агрегованими метриками встановлено, що середній час обробки запиту (I) має перевагу над матчастиною ($I > A$), бо I інтегрує вплив численних підкомпонентів, і аналогічно середній час на запит (J) переважає над A ($J > A$), оскільки J агрегує багато складових і краще відображає кінцеву латентність.

У порівняннях, де в центрі – алгоритмічні витрати (B), встановлено такі пріоритети: B переважає над споживанням пам'яті масивами (C) ($B > C$) і над споживанням пам'яті об'єктами (D) ($B > D$), оскільки алгоритміка визначає масштабований характер роботи і сильніше впливає на продуктивність при великих навантаженнях; за умов обмежених SLA нестабільність часу (E) має перевагу над B ($E > B$), оскільки проблеми варіативності часу частіше дають критичні наслідки, ніж оптимізація одного алгоритмічного компонента; B переважає над середнім використанням пам'яті (F) ($B > F$), бо алгоритмічні вузькі місця безпосередньо впливають на затримку і пропускну здатність; B також має перевагу над відхиленням пам'яті ($B > G$), оскільки відхилення пам'яті проявляються повільніше; проте при наявності стійких витоків пам'яті (H) їхній ризик іноді перевищує виграш від алгоритміки ($H > B$); зрештою, комплексні агреговані показники переважають B: середній час обробки запиту (I) ($I > B$) і середній час на запит (J) ($J > B$) є більш комплексними індикаторами продуктивності та кращими для оцінки кінцевої поведінки системи.

У зоні порівнянь локальних пам'ятевих метрик зафіксовано наступне: споживання пам'яті масивами (C) і об'єктами (D) часто корельовані, тому для загального ранжування їх можна вважати еквівалентними ($C = D$); однак ненадійна латентність (E) має більший негайний вплив на SLA і UX ніж локальне споживання пам'яті масивами ($E > C$); загальне середнє використання пам'яті (F) має перевагу над C ($F > C$), бо краще відображає навантаження на пам'ять програми; нестабільність пам'яті (G) більш тривожна, ніж C ($G > C$), оскільки вона створює більше невизначеності та ризиків; тренд витоку (H) акумулюється й має вищі довгострокові ризики за локальне споживання масивами ($H > C$); крім того, агреговані показники часу переважають C – як час обробки запиту (I) ($I > C$), так і середній час на запит (J) ($J > C$), оскільки вони краще відображають кінцеву поведінку системи.

У зіставленнях, що стосуються споживання пам'яті об'єктами (D), зафіксовано: варіативність часу (E) створює негайні SLA-проблеми частіше, ніж абсолютне споживання пам'яті для об'єктів ($E > D$); загальне середнє використання пам'яті (F) краще відображає стан системи і тому переважає D ($F > D$); відхилення пам'яті (G) означають нестабільність, якої важче компенсувати, ніж передбачуване споживання ($G > D$); тренд витоку (H) має довгостроковий негативний ефект, що переважає локальні споживання пам'яті ($H > D$); агреговані показники часу також важливіші за D: час обробки ($I > D$) і час на запит ($J > D$) мають пряміший вплив на UX та пропускну здатність.

Щодо варіативності та агрегованих часових індикаторів, встановлено: велике відхилення часу (E) швидше створює оперативні проблеми та незадоволення користувачів ніж середні показники пам'яті (F) ($E > F$); нестабільність часу зазвичай має більш очевидні наслідки для сервісу ніж відхилення пам'яті ($E > G$), хоча обидві метрики важливі; у випадку зіставлення відхилення часу й тренду витоку логічним є визнати їх рівнозначними за потенційним негативним впливом ($E = H$) – відхилення часу дає негайний негатив для UX/SLA, тоді як тренд витоку є накопичувальною серйозною загрозою; велика варіативність часу також має перевагу над середнім часом обробки (I) ($E > I$) і над середнім часом на запит (J) (E

> J), оскільки флуктуації породжують невідтворювані інциденти, тоді як середні величини є більш прогнозованими.

У відношенні метрик, пов'язаних із середнім використанням пам'яті (F), визнано: відхилення пам'яті (G) більш тривожні за саме середнє споживання ($G > F$), оскільки нестабільність загрожує непередбачуваними підвищеннями споживання; тренд витоку пам'яті (H) переважає над F ($H > F$), оскільки витoki створюють накопичувальну загрозу; середній час обробки запиту (I) має пріоритет над F ($I > F$) через безпосередній вплив на затримку та UX; аналогічно, середній час на запит (J) переважає над F ($J > F$), будучи більш прикладною інтегрованою метрикою відгуку.

У порівняннях, що стосуються відхилень пам'яті (G) та тренду витоку (H), виявлено: тренд витоку є системною наростаючою проблемою і має більший потенційний негативний ефект ніж випадкові флуктуації пам'яті ($H > G$); одночасно агреговані часові метрики мають перевагу над G – середній час обробки запиту (I) ($I > G$) і середній час на запит (J) ($J > G$) краще відображають UX і пропускну здатність; у порівнянні H з часовими метриками встановлено, що тренд витоку часто має більшу значущість для довгострокової надійності ніж помірні відмінності в обробці ($H > I$) і може переважати над середнім часом на запит в ризик-орієнтованому ранжуванні ($H > J$), оскільки витoki, що акумулюються, здатні спричинити відключення або рестарти.

Насамкінець, у співставленні агрегованих часових індикаторів встановлено, що середній час на запит (I) має дещо вищу значущість, ніж середній час обробки запиту (J) ($J > I$), оскільки J охоплює ширший контекст і несе інформацію про сумарний відгук системи, що робить його трохи важливішим для прийняття остаточних рішень щодо вибору стеку [16, 21, 22, 23, 24].

Провівши дане порівняння, підраховано таку кількість балів:

- середній час математичного навантаження – 4;
- середній час алгоритмічного навантаження – 5;
- середнє споживання пам'яті масивами – 0.5;
- середнє споживання пам'яті об'єктами – 0.5;

- відхилення часу – 8.5;
- середнє використання пам'яті – 2;
- відхилення пам'яті – 3;
- тренд витоку пам'яті – 8.5;
- середній час обробки запиту – 6;
- середній час на запит – 7.

Згідно отриманих балів виконано калібрування вагів:

- середній час математичного навантаження: 0.016;
- середній час алгоритмічного навантаження: 0.02;
- середнє споживання пам'яті масивами: 0.002;
- середнє споживання пам'яті об'єктами: 0.002;
- відхилення часу: 0.034;
- середнє використання пам'яті: 0.008;
- відхилення пам'яті: 0.012;
- тренд витоку пам'яті: 0.034;
- середній час обробки запиту: 0.024;
- середній час на запит: 0.028.

Перевіримо загальну суму критеріїв:

$$\begin{aligned} \text{Сума критеріїв} &= 0.016 + 0.02 + 0.002 + 0.002 + 0.034 + 0.008 + \\ &+ 0.012 + 0.034 + 0.024 + 0.028 = 0.18 \end{aligned}$$

У межах останньої групи технічних показників визначено наступні метрики: середній коефіцієнт деградації (A), середній час виконання однієї асинхронної задачі (B), середній коефіцієнт зміни часу при зростанні кількості задач (C), середній загальний час виконання пакетів задач (D), середня мережна затримка (E), середній джиттер (F), середня втрата пакетів (G) та середня пропускна здатність (H). Нижче наведено результати попарних зіставлень і коротку аргументацію для кожного порівняння.

Порівняння з участю коефіцієнта деградації (A) показали: A переважає над одиничним часом асинхронної задачі (B) ($A > B$), оскільки деградація відображає загальне погіршення продуктивності при зростанні навантаження, тоді як B важливий тільки локально; середній коефіцієнт зміни часу при зростанні кількості задач (C) має більшу практичну цінність для прогнозування масштабованості і тому переважає над A ($C > A$); середній загальний час виконання пакетів задач (D) більш релевантний для батчевих сценаріїв і віддається перевага над A ($D > A$); у порівнянні A і середньої мережевої затримки (E) критерій деградації вважається критичнішим для довгострокової стійкості платформи ($A > E$); A також має перевагу над середнім джиттером (F) ($A > F$), бо джиттер важливий переважно для реального-часових потоків, тоді як системна деградація має ширший негативний ефект; A переважає над середньою втратою пакетів (G) у типових HTTP/REST сценаріях ($A > G$), оскільки втрати пакетів зустрічаються рідше, а накопичувальна деградація загрожує довготерміновій експлуатації; водночас середня пропускна здатність (H) у багатьох контекстах більш визначальна для обслуговування навантаження і має невелику перевагу над A ($H > A$).

У зіставленнях, де ключовим є одиничний час асинхронної задачі (B), встановлено: чутливість до навантаження (C) важливіша за B при ухваленні рішень для навантажених сценаріїв ($C > B$); агрегований час пакета (D) краще відображає продуктивність у батчевих/паралельних сценаріях і має перевагу над B ($D > B$); мережева затримка формує кінцеву латентність і часто важливіша за локальний час задачі (E) ($E > B$); водночас одиничний час асинхронної задачі важливіший за середній джиттер (F) у типових веб-сценаріях ($B > F$) та за поодинокі випадки втрат пакетів ($B > G$); проте пропускна здатність визначає масштаб та має перевагу над B в системах з високим потоком даних ($H > B$).

Аналіз метрик, пов'язаних із середнім коефіцієнтом зміни часу при зростанні кількості задач (C), показав послідовну перевагу цієї метрики у контексті масштабованості: C переважає над середнім загальним часом виконання пакетів задач (D) ($C > D$), оскільки дає загальніший сигнал про деградацію під навантаженням; C також має пріоритет над середньою мережевою затримкою (E)

($C > E$), над середнім джиттером (F) ($C > F$) та над втратою середньою втратою пакетів (G) ($C > G$), оскільки чутливість до навантаження є ключовим чинником при виборі архітектури для потенційно високонавантажених систем; крім того, для архітектурних рішень поведінка при навантаженні важливіша за сиру пропускну здатність (H) ($C > H$).

Детальніше про оцінку середнього загального часу виконання пакетів (D): у батчевих сценаріях D є більш комплексною і релевантною метрикою, тому D переважає над середньою мережевою затримкою (E) ($D > E$), над середнім джиттером (F) ($D > F$) та над середньою втратою пакетів (G) ($D > G$), оскільки локальні обчислення і внутрішня взаємодія компонентів визначають ефективність пакетної обробки; у цьому ж контексті D вважається більш релевантною для оцінки батчевих робіт ніж пропускну здатність (H) ($D > H$), хоча в загальних питаннях масштабованості H також має велике значення.

Порівняння, що стосуються мережевих якостей та їхньої ролі у відгуку системи, дали такі висновки: мережева затримка (E) загалом має більш помітний вплив на UX, ніж середній джиттер (F) у типовій веб-моделі ($E > F$), і важливіша за періодичну втрату пакетів (G) у більшості HTTP API ($E > G$); для індивідуальної реакції користувача затримка часто є важливішою за сиру пропускну здатність (H) ($E > H$), хоча остаточний вибір між E і H залежить від конкретного сценарію застосування.

Щодо взаємозв'язку джиттера (F), втрати пакетів (G) і пропускну здатності (H): втрата пакетів може призводити до повторних передач і помилок, бути критичнішою для надійності ніж невеликий середній джиттер (F) у нетривких HTTP-запитах ($G > F$); пропускну здатність у більшості сценаріїв є фундаментальною для масштабування і має перевагу над F ($H > F$); у системах з великими обсягами трафіку сира пропускну здатність визначає можливості системи більше, ніж рідкісні випадки втрат, особливо за наявності механізмів повторної передачі/корекції ($H > G$) [23, 29, 30, 31].

Провівши дане порівняння, підраховано таку кількість балів:

- середній коефіцієнт деградації – 4;

- середній час виконання 1 асинхронної задачі – 2;
- середній коефіцієнт зміни часу при зростанні кількості задач – 7;
- середній загальний час виконання пакетів задач – 6;
- середня затримка – 4;
- середній джиттер – 0;
- середня втрата пакетів – 1;
- середня пропускна здатність – 4;

Згідно отриманих балів виконано калібрування вагів, включаючи абсолютну вагу 0.00215:

- середній коефіцієнт деградації : 0.016856;
- середній час виконання 1 асинхронної задачі: 0.008428;
- середній коефіцієнт зміни часу при зростанні кількості задач: 0.029498;
- середній загальний час виконання пакетів задач: 0.025284;
- середня затримка: 0.016856;
- середній джиттер: 0.00215;
- середня втрата пакетів: 0.004214;
- середня пропускна здатність: 0.016856;

Перевіримо загальну суму критеріїв:

$$\begin{aligned} \text{Сума критеріїв} = & 0.016856 + 0.008428 + 0.029498 + 0.025284 + 0.016856 + \\ & + 0.00215 + 0.004214 + 0.016856 \approx 0.12 \end{aligned}$$

Для групи високої важливості серед освітніх критеріїв, в яку входять два критерії: попит на ринку праці, актуальність, були проведено одне порівняння, в якому переможцем є попит на ринку праці, оскільки при прийнятті рішень про освоєння технології чи вибір стеку роботодавці й кандидати орієнтуються передусім на наявні робочі місця – це прямий економічний сигнал про затребуваність навичок. Актуальність, у нашому випадку, є похідною величиною (комбінується з показниками спільноти та попиту) і служить додатковим

контекстом як спроба прогнозу, але сама по собі менш безпосередньо впливає на можливість працевлаштування.

Отже, скореговані ваги такі:

- попит на ринку праці – 0.125;
- актуальність – 0.075;

При таких вагах перевіримо суму:

$$\text{Сума критеріїв} = 0.125 + 0.075 = 0.2$$

Для групи середньої важливості серед освітніх критеріїв, в яку входять два критерії: розвиненість спільноти, доступність документації, були проведено одне порівняння, в якому визначено, що обидва критерії є однаковими за важливістю. Якщо документація написана доступною мовою і не викликатиме проблем з опанування мови на базовому рівні, то і питати про допомогу на платформах зі спільнотою не потрібно, але якщо потребуватиме більш докладна інформація щодо складніших для розуміння чи проєктування речей, то досвід спільноти програмістів стане у нагоді. Враховуючи це, то ваги залишаються якими і були (по 0.06 кожний).

У межах групи освітніх критеріїв низької важливості виділено наступні показники: складність освоєння (А), вартість навчання (В) та час на навчання (С). Проведене попарне зіставлення цих показників дозволяє визначити їхній відносний пріоритет з урахуванням впливу на доступність та ефективність навчального процесу для потенційних слухачів.

Аналіз показав, що складність освоєння (А) має перевагу над вартістю (В) ($A > B$): висока концептуальна складність матеріалу створює суттєвий бар'єр входу, який не компенсується навіть наявністю безкоштовних курсів; отже, саме складність визначає можливість і швидкість опанування технології. Також складність має перевагу над часом на навчання (С) ($A > C$): тривалість курсу можна частково компенсувати організаційно або шляхом розбиття програми на модулі, тоді як висока складність робить процес навчання повільним і менш передбачуваним. У підсумку в класичному зіставленні вартості та тривалості

курсів вартість переважає над часом ($B > C$): відсутність фінансової доступності може взагалі заблокувати початок навчання, тоді як часові витрати для багатьох кандидатів є відносно гнучким ресурсом.

Визначено таку суму балів:

- складність освоєння – 2;
- вартість навчання – 1;
- час на навчання – 0.

Згідно отриманих балів виконано калібрування вагів, включаючи абсолютну вагу 0.0135:

- складність освоєння: 0.045;
- вартість навчання: 0.0225;
- час на навчання: 0.0135.

Перевіримо загальну суму критеріїв:

$$\text{Сума критеріїв} = 0.045 + 0.0225 + 0.0135 = 0.81$$

В результаті проведення даних етапів отримання вагових коефіцієнтів отримано таблицю ваг (Таблиця 4.16), що пройшла коригування порівняльним методом. Дана таблиця наведена нижче.

Таблиця 4.16 – Скореговані ваги методом порівняння

Назва критерію	Вага
Середній загальний час виконання коду	0.057
Середній час виконання	0.038
Індекс стабільності часу	0.095
Індекс стабільності пам'яті	0.019
Середнє загальне споживання пам'яті	0.015
Середній витік пам'яті	0.076
Середній час математичного навантаження	0.016
Середній час алгоритмічного навантаження	0.02
Середнє споживання пам'яті масивами	0.002
Середнє споживання пам'яті об'єктами	0.002
Відхилення часу	0.034

Продовження таблиці 4.16

Назва критерію	Вага
Середнє використання пам'яті	0.008
Відхилення пам'яті	0.015
Тренд витоку пам'яті	0.034
Середній час обробки запиту	0.024
Середній час на запит	0.028
Середній коефіцієнт деградації	0.0170
Середній час виконання 1 асинхронної задачі	0.0086
Середній коефіцієнт зміни часу при зростанні кількості задач	0.0301
Середній загальний час виконання пакетів задач	0.0258
Середня затримка	0.0170
Середній джиттер	0.0021
Середня втрата пакетів	0.0043
Середня пропускна здатність	0.0170
Попит на ринку праці	0.125
Актуальність	0.075
Розвиненість спільноти	0.06
Доступність документації	0.06
Складність освоєння	0.045
Вартість навчання	0.0225
Час на навчання	0.0135

Провівши дане та попередні етапи розробки, доцільно побудувати дерево прийняття рішень.

4.4 Отримання та представлення результатів дослідження

Побудова дерева прийняття рішення починалася з підготовки і узгодження трьох компонентів вихідних даних, які й лягли в основу подальшої логіки відбору. По-перше, були отримані усереднені результати серійних прогонів для всіх припустимих комбінацій технологій. По-друге, паралельно з експериментальною частиною зібрано освітні характеристики, що описують складність освоєння, часові та фінансові витрати на вивчення, доступність документації, розвиненість

спільноти та інші навчальні аспекти мов і фреймворків. По-третє, для зважування окремих показників було визначено набір ваг, що визначають відносний вклад кожної метрики в підсумкову оцінку. Ці ваги служать масштабом, який узгоджує технічні й освітні домени в єдиному ранжуванні.

Дерево відтворює систему послідовного відбору критеріїв, орієнтовану на потреби новачка: воно дозволяє крок за кроком фільтрувати множину стеків технологій веброзробки за набором бажаних властивостей. Для кожного критерію, що розглядається, було збудовано емпіричний розподіл значень по всій сукупності стеків і на його основі визначено опорні (порогові) інтервали. Значення, що потрапляють до нижньої крайової ділянки розподілу, інтерпретуються як переваги, а значення з верхньої крайової ділянки відповідно класифікуються як недоліки. Такий підхід дозволяє об'єктивно виділяти сильні та слабкі сторони кожного варіанта на підставі реальних вимірювань і не вимагає ручного встановлення абстрактних меж.

Агрегований індекс кожного стеку обчислюється як сума внесків окремих метрик, де внесок визначається добутком та її ваги. Внаслідок у вузлі дерева поряд із списком стеків зберігається і ранжування за цим індексом: нижчий індекс означає вищу відповідність обраним перевагам. Одночасно для кожного стеку фіксується набір метрик, які потрапили в крайові зони розподілу – ці набори і становлять формальну основу для позначення переваг і недоліків, що відображається в інтерфейсі перегляду дерева. Через значну комбінаторну розмірність повного дерева подати його в графічному вигляді повністю неможливо, тому у роботі наводиться вибіркоковий фрагмент дерева (рисунок 4.2), що ілюструє типовий шлях відбору критеріїв та демонструє формат подання результатів у вузлі.

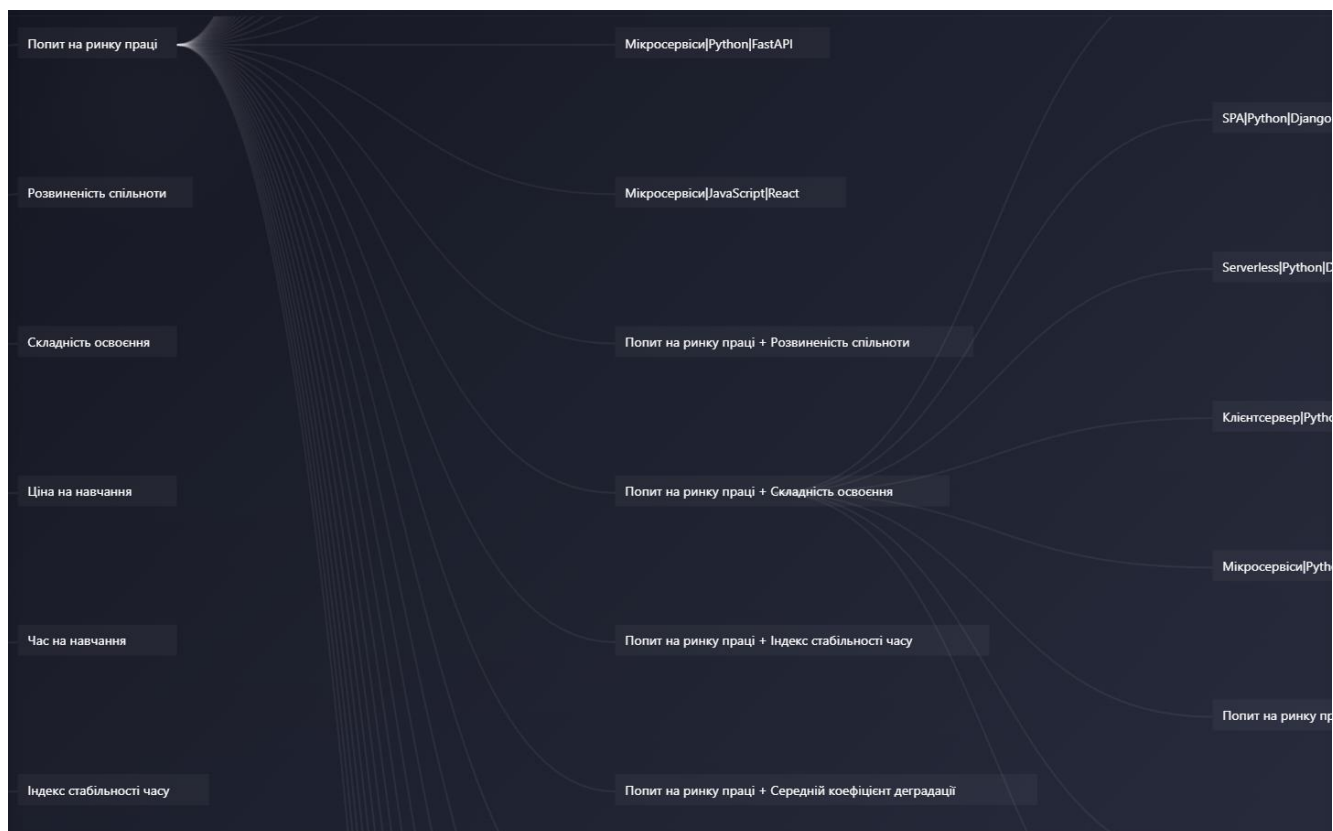


Рисунок 4.2 – Відображення роботи дерева прийняття рішення

На рисунку 4.3 проілюстровано приклад відображення стеку технологій, що включає назву, рахунок (score), переваги та недоліки.

Details

SPA|Python|Django

Score: 0.326

Переваги:

- Індекс стабільності часу
- Складність освоєння**
- Попит на ринку праці**
- Доступність документації
- Актуальність

Недоліки:

- Середній витік пам'яті (MB)
- Середнє використання пам'яті (MB)
- Відхилення пам'яті (MB)
- Середня затримка (ms)
- Середній джиттер (ms)
- Час на навчання

Рисунок 4.3 – Характеристики стеку технологій “Моноліт, Python, Django”

Запропонована методика до формування дерева має практичну цінність для цільової аудиторії – початківців у веброзробці. Послідовність відбору критеріїв забезпечує інтуїтивно зрозумілий механізм фільтрації, а одночасне відображення переваг і недоліків робить прозорими компроміси між альтернативами. У висновку слід підкреслити, що результати дерева залежать від якості вхідних вимірювань і встановлених ваг – тому вони найкраще служать як інструмент підтримки прийняття рішення, що надає структуроване і візуально зручне уявлення про співвідношення технічних та освітніх факторів при виборі технологічного стеку.

4.5 Порівняння витрат часу на визначення метрики

В даному підрозділі наведено порівняння результату дослідження, дерева прийняття рішень, з аналогічними ручними методиками отримання метрик. Для порівняння було виміряно час, необхідний користувачу для знаходження однакових показників: «Середня загальна швидкість виконання коду» та «Попит на ринку праці».

Як аналог, для визначення технології за технічним критерієм, було використано бенчмарк TechEmpower, а для визначення за освітнім критерієм було використано подвійний метод, який включає запит у пошукачі браузера популярних технологій веброзробки та перевірка даних на платформі з вакансіями. Для цього найкраще підходить платформа Dou, оскільки на ній наводяться статті оглядів вебтехнологій і наявне середовище з вакансіями.

Оскільки неможливо визначити даний час, який може використати середньостатистичний користувач, було прийнято використовувати приблизний час пошуку даних з технології в якості користувача, який вже мав у цьому досвід, таким чином уникаючи похибку у визначенні часу, який витрачається на ознайомлення з платформами.

Загальний час на пошук технології за критерієм включає суму усереднених часових витрат на типові дії: запуск браузера, знаходження потрібного сайту, завантаження сторінки, пошук критерію на сторінці, застосування фільтрування і фіксацію результату.

Результати наведені на рисунку 4.4, а якому зображена гістограма з часовими витратами на пошук новачком технології за типами критеріїв. Дана гістограма включає в себе пошук за технічним, освітнім критерієм і їх сумою. Таке рішення у відображенні було прийнято, оскільки дерево прийняття рішень охоплює пошук за технічними та освітніми метриками одночасно.

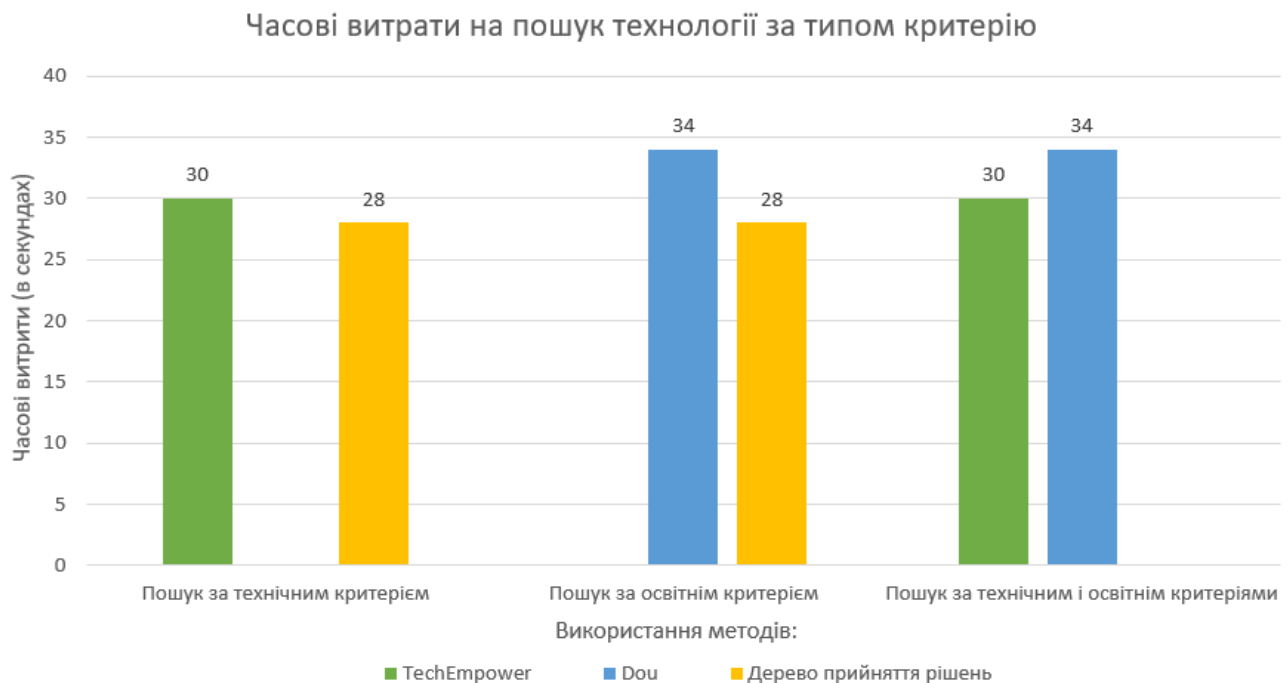


Рисунок 4.4 – Часові витрати на пошук за типом критерію

За даними вимірами часових замірів можна визначити приблизний відсоток покращення методу обрання технології веброзробки:

- обрання за технічним критерієм ~ 6,6%;
- обрання за освітнім критерієм ~ 17,6%;
- обрання за технічним і освітніми критеріями ~ 12,5%.

Таким чином, отримані результати доводять, що запропонована методика, яка наведена у дипломній роботі, дозволяє швидше отримувати релевантні дані без втрати якості оцінки, що робить її корисною в умовах обмеженого часу – наприклад, під час прийняття технічних рішень у проєктних командах або при оперативній підготовці рекомендацій для рекрутингу та навчальних програм. Крім того, стандартизація кроків збору та нормалізації зменшує залежність результату від індивідуального досвіду користувача, підвищуючи відтворюваність і зручність інтеграції методики в робочі процеси.

Із цього можна відзначити, що наукова цінність роботи полягає в розробці інтегрованої двошарової методики, яка поєднує технічні та освітні метрики в єдине формалізоване дерево прийняття рішень, а також у створенні відтворюваної методики збору й нормалізації даних із поєднанням алгоритмічної обробки та якісної калібрувальної процедури для тонкого налаштування ваг. Це нововведення – разом із введенням часових витрат користувача як кількісного критерію ефективності – забезпечує більш обґрунтований, реплікований і адаптований до практики підхід до порівняння технологій, що відкриває можливості для подальших досліджень і масштабування у суміжні домени.

Отже, ціль, що була поставлена у даній дипломній роботі, досягнута.

ВИСНОВКИ

У ході виконання дипломного проєкту здійснено послідовний цикл робіт, що охопив теоретичне обґрунтування, практичну реалізацію інструменту для збору експериментальних даних і формалізацію процедури вибору технологічного стеку у вигляді дерева прийняття рішень.

Технічна частина дипломної роботи полягала в автоматизованому зборі показників продуктивності й стабільності для набору допустимих комбінацій «архітектура – мова – фреймворк» шляхом багаторазових прогонів з подальшим усередненням результатів. Реалізація цього етапу виконана як вебінструмент, що забезпечує відтворюваність і централізацію експериментальної бази.

Паралельно з експериментальною частиною проведено збір освітніх метрик – показників простоти входу, вартості та часу навчання, рівня попиту на ринку, доступності документації та інших факторів, що визначають практичну придатність стеку для студентів і початківців. Отримані освітні дані агреговано і підготовлено для інтеграції з технічними індикаторами.

Отримавши результати випробувань за зібрані метрики було побудовано вагову модель в два етапи: початкове присвоєння ваг, яке виконано за принципом «мін-макс» для прозорого і відтворюваного старту, далі здійснено якісну калібрувальну корекцію на підставі експериментальних спостережень і логіки застосування. Остаточного застосовано баланс між технічним та освітнім блоками, що визначає внесок кожної групи у загальний індекс. Такий підхід поєднує простоту початкової конфігурації та можливість її обґрунтованого налаштування після отримання реальних даних.

На базі підготовлених входів було реалізовано дерево прийняття рішень у вигляді послідовної процедури відбору та ранжування: для кожного критерію визначено порогові інтервали (нижні і верхні крайні частки розподілу), за якими формується набір виявлених переваг і недоліків стеку. Далі альтернативи фільтруються за обраними уподобаннями користувача і ранжуються за

агрегованим балом (сума внесків технічних та освітніх метрик з урахуванням ваг). Візуалізація дерева спрощує інтерпретацію результатів і дозволяє початківцю ознайомитися з варіантами стеків веброзробки та знаходити компроміси між ними.

У підсумку проведених робіт отримано метод для визначення стеку технологій для початківця у вигляді дерева прийняття рішень, практична користь якого полягає у зниженні ризику неправильного вибору технології для навчальних і прикладних задач: користувач отримує збалансовану рекомендацію, що поєднує продуктивність, стабільність та освітню придатність.

Серед можливих напрямів подальшого вдосконалення слід виділити: розширення набору експериментів (збільшення числа прогонів і різноманітності сценаріїв) для зростання надійності вимірів; формалізоване чутливісне дослідження впливу ваг (статистична валідація), а також застосування методів багатокритеріального аналізу (наприклад, АНР/TOPSIS) або машинного навчання для адаптивного визначення ваг на основі реального використання. Крім того доцільно розробити інтерфейс персоналізації та регулярне оновлення освітніх даних для збереження релевантності рекомендацій у динамічному ринку праці.

Отже, виконані в роботі дослідження і реалізація забезпечили очікуваний результат: створено відтворювану, інтерпретовану та практично корисну процедуру вибору стеку, що може бути використана як інструмент підтримки прийняття рішень у навчальному процесі та при проєктному плануванні, а також слугить платформою для подальших досліджень і розвитку.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Веб-розробка - з чого розпочати навчання. URL: <https://kiev.itstep.org/blog/web-development-where-to-start-learning> (дата звернення: 01.12.2025);
2. Архітектура програмного забезпечення: як правильне проєктування забезпечує ефективність та безпеку. URL: <https://wezom.com.ua/ua/blog/arhitektura-programnogo-zabezpechennya> (дата звернення: 01.12.2025);
3. Найпопулярніші мови програмування в 2025 році. URL: <https://dan-it.com.ua/uk/blog/samye-populjarnye-jazyki-programmirovaniya-v-2025-godu/> (дата звернення: 01.12.2025);
4. Рейтинг фреймворків для веброзробки. URL: <https://brander.ua/blog/reytynh-freymvorkiv-dlya-vebrozrobky> (дата звернення: 01.12.2025);
5. Архітектура вебзастосунків 2024: Ультимативний гайд для розробників. URL: <https://robotdreams.cc/uk/blog/567-arhitektura-vebzastosunkiv> (дата звернення: 01.12.2025);
6. Клієнт-серверна архітектура. URL: <https://training.qatestlab.com/blog/technical-articles/client-server-architecture/> (дата звернення: 02.12.2025);
7. Рейтинг мов програмування 2025. URL: <https://dou.ua/lenta/articles/language-rating-2025/> (дата звернення: 02.12.2025);
8. Топ-10 мов програмування, які будуть актуальні в 2025. URL: <https://career.softserveinc.com/uk-ua/stories/top-10-programming-languages-to-learn> (дата звернення: 02.12.2025);
9. Найпопулярніші мови програмування у 2024. URL: <https://robotdreams.cc/uk/blog/588-naypopulyarnishi-movy-prohramuvannya-u-2024> (дата звернення: 02.12.2025);

10. Яку мову програмування вчити початківцю у 2025 році. URL: <https://goit.global/ua/articles/yaku-movu-prohramuvannia-vchyty-pochatktivtsiu/> (дата звернення: 03.12.2025);
11. Go language rises in Tiobe popularity index. URL: <https://www.infoworld.com/article/3603503/go-language-rises-in-tiobe-popularity-index.html> (дата звернення: 03.12.2025);
12. Libraries. URL: <https://2024.stateofjs.com/en-US/libraries/> (дата звернення: 03.12.2025);
13. React (and React Native) for SMEs: How It Compares to Vue and Angular. URL: <https://mainstay.digital/insights/react-vs-vue-vs-angular-smes-2025> (дата звернення: 03.12.2025);
14. A Year In Review: Flask in 2024: Flask in 2024. URL: <https://blog.miguelgrinberg.com/post/a-year-in-review-flask-in-2024> (дата звернення: 03.12.2025);
15. go-web-framework-stars. URL: <https://github.com/mingrammer/go-web-framework-stars> (дата звернення: 05.12.2025);
16. Web Framework Benchmarks. URL: <https://www.techempower.com/benchmarks> (дата звернення: 08.12.2025).
17. State of JS 2024. URL: <https://2024.stateofjs.com> (дата звернення: 05.12.2025);
18. Stack Overflow Survey 2024. URL: <https://survey.StackOverflow.co/2024/technology> (дата звернення: 08.12.2025).
19. Courses. URL: <https://www.coursera.org/courses> (дата звернення: 05.12.2025);
20. Web Development Courses. URL: <https://www.udemy.com/courses/development/web-development/> (дата звернення: 05.12.2025);
21. MDPI. URL: <https://www.mdpi.com/2227-7390/9/4/308> (дата звернення: 05.12.2025);

22. Technology stack selection. URL: <https://hj.diva-portal.org/smash/get/diva2%3A1681404/FULLTEXT01.pdf> (дата звернення: 05.12.2025);

23. Flask vs FastAPI: An In-Depth Framework Comparison. URL: <https://betterstack.com/community/guides/scaling-python/flask-vs-fastapi/> (дата звернення: 05.12.2025);

24. How to Choose Right Technology Stack for Web Product Development. URL: <https://www.techmagic.co/blog/technology-stack-for-web-development> (дата звернення: 05.12.2025);

25. 5 Steps to Choosing the Best Tech Stack for Your Project. URL: <https://www.leanware.co/insights/5-steps-to-choosing-the-best-tech-stack-for-your-project> (дата звернення: 05.12.2025);

26. An Overview of MEAN, MERN, and LAMP Stacks: A Comparative Analysis. URL: https://www.researchgate.net/publication/386088433_An_Overview_of_MEAN_MERN_and_LAMP_Stacks_A_Comparative_Analysis_of_Architecture_Cost_Learning_Curve_and_Community_Support (дата звернення: 05.12.2025);

27. Rethinking Technology Stack Selection with AI Coding Proficiency. URL: <https://www.arxiv.org/pdf/2509.11132> (дата звернення: 05.12.2025);

28. React vs Angular vs Vue Performance in 2025. URL: <https://medium.com/%4oreactmasters.in/react-vs-angular-vs-vue-performance-in-2025-7590f673494b> (дата звернення: 05.12.2025);

29. Introduction to TechEmpower Web Benchmarks. URL: <https://goframe.org/en/articles/techempower-web-benchmarks-r23> (дата звернення: 05.12.2025);

30. TechEmpower Framework Benchmarks. URL: <https://tfb-status.techempower.com/> (дата звернення: 05.12.2025);

31. Implementation of TypeScript: <https://www.preprints.org/manuscript/202407.0051> (дата звернення: 05.12.2025).

ДОДАТОК А

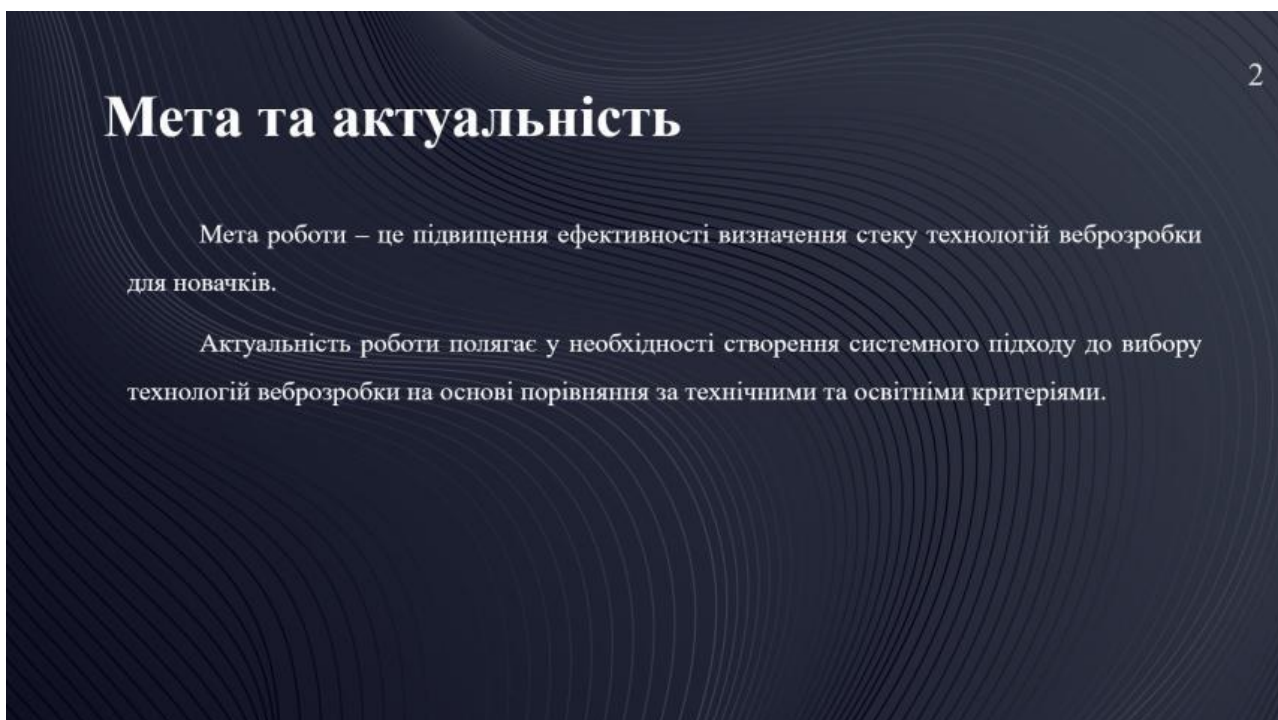


Рисунок А.1 – Слайд “Мета та актуальність”



Рисунок А.2 – Слайд “Аналоги”

Постановка задачі

Складність об'єктивного вибору технологій веброзробки опираючись на технічні та освітні критерії і їх порівняння між собою.



Рисунок А.3 – Слайд “Постановка задачі”

Вирішення проблеми

Створення комплексної методики оцінювання технологій веброзробки з метою отримання повноцінної оцінки.



Рисунок А.4 – Слайд “Вирішення проблеми”

Проектування системи

6

Дана система має охоплювати:

- Методику збору метрик за технічними критеріями;
- Методу збору метрик за освітніми критеріями;
- Систему порівняння і відтворення результатів.

Результатом даної системи є:

- Відображення стеків технологій за допомогою дерева прийняття рішення;
- Показувати переваги і недоліки кожного стеку для порівняння між іншими.

Рисунок А.5 – Слайд “Проектування системи”

Методики збору метрик за технічними і освітніми критеріями

7

Дані методики визначають такі етапи:

- Визначення критеріїв оцінювання;
- Проведення випробувань за технічними критеріями;
- Збір освітніх метрик на різних платформах надавачів освіти, з вакансіями, з документацією та сайту спільноти програмістів.



Рисунок А.6 – Слайд “Методи збору метрик за технічними і освітніми критеріями”

Проведення випробувань

Проведення випробувань було здійснено за допомогою авторського інструмента. Даний інструмент являє собою вебзастосунок, який складається з клієнта, сервера і бази даних. Поруч наведено алгоритм роботи вебзастосунку.

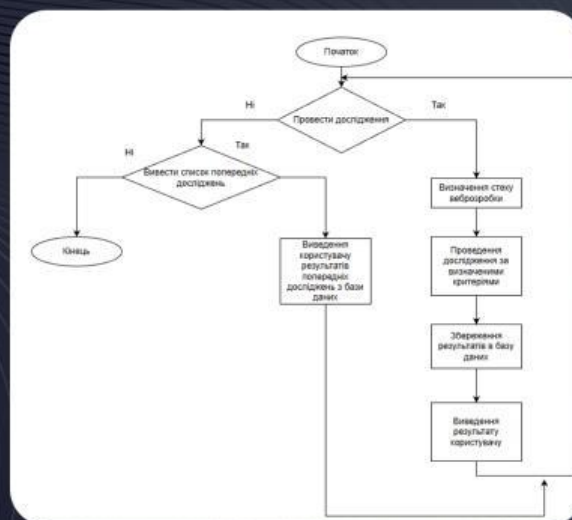


Рисунок А.7 – Слайд “Проведення випробувань”

Реалізація вебзастосунку



Початковий екран

Параметр	Значення
Середній час математичного навантаження (мс)	16,671
Середній час асинхронного навантаження (мс)	1,071
Середній загальний час виконання коду (мс)	17,763
Середнє споживання пам'яті (МБ)	3,503
Середнє споживання пам'яті об'єктів (МБ)	3,719
Середнє загальне споживання пам'яті (МБ)	7,219
Середній час виконання (мс)	56,989
Виконання часу (мс)	5,695
Середнє використання пам'яті (МБ)	442,676
Виконання пам'яті (МБ)	156,482
Середній загальний час виконання (мс)	22,836
Трива виконання пам'яті (МБ)	515,311
Віднос. стабільності часу	0,9
Віднос. стабільності пам'яті	-0,184
Середній час обробки запиту (мс)	247,474
Середній час на запит (мс)	1,627
Середній коефіцієнт затримки	2,012
Середній час виконання і кешування даних (мс)	0,348
Середній коефіцієнт затримки при зчитуванні даних	1,489
Середній загальний час виконання і кешування даних (мс)	35,213
Середня затримка (мс)	35,93
Середній джоттер (мс)	0,916
Середня тривота пакетів	9
Середня пропускна здатність (Kb/s)	177,831

Екран результату з таблицею

Рисунок А.8 – Перший слайд з групи слайдів “Реалізація вебзастосунку”

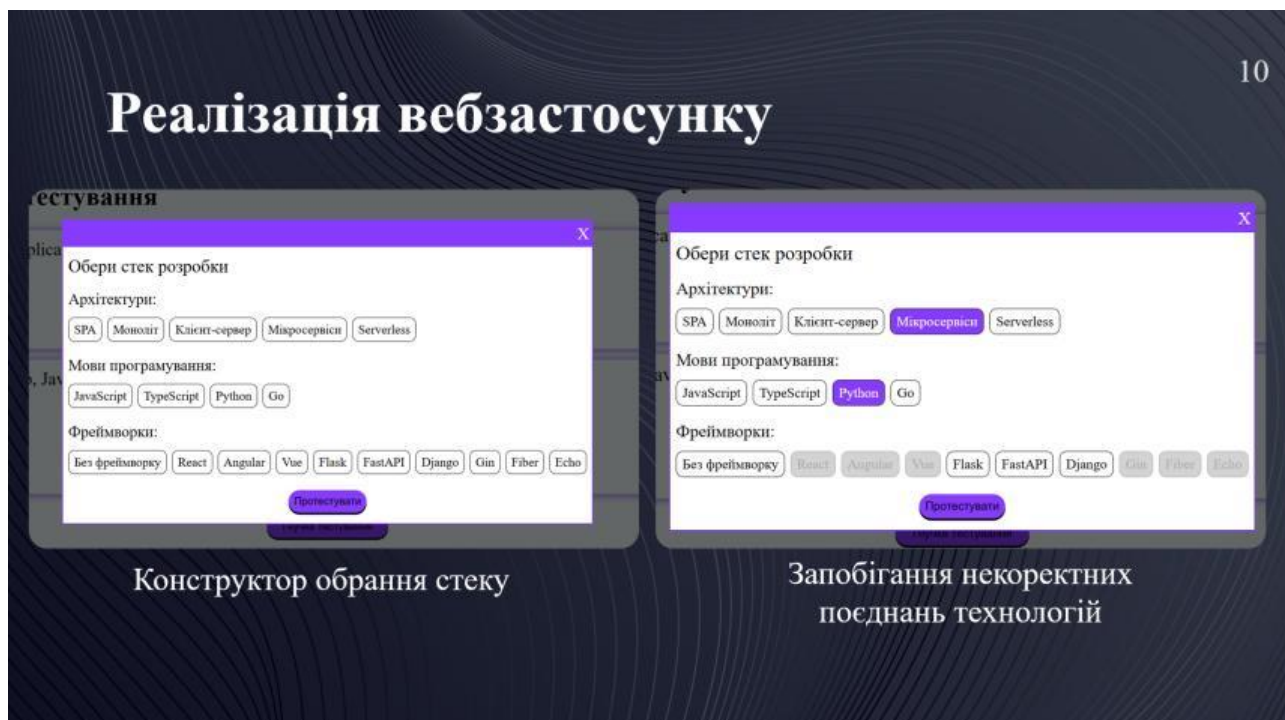


Рисунок А.9 – Останній слайд з групи слайдів “Реалізація вебзастосунку”

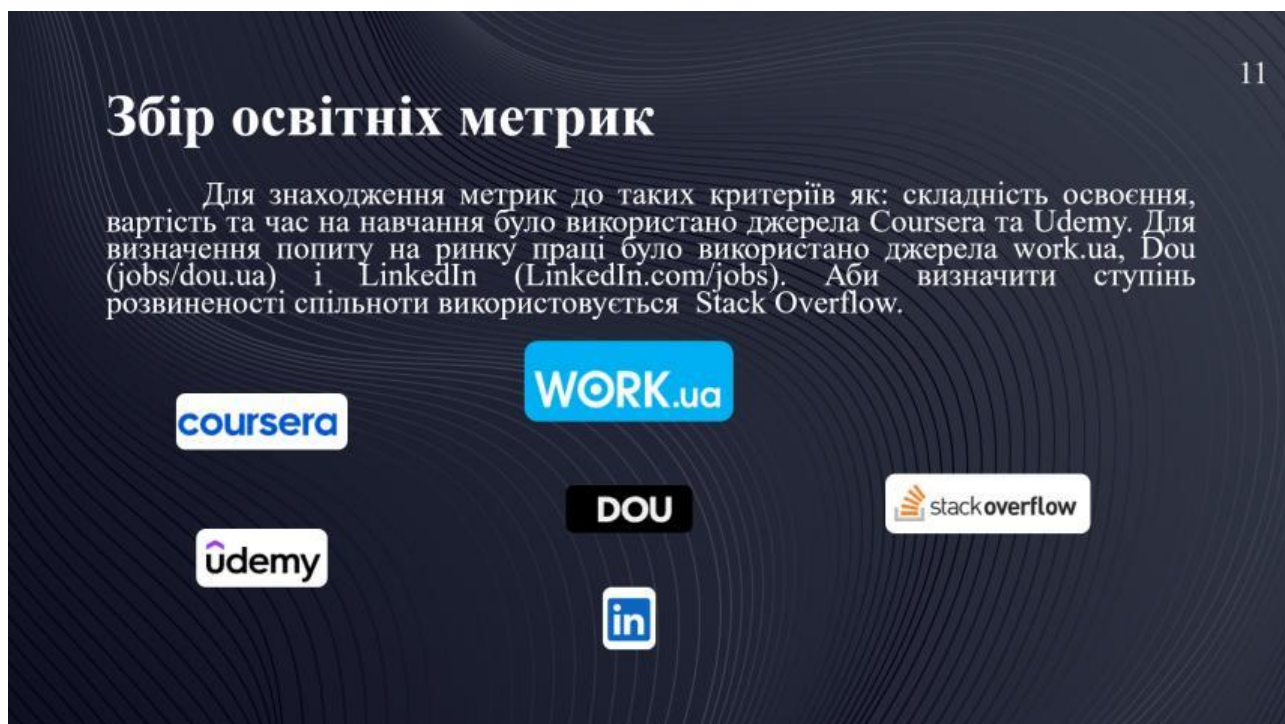


Рисунок А.10 – Слайд “Збір освітніх метрик”

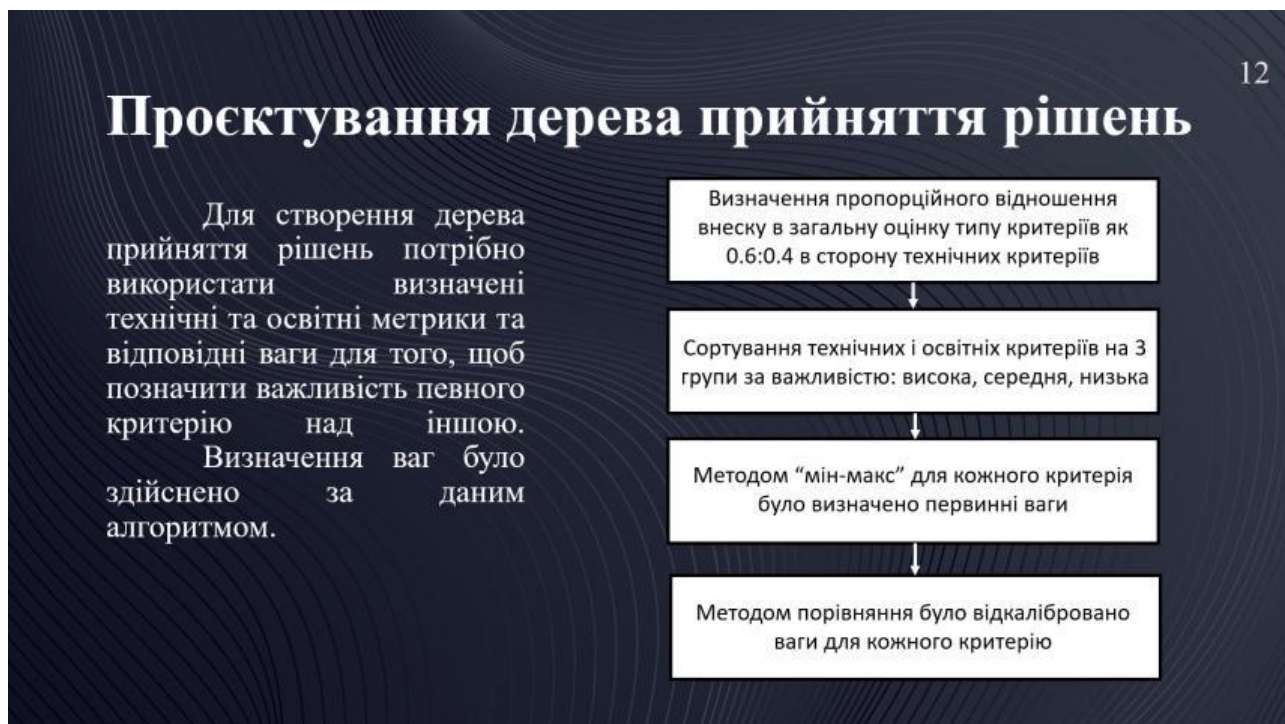


Рисунок А.11 – Слайд “Проектування дерева прийняття рішень”

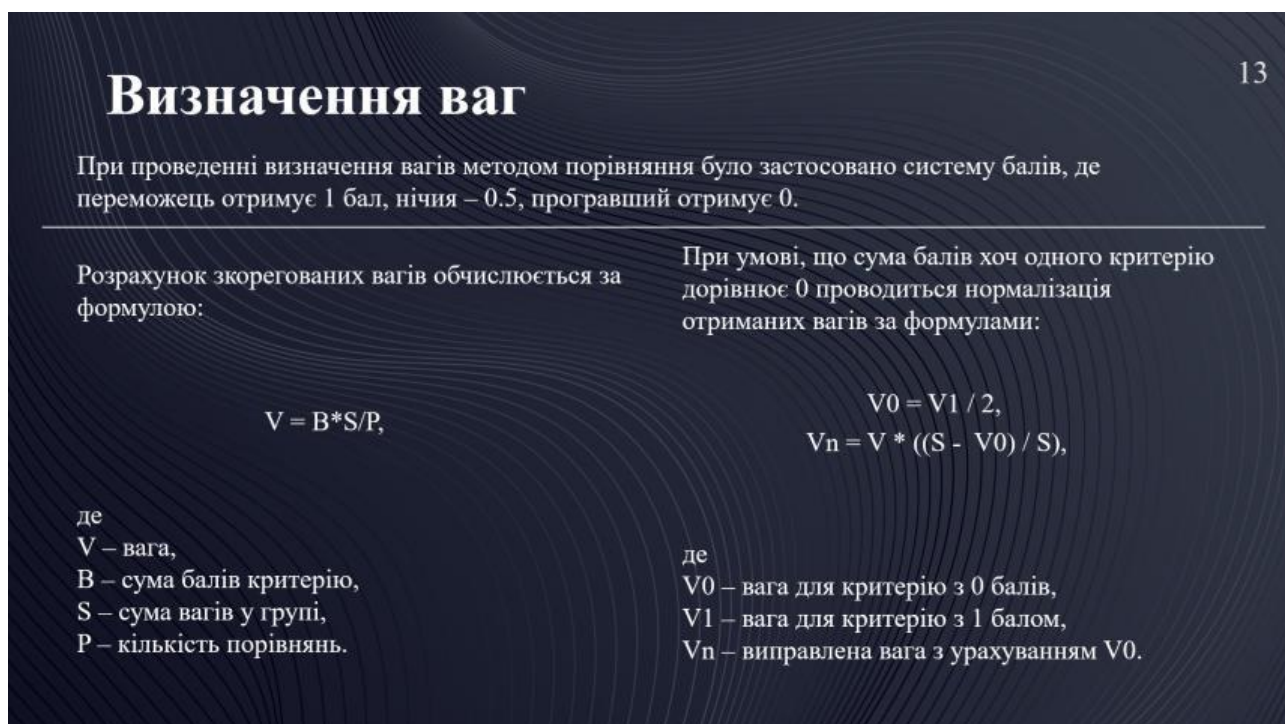


Рисунок А.12 – Перший слайд з групи слайдів “Визначення ваг”



Рисунок А.13 – Останній слайд з групи слайдів “Визначення ваг”

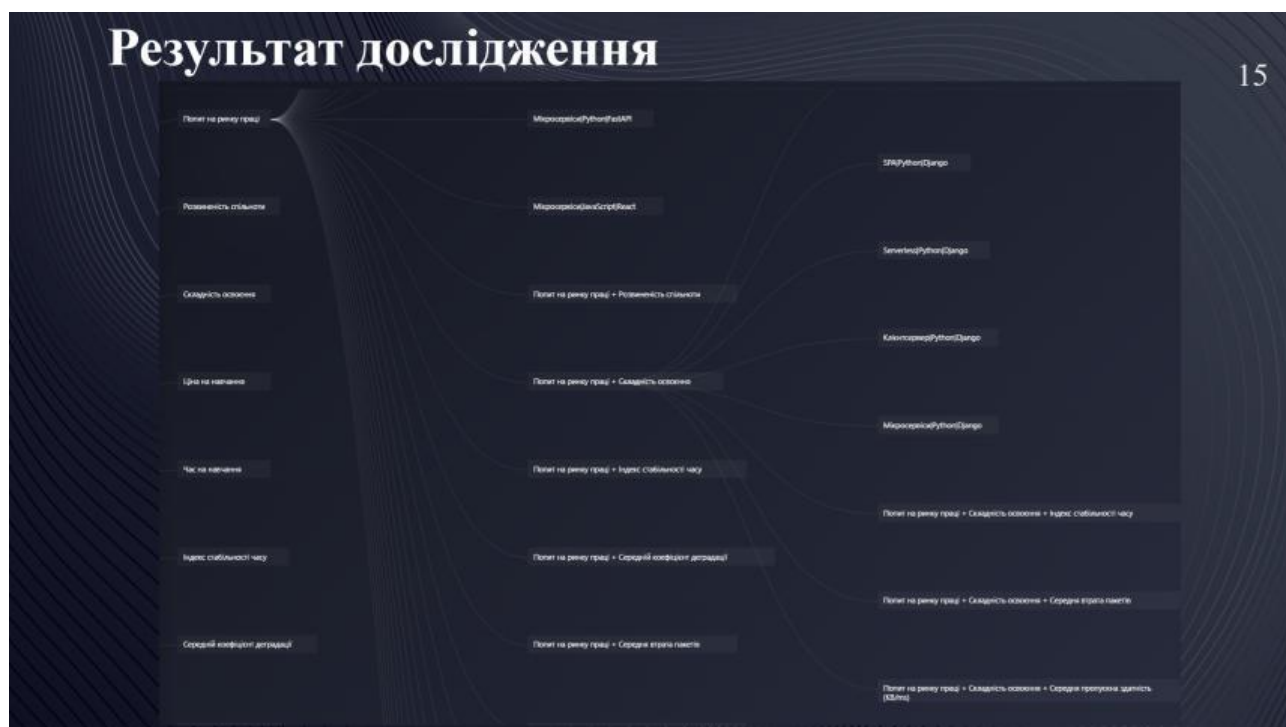


Рисунок А.14 – Перший слайд з групи слайдів “Результат дослідження”

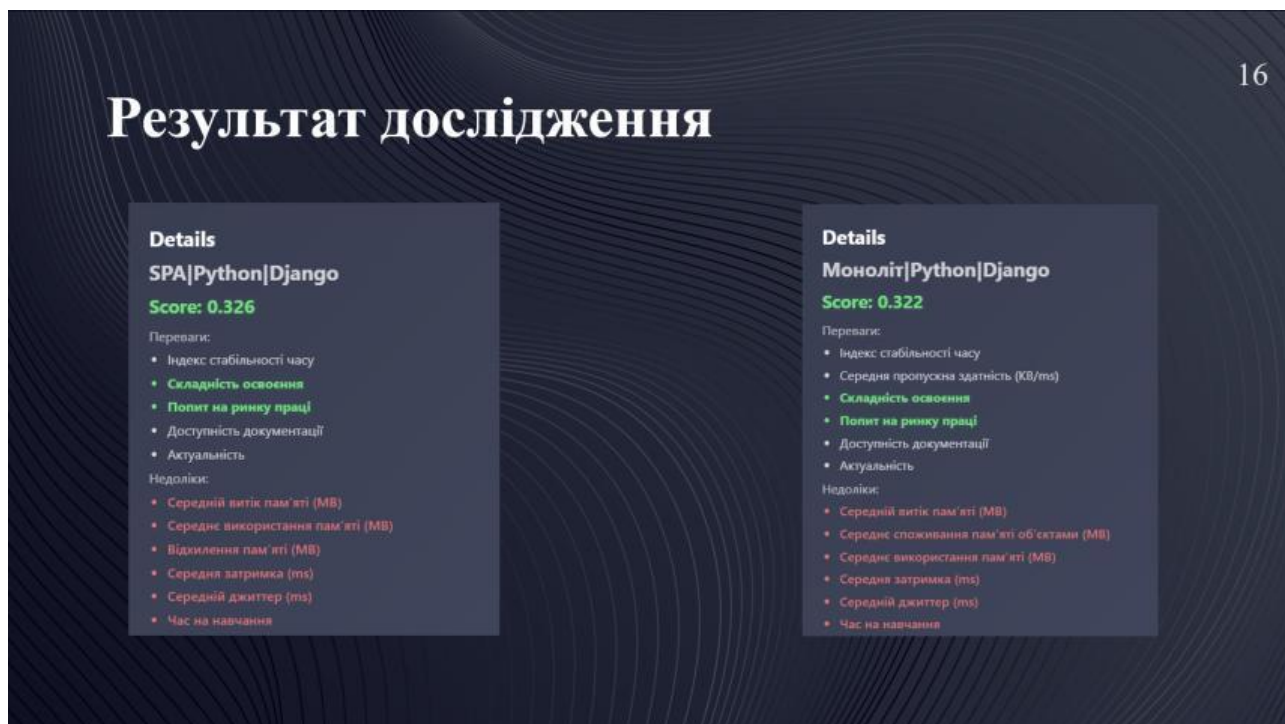


Рисунок А.15 – Останній слайд з групи слайдів “Результат дослідження”



Рисунок А.16 – Слайд “Порівняння результату дослідження з аналогічними інструментами”

Напрями вдосконалення

- Розширити експерименти: збільшити число прогонів і різноманітність сценаріїв для підвищення надійності вимірів;
- Провести формалізований чутливий аналіз ваг з статистичною валідацією результатів;
- Інтегрувати багатокритеріальні методи (АНР, TOPSIS) та/або підходи машинного навчання для адаптивного визначення ваг;
- Додати інтерфейс персоналізації (налаштування пріоритетів користувача) для індивідуальних рекомендацій;
- Регулярно оновлювати освітні дані (курси, вакансії, спільноти) для збереження релевантності рекомендацій.

Рисунок А.17 – Слайд “Напрямки вдосконалення”

Висновки

Було зроблено:

- Реалізовано автоматизований збір технічних метрик шляхом багаторазових прогонів з усередненням і централізованим збереженням результатів;
- Зібрано освітні метрики та інтегровано їх із технічними індикаторами;
- Розроблено двоетапну вагову і реалізовано логіку дерева з пороговими інтервалами, фільтрацією за вподобаннями користувача й ранжуванням за агрегованим балом.

Виконані в роботі дослідження і реалізація забезпечили очікуваний результат: створено відтворювану, інтерпретовану та практично корисну методику до вибору стеку, що може бути використана як інструмент підтримки прийняття рішень у навчальному процесі та при проєктному плануванні, а також служить платформою для подальших досліджень і розвитку.

Рисунок А.18 – Слайд “Висновки”