

ПОЛТАВСЬКИЙ ДЕРЖАВНИЙ АГРАРНИЙ УНІВЕРСИТЕТ
Навчально-науковий інститут економіки, управління, права та
інформаційних технологій
Кафедра інформаційних систем та технологій

КВАЛІФІКАЦІЙНА РОБОТА

на здобуття ступеня вищої освіти магістр

на тему: «Дослідження хмарових технологій розробки вебзастосунків»

Виконав: здобувач вищої освіти
за освітньою програмою
Інформаційні управляючі системи та
технології
спеціальності 126 Інформаційні системи та
технології
ступеня вищої освіти магістр
групи 126ІСТ_мд_2023
Тарбеєв Олексій Павлович
Керівник: Флегантов Леонід Олексійович
Рецензент: Ковальчук Станіслав Богданович

Полтава – 2024 року

ВСТУП

Актуальність теми. Застосування хмарових технологій у розробці вебзастосунків є актуальним рішенням через зростаючий попит на гнучкість, масштабованість та швидкість розгортання сучасних застосунків. Сучасні компанії часто обирають хмарові платформи AWS, Google Cloud Platform, Microsoft Azure завдяки їх можливості забезпечити високопродуктивну інфраструктуру, автоматичне масштабування, резервне копіювання та підтримку безперервного циклу інтеграції та доставки (CI/CD). Це дозволяє швидко реагувати на зростаюче навантаження, не витрачаючи значних ресурсів на підтримку локальних серверів.

Використання хмарових технологій особливо актуальне в умовах розвитку віддаленої роботи та розподілених команд, де швидкий доступ до даних і послуг із різних точок світу стає критично важливим. Крім того, хмарові сервіси дозволяють інтегрувати вебзастосунки з іншими сервісами, забезпечуючи безпечний обмін даними та доступ до аналітичних інструментів, необхідних для бізнес-рішень.

Безпека даних також є одним із важливих факторів, оскільки хмарові платформи надають розширені можливості шифрування даних і управління доступом, що відповідає сучасним стандартам інформаційної безпеки. Завдяки автоматичному масштабуванню та високій надійності хмарові рішення оптимізують витрати й підвищують ефективність бізнесу, надаючи можливість швидкої та безперебійної роботи вебзастосунків. Отже, дослідження хмарових технологій у веброзробці має суттєве значення для створення адаптивних, продуктивних і захищених вебзастосунків у сучасних умовах.

Зв'язок роботи з науковими програмами, планами, темами. Робота виконана у відповідності до науково-дослідної ініціативної теми «Організаційно-методологічні аспекти впровадження інформаційно-комунікаційних систем і технологій в управлінні діяльністю сучасних організацій та підприємств за умов переходу до цифрової економіки» ДРН 0123U105060.

Мета роботи: дослідити ефективність використання хмарових технологій у розробці вебзастосунків.

Завдання роботи:

1. Виконати огляд основних хмарових технологій;
2. Дослідити конкретні хмарові платформи (AWS, Google Cloud Platform, Microsoft Azure) з метою визначення їх можливостей для веброзробки;
3. Виконати тестову розробку вебзастосунку на базі хмарової платформи, оцінити ефективність її використання;
4. Сформулювати висновки та рекомендації щодо використання хмарових технологій для розробки сучасних вебзастосунків.

Об'єкт дослідження: хмарові технології розробки вебзастосунків.

Предмет дослідження: хмарові платформи (AWS, Google Cloud Platform, Microsoft Azure) як інструменти для розробки та масштабування вебзастосунків.

Методи дослідження:

- огляд літератури, наукових статей;
- дослідження можливостей сервісів AWS, Google Cloud Platform, Microsoft Azure для розробки вебзастосунків; порівняльний аналіз продуктивності, витрат, масштабованості та функціональних можливостей різних хмарних платформ;
- практична розробка вебзастосунку із використанням мікросервісної архітектури, AWS Lambda, RDS, S3; аналіз та оцінка продуктивності, стабільності та зручності використання хмарових сервісів для вебзастосунку;
- синтез висновків і рекомендацій щодо ефективності хмарових технологій та їх впровадження в сучасних ІТ-проектах.

Інформаційна база дослідження: науково-технічна література; наукові статті та публікації з хмарових технологій, доступні в наукових базах даних Google Scholar, IEEE Xplore, SpringerLink, ScienceDirect, Scopus, ResearchGate; вітчизняні та зарубіжні стандарти та методології розробки вебзастосунків; офіційна документація та ресурси хмарових платформ AWS, Google Cloud Platform, Microsoft Azure; документація з технологій веброзробки (HTML, CSS, JavaScript) та мов програмування Python, Java, а також рекомендації з розробки

вебінтерфейсів; інтернет-джерела аналітичного характеру експертів у галузі ІТ, хмарових обчислень та DevOps.

Елементи наукової новизни: порівняльний аналіз популярних хмарових платформ (AWS, Google Cloud Platform, Microsoft Azure) дозволив виявити їх переваги та недоліки в контексті розробки вебзастосунків, зокрема щодо масштабованості, продуктивності та вартості; розробка хмарового середовища для вебзастосунку TaskFlow із використанням сучасної мікросервісної архітектури та безсерверних рішень (AWS Lambda, Amazon RDS, S3) дала змогу оцінити ефективність цих технологій у реальних умовах.

Практична значущість: робота надає практичні рекомендації з проєктування та оптимізації хмарового середовища для вебзастосунків, зокрема, з використання AWS сервісів (EC2, Lambda, RDS, S3) і технологій автоматичного масштабування, моніторингу та резервного копіювання; вебзастосунок TaskFlow демонструє практичне застосування хмарових технологій, включаючи інтеграцію з API, контейнеризацію (Docker, Kubernetes) та функціональні модулі, такі як реєстрація/авторизація через OAuth.

Апробація результатів дослідження. За результатами проведеного дослідження опубліковано тези доповіді: «Аналіз ефективності хмарових технологій у розробці вебзастосунків». Матер. ХХІ щорічного міждисциплінарного семінару «Студентські роботи за науковою тематикою кафедри інформаційних систем та технологій ННІ ЕУП та ІТ ПДАУ», 20 листопада 2024 року, м. Полтава.

Структура та обсяг кваліфікаційної роботи. Робота складається зі вступу, трьох розділів, висновків, списку використаних джерел та додатків. Основний текст роботи викладений на 69 сторінках, містить 19 рисунків і 28 таблиць. Список використаних джерел налічує 91 найменування.

РОЗДІЛ 1

ТЕОРЕТИКО-МЕТОДОЛОГІЧНІ АСПЕКТИ ВИКОРИСТАННЯ ХМАРОВИХ ТЕХНОЛОГІЙ У РОЗРОБЦІ ВЕБЗАСТОСУНКІВ

1.1 Основи веброзробки та вебзастосунків

Веброзробка, як процес створення вебсайтів, вебзастосунків та сервісів, які працюють у локальній комп'ютерній мережі або у глобальній мережі Інтернет, включає програмування клієнтської (front-end) і серверної частини (back-end) вебзастосунків, налаштування баз даних і забезпечення взаємодії між ними.

Сучасна роль та значення веброзробки полягає у забезпеченні ефективної інтерактивної взаємодії користувача з інформаційними системами (наприклад, CRM-системи, інтернет-магазини), розробці сервісів, що підтримують віддалену роботу (наприклад, вебплатформи для командної роботи), впровадженні технологій для автоматизації бізнес-процесів, забезпеченні мобільності через адаптивні вебзастосунки, підтримці глобальних сервісів, які доступні з будь-якого місця тощо [1-3].

Вебзастосунки класифікуються за різними критеріями, такими як рівень динамічності, функціональне призначення, архітектура, тип платформи та рівень інтерактивності [4]. Така класифікація дозволяє охопити існуюче різноманіття вебзастосунків і обирати оптимальні підходи для їх розробки та впровадження залежно від цілей та потреб користувачів (таблиця 1.1).

Таблиця 1.1 – Класифікація вебзастосунків

Критерій класифікації	Категорії	Опис	Приклади
За рівнем динамічності	Статичні Динамічні SPA (Single Page Application) PWA (Progressive Web Application)	Статичні мають фіксований контент, динамічні змінюють контент на основі запитів, SPA оновлюються без перезавантаження, PWA мають функціонал мобільних застосунків.	Вебсайт-візитка (статичний) Соціальні мережі (динамічні) Trello (SPA) Spotify Web (PWA)

Продовження таблиці 1.1.

За функціональним призначенням	Інформаційні Комерційні Соціальні Розважальні Корпоративні	Поділяються за основною метою: інформування, продаж, взаємодія користувачів, розваги чи підтримка внутрішніх процесів в організаціях.	Блоги (інформаційні) Інтернет-магазини (комерційні) Facebook (соціальні) YouTube (розважальні) CRM-системи (корпоративні)
За архітектурою	Монолітні Мікросервісні Безсерверні	Монолітні мають єдину структуру, мікросервісні складаються з незалежних компонентів, безсерверні виконуються на хмарових платформах.	Традиційні вебсайти (монолітні) Netflix (мікросервісні) Slack (безсерверні)
За типом платформи	Для настільних браузерів Мобільні Кросплатформенні	Вебзастосунки для використання на ПК, мобільних пристроях або адаптовані для всіх платформ завдяки адаптивному дизайну.	Gmail (для браузерів) Мобільні версії вебсайтів (мобільні) Facebook (кросплатформенні)
За рівнем інтерактивності	Інформаційні Інтерактивні Транзакційні	Інформаційні лише передають контент, інтерактивні дозволяють взаємодію (форми, запити), транзакційні підтримують складні операції (оплати, реєстрація).	Новинні портали (інформаційні) Форми для реєстрації (інтерактивні) Платіжні сервіси (транзакційні)

Технологічною основою веброзробки є технології HTML, CSS, JavaScript, серверні мови програмування та фреймворки, які загалом забезпечують створення вебзастосунків, дозволяють реалізувати їх структуру, оформлення, інтерактивність та серверну логіку [5, 6].

HTML (HyperText Markup Language) є основною мовою розмітки для створення вебсторінок. HTML визначає структуру вебдокументу (ієрархію та розташування заголовків, абзаців, посилань, форм тощо) та його семантику (сучасний HTML5 підтримує семантичні теги, такі як <header>, <footer>, <article>, які поліпшують читабельність коду для пошукових систем і розробників) [7-14].

CSS (Cascading Style Sheets) відповідає за оформлення вебсторінок: кольори, шрифти, відступи, вирівнювання, анімацію та адаптивність. CSS дозволяє відокремити стиль оформлення від структури, надає естетичний вигляд

вебзастосунку, забезпечує адаптивність дизайну для різних пристроїв (десктопів, планшетів, смартфонів) [15-20]. Використовує CSS-фреймворки, такі як Bootstrap, Tailwind CSS та ін., що спрощують створення адаптивного дизайну [21-25], підтримує препроцесори, такі як SASS і LESS [26-29]. Діаграма, яка наочно представляє базову класифікацію вебзастосунків, представлена на рисунку 1.1.

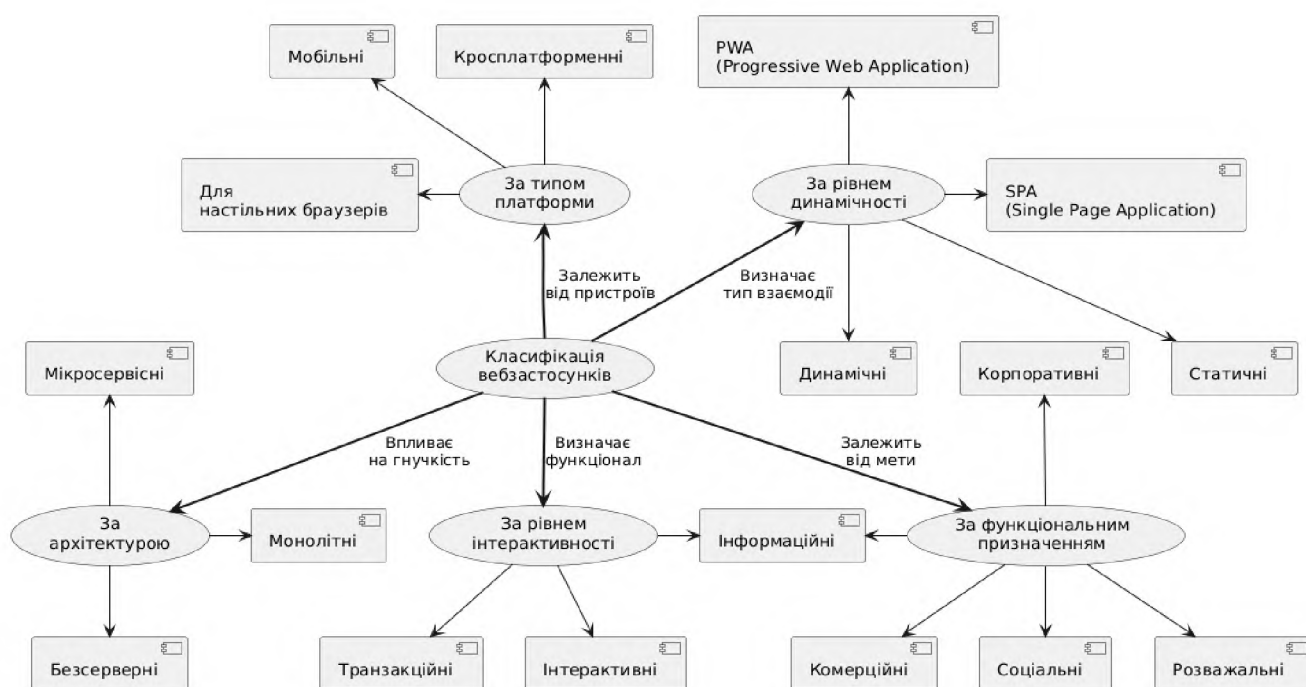


Рисунок 1.1 – Класифікація вебзастосунків

JavaScript забезпечує інтерактивність і динамічну поведінку вебзастосунків. Це мова програмування, яка виконується в браузері, що забезпечує обробку подій на боці клієнта (реагування на дії користувача, наприклад, кліки чи введення даних), маніпуляцію DOM (зміна HTML-елементів без перезавантаження сторінки), створення динамічних візуальних ефектів, забезпечує зв'язок з API для обробки даних у реальному часі, використовує бібліотеки та фреймворки (наприклад, React, Angular, Vue.js спрощують розробку складних SPA) [30-37].

Серверні мови програмування використовують виконання для завдань на стороні сервера: сервер обробляє запити клієнтів, працює з базами даних, виконує бізнес-логіку тощо. Популярні серверні мови: PHP (широко використовується для створення CMS, таких як WordPress) [38, 39], Python (підходить для обробки

даних, інтеграції AI та створення API, наприклад, з використанням Flask, Django) [40-42], , Node.js (використовує JavaScript для серверної частини, що спрощує розробку повного стеку (front-end і back-end)) [43], Java (застосовується у великих корпоративних проєктах завдяки надійності та масштабованості) [44-45].

Фреймворк – це програмне середовище або набір інструментів, що спрощує розробку програмного забезпечення, надаючи готові компоненти, шаблони та бібліотеки для виконання повторюваних задач. У веброзробці фреймворки використовуються для створення як клієнтської, так і серверної частини вебзастосунків. Фреймворки спрощують розробку, надаючи готові рішення для повторюваних задач. Популярні фреймворки: React, Angular, Vue.js (front-end); Laravel (PHP), Django (Python), Spring (Java) (back-end), Next.js, Nuxt.js (full-stack). Загалом, фреймворки забезпечують прискорення розробки (менше ручної роботи завдяки шаблонам і бібліотекам) та уніфікацію (дотримання стандартів для структури проєктів) [46, 47].

Таким чином, HTML створює основу вебзастосунків, CSS додає стиль, JavaScript забезпечує динамічність, серверні мови відповідають за обробку даних, а фреймворки спрощують розробку складних систем (таблиця 1.2).

Таблиця 1.2 – Базові технології веброзробки

Технологія	Призначення	Особливості
HTML	Створення структури вебсторінки.	- Визначення елементів: заголовки, абзаци, списки, таблиці, форми. - Семантична розмітка для SEO.
CSS	Оформлення зовнішнього вигляду вебсторінки.	- Стилзація: кольори, шрифти, відступи. - Адаптивний дизайн через медіазапити. - Анімація та ефекти.
JavaScript	Додавання інтерактивності та динамічності.	- Реагування на події користувача. - Маніпуляція DOM. - Інтеграція з API. - Використання фреймворків (React).
Серверні мови програмування	Обробка даних, робота з базами даних, виконання бізнес-логіки.	- Обробка запитів від клієнта. - Робота з базами даних. - Інтеграція API. - Масштабованість.
Фреймворки	Спрощення та прискорення розробки за рахунок готових рішень.	- Інструменти для роботи з базами даних. - Роутинг та обробка запитів. - Шаблонізація інтерфейсів.

Основні відомості про популярні серверні мови програмування та їх особливості представлені у таблиці 1.3.

Таблиця 1.3 – Популярні серверні мови програмування та їх особливості

Мова програмування	Характеристика	Приклади застосування	Фреймворки
PHP	Простота у використанні. Висока інтеграція з MySQL та іншими базами даних. Широке ком'юніті.	CMS (WordPress, Joomla, Drupal). Вебзастосунки з середнім навантаженням.	Laravel, Symfony, CodeIgniter
Python	Зручний і читабельний синтаксис. Підтримка машинного навчання та наукових обчислень. Легка інтеграція з базами даних.	Вебфреймворки (Django, Flask). Наукові розрахунки, аналітика даних (NumPy, pandas).	Django, Flask, FastAPI
Node.js	Асинхронна обробка запитів. Підтримка JavaScript на серверній стороні. Висока продуктивність у реальних застосунках.	Чати в реальному часі (Slack). Поточкові сервіси (Netflix). Інтернет-магазини.	Express, NestJS, Koa
Ruby	Простота у створенні REST API. Орієнтованість на розробку MVP (Minimum Viable Product). Легкість у підтримці коду.	Соціальні платформи (GitHub, Basecamp). Стартапи. Вебфреймворки для швидкої розробки.	Ruby on Rails, Sinatra
Java	Висока продуктивність і масштабованість. Підтримка багатопотоковості. Використовується в корпоративному середовищі.	Банківські системи. CRM та ERP платформи. Великі корпоративні вебзастосунки.	Spring Boot, Hibernate, Struts
Go (Golang)	Висока продуктивність. Статична типізація. Найкраще підходить для мікросервісів.	Хмарні сервіси (Docker, Kubernetes). Мікросервісна архітектура. Рішення для високих навантажень.	Gin, Echo, Beego
C#	Інтеграція з .NET платформою. Підходить для Windows-середовищ. Можливість кросплатформної розробки.	Вебзастосунки на платформі ASP.NET. Інтеграція корпоративних сервісів. Ігри (Unity).	ASP.NET, Blazor
Rust	Фокус на безпеку пам'яті. Висока продуктивність. Підходить для системного програмування.	Розробка вебсерверів. Побудова високонавантажених систем. Інтеграція з системами IoT.	Actix, Rocket

У таблиці вище представлені особливості кожної мови вебпрограмування, що роблять її придатною для певних задач. Приклади застосування вказують на типові сценарії використання мов. Вказані найпопулярніші фреймворки, які

працюють з відповідною мовою. Мову Java включено завдяки її популярності в корпоративному середовищі; Go (Golang) додано як одну з мов для високонавантажених систем; мова C# забезпечує інтеграцію з платформою .NET; мову Rust додано для опису системного програмування та безпеки пам'яті. Узагальнення відомостей за технологіями для різних типів вебзастосунків представлено у таблиці 1.4.

Таблиця 1.4 – Використання технологій для різних типів вебзастосунків

Тип вебзастосунків	Технології	Особливості	Приклади застосування
Статичні	HTML, CSS	Фіксований контент. Мінімальна інтерактивність. Швидке завантаження.	Лендінги, сторінки-візитки, статичні сторінки документації.
Динамічні	HTML, CSS, JavaScript, PHP, Python, Ruby	Генерація контенту на сервері або в браузері. Взаємодія з базами даних.	Блоги, онлайн-магазини, соціальні мережі.
SPA (Single Page Application)	React, Angular, Vue.js, Svelte, Ember.js	Односторінкова структура. Асинхронне оновлення контенту. Підвищена швидкість роботи.	Gmail, Trello, Google Docs.
PWA (Progressive Web Application)	Workbox, Service Workers, IndexedDB, Web App Manifest	Робота офлайн. Інтеграція з пристроями користувача. Можливість встановлення як додатка.	Spotify Web, Pinterest, Uber Lite.
Монолітні	PHP, ASP.NET, Python (Django), Java	Єдина структура. Легкість у розробці. Складність у масштабуванні.	Корпоративні вебсайти, CRM.
Мікросервісні	Docker, Kubernetes, REST API, gRPC, RabbitMQ	Розподілена архітектура. Легка масштабованість. Кожен компонент незалежний.	Netflix, Amazon, Uber.
Безсерверні	AWS Lambda, Google Functions, Firebase, Azure Functions	Логіка обробки подій у хмарі. Оплата за використання. автоматичне масштабування.	Event-driven системи, API-шлюзи.
Для настільних браузерів	HTML, CSS, JavaScript, WebAssembly, Electron	Оптимізація під десктопні екрани. Можливість використання локальних ресурсів.	Figma, Visual Studio Code.
Мобільні	React Native, Flutter Web, PWA	Оперативність і адаптивність. Інтеграція з мобільними пристроями.	WhatsApp Web, мобільні версії e-commerce сайтів.
Кросплатформенні	React Native, Flutter, Ionic	Єдиний код для різних платформ. Легка підтримка та розвиток.	Facebook, Twitter Lite, Airbnb.

Фреймворки забезпечують зручність розробки за рахунок стандартизації процесів. Таблиця 1.5 систематизує основні відомості про фреймворки для фронтенду та бекенду вебзастосунків.

Таблиця 1.5 – Популярні фреймворки для побудови вебзастосунків

Тип фреймворка	Популярні фреймворки	Ключові особливості
Фронтенд	React, Angular, Vue.js	Компонентна структура, покращений UX/UI.
Бекенд	Django, Flask (Python), Express.js (Node.js),	Швидка розробка API, інтеграція з базами даних. Підтримка RESTful API.

Фронтенд-фреймворки:

- React – популярний фреймворк від Facebook, що базується на компонентній архітектурі, дозволяє створювати динамічний інтерфейс користувача з мінімальними оновленнями DOM;
- Angular – повноцінний фронтенд-фреймворк від Google з підтримкою двостороннього зв'язування даних, найкраще підходить для великих вебпроектів з високою інтерактивністю;
- Vue.js – легкий у вивченні фреймворк з гнучкою структурою, що поєднує переваги React і Angular, забезпечує високу продуктивність та ефективне управління компонентами вебзастосунку.

Головними особливостями використання фронтенд-фреймворків є: компонентна структура, висока швидкість розробки та можливість побудови зручного UX/UI.

Бекенд-фреймворки:

- Django (Python) – потужний фреймворк з підтримкою швидкої розробки та вбудованими інструментами для інтеграції з базами даних, підходить для складних проектів, які потребують масштабування;
- Flask (Python) – легкий, мінімалістичний фреймворк, дозволяє створювати RESTful API для невеликих або середніх проектів;
- Express.js (Node.js) – гнучкий фреймворк для створення REST API на базі JavaScript, характеризується простотою та високою продуктивністю для обробки запитів у реальному часі.

Головними особливостями бекенд-фреймворків є те, що вони забезпечують швидке створення API, інтеграцію з базами даних і ефективну обробку серверних запитів. Загалом, фреймворки для фронтенду та бекенду спрощують процес розробки вебзастосунків завдяки своїм особливостям, таким як компонентна архітектура, швидка інтеграція API та підтримка сучасних стандартів веброзробки. Вибір фреймворка залежить від складності проекту, потреб масштабування та вимог до UX/UI.

Діаграма взаємодії клієнтської та серверної частини вебзастосунків представлена на рисунку 1.2 демонструє чіткий потік даних між клієнтською частиною, сервером і базою даних, що є основою роботи вебзастосунків у архітектурі клієнт-сервер.

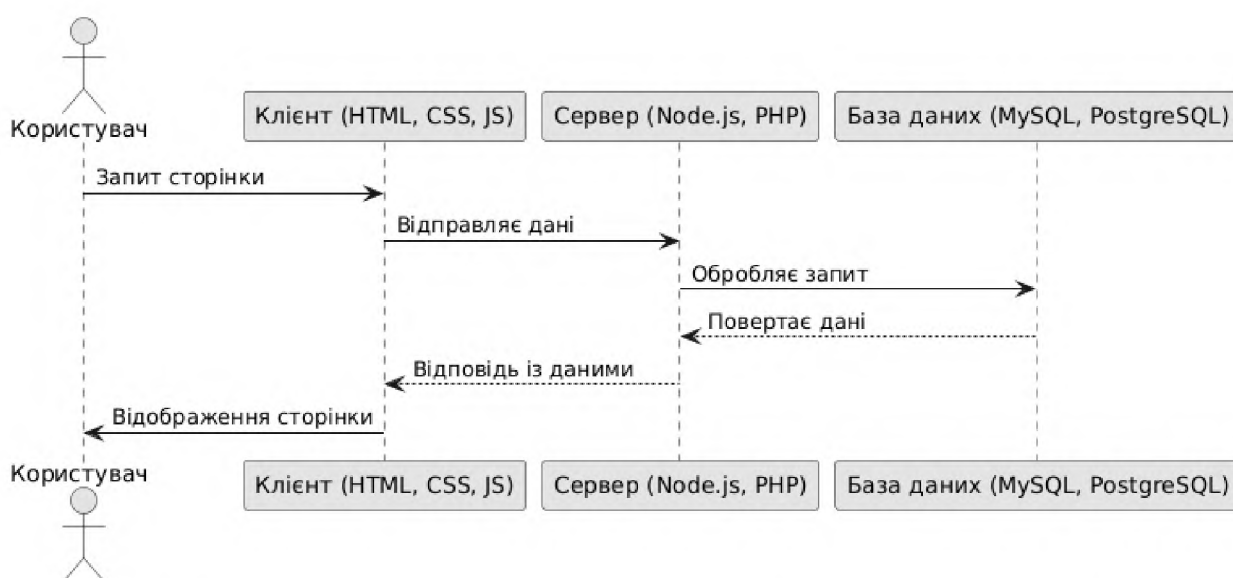


Рисунок 1.2 – Діаграма взаємодії клієнтської та серверної частини вебзастосунків

Діаграма на рисунку 1.2 відображає ключові етапи взаємодії фронтенду (клієнтської частини), бекенду (серверної частини) та бази даних у процесі роботи вебзастосунку. Основна мета – демонстрація послідовності обміну даними між компонентами. Основні компоненти діаграми:

- клієнтська частина (Клієнт) реалізована за допомогою HTML, CSS, JavaScript та фронтенд-фреймворків (React, Angular, Vue.js), відповідає за відображення інтерфейсу користувача та надсилання запитів до сервера;

- серверна частина (Сервер) виконує обробку запитів, бізнес-логіку та інтеграцію з базою даних, реалізується з використанням фреймворків, таких як Django, Flask, Express.js;

- база даних (База даних) – сховище даних вебзастосунку, що підтримує роботу з SQL (MySQL, PostgreSQL) або NoSQL (MongoDB), використовується для зберігання й отримання інформації відповідно до запитів від сервера.

Послідовність взаємодії:

- запит клієнта – користувач взаємодіє з вебзастосунком через інтерфейс на фронтенді (наприклад, натискає кнопку, заповнює форму);

- HTTP-запит до сервера – клієнтська частина надсилає HTTP-запит до серверної частини за допомогою REST API або GraphQL;

- обробка на сервері – бекенд обробляє запит, виконує необхідну бізнес-логіку та формує звернення до бази даних;

- запит до бази даних – сервер звертається до бази даних для отримання, збереження або оновлення даних;

- повернення відповіді – база даних надсилає потрібні дані серверу;

- формування відповіді сервером – бекенд обробляє отриману інформацію та формує HTTP-відповідь у форматі JSON або HTML;

- відображення на клієнті – клієнтська частина отримує відповідь та оновлює інтерфейс користувача відповідно до результатів (наприклад, показує завантажені дані або повідомлення про успішну операцію).

HTML – основа будь-якого вебзастосунку, що означає структуру сторінки за допомогою тегів, таких як <div>, <h1> та <p>. HTML є базовою технологією, без якої неможливо створити жоден вебінтерфейс.

CSS – відповідає за візуальне оформлення вебсторінок. Використання допоміжних CSS-технологій, таких як Flexbox та Grid, дозволяє реалізувати адаптивний дизайн, що є важливим для коректного відображення інтерфейсу на різних пристроях. Додатково застосування CSS-препроцесорів (SASS) забезпечує зручність та гнучкість у написанні стилів.

JavaScript – забезпечує динамічність сторінок та взаємодію користувача з інтерфейсом. Основні можливості JavaScript включають маніпуляції з об'єктами DOM для зміни елементів сторінки в реальному часі та підтримку Fetch API для асинхронного отримання даних із серверу без перезавантаження сторінки.

Систематизовані відомості про базові технології вебпрограмування, їх роль у процесі веброзробки та приклади використання, що дозволяє чітко розділити завдання кожної технології та зрозуміти її значення для побудови сучасних вебзастосунків представлені у таблиці 1.6.

Таблиця 1.6 – Зведені відомості щодо основних технологій вебпрограмування

Технологія	Роль	Приклади
HTML	Структура сторінки	Теги <div>, <h1>, <p>.
CSS	Оформлення сторінки	Flexbox, Grid, SASS, адаптивність.
JavaScript	Динаміка сторінки	DOM-маніпуляції, Fetch API.
PHP	Серверна обробка	Взаємодія з MySQL, рендеринг HTML.
Node.js	Асинхронна обробка запитів	WebSocket, REST API.
React	Фронтенд-фреймворк	Компонентна структура.
Django	Бекенд-фреймворк	Розробка RESTful API.

PHP – є популярною серверною мовою програмування. Основні завдання, що, зазвичай, реалізуються на PHP включають рендеринг HTML-коду та взаємодію з базами даних, такими як MySQL. PHP часто використовується для обробки запитів користувача та створення динамічного контенту.

Node.js – серверна технологія на основі JavaScript, що дозволяє обробляти асинхронні запити та створювати RESTful API. Підтримує використання протоколу WebSocket, який забезпечує швидку передачу даних у реальному часі, що є важливим для динамічних застосунків.

React – фронтенд-фреймворк, який базується на компонентній структурі. Він дозволяє створювати багаторазові компоненти інтерфейсу, що спрощує розробку складних вебзастосунків із великою кількістю інтерактивних елементів.

Django – потужний бекенд-фреймворк, орієнтований на швидку розробку. Забезпечує зручну розробку RESTful API для взаємодії фронтенду з сервером та підтримує інтеграцію з базами даних.

Стек основних технологій веброзробки систематизовано на діаграмі, зображеній на рисунку 1.3.

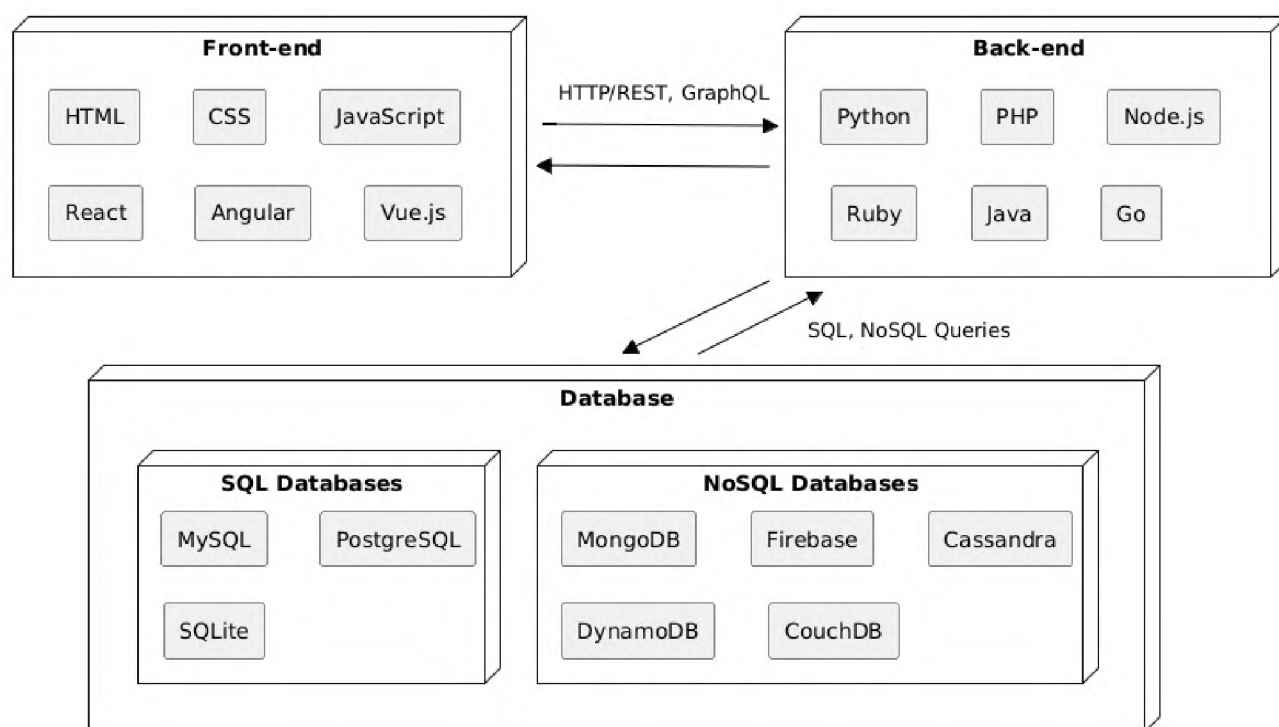


Рисунок 1.3 – Стек технологій веброзробки

Отже, до основних технологій веброзробки належать: HTML – забезпечує структуру вебдокументів, CSS – відповідає за їх візуальне оформлення, JavaScript – додає динаміку на стороні клієнта, серверні мови та фреймворки (PHP, Node.js, Django) – реалізують логіку на стороні сервера. Зокрема, React є прикладом фронтенд-фреймворку, що забезпечує ефективне створення інтерфейсу. Усі ці технології працюють у комплексі для розробки функціональних, гнучких і масштабованих вебзастосунків.

1.2 Архітектура вебзастосунків

1.2.1 Компоненти вебзастосунків

Сучасні вебзастосунки побудовані на трирівневій архітектурі [48, 49]:

- front-end – клієнтська частина, де користувач взаємодіє з інтерфейсом;
- back-end – обробляє дані, відповідає за логіку застосунку та взаємодіє з базою даних;
- база даних – забезпечує зберігання інформації та підтримує доступ до необхідних ресурсів.

Front-end відповідає за інтерфейс взаємодії користувача з вебзастосунком, використовує технології HTML, CSS і JavaScript, фреймворки React, Angular, Vue.js, забезпечує взаємодію з сервером через API.

Back-end обробляє запити від front-end, виконує бізнес-логіку, взаємодіє з базою даних, забезпечує обробку даних, безпеку і масштабованість, використовує мови Python (Django, Flask), JavaScript (Node.js), PHP.

База даних відповідає за зберігання та управління даними, використовує SQL-бази (MySQL, PostgreSQL) та NoSQL-бази (MongoDB), використовується ORM (Object-Relational Mapping) для інтеграції з бекендом.

Ця архітектура забезпечує чітке розділення, що дозволяє ефективно масштабувати вебзастосунки, підтримувати їх продуктивність і забезпечувати гнучкість розробки. Взаємодія між компонентами архітектури відбувається через API, що є важливою частиною інтеграції в веброзробці [50-52].

Основні відомості про компоненти вебзастосунків узагальнені у таблиці 1.7.

Таблиця 1.7 – Основні відомості про компоненти вебзастосунків

Компонент	Роль	Приклади технологій
Front-end	Інтерфейс користувача	HTML, CSS, JavaScript, React
Back-end	Обробка запитів, бізнес-логіка	Python (Django), Node.js, PHP
База даних	Зберігання даних	MySQL, PostgreSQL, MongoDB

Таблиця 1.5 узагальнює ключові компоненти архітектури вебзастосунків, їх ролі та технології, що використовуються для їх реалізації. Ця структура

відображає принцип багаторівневої архітектури (front-end, back-end, база даних), яка є основою сучасних вебзастосунків.

Front-end (інтерфейс користувача) відповідає за візуальне відображення даних та взаємодію користувача із застосунком, використовує технології, такі як HTML (структура сторінки), CSS (стилі), JavaScript (логіка та динамічність), забезпечують створення зручного і функціонального інтерфейсу. Використання React дозволяє будувати компонентно-орієнтовані інтерфейси, що підвищують продуктивність і зручність розробки. Взаємодія front-end з back-end відбувається через API (RESTful або GraphQL) [53, 54].

Back-end (серверна частина) реалізує бізнес-логіку, обробляє HTTP-запити від клієнта та відповідає за інтеграцію з базами даних. Використовує технології, такі як Python (Django), Node.js, PHP, що забезпечують обробку запитів, генерацію відповідей і виконання складних операцій на сервері. Back-end відіграє ключову роль у забезпеченні функціональності вебзастосунку, включаючи перевірку автентифікації, управління ресурсами та обробку даних.

База даних (зберігання даних) відповідає за сховище структурованих або неструктурованих даних вебзастосунку. Реляційні бази даних, такі як MySQL та PostgreSQL, використовуються для структурованих даних з чіткими зв'язками. MongoDB, як NoSQL-рішення, забезпечує гнучке зберігання даних у документній моделі, що корисно для динамічних і масштабованих проєктів [55-59]. Бекенд отримує дані з бази, обробляє їх і надсилає до фронтенду для відображення користувачу (рисунок 1.4).

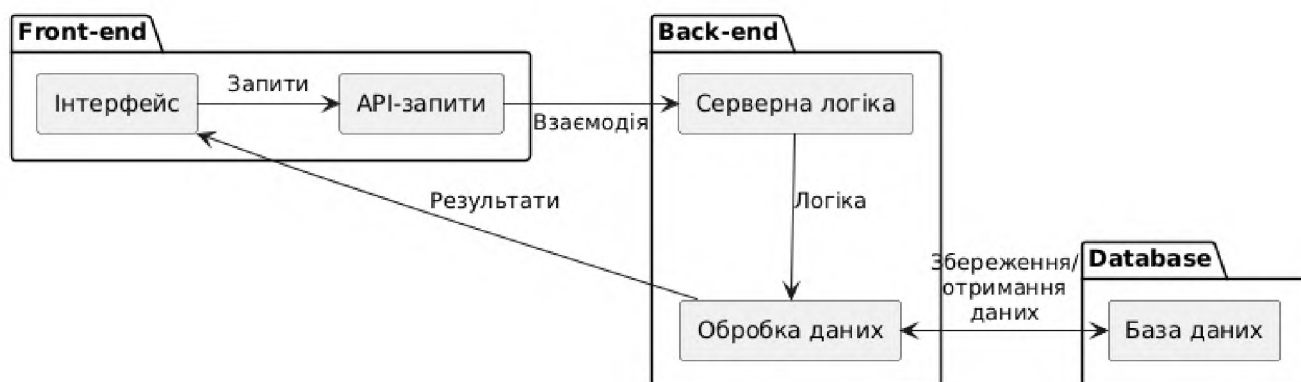


Рисунок 1.4 – Діаграма взаємодії основних компонентів вебзастосунку

Діаграма взаємодії основних компонентів вебзастосунки відображає чіткий потік даних між ключовими складовими: front-end (клієнтська частина), back-end (серверна частина) та базою даних. Діаграма демонструє логічну структуру взаємодії між компонентами вебзастосунку у клієнт-серверній архітектурі, і показує, як чітке розділення функцій забезпечує гнучкість, масштабованість і підтримку сучасних вебрішень.

Front-end (клієнтська частина) – ініціює HTTP-запити до сервера через API для отримання або надсилання даних, відображає результати у зручному інтерфейсі користувача, використовуючи HTML, CSS, JavaScript та фреймворки (React, Angular, Vue.js).

Back-end (серверна частина) – обробляє запити клієнта, виконує бізнес-логіку та взаємодіє з базою даних, відповідає за автентифікацію, авторизацію, обробку даних та формування відповіді у форматі JSON або HTML.

База даних – зберігає, оновлює та надає дані на запит сервера. Взаємодія з базою даних реалізується через SQL (MySQL, PostgreSQL) або NoSQL (MongoDB) технології.

Послідовність взаємодії: front-end надсилає запит; back-end обробляє його і звертається до бази даних; отримані дані передаються назад до front-end для відображення користувачу.

1.2.2 API та інтеграція між компонентами

API (Application Programming Interface) – це набір правил та інструментів, що забезпечують інтеграцію між компонентами. API у розробці вебзастосунків виконує такі основні функції:

- забезпечує обмін даними через запити між front-end і back-end;
- забезпечує стандартизацію обміну даними (JSON, XML).

Відрізняють три основні види API, кожен із яких має свої особливості та сферу застосування в сучасній веброзробці:

- RESTful API – це стандартизований підхід до побудови API, що використовує протокол HTTP та його методи: GET (отримання даних), POST (створення), PUT (оновлення) та DELETE (видалення). RESTful API є простим у

використанні, легко інтегрується з різними технологіями й широко застосовується для створення вебсервісів завдяки своїй ефективності та доступності. Підходить для сценаріїв, де основне завдання – передача структурованих даних між клієнтом і сервером [53];

- GraphQL – це гнучкий підхід до взаємодії з API, який дозволяє клієнту вибрати тільки ті дані, які йому потрібні. На відміну від RESTful API, де сервер відправляє фіксований набір даних, у GraphQL запити більш конкретизовані й оптимізовані, що знижує обсяг переданих даних. Підходить для складних систем із великою кількістю взаємопов'язаних даних та мобільних застосунків, де важливо мінімізувати кількість запитів і зекономити трафік [54];

- WebSocket API – це протокол для двонапрямленої передачі даних у реальному часі між клієнтом і сервером. На відміну від RESTful API, який працює за схемою «запит-відповідь», WebSocket забезпечує постійне з'єднання для обміну даними. Використовується в застосунках, де важлива швидка передача оновлень, наприклад, у чатах, біржових платформах та онлайн-іграх [60].

Кожен вид API має свої переваги:

- RESTful API – простий і універсальний для більшості вебсервісів;
- GraphQL – забезпечує гнучкість і оптимізацію передачі даних;
- WebSocket API – найкраще для обміну даними у режимі реального часу та двосторонньої комунікації.

Вибір виду API залежить від вимог до функціональності, продуктивності та архітектури вебзастосунку.

Таблиця 1.8 систематизує основні методи API, які є фундаментальними для роботи вебзастосунків у рамках RESTful архітектури.

Таблиця 1.8 – Основні методи API та їх призначення

Метод	Призначення	Приклад
GET	Отримання даних	Отримати список користувачів
POST	Створення нового ресурсу	Додати нового користувача
PUT	Оновлення існуючого ресурсу	Змінити інформацію користувача
DELETE	Видалення ресурсу	Видалити користувача

Методи GET, POST, PUT і DELETE є основою для взаємодії між клієнтом та сервером через API. Вони відповідають основним операціям CRUD (Create, Read, Update, Delete), що забезпечують повний цикл роботи з даними у вебзастосунках. Використання цих методів дозволяє будувати стандартизовані та ефективні RESTful API, які є надзвичайно важливими у сучасній веброзробці:

- GET – використовується для отримання даних із сервера. Метод є безпечним і не змінює стан ресурсу. Наприклад, запит для отримання списку користувачів передає дані у форматі JSON або XML [61-65];
- POST – призначений для створення нового ресурсу на сервері. Використовується для надсилання даних, що будуть збережені або оброблені сервером, наприклад, додавання нового користувача до бази даних;
- PUT – використовується для оновлення існуючого ресурсу. Замінює поточні дані ресурсу новими, наприклад, зміна інформації користувача в системі;
- DELETE – призначений для видалення ресурсу з сервера. Прикладом є видалення користувача або елемента даних із бази.

Діаграма інтеграції API відображає взаємодію між основними компонентами системи, які використовують API для обміну даними. Це дозволяє визначити структуру запитів, послідовність передачі інформації та роль кожного компонента у системі.

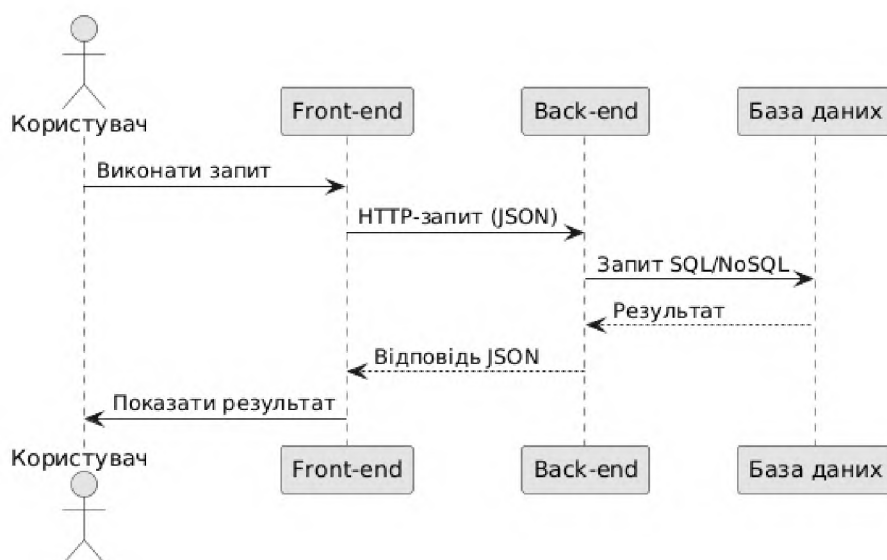


Рисунок 1.5 – Інтеграція API у вебзастосунку

Послідовність взаємодії: front-end ініціює запит (наприклад, метод GET для отримання даних); API передає запит на сервер; сервер обробляє запит і взаємодіє з базою даних для отримання необхідної інформації.; база даних повертає результат серверу; сервер через API надсилає відповідь front-end, де вона обробляється і відображається користувачу.

Основна функція API – забезпечити стандартизовану комунікацію між різними частинами вебзастосунку. Діаграма на рисунку 1.5 наочно демонструє співпрацю клієнта, сервера та бази даних у межах єдиної архітектури, де API виступає ключовим інструментом для обміну інформацією та інтеграції компонентів вебзастосунку. Це дозволяє ефективно розподілити ролі та полегшує масштабування системи.

1.2.3 Види архітектури вебзастосунків

Монолітна архітектура – усі компоненти (front-end, back-end, база даних) інтегровані в єдину систему. Переваги: простота розробки, легкість тестування. Недоліки: складно масштабувати, будь-які зміни впливають на всю систему.

Мікросервісна архітектура – система розділена на окремі сервіси, що працюють незалежно; кожен сервіс виконує свою задачу (автентифікація, обробка даних). Переваги: легше масштабувати, швидше оновлювати окремі сервіси. Недоліки: складна інтеграція, потребує оркестрації (наприклад, Kubernetes).

Безсерверна архітектура – серверна логіка виконується у хмарі (AWS Lambda, Azure Functions). Переваги: зниження витрат, автоматичне масштабування. Недоліки: залежність від хмарових провайдерів, обмеження продуктивності.

Монолітна архітектура проста у реалізації, легко тестується й розгортається, підходить для невеликих проєктів з обмеженим функціоналом, але має обмежену масштабованість та складність у підтримці великих систем, оскільки оновлення вимагає перезапуску всього застосунку. Прикладами використання є невеликі вебзастосунки або стартапи, де швидкість розробки є критичною.

Мікросервісна архітектура має високу масштабованість, можливість незалежного оновлення й розробки окремих сервісів. Підходить для великих і

складних систем, що потребують гнучкості. Обмеженнями є складність інтеграції між сервісами, підвищені витрати на підтримку та необхідність у професійних командах для налаштування. Застосовується у великих корпораціях, таких як Netflix і Amazon, де важливі стабільність та продуктивність системи.

Безсерверна архітектура мінімізує витрати на інфраструктуру, оскільки ресурси надаються провайдером за принципом «оплата за використання», що підходить для проєктів із непередбачуваним навантаженням. Обмеженнями є залежність від хмарового провайдера та обмеження функціоналу, які накладаються конкретною платформою. Приклади використання: AWS Lambda та Google Cloud, де серверна логіка виконується за потреби, забезпечуючи високу ефективність.

Таблиця 1.9 систематизує та порівнює відомості про три основні архітектури вебзастосунків.

Таблиця 1.9 – Порівняння архітектур вебзастосунків

Архітектура	Переваги	Недоліки	Приклади
Монолітна	Простота розробки, легкість тестування	Складно масштабувати, оновлення всієї системи	Невеликі проєкти
Мікросервісна	Масштабованість, швидке оновлення сервісів	Складність інтеграції, висока вартість підтримки	Netflix, Amazon
Безсерверна	Мінімальні витрати на інфраструктуру	Залежність від провайдера, обмеження функціоналу	AWS Lambda, Google Cloud

Приклад мікросервісної архітектури вебзастосунку ілюструє діаграма на рисунку 1.6.



Рисунок 1.6 – Приклад мікросервісної архітектури вебзастосунку

Кожна архітектура має свої переваги і недоліки, що визначають доцільність їїнього використання залежно від масштабів проєкту та його вимог: вибір архітектури вебзастосунку залежить від масштабів проєкту, ресурсних можливостей та вимог до продуктивності.

У контексті даної роботи мікросервісна та безсерверна архітектури є найбільш актуальними, оскільки вони забезпечують гнучкість, масштабованість та оптимізацію витрат, що є критично важливим при використанні хмарових технологій.

1.3 Огляд хмарових технологій у веброзробці

1.3.1 Поняття хмарових обчислень і їх значення для веброзробки

Хмарові обчислення (Cloud Computing) – це модель надання обчислювальних ресурсів (серверів, сховищ даних, мереж, програм) через Інтернет, що забезпечує доступність і масштабованість. Дана модель забезпечує наступні можливості:

- гнучкість – ресурси автоматично адаптуються до змін у навантаженні;
- інтеграція – можливість використовувати готові сервіси для автентифікації, аналізу даних, обробки платежів тощо;
- доступність – доступ до ресурсів і даних з будь-якого місця в режимі 24/7;
- економія – відсутність потреби в закупівлі та підтримці фізичної інфраструктури.

На рисунку 1.7 відображено загальний процес хмарових обчислень для розуміння принципів роботи хмарових технологій та їх інтеграції у вебзастосунки. Діаграма показує взаємодію основних компонентів: користувачів, хмарової інфраструктури та хмарових сервісів, наочно ілюструє, як користувачі взаємодіють із хмаровою інфраструктурою для виконання задач. Хмарові

обчислення забезпечують гнучке масштабування, доступність ресурсів та оптимізацію витрат, що є ключовими перевагами для сучасних вебзастосунків.

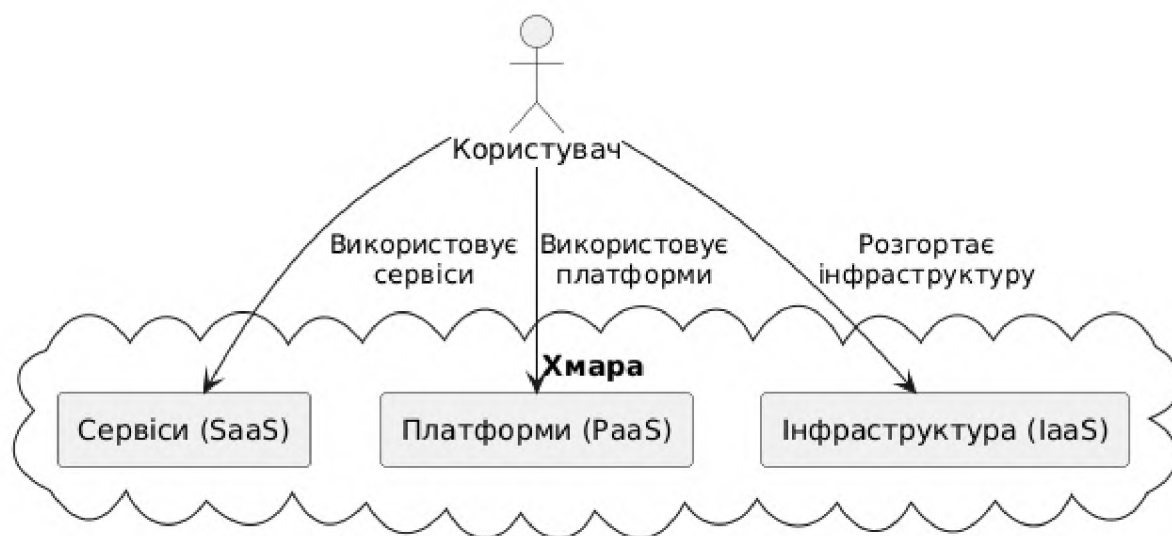


Рисунок 1.7 – Діаграма загального процесу хмарових обчислень

Основні етапи процесу хмарових обчислень:

- клієнт (Користувач) ініціює запит на виконання певної задачі чи доступу до ресурсу, використовує інтерфейси або програмні інструменти для взаємодії з хмаровою інфраструктурою;
- хмарова інфраструктура (Хмара) забезпечує віртуальні сервери, сховища даних та обчислювальні ресурси, які надаються хмаровим провайдером (наприклад, AWS, Google Cloud, Azure), автоматично обробляє запит користувача, розподіляючи ресурси за принципом оплати за використання;
- хмарові сервіси надають доступ до сервісів IaaS (інфраструктура як послуга), PaaS (платформа як послуга) та SaaS (програмне забезпечення як послуга). Сервіси обробляють дані, виконують операції та надсилають результат клієнту;
- зворотна передача результатів – оброблені дані або виконані задачі надсилаються користувачу через мережу Інтернет, користувач отримує результат у вигляді збережених файлів, оброблених запитів або відповіді на API-запит.

1.3.2 Типи хмарових рішень: IaaS, PaaS, SaaS

До основних типів хмарових рішень належать:

- IaaS (Infrastructure as a Service) – надає віртуальні сервери, сховища, мережі, використовується для побудови власних додатків. Приклади: Amazon EC2, Google Compute Engine;
- PaaS (Platform as a Service) – надає платформу для розробки, тестування та розгортання вебзастосунків, забезпечує інтеграцію з інструментами розробки. Приклади: Google App Engine, Heroku;
- SaaS (Software as a Service) – надає готові програмні рішення, доступні через браузер, не потребує розгортання інфраструктури. Приклади: Gmail, Google Docs, Dropbox.

Основні відомості про типи хмарових рішень узагальнює таблиця 1.10.

Таблиця 1.10 – Типи хмарових рішень (сервісів)

Тип	Що надається	Рівень контролю	Приклади
IaaS	Інфраструктура (сервери, ВМ)	Повний контроль	Amazon EC2, Microsoft Azure
PaaS	Платформа для розробки	Частковий контроль	Heroku, Google App Engine
SaaS	Готове програмне забезпечення	Мінімальний контроль	Gmail, Dropbox

Взаємодію основних типів хмарових рішень IaaS, PaaS та SaaS демонструє рисунок 1.8. Схема наочно демонструє ієрархічну структуру хмарових рішень і їх взаємозалежність, показує рівні абстракції та розподіл відповідальності між користувачем і хмаровим провайдером. Кожен рівень забезпечує певний набір можливостей та рівень відповідальності. Вибір між IaaS, PaaS та SaaS залежить від завдань користувача, ресурсів та технічних вимог проєкту.

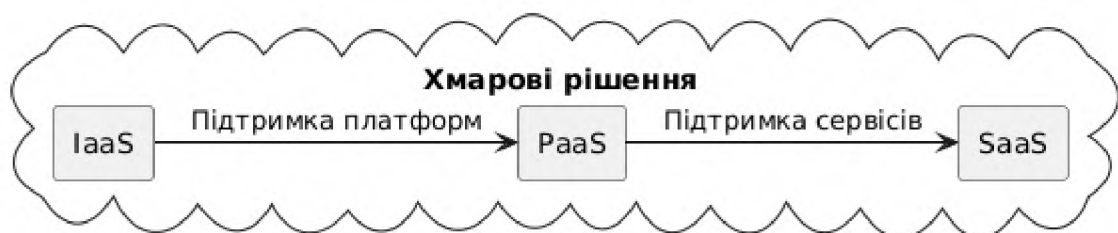


Рисунок 1.8 – Схема взаємодії хмарових рішень різних типів

Взаємодія між рівнями: IaaS є базовим рівнем, що надає ресурси для PaaS та SaaS; PaaS будується на основі IaaS, надаючи середовище для розробки; SaaS використовує можливості PaaS для кінцевого надання програмного продукту користувачу.

На рівні IaaS користувач отримує доступ до віртуалізованої інфраструктури (серверів, сховищ даних і мережевих ресурсів), користувач відповідає за розгортання операційної системи, програмного забезпечення та управління ресурсами. Приклади: Amazon EC2, Microsoft Azure Virtual Machines, Google Compute Engine.

На рівні PaaS надається готова платформа для розробки, тестування та розгортання за стосунків, користувач відповідає за код програми та його функціональність, тоді як інфраструктура та операційна система керуються провайдером. Приклади: AWS Elastic Beanstalk, Google App Engine, Microsoft Azure App Services.

На рівні SaaS користувач отримує доступ до готового програмного рішення через вебінтерфейс або API, провайдер послуг відповідає за інфраструктуру, платформу та програмне забезпечення, забезпечує його підтримку та оновлення. Приклади: Google Workspace, Microsoft Office 365, Salesforce, Dropbox.

1.3.3 Переваги хмарових сервісів у розробці вебзастосунків

Характеристики хмарових сервісів є важливими чинниками їх впровадження у різних галузях. Представлені характеристики показують, що хмарові технології сприяють оптимізації витрат, підвищенню продуктивності, забезпеченню доступності даних і розширенню можливостей інтеграції. Особливе значення мають аспекти безпеки, які є критично важливими для захисту інформації в сучасному цифровому середовищі [66-75].

Таблиця 1.11 наочно ілюструє ключові переваги хмарових сервісів, які забезпечують їх популярність у сучасній IT-інфраструктурі. Зниження витрат досягається завдяки моделі «оплати за використання», яка усуває необхідність утримання дорогого обладнання, як у випадку з AWS Lambda. Масштабованість дозволяє автоматично адаптувати ресурси до потреб проєкту, що особливо

корисно для динамічних систем, таких як Google Kubernetes Engine. Постійна доступність забезпечує безперервну роботу користувачів із даними незалежно від місця знаходження, як це реалізовано у Dropbox і Google Drive. Інтеграція через API спрощує з'єднання хмарових сервісів із зовнішніми системами, прикладом чого є Stripe і Twilio. Високий рівень безпеки, включаючи шифрування даних і управління доступом, гарантується завдяки таким інструментам, як AWS IAM і Google Cloud Security (таблиця 1.11).

Таблиця 1.11 – Основні переваги хмарових сервісів

Перевага	Опис	Приклади
Зниження витрат	Оплата лише за використані ресурси.	AWS Lambda (безсерверна обробка)
Масштабованість	Автоматичне масштабування в залежності від навантаження.	Google Kubernetes Engine
Доступність	Постійний доступ до даних з будь-якого місця.	Dropbox, Google Drive
Інтеграція	Легка інтеграція з іншими сервісами через API.	Stripe (платежі), Twilio (комунікація).
Безпека	Вбудовані механізми шифрування та управління доступом.	AWS IAM, Google Cloud Security

Діаграма процесу масштабування ресурсів з використанням хмарових обчислень представлена на рисунку 1.9.



Рисунок 1.9 – Діаграма процесу масштабування ресурсів з використанням хмарових обчислень

Діаграма 1.9 ілюструє автоматизований процес масштабування ресурсів у хмаровому середовищі, що забезпечує адаптацію системи до змін у навантаженні. Користувач (User) взаємодіє із додатком, створюючи підвищене навантаження, наприклад, під час пікового періоду роботи системи.

Додаток, який розміщений у хмаровій інфраструктурі, передає запит на масштабування до інфраструктури хмарового провайдера. Це може включати збільшення кількості віртуальних машин, підвищення обчислювальної потужності чи оптимізацію сховища даних. У відповідь інфраструктура забезпечує розширення ресурсів і передає оновлені параметри продуктивності назад до додатка.

Цей зворотний зв'язок дозволяє забезпечити збалансовану продуктивність додатка навіть під час значних навантажень, що критично важливо для підтримання якості обслуговування. Діаграма відображає ключову перевагу хмарових обчислень — автоматичне масштабування, яке оптимізує використання ресурсів і забезпечує стабільну роботу додатка.

Таким чином, хмарові технології забезпечують значні переваги для веброзробки, дозволяючи зменшувати витрати, забезпечувати масштабованість і доступність. Типи хмарових рішень (IaaS, PaaS, SaaS) дозволяють розробникам вибирати відповідний рівень інтеграції в залежності від потреб проекту. Візуалізації, такі як UML-діаграми та таблиці, дозволяють чітко уявити процеси та структуру хмарових рішень.

1.4 Хмарові платформи для розробки вебзастосунків

1.4.1 Огляд платформ AWS, Google Cloud Platform та Microsoft Azure

Amazon Web Services (AWS) – лідер ринку хмарових технологій. Що підтримує широкий спектр сервісів: обчислення (EC2), зберігання (S3), бази даних (RDS, DynamoDB), аналітику, штучний інтелект. Пропонує безсерверні

обчислення (AWS Lambda) та можливість автоматичного масштабування (Elastic Load Balancer, Auto Scaling) [66].

Google Cloud Platform (GCP) – має інтеграцію з іншими продуктами Google. Основні сервіси: обчислення (Compute Engine), бази даних (Cloud SQL, Firestore), штучний інтелект і машинне навчання (AI Platform). Особливості: BigQuery для аналізу великих даних, App Engine для безсерверних застосунків [67].

Microsoft Azure – популярний вибір серед корпоративних користувачів. Має широкий спектр інструментів для інтеграції з Windows Server, Active Directory. Основні сервіси: віртуальні машини (Virtual Machines), бази даних (Azure SQL Database), аналітика (Azure Synapse Analytics) [68-70].

Порівняння хмарових платформ представлено у таблиці 1.12.

Таблиця 1.12 – Порівняння хмарових платформ

Характеристика	AWS	Google Cloud Platform	Microsoft Azure
Ринок	Лідер ринку	Сильна аналітика	Корпоративний сегмент
Популярні сервіси	EC2, S3, RDS, Lambda	Compute Engine, BigQuery	Virtual Machines, Azure SQL
Інтеграція	Підтримка широкого спектру зовнішніх інструментів	Тісна інтеграція з Google	Інтеграція з Windows Server
Сфера використання	Різні бізнеси	Дані, AI	Корпоративні клієнти

Таблиця 1.10 демонструє ключові особливості трьох провідних хмарових платформ: AWS, Google Cloud Platform та Microsoft Azure. AWS є лідером ринку, пропонуючи широкий набір популярних сервісів, таких як EC2, S3 і Lambda, які підходять для різноманітних бізнес-задач. Google Cloud Platform відзначається потужними інструментами для роботи з даними та AI, як-от BigQuery, а також тісною інтеграцією з екосистемою Google, що робить її привабливою для задач аналітики та обробки даних. Microsoft Azure зосереджена на корпоративному сегменті завдяки інтеграції з Windows Server і популярним рішенням, як-от Virtual Machines і Azure SQL, що робить її оптимальним вибором для великих організацій. Таблиця підкреслює різницю у фокусі та спеціалізації кожної

платформи, що дозволяє вибрати оптимальне рішення залежно від потреб проекту.

Діаграма класифікації хмарових платформ представлена на рисунку 1.10.

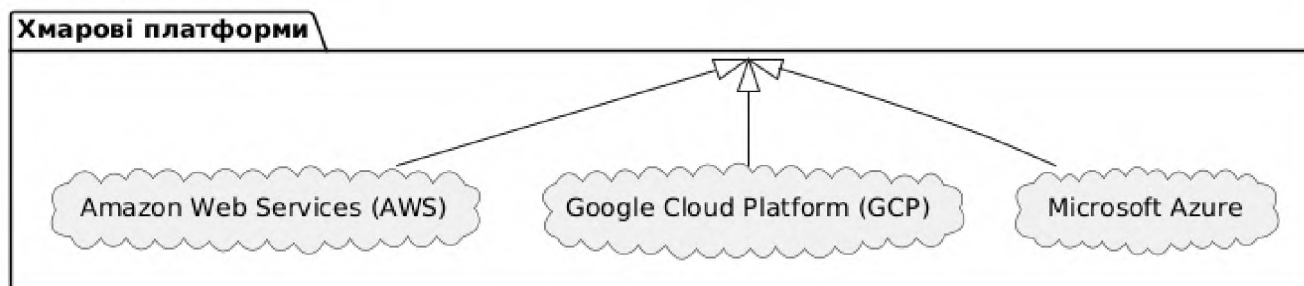


Рисунок 1.10 – Основні хмарові платформи

Діаграма відображає загальну класифікацію основних хмарових платформ, представлених трьома провідними провайдерами: Amazon Web Services (AWS), Google Cloud Platform (GCP) та Microsoft Azure. Вони належать до категорії «Хмарові платформи» та є окремими підкласами у цій ієрархії:

- AWS виступає найбільшим гравцем на ринку, надаючи широкий спектр послуг для різних бізнес-потреб;
- GCP спеціалізується на аналітиці даних та технологіях штучного інтелекту, пропонуючи рішення для обробки великих обсягів даних;
- Microsoft Azure орієнтована на корпоративний сегмент завдяки інтеграції з продуктами Microsoft і підтримці великих організацій.

Діаграма відображає, як усі три платформи входять до загальної категорії хмарових рішень, підкреслюючи їх унікальні особливості та цільові сегменти ринку. Ця класифікація є основою для аналізу функціональних можливостей і вибору оптимальної платформи залежно від потреб користувачів.

1.4.2 Хмарові інструменти веброзробки

Хмарові платформи AWS, Google Cloud Platform і Microsoft Azure надають широкий спектр інструментів для розробки вебзастосунків. Вибір платформи залежить від потреб проекту: AWS пропонує універсальні інструменти, GCP

відзначається сильними аналітичними сервісами, Microsoft Azure орієнтована на корпоративний сегмент.

Основні інструменти AWS:

- AWS Lambda – виконання коду без серверів;
- Amazon S3 – зберігання файлів;
- Amazon RDS – керовані SQL-бази даних;
- Elastic Beanstalk – автоматичне розгортання вебзастосунків.

Основні інструменти GCP:

- Google App Engine – створення безсерверних вебзастосунків;
- Firestore – NoSQL-база даних;
- BigQuery – сервіс аналітики великих даних;
- Kubernetes Engine – оркестрація контейнерів.

Основні інструменти Microsoft Azure:

- Azure App Service – розробка та хостинг вебзастосунків;
- Azure SQL Database – керована база даних;
- Azure Functions – виконання коду без серверів;
- Azure DevOps – інструменти CI/CD.

Використання цих платформ дозволяє підвищити продуктивність і забезпечити масштабованість вебзастосунків. Приклади хмарових інструментів для веброзробки, які пропонують ці платформи представлені у таблиці 1.13.

Таблиця 1.13 – Хмарові інструменти для веброзробки (приклади)

Платформа	Інструмент	Призначення
AWS	AWS Lambda	Безсерверна обробка
Google Cloud	Google App Engine	Хостинг вебзастосунків
Microsoft Azure	Azure Functions	Віддалене виконання коду
AWS	Elastic Beanstalk	Автоматичне розгортання
Google Cloud	Kubernetes Engine	Оркестрація контейнерів

Таблиця 1.11 демонструє різноманітні хмарові інструменти для веброзробки, які пропонують провідні хмарові платформи. AWS Lambda забезпечує можливості безсерверної обробки, що дозволяє виконувати код у

відповідь на події без необхідності управління інфраструктурою. Google App Engine є зручним рішенням для хостингу вебзастосунків, тоді як Azure Functions дозволяє виконувати код у хмарі для автоматизації задач і інтеграції систем. Elastic Beanstalk від AWS спрощує автоматичне розгортання вебзастосунків, а Kubernetes Engine від Google Cloud пропонує потужний інструмент для оркестрації контейнерів. Ці інструменти ілюструють, як хмарові платформи допомагають оптимізувати процеси розробки та підтримки вебзастосунків. Діаграма інтеграції хмарових інструментів представлена на рисунку 1.1.

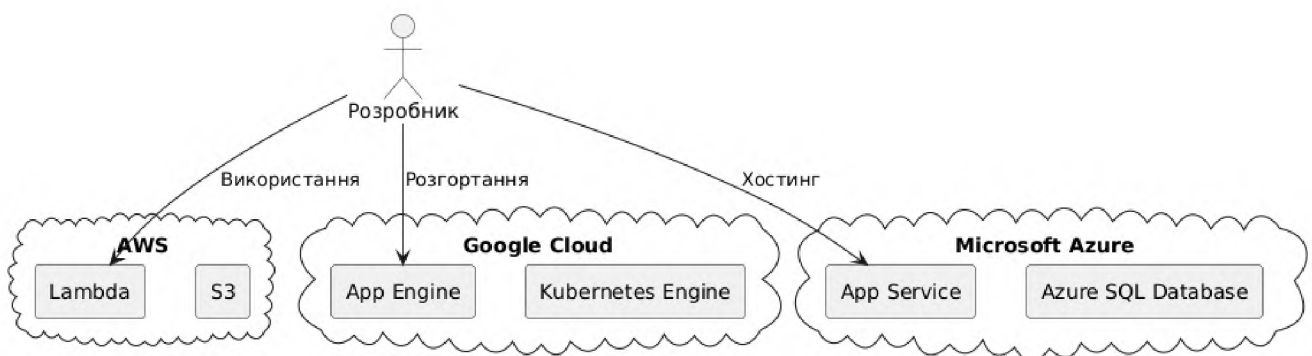


Рисунок 1.11 – Діаграма інтеграції хмарових інструментів

Основні відомості про хмарові платформи, що надають інструменти для розробки вебзастосунків, узагальнені у таблиці 1.14.

Таблиця 1.14 – Популярні хмарові платформи для розробки вебзастосунків

Платформа	Популярні сервіси	Переваги	Сфера використання
Amazon Web Services (AWS)	EC2, S3, Lambda, RDS, Elastic Beanstalk	Лідер ринку, широкий спектр сервісів, гнучкість, масштабованість	Універсальна для різних бізнесів
Google Cloud Platform (GCP)	Compute Engine, App Engine, BigQuery, Firestore	Сильна аналітика, інтеграція з продуктами Google, оптимізація для великих даних	Дані, AI, вебзастосунки
Microsoft Azure	Azure App Service, Azure SQL Database, Virtual Machines, Azure DevOps	Інтеграція з Windows Server, корпоративні рішення, підтримка DevOps	Корпоративний сегмент, CI/CD

Ця таблиця допомагає узагальнити ключову інформацію про платформи, їх переваги та основні сфери застосування.

1.5 Методології розробки вебзастосунків у хмаровому середовищі

1.5.1 Agile і DevOps у хмаровій розробці

Методологія Agile – це підхід до розробки, заснований на ітеративному процесі з частими релізами та адаптацією до змін. Особливості Agile в хмаровому середовищі: можливість швидкого розгортання кожної ітерації завдяки хмаровим платформам; підтримка командної роботи через інтеграцію інструментів керування проєктами (Jira, Trello) [71, 72].

Методологія DevOps – це культура та набір практик, що забезпечують співпрацю між командами розробки (Dev) і операцій (Ops). Роль у хмаровій розробці: автоматизація процесів (тестування, розгортання та моніторинг); використання хмарових інструментів для безперервного розгортання та масштабування [73-75].

Основна мета, особливості та інструменти Agile та DevOps представлені у таблиці 1.15.

Таблиця 1.15 – Порівняння Agile і DevOps

Методологія	Основна мета	Особливості	Інструменти
Agile	Гнучкість і швидкість	Ітеративний підхід, розподіл на спринти	Jira, Trello, Slack
DevOps	Автоматизація та інтеграція	CI/CD, спільна відповідальність, автоматизація	Jenkins, Docker, Kubernetes, Git

Таблиця 1.13 порівнює дві провідні методології розробки програмного забезпечення: Agile і DevOps, підкреслюючи їх основні цілі, особливості та інструменти. Agile спрямована на забезпечення гнучкості та швидкої адаптації до змін завдяки ітеративному підходу та роботі в рамках спринтів. Для управління процесами Agile широко використовуються інструменти, такі як Jira, Trello та

Slack, які спрощують комунікацію й координацію команд. DevOps орієнтована на автоматизацію процесів і інтеграцію розробки та операцій, забезпечуючи безперервне розгортання (CI/CD) і спільну відповідальність між розробниками й адміністраторами. У DevOps застосовуються потужні інструменти, як-от Jenkins для автоматизації, Docker для контейнеризації та Kubernetes для оркестрації. Таблиця ілюструє, як ці методології доповнюють одна одну, створюючи ефективні процеси розробки й підтримки програмного забезпечення.

1.5.2 Роль CI/CD для автоматизації процесів розгортання вебзастосунків у хмаровому середовищі

CI/CD (Continuous Integration/Continuous Delivery) – це підхід, що забезпечує автоматизацію інтеграції коду, тестування та розгортання. Має велике значення у хмаровій розробці [76-80].

Безперервна інтеграція (CI) забезпечує автоматичне тестування коду після кожного коміту, а також виявлення помилок на ранніх етапах. Основні інструменти CI: Jenkins та GitHub Actions.

Безперервне розгортання (CD) забезпечує автоматичне розгортання стабільної версії застосунку у хмарі, швидкий цикл оновлень. Основні інструменти CD: Docker, Kubernetes, AWS CodeDeploy.

Діаграма CI/CD у хмаровому середовищі представлена на рисунку 1.12.

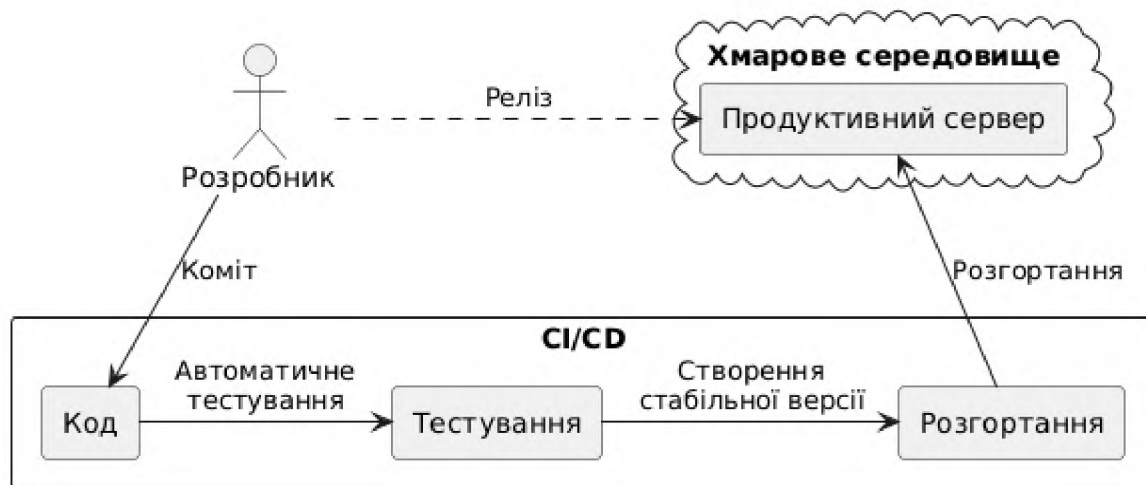


Рисунок 1.12 – Діаграма процесу CI/CD у хмаровому середовищі

Ця діаграма ілюструє процес автоматизації розробки, тестування та розгортання програмного забезпечення за допомогою CI/CD у хмаровому середовищі. Розробник починає процес із внесення змін у код і комітує їх у систему керування версіями. Далі код автоматично проходить стадію тестування, де перевіряється його якість та стабільність. Після успішного тестування створюється стабільна версія програмного забезпечення, яка автоматично переходить на етап розгортання. У хмаровому середовищі ця версія розгортається на продуктивному сервері, роблячи її доступною для кінцевих користувачів. Розробник завершує цикл процесу, контролюючи реліз та функціонування нової версії. Діаграма наочно демонструє переваги автоматизації CI/CD, які забезпечують безперервний розвиток і стабільну роботу програмного забезпечення в хмаровому середовищі.

Методології та підходи, такі як Agile, DevOps і CI/CD, є основою сучасної розробки програмного забезпечення, особливо у хмаровому середовищі. Їх використання дозволяє не лише підвищити ефективність командної роботи, але й забезпечити безперервний процес розробки, тестування і розгортання програмних рішень. У хмаровій розробці ці підходи надають особливі переваги, оскільки інтегруються з автоматизованими інструментами для масштабування, моніторингу та управління проєктами. Ключові аспекти цих методологій, їх цілі, переваги у хмаровій розробці та основні інструменти для реалізації. Узагальнено у таблиці 1.16.

Таблиця 1.16 – Ключові аспекти методологій розробки вебзастосунків

Методологія/підхід	Мета	Переваги у хмаровій розробці	Інструменти
Agile	Гнучка розробка	Швидке реагування на зміни, спринти, інтеграція команд	Jira, Trello
DevOps	Інтеграція Dev і Ops	Автоматизація, співпраця, моніторинг	Jenkins, Docker, Kubernetes
CI/CD	Автоматизація інтеграції і релізів	Швидкість релізів, зменшення ризику помилок	GitHub Actions, AWS CodeDeploy

Ця таблиця підсумовує роль методологій Agile і DevOps та підходу CI/CD у хмаровій розробці, демонструючи їх переваги та відповідні інструменти.

Висновки до розділу 1

У першому розділі розглянуто теоретико-методологічні аспекти використання хмарових технологій у розробці вебзастосунків. Основні висновки полягають у наступному:

- веброзробка є ключовою складовою сучасних інформаційних технологій, забезпечуючи створення інтерактивних та функціональних вебзастосунків;
- вебзастосунки класифікуються за рівнем динамічності (статичні, динамічні, SPA, PWA), що визначає сферу їх застосування;
- для реалізації вебзастосунків використовуються такі базові технології, як HTML для структури, CSS для оформлення, JavaScript для інтерактивності, а також серверні мови та фреймворки для обробки даних і взаємодії з базами даних;
- сучасні вебзастосунки використовують трирівневу архітектуру. Основними компонентами архітектури вебзастосунків є front-end (інтерфейс користувача), back-end (серверна логіка) та бази даних (зберігання даних);
- API є ключовим елементом інтеграції між компонентами, що забезпечує стандартизований обмін даними;
- архітектури вебзастосунків можуть бути монолітними, мікросервісними або безсерверними, кожна з яких має свої переваги та недоліки залежно від потреб проєкту.

Хмарові технології у веброзробці:

- хмарові обчислення забезпечують доступ до обчислювальних ресурсів через Інтернет, що підвищує гнучкість, масштабованість і доступність розробки;
- розрізняють три основні типи хмарових рішень: IaaS (інфраструктура як послуга), PaaS (платформа як послуга) та SaaS (програмне забезпечення як послуга), кожен з яких відповідає різним рівням потреб;

- використання хмарових сервісів значно знижує витрати, забезпечує автоматичне масштабування та інтеграцію з іншими сервісам.

Популярні хмарові платформи для веброзробки:

- найпопулярнішими платформами є AWS, Google Cloud Platform і Microsoft Azure, які надають широкий спектр інструментів для розробки, розгортання та підтримки вебзастосунків;
- інструменти, такі як AWS Lambda, Google App Engine і Azure Functions, дозволяють оптимізувати процеси розробки завдяки безсерверним технологіям;

Методології розробки у хмаровому середовищі:

- Agile забезпечує ітеративний підхід до розробки, дозволяючи швидко адаптуватися до змін;
- DevOps сприяє автоматизації та інтеграції процесів розробки і розгортання;
- використання CI/CD дозволяє автоматизувати процеси інтеграції, тестування та розгортання, забезпечуючи стабільність і швидкість релізів.

РОЗДІЛ 2

ЗАСТОСУВАННЯ ХМАРОВИХ ТЕХНОЛОГІЙ У РОЗРОБЦІ ВЕБЗАСТОСУНКІВ

2.1 Аналіз ефективності хмарових технологій у розробці вебзастосунків

2.1.1 Сучасні тенденції використання хмарових технологій

Сучасні тенденції використання хмарових технологій [81-90]:

- згідно з даними Gartner (2023), 67% організацій використовують хмарові платформи для веброзробки;
- статистика RightScale (Flexera) показує, що AWS залишається лідером ринку з часткою 32%, за ним йдуть Azure (21%) та Google Cloud (10%);
- близько 92% компаній використовують багатохмарний підхід (multi-cloud) для підвищення стійкості;
- використання безсерверних обчислень зростає на 15% щороку завдяки зниженню витрат і високій гнучкості.

Таблиця 2.1 – Статистика використання хмарових технологій

Показник	Дані
Рівень використання хмарових платформ	67% серед усіх компаній
Частка ринку AWS	32%
Зростання використання безсерверних рішень	15% щорічно
Частка багатохмарних стратегій	92%

Таблиця 2.1 демонструє ключові статистичні показники використання хмарових технологій у сучасному бізнесі, підкреслюючи їхню зростаючу популярність. Дані свідчать, що 67% компаній активно використовують хмарові платформи для реалізації своїх потреб, причому AWS утримує лідерську позицію з часткою ринку 32%. Зростання популярності безсерверних рішень на рівні 15% щороку вказує на переваги таких підходів, як економічна ефективність та гнучкість. Водночас 92% компаній застосовують багатохмарні стратегії для підвищення стійкості своїх систем і забезпечення безперервності бізнес-процесів.

Ці показники підкреслюють важливість хмарових технологій як невід’ємної частини сучасної IT-інфраструктури.

2.1.2 Аналіз продуктивності та витрат на розробку

Розглянемо порівняння продуктивності та витрат на розробку вебзастосунку з хмаровими сервісами та традиційними підходами.

Продуктивність хмарових сервісів забезпечує:

- швидке розгортання середовища розробки;
- автоматичне масштабування під час високих навантажень;
- інструменти для моніторингу (CloudWatch, BigQuery) забезпечують

стабільність.

Традиційні підходи характеризуються наступними показниками:

- тривалий час налаштування фізичної інфраструктури;
- обмежена масштабованість через залежність від серверного обладнання.

Характеристики витрат, пов’язані з використанням хмарових сервісів:

- оплата лише за використані ресурси (Pay-as-you-go);
- економія на утриманні фізичної інфраструктури;
- гнучкість тарифних планів (S3 Standard, S3 Infrequent Access).

Традиційні підходи:

- високі витрати на апаратне забезпечення та технічне обслуговування;
- витрати на персонал для підтримки серверів.

Таблиця 2.2 – Таблиця порівняння продуктивності та витрат

Критерій	Хмарові сервіси	Традиційні підходи
Час налаштування	Мінімальний	Тривалий
Масштабованість	Автоматична	Обмежена фізичною інфраструктурою
Витрати	Оплата за використані ресурси	Високі початкові витрати
Моніторинг	Вбудовані інструменти	Додаткове програмне забезпечення

Таблиця 2.2 порівнює продуктивність та витрати між хмаровими сервісами і традиційними підходами, відображаючи їх ключові переваги та обмеження. Хмарові сервіси забезпечують мінімальний час налаштування завдяки готовим до використання рішенням і автоматичному масштабуванню, що дозволяє легко

адаптуватися до змін у навантаженні. У той же час, традиційні підходи обмежені фізичною інфраструктурою, яка вимагає значних початкових витрат і часу на налаштування. Крім того, хмарові сервіси включають вбудовані інструменти моніторингу, що знижує витрати на додаткове програмне забезпечення, тоді як традиційні системи потребують окремих рішень для моніторингу. Це порівняння підкреслює економічну доцільність і технічну перевагу хмарових технологій у сучасній розробці.

У сучасній веброботці вибір між хмаровими технологіями та традиційними підходами значною мірою залежить від потреб проєкту, доступних ресурсів та вимог до гнучкості й масштабованості. Хмарові рішення пропонують значні переваги завдяки автоматизації, швидкому впровадженню та адаптивності до змін, тоді як традиційні підходи часто обмежені фізичною інфраструктурою і потребують більших інвестицій на початкових етапах. У таблиці 2.3 узагальнено основні відмінності між цими підходами, що дозволяє оцінити їхню доцільність залежно від конкретних умов і задач.

Таблиця 2.3 – Порівняння хмарових технологій і традиційних підходів до веброботки

Параметр	Хмарові технології	Традиційні підходи
Швидкість впровадження	Висока	Низька
Ефективність	Максимальна завдяки автоматизації	Середня через ручне керування
Витрати	Знижені завдяки оплаті за ресурси	Високі через витрати на обладнання
Масштабованість	Автоматична	Обмежена

Таким чином, хмарові технології демонструють значну перевагу над традиційними підходами в контексті продуктивності, гнучкості та економічності, що робить їх оптимальним вибором для сучасної веброботки.

2.2 Практичне використання хмарових платформ на прикладі вебзастосунку для керування завданнями

2.2.1 Обґрунтування вибору хмарової платформи

Для успішної реалізації сучасних вебзастосунків, орієнтованих на командну роботу та управління проектами, необхідно ретельно обирати хмарову платформу, яка забезпечить ефективність, надійність і масштабованість. Такий вибір залежить від особливостей самого застосунку, його функціональних вимог, цільової аудиторії та очікуваних навантажень. У цьому контексті застосунок TaskFlow, розроблений для командного управління завданнями, має чітко визначену мету та функціонал, що формує основу для обґрунтування вибору відповідної хмарової платформи.

TaskFlow – це вебзастосунок для командного управління завданнями, орієнтований на малий і середній бізнес. Він дозволяє створювати проекти, призначати завдання членам команди, відстежувати статус виконання, встановлювати дедлайни та отримувати сповіщення про важливі події.

Основні функції вебзастосунку:

- реєстрація та авторизація користувачів з використанням безпечної автентифікації (OAuth 2.0);
- управління завданнями: створення, редагування, видалення завдань, призначення виконавців, встановлення пріоритетів;
- дошка завдань – інтерактивний перегляд статусу завдань (наприклад, Kanban-дошка);
- Push-сповіщення для важливих подій (нові завдання, зміна статусу);
- API для інтеграції з календарями (Google Calendar) та сервісами (Slack).

Для реалізації TaskFlow вибір зроблено на користь AWS.

Обґрунтування вибору AWS:

- широкий спектр сервісів: AWS Lambda для реалізації безсерверної логіки (наприклад, обробка подій для сповіщень); Amazon S3 для зберігання статичних файлів (зображень, документів); Amazon RDS (Relational Database Service) для

керованої SQL-бази даних; Amazon SNS (Simple Notification Service) для сповіщень;

- автоматичне масштабування (Auto Scaling) дозволяє динамічно збільшувати ресурси при високому навантаженні;
- безпека: підтримка автентифікації через AWS Cognito; налаштування IAM (управління доступом) для безпеки даних;
- економічність – модель оплати «Pay-as-you-go» знижує витрати для проєкту на початкових етапах;
- інтеграція з API – API Gateway для управління запитами між front-end і back-end.

Порівняння хмарових платформ представлено у таблиці 2.4.

Таблиця 2.4 – Таблиця порівняння хмарових платформ

Критерій	AWS	Google Cloud Platform (GCP)	Microsoft Azure
Сервіси для безсерверних обчислень	Lambda	Cloud Functions	Azure Functions
Система сповіщень	Amazon SNS	Pub/Sub	Azure Notification Hubs
Сервіси баз даних	RDS, DynamoDB	Cloud SQL, Firestore	Azure SQL Database
Масштабування	Auto Scaling	Kubernetes Engine	Azure Autoscale
Цінова політика	Гнучка, «Pay-as-you-go»	Залежить від аналітичних потреб	Переважно для корпоративних клієнтів

Для забезпечення ефективної роботи вебзастосунку TaskFlow було обрано набір сервісів AWS, які відповідають основним функціональним вимогам застосунку, таким як обробка запитів, управління даними, зберігання файлів та сповіщення. Кожен із сервісів виконує свою чітко визначену роль, що дозволяє досягти високої продуктивності, гнучкості та масштабованості системи. UML-діаграма нижче наочно демонструє взаємодію між обраними сервісами AWS і користувачем. Діаграма обраних сервісів AWS для вебзастосунку TaskFlow представлена на рисунку 2.1.

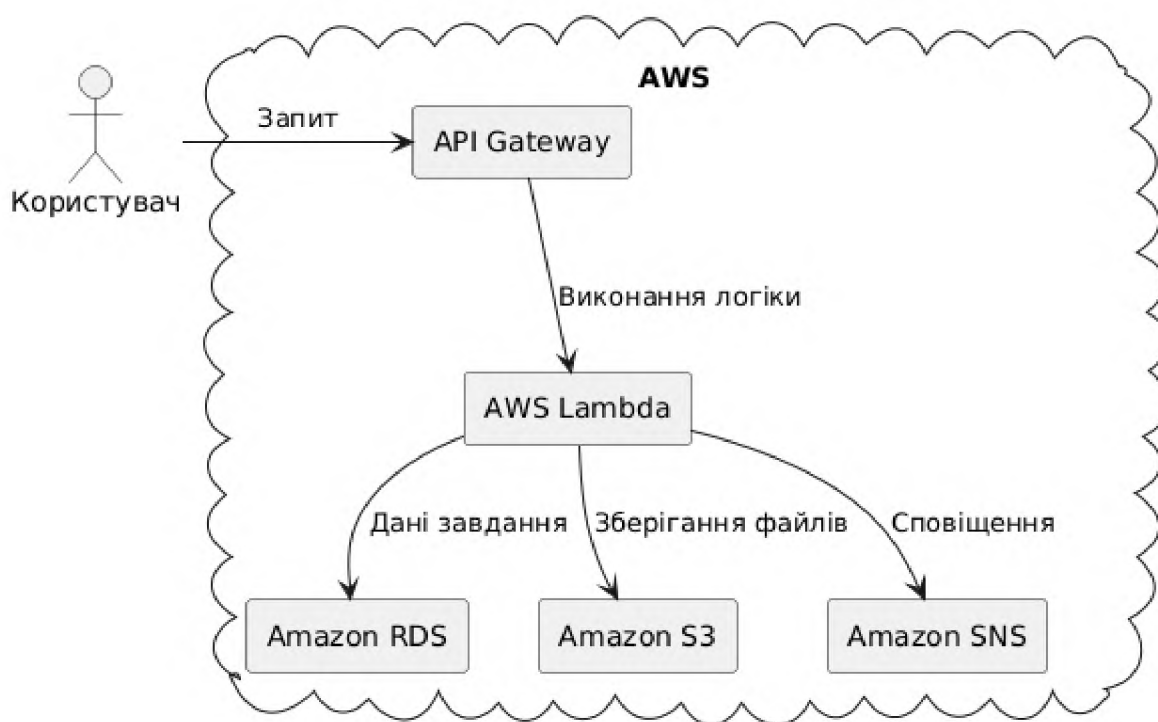


Рисунок 2.1 – Діаграма інтеграції сервісів AWS, обраних для вебзастосунку TaskFlow

Діаграма ілюструє, як користувач взаємодіє із застосунком через API Gateway, який перенаправляє запити на AWS Lambda для виконання бізнес-логіки. Lambda звертається до Amazon RDS для роботи з базою даних завдань, до Amazon S3 для зберігання файлів і до Amazon SNS для надсилання сповіщень. Така архітектура забезпечує ефективне виконання запитів, централізоване управління даними, зручне зберігання великих файлів і оперативну комунікацію з користувачами.

Таким чином, платформу AWS було обрано через її гнучкість, економічність і широкий спектр сервісів, необхідних для реалізації TaskFlow. Ця платформа забезпечує всеосяжну підтримку для створення, розгортання та масштабування вебзастосунків, що ідеально відповідає потребам проєкту.

2.2.2 Архітектура застосунку

Архітектура вебзастосунку TaskFlow базується на принципах розподіленої системи з інтеграцією хмарових сервісів.

Front-end:

- використовує React для створення інтерактивного інтерфейсу користувача;

- компоненти інтерфейсу: панель завдань (Kanban), календар завдань, форма реєстрації/авторизації;

- зв'язок із back-end через API.

Back-end:

- реалізовано на Node.js для обробки запитів і виконання бізнес-логіки;

- логіка управління завданнями (створення, оновлення, видалення);

- робота з хмаровою базою даних (Amazon RDS).

API:

- використовується AWS API Gateway для маршрутизації запитів між front-end і back-end;

- формат передачі даних: JSON;

- RESTful API підтримує стандартні методи: GET, POST, PUT, DELETE.

Контейнеризація:

- використовуються Docker-контейнери для ізольованого розгортання компонентів;

- оркестрація контейнерів через Kubernetes для автоматичного масштабування та моніторингу стану.

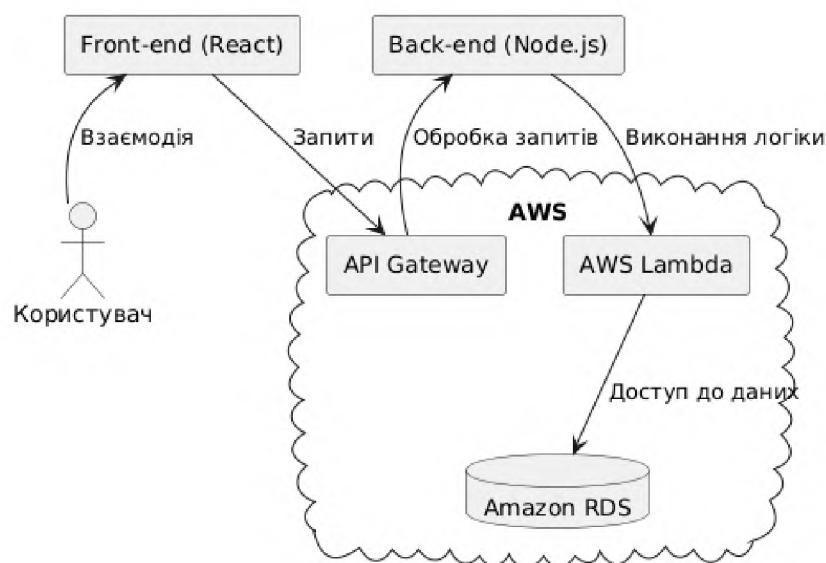


Рисунок 2.2 – Архітектура вебзастосунку TaskFlow

Для забезпечення повноцінної роботи вебзастосунку TaskFlow розроблена архітектура, яка поєднує фронтенд, бекенд і хмарові сервіси AWS (рисунок 2.2). Ця архітектура забезпечує ефективну взаємодію між користувачами та системою, спрощуючи обробку запитів, виконання бізнес-логіки та управління даними. UML-діаграма нижче відображає основні компоненти архітектури та їх взаємозв'язки.

Діаграма демонструє, як користувач взаємодіє з фронтендом, створеним на основі React, який обробляє введення даних і надсилає запити через API Gateway. Бекенд, побудований на Node.js, отримує ці запити для обробки і виконує бізнес-логіку через AWS Lambda. Lambda звертається до бази даних Amazon RDS для роботи з інформацією. Така архітектура дозволяє розділити обов'язки між компонентами, забезпечити гнучкість, продуктивність і масштабованість вебзастосунку.

2.2.3 Функціональні компоненти

Реєстрація/авторизація (OAuth) реалізована через AWS Cognito:

- користувачі можуть реєструватися через email або використовувати сторонні сервіси (Google, Facebook);
- користувач проходить автентифікацію при вході, після чого отримує доступ до функціоналу;
- забезпечення безпеки реалізоване через токени доступу (JWT).

Модуль управління завданнями:

- функціонал: створення нового завдання (введення назви, дедлайну, опису); призначення завдань учасникам команди; відстеження статусу завдання (нове, у прогресі, виконане);
- дані про завдання зберігаються у хмаровій базі (Amazon RDS);
- push-сповіщення про зміни статусу завдання – через Amazon SNS.

Хмарова база даних:

- використовується Amazon RDS (PostgreSQL): зберігання структурованих даних (користувачі, завдання, проєкти); надійність і автоматичне резервне копіювання;

- таблиці: users – зберігає інформацію про користувачів; tasks – інформація про завдання; projects – метадані проєктів (таблиця 2.5).

Таблиця 2.5 – Структура таблиць бази даних вебзастосунку

Назва таблиці	Поля таблиці
Users	id, name, email, password_hash
Tasks	id, title, description, status, deadline, project_id
Projects	id, name, created_at

Функціональні компоненти вебзастосунку TaskFlow спрямовані на забезпечення ключових операцій, необхідних для ефективного управління завданнями користувачів. Архітектура цих компонентів інтегрує реєстрацію, авторизацію, управління завданнями та роботу з базою даних. UML-діаграма нижче відображає основні функціональні частини TaskFlow та їхню взаємодію.

UML-діаграма діаграма функціональних компонентів вебзастосунку TaskFlow представлена на рисунку 2.3.

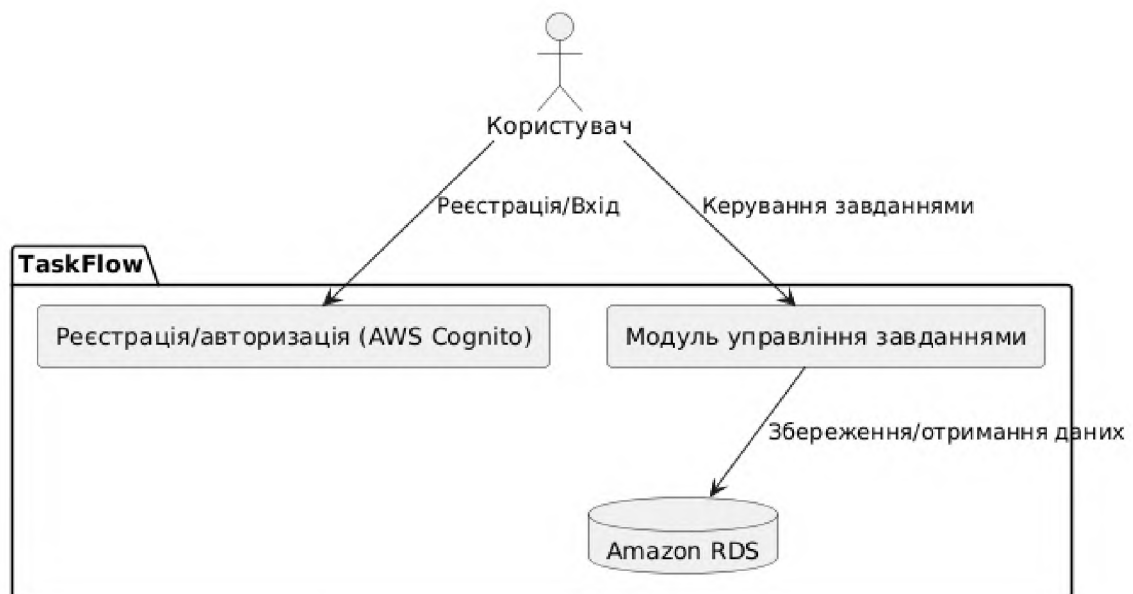


Рисунок 2.3 – Діаграма функціональних компонентів вебзастосунку TaskFlow

Діаграма показує, як користувач реєструється або авторизується через компонент реєстрації/авторизації, побудований на основі AWS Cognito. Після входу користувач може взаємодіяти з модулем управління завданнями, який

дозволяє створювати, оновлювати, видаляти або переглядати завдання. Цей модуль звертається до бази даних Amazon RDS для збереження або отримання відповідних даних. Така структура забезпечує безпечний доступ до системи, надійне управління даними і зручність для користувачів.

Таким чином, архітектура вебзастосунку TaskFlow побудована з використанням сучасних хмарових технологій, що забезпечують гнучкість, масштабованість і надійність. Функціональні компоненти, такі як реєстрація через OAuth, модуль управління завданнями та хмарова база даних, дозволяють створити зручний і продуктивний інструмент для командної роботи.

2.2.4 Практичні аспекти інтеграції

Практичні аспекти інтеграції вебзастосунку TaskFlow включають масштабування, моніторинг та push-сповіщення.

Масштабування реалізується за допомогою Auto Scaling в AWS:

- горизонтальне масштабування передбачає додавання нових інстанцій вебсерверів (наприклад, EC2) при збільшенні навантаження та зменшення кількості інстанцій у періоди низького навантаження;
- вертикальне масштабування передбачає динамічне збільшення обчислювальних ресурсів (CPU, RAM) для серверів при зростанні вимог до продуктивності.

Процес масштабування включає:

- моніторинг навантаження через Amazon CloudWatch;
- використання метрик (наприклад, CPU Utilization > 75%) для автоматичного запуску масштабування;
- балансування навантаження через Elastic Load Balancer (ELB).

Для моніторингу стану вебзастосунку використовується Amazon CloudWatch:

- ключові метрики: навантаження на процесор (CPU utilization); відсоток використання пам'яті; час відповіді API; кількість запитів на хвилину (Requests per Minute, RPM);

- функціональність: налаштування алертів (наприклад, при перевищенні часу відповіді API); візуалізація метрик у вигляді графіків і діаграм; логування подій через CloudWatch Logs для аналізу помилок.

Push-сповіщення забезпечують користувачам оперативну інформацію про важливі події в застосунку. Реалізація – через Amazon SNS (Simple Notification Service):

- сценарії використання: нагадування про дедлайн завдання; сповіщення про нові завдання або зміну їх статусу;

- процес роботи:

- подія (наприклад, створення нового завдання) генерує повідомлення;
- AWS Lambda обробляє подію та відправляє її через SNS;
- користувач отримує push-сповіщення на пристрій.

Таким чином, для забезпечення стабільності та ефективності роботи вебзастосунку TaskFlow в умовах змінного навантаження використовується інтеграція сервісів AWS для масштабування, моніторингу та надсилання push-сповіщень. Така інтеграція дозволяє автоматично регулювати кількість серверних інстансів, оперативно відстежувати стан системи та сповіщати користувачів про важливі події. UML-діаграма нижче демонструє взаємодію основних компонентів, які забезпечують ці процеси. Діаграма інтеграції масштабування, моніторингу та push-сповіщень представлена на рисунку 2.4.

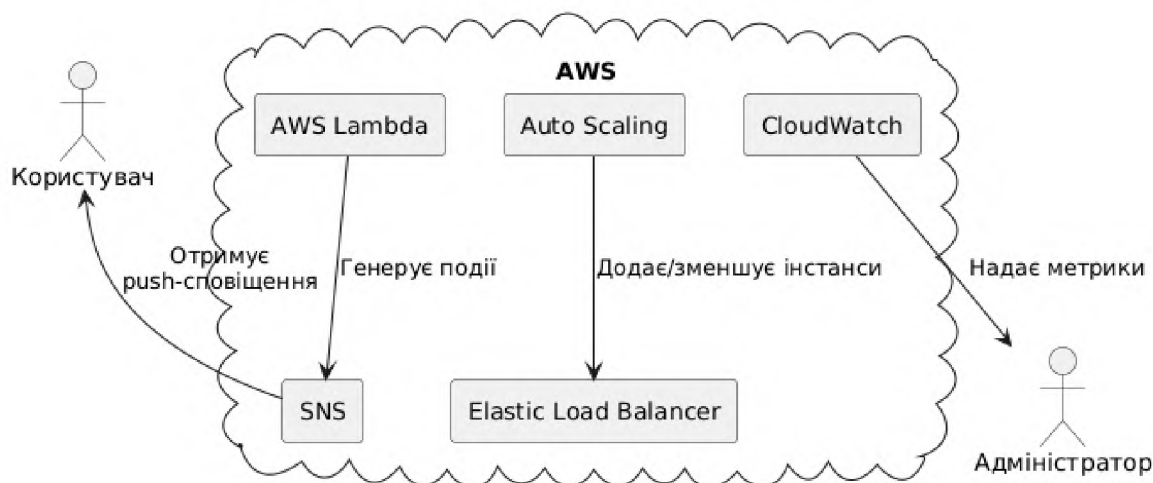


Рисунок 2.4 – Діаграма інтеграції масштабування, моніторингу та push-сповіщень

Діаграма ілюструє, як Auto Scaling через Elastic Load Balancer додає або зменшує кількість інстансів для підтримки необхідного рівня продуктивності. Amazon CloudWatch надає адміністратору метрики для моніторингу стану системи, а AWS Lambda генерує події, які через Amazon SNS транслиються користувачам у вигляді push-сповіщень. Це рішення забезпечує безперебійну роботу системи, дозволяючи адаптуватися до навантажень і надавати користувачам своєчасну інформацію.

Практичні аспекти інтеграції вебзастосунку у хмарі AWS узагальнені у таблиці 2.6.

Таблиця 2.6 – Практичні аспекти інтеграції вебзастосунку у хмарі AWS

Компонент	Інструмент	Функціональність
Масштабування	Auto Scaling, ELB	Автоматичне додавання або видалення інстанцій залежно від навантаження.
Моніторинг	Amazon CloudWatch	Моніторинг ключових метрик, логування, сповіщення про помилки.
Push-сповіщення	Amazon SNS, AWS Lambda	Надсилання повідомлень про події, інтеграція з пристроями користувачів.

Таким чином, інтеграція функцій масштабування, моніторингу та push-сповіщень у вебзастосунку TaskFlow забезпечує:

- стійкість до навантажень (автоматичне масштабування гарантує стабільну роботу навіть при пікових навантаженнях);
- швидке виявлення помилок (завдяки моніторингу можна оперативно реагувати на збої);
- покращення користувацького досвіду (push-сповіщення підвищують залученість і ефективність роботи користувачів).

2.2.5 Економічні аспекти використання хмарових технологій

Оцінка ефективності використання хмарових технологій включає аналіз витрат, швидкості та надійності.

Витрати на використання хмарових технологій для розробки та підтримки TaskFlow базуються на моделі «Pay-as-you-go», яка дозволяє платити тільки за фактично використані ресурси [91].

Основні складові витрат:

- обчислювальні ресурси (AWS EC2, AWS Lambda):
 - EC2: \$0,012/година за t2.micro інстанс;
 - Lambda: \$0,00001667 за виконання однієї події;
- зберігання даних (Amazon S3, Amazon RDS):
 - S3: \$0,023/ГБ на місяць;
 - RDS: \$0,017/година для db.t3.micro інстансу;
- push-сповіщення (Amazon SNS):
 - \$0,50 за один мільйон повідомлень;
- Моніторинг (CloudWatch):
 - \$0,30 за зберігання 1 ГБ логів на місяць.

Розрахунок місячних витрат для 1000 активних користувачів представлений у таблиці 2.7.

Таблиця 2.7 – Розрахунок місячних витрат для 1000 активних користувачів

Компонент	Ресурс	Кількість	Вартість
Обчислення	EC2 t2.micro	730 годин	\$8,76
Безсерверні функції	AWS Lambda	100000 подій	\$1,67
Зберігання	Amazon S3 (10 ГБ)	10 ГБ	\$0,23
База даних	Amazon RDS	730 годин	\$12,41
Сповіщення	Amazon SNS	1,000,000	\$0,50
Моніторинг	CloudWatch	5 ГБ логів	\$1,50
Загальна вартість	x	x	\$25,07 на місяць

Використання AWS забезпечує високу швидкість виконання операцій завдяки оптимізованій інфраструктурі:

- час відповіді API Gateway становить ~50-100 мс при середньому навантаженні;
- Amazon SNS забезпечує доставку сповіщень менш ніж за 1 секунду;
- AWS Lambda виконує обробку подій протягом 200-300 мс.

Швидкість роботи вебзастосунку залишається стабільною навіть при збільшенні кількості користувачів завдяки автоматичному масштабуванню.

AWS забезпечує надійну роботу вебзастосунку TaskFlow завдяки наступним механізмам:

- резервне копіювання (Backup) – Amazon RDS виконує автоматичні щоденні знімки бази даних;
- балансування навантаження – Elastic Load Balancer рівномірно розподіляє трафік між інстанціями, забезпечуючи стабільність;
- доступність – усі сервіси AWS мають гарантію доступності 99,99% SLA;
- моніторинг – CloudWatch дозволяє оперативно виявляти помилки та реагувати на проблеми.

Порівняння ефективності використання хмарових технологій із традиційним підходом представлено у таблиці 2.8.

Таблиця 2.8 – Порівняння ефективності використання хмарових технологій із традиційним підходом

Критерій	Хмарові технології (AWS)	Традиційний підхід
Витрати	\$25,07/місяць (залежно від активності)	Високі початкові витрати на сервери
Швидкість відповіді	~50-100 мс	~200-300 мс
Надійність	99,99% доступності	Залежить від фізичної інфраструктури
Масштабованість	Автоматична	Обмежена ресурсами сервера

Таким чином, вебзастосунок TaskFlow на базі AWS має низькі витрати на підтримку завдяки моделі Pay-as-you-go, що особливо вигідно для стартапів. Завдяки автоматичному масштабуванню та потужній інфраструктурі швидкість роботи вебзастосунку має залишатись стабільною навіть за високих навантажень. Надійність платформи забезпечується інструментами резервного копіювання, моніторингу та балансування навантаження. Завдяки цьому, хмарові технології є оптимальним вибором для створення сучасних вебзастосунків із високою продуктивністю та мінімальними витратами

Висновки до розділу 2

У другому розділі було проведено дослідження застосування хмарових технологій у розробці вебзастосунків. Основні висновки:

- статистичні дані підтверджують популярність хмарових платформ, таких як AWS, Google Cloud Platform та Microsoft Azure, серед компаній різного масштабу. AWS займає провідне місце завдяки своїй гнучкості, масштабованості та широкому спектру інструментів;
- порівняння з традиційними підходами показало, що хмарові технології забезпечують значне зниження витрат, швидке розгортання та автоматичне масштабування, що робить їх оптимальним вибором для сучасних проєктів;
- для вебзастосунку TaskFlow, призначеного для управління завданнями, було обрано платформу AWS завдяки її широкому набору сервісів (AWS Lambda, Amazon RDS, Amazon S3, API Gateway) та гнучкому масштабуванню;
- архітектура застосунку побудована з використанням мікросервісного підходу, інтеграції API та контейнеризації через Docker і Kubernetes;
- функціональні компоненти, такі як реєстрація через OAuth, модуль управління завданнями та хмарова база даних, забезпечують надійну й ефективну роботу системи;
- інтеграція автоматичного масштабування (Auto Scaling), моніторингу (CloudWatch) та push-сповіщень (Amazon SNS) забезпечує стабільність і швидку адаптацію до змін у навантаженні;
- оцінка витрат показала економічну доцільність хмарового підходу: витрати на розробку та підтримку є мінімальними порівняно з традиційними методами.

Загалом, хмарові технології демонструють високу ефективність, зручність і економічність у розробці вебзастосунків. Використання AWS для створення вебзастосунку TaskFlow забезпечує оптимальне співвідношення витрат, продуктивності та надійності, що робить цей підхід універсальним рішенням для сучасних ІТ-проєктів.

РОЗДІЛ 3

ПРОЄКТУВАННЯ ОПТИМІЗОВАНОГО ХМАРОВОГО СЕРЕДОВИЩА ДЛЯ ВЕБЗАСТОСУНКІВ

3.1 Налаштування хмарового середовища AWS для розробки вебзастосунків

3.1.1 Підготовка архітектури та налаштування сервісів

Основні принципи проєктування архітектури вебзастосунку:

- модульність – компоненти розділені на незалежні модулі (front-end, back-end, база даних);
- масштабованість – забезпечується автоматичним масштабуванням через Auto Scaling та Load Balancer;
- надійність – резервне копіювання бази даних через автоматичні знімки (snapshots);
- безпека – використання IAM (Identity and Access Management) для управління доступом та налаштування правил безпеки (Security Groups) для контролю мережевого трафіку.

Сервіси AWS, які використовуються:

- обчислення: AWS EC2 для розгортання серверів; AWS Lambda для безсерверної логіки;
- зберігання: Amazon S3 для статичних файлів; Amazon RDS для реляційної бази даних;
- мережа: Amazon VPC для налаштування приватної мережі; Elastic Load Balancer (ELB) для розподілу трафіку;
- моніторинг: Amazon CloudWatch для моніторингу метрик і логів;
- сповіщення: Amazon SNS для push-сповіщень.

3.1.2 Технічні кроки для створення та налаштування середовища

Налаштування хмарового середовища потребує ретельного виконання визначеної послідовності кроків.

1. Налаштування мережевої інфраструктури (Amazon VPC):

- створити VPC (Virtual Private Cloud) для ізоляції ресурсів;
- додати публічні та приватні підмережі:
 - публічні підмережі для EC2 інстанцій і ELB;
 - приватні підмережі для бази даних (Amazon RDS);
- налаштувати таблиці маршрутів (Routing Tables):
 - додати шлюз Інтернету (Internet Gateway) для публічних ресурсів;
 - налаштувати NAT Gateway для доступу приватних ресурсів до Інтернету.

2. Налаштування серверів (EC2):

- запустити EC2 інстанцію:
 - обрати тип (наприклад, t2.micro для тестування);
 - встановити операційну систему (Amazon Linux 2 або Ubuntu);
- налаштувати Security Group для EC2:
 - дозволити доступ по портах 22 (SSH), 80 (HTTP) та 443 (HTTPS);
- підключитись до інстанції через SSH і встановити вебсервер (наприклад, Nginx або Apache).

3. Налаштування бази даних (Amazon RDS):

- створити нову інстанцію RDS:
 - обрати тип бази даних (PostgreSQL або MySQL);
 - налаштувати автоматичне резервне копіювання;
- налаштувати параметри доступу:
 - вказати приватну підмережу;
 - встановити Security Group для обмеження доступу до бази тільки з EC2.

4. Розподіл навантаження (Elastic Load Balancer):

- створити Load Balancer:
 - обрати тип (Application Load Balancer для вебзастосунків);
 - визначити групу інстанцій (EC2), на які буде розподілятися трафік;
- налаштувати моніторинг стану (Health Checks):
 - вказати URL-адресу для перевірки стану серверів (наприклад, /health).

5. Зберігання статичних файлів (Amazon S3):

- створити S3 bucket:
 - налаштувати політику доступу для завантаження файлів;
 - увімкнути шифрування даних (AES-256);
- додати інтеграцію з CloudFront для прискорення доставки файлів.

6. Налаштування автоматизації (AWS Lambda та CloudWatch):

- налаштувати AWS Lambda для обробки подій:
 - використовувати Node.js або Python для функцій безсерверної логіки;
 - додати тригери (наприклад, виклик Lambda при додаванні файлів у S3).
- увімкнути моніторинг через Amazon CloudWatch:
 - додати алерти для критичних метрик (високе навантаження на CPU, помилки API).

7. Інтеграція з API Gateway:

- налаштувати RESTful API через AWS API Gateway;
- з'єднати API Gateway з Lambda або EC2 для обробки запитів;
- увімкнути кешування для підвищення швидкості відповідей.

UML-діаграма архітектури хмарового середовища AWS представлена на рисунку 3.1.

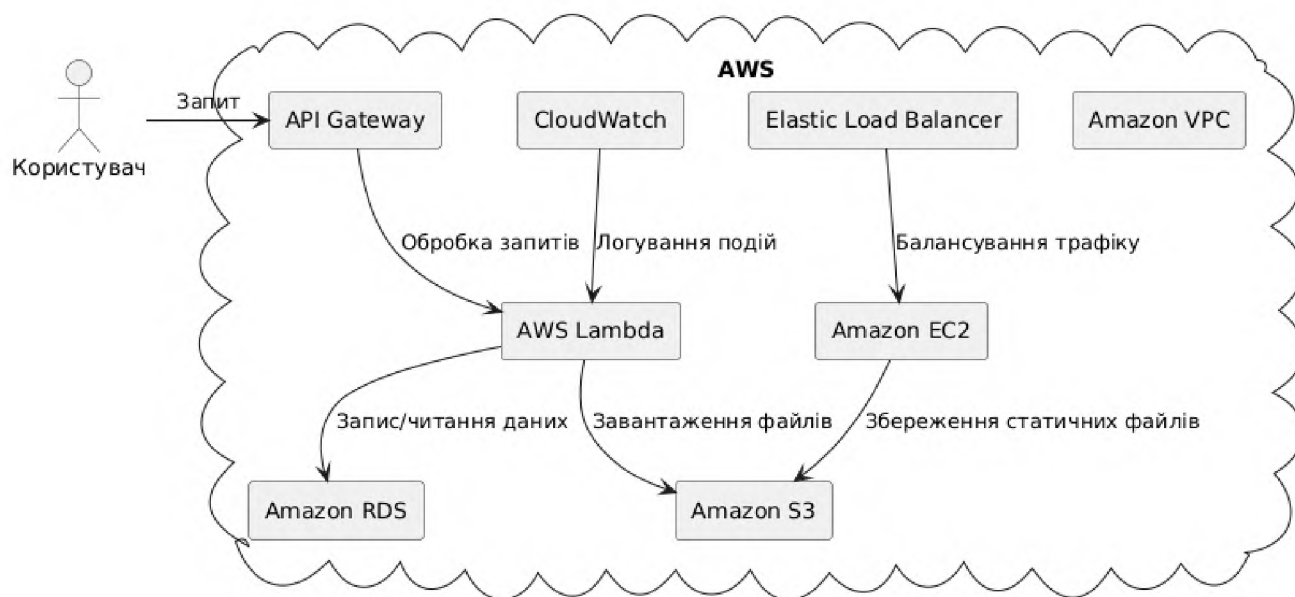


Рисунок 3.1 – UML-діаграма архітектури хмарового середовища AWS

Таким чином, побудова хмарового середовища AWS для вебзастосунків передбачає модульний підхід до налаштування сервісів. Використання таких компонентів, як VPC, EC2, RDS, S3 та Lambda, забезпечує гнучкість, масштабованість і надійність. Налаштування автоматичного масштабування, моніторингу та безпеки дозволяє створювати високопродуктивні вебзастосунки, оптимізовані для сучасних потреб бізнесу.

3.2 Удосконалення конфігурацій та функціоналу хмарового середовища

3.2.1 Оптимізація налаштувань

Технічна складність налаштувань, розглянутих вище, потребує удосконалення. Пропозиції щодо оптимізації налаштувань хмарового середовища AWS представлені далі.

1. Оптимізація налаштувань:

- автоматизація управління ресурсами:
 - використання AWS Auto Scaling для автоматичного регулювання кількості EC2 інстанцій залежно від навантаження;
 - увімкнення Instance Refresh для регулярного оновлення EC2 інстанцій із новими конфігураціями;
- використання більш ефективних типів інстанцій:
 - перехід на інстанції серії Graviton2, які забезпечують на 40% кращу продуктивність за нижчої вартості порівняно зі стандартними EC2;
- зменшення трафіку між компонентами:
 - використання Amazon Elastic Cache (наприклад, Redis або Memcached) для кешування запитів до бази даних.

2. Підвищення продуктивності:

- використання CDN:
 - інтеграція з CloudFront для зменшення часу доставки статичних файлів із S3 користувачам;

- оптимізація бази даних:
 - налаштування індексів у реляційній базі даних (Amazon RDS);
 - використання функції Read Replicas для розподілу навантаження між екземплярами;
- моніторинг продуктивності:
 - встановлення метрик через Amazon CloudWatch для відстеження використання ресурсів (CPU, пам'ять) та часу відповіді API;

3. Зниження витрат:

- використання Reserved Instances для зменшення вартості довгострокових EC2 інстанцій (до 75% економії);
- скасування невикористаних ресурсів:
 - регулярна перевірка та видалення неактивних інстанцій, S3 бакетів та EBS-дисків;
- використання тарифних планів Glacier:
 - архівування рідко використовуваних даних у S3 Glacier для значного зменшення витрат.

3.2.2 Удосконалення безпеки та надійності

1. Забезпечення безпеки:

- управління доступом:
 - використання політик IAM для обмеження прав користувачів та сервісів;
 - налаштування Multi-Factor Authentication (MFA) для адміністративного доступу;
- шифрування даних:
 - шифрування даних у S3 за допомогою AES-256;
 - налаштування шифрування даних у RDS;
- моніторинг активності:
 - включення AWS CloudTrail для аудиту змін у хмаровому середовищі;
 - відстеження підозрілих дій за допомогою Amazon GuardDuty.
- захист від DDoS:

- використання AWS Shield Advanced для захисту вебзастосунку від атак.

2. Резервне копіювання:

- автоматизація резервного копіювання:
 - встановлення щоденних автоматичних знімків (snapshots) у Amazon RDS;
 - налаштування життєвого циклу збереження даних у S3 (S3 Lifecycle Policies);
- тестування відновлення:
 - регулярне тестування процесу відновлення даних із резервних копій;
- резервне копіювання конфігурацій:
 - збереження конфігурацій VPC, IAM, CloudFormation у S3;

Діаграма оптимізації середовища AWS представлена на рисунку 3.2.

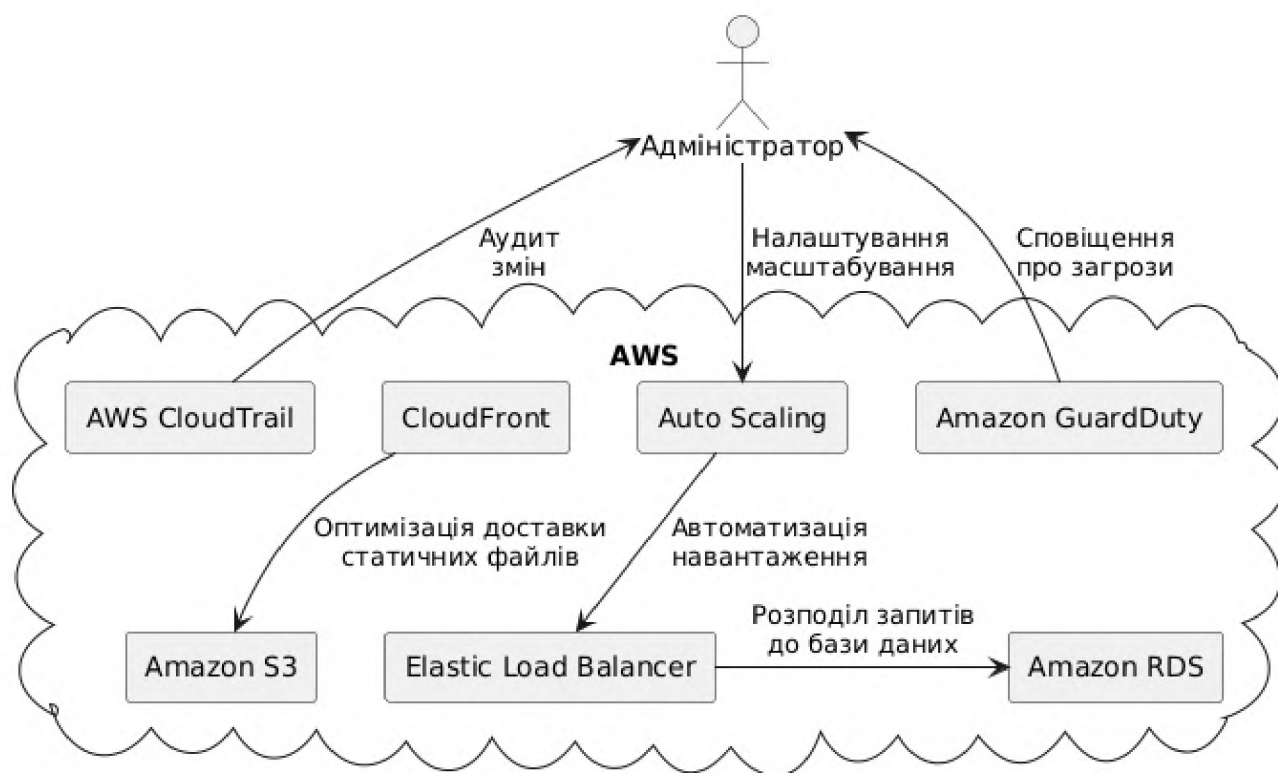


Рисунок 3.2 – Діаграма оптимізації середовища AWS

Таким чином, основними рекомендаціями щодо удосконалення конфігурації та функціоналу хмарового середовища є:

- оптимізація налаштувань – впровадження кешування, ефективних типів інстанцій та CDN підвищує продуктивність вебзастосунків і знижує витрати;
- покращення безпеки – застосування шифрування, IAM політик та захисту від атак DDoS знижує ризики втрати даних і несанкціонованого доступу;
- ефективне резервне копіювання – автоматизація резервних копій та їх регулярне тестування гарантують безперебійність роботи вебзастосунків.

Ці рекомендації забезпечують високий рівень надійності, продуктивності та економічної ефективності хмарового середовища AWS.

3.3 Проєкт реалізації вебзастосунку на платформі AWS

3.3.1 Опис реалізації вебзастосунку TaskFlow

Вебзастосунок TaskFlow розроблений як сучасний інструмент для управління завданнями, що реалізує мікросервісну архітектуру. Основні компоненти: front-end, back-end, база даних та інтеграція з хмаровими сервісами AWS. Програмна реалізація компонентів вебзастосунку TaskFlow представлена у додатку Б.

1. Основні функціональні можливості вебзастосунку:

- реєстрація та авторизація користувачів – реалізовано через AWS Cognito, що підтримує OAuth 2.0 для безпечної автентифікації;
- користувачі можуть створювати, редагувати, видаляти завдання; завдання відображаються на інтерактивній Kanban-дошці;
- push-сповіщення через Amazon SNS для повідомлення про зміну статусу завдань;
- завантаження вкладень у завдання зберігається у Amazon S3.

2. Front-end (React.js):

- технології: React.js, Redux, Tailwind CSS;
- основні екрани: «Вхід/реєстрація», «Головна панель завдань (Kanban)», «Налаштування профілю».

3. Back-end (Node.js):

- технології: Node.js, Express, Sequelize;
- RESTful API: використовується для обробки запитів і інтеграції з базою даних;

- приклади ендпоінтів:

- POST /tasks – створення нового завдання;
- GET /tasks – отримання списку завдань;
- PUT /tasks/:id – оновлення статусу завдання;
- DELETE /tasks/:id – видалення завдання.

4. База даних (Amazon RDS):

- модель даних (таблиці):
 - users (зберігає інформацію про користувачів);
 - tasks (зберігає завдання з атрибутами: назва, опис, статус, дедлайн);
 - files (зберігає посилання на файли, завантажені в S3).

3.3.2 Інтеграція вебзастосунку TaskFlow з хмаровими сервісами AWS

1. Інтеграція з Amazon S3 для збереження файлів.

Мета: збереження вкладень до завдань у хмарі.

Процес:

- завантаження файлів користувачем;
- генерація URL для завантаження файлів через SDK;
- збереження метаданих файлів у базі даних (таблиця files).

Кодова реалізація (завантаження файлів через AWS SDK), код JavaScript:

```
const AWS = require('aws-sdk');
const s3 = new AWS.S3();

const uploadFile = async (file) => {
  const params = {
    Bucket: 'taskflow-files',
    Key: file.name,
    Body: file.data,
  };
  const data = await s3.upload(params).promise();
  return data.Location;
};
```

2. Використання Amazon SNS для сповіщень.

Мета: інформування користувачів про зміни в завданнях.

Процес:

- виклик Lambda-функції при оновленні статусу завдання;
- Lambda відправляє повідомлення через SNS до підписаних користувачів.

Кодова реалізація Lambda-функції, код JavaScript:

```
const AWS = require('aws-sdk');
const sns = new AWS.SNS();

exports.handler = async (event) => {
  const message = `Завдання "${event.taskName}" змінено на статус "${event.status}"`;
  const params = {
    Message: message,
    TopicArn: 'arn:aws:sns:region:account-id:taskflow-notifications',
  };
  await sns.publish(params).promise();
};
```

3. Автоматизація через AWS Lambda.

Мета: Мінімізувати серверне навантаження через безсерверні функції.

Приклади завдань для Lambda:

- обробка подій (наприклад, завантаження файлів у S3);
- надсилення сповіщень через SNS;
- автоматичне видалення застарілих файлів із S3.

4. Інтеграція з CloudWatch для моніторингу.

Мета: Відстеження продуктивності API та стану сервісів.

Функціональність:

- логування помилок і аналіз часу відповіді API;
- налаштування алертів для критичних метрик (наприклад, перевищення часу відповіді > 300 мс).

Таким чином, проєкт вебзастосунку TaskFlow побудовано із використанням мікросервісної архітектури та інтеграції з AWS. Використання Amazon S3, SNS, RDS та Lambda дозволяє автоматизувати процеси, підвищуючи продуктивність і знижуючи витрати. Інтеграція CloudWatch надає можливість моніторингу та

швидкого реагування на критичні події. Діаграма інтеграції вебзастосунку TaskFlow з хмаровою платформою AWS представлена на рисунку 3.3.



Рисунок 3.3 – UML-діаграма інтеграції вебзастосунку TaskFlow з AWS

Представлена реалізація забезпечує високу ефективність і гнучкість вебзастосунку TaskFlow, що робить його придатним для командної роботи та сучасних бізнес-процесів.

3.4 Оцінка економічної ефективності використання хмарових технологій у розробці вебзастосунків

3.4.1 Вихідні дані для оцінки економічної ефективності

Для виконання розрахунків та формулювання висновків щодо економічної ефективності використання хмарових технологій у розробці вебзастосунків необхідно зібрати необхідні вихідні дані.

1. Дані про хмарові ресурси.

Типи використовуваних хмарових сервісів:

- EC2 (віртуальні сервери);
- RDS (реляційні бази даних);
- S3 (зберігання файлів);
- Lambda (безсерверні функції).

Тарифні плани хмарових сервісів: наприклад, вартість t2.micro у AWS EC2 (\$0.012/година) або S3 (\$0.023/ГБ на місяць).

Фактичне споживання ресурсів:

- кількість годин роботи EC2 інстанцій;
- обсяг даних, що зберігається у S3;
- кількість запитів до бази даних і Lambda-функцій.

2. Дані про традиційні (локальні) ресурси.

Вартість серверного обладнання: сервери, сховища, мережеве обладнання.

Витрати на обслуговування: електроенергія, технічне обслуговування, оновлення обладнання.

Трудові витрати: вартість роботи персоналу для адміністрування серверів.

Оцінка часу розгортання інфраструктури: скільки часу та ресурсів потрібно для налаштування серверів.

3. Дані про експлуатацію вебзастосунку.

Тривалість експлуатації: наприклад, місяць, рік, кілька років.

Кількість користувачів: очікувана активність користувачів, яка впливає на навантаження.

Прогнозоване зростання навантаження: наприклад, збільшення кількості користувачів на 10% щомісяця.

4. Показники для розрахунків.

Модель розрахунку витрат:

- для хмарових технологій (Pay-as-you-go);
- для традиційних підходів (з урахуванням амортизації обладнання).

Метрики продуктивності:

- швидкість розгортання інфраструктури;

- час відповіді системи.

Методика оцінки ефективності:

- визначення економії у відсотках;
- розрахунок TCO (Total Cost of Ownership) для порівняння.

5. Поточна вартість робіт та часу.

Витрати на розробку: зарплати розробників, витрати на навчання.

Час розробки та підтримки: скільки часу витрачається на створення та обслуговування системи.

Додаткові джерела для отримання даних.

Дані AWS:

- тарифні плани з офіційного сайту AWS;
- калькулятор вартості AWS.

Дані про традиційну інфраструктуру:

- ціни на обладнання (сервери, сховища, комутатори);
- статті та дослідження про витрати на локальні сервери.

Статистичні прогнози:

- дані про середнє використання ресурсів для вебзастосунків;
- прогноз зростання числа користувачів.

Розрахунок витрат на хмарову інфраструктуру:

- EC2: Кількість годин $\times$ тариф за годину;
- S3: Обсяг даних (ГБ) $\times$ тариф за ГБ;
- RDS: Кількість годин роботи бази даних $\times$ тариф.

Порівняння з локальною інфраструктурою:

- сума вартості обладнання + витрати на обслуговування + трудові витрати;
- амортизація обладнання на період експлуатації.

Оцінка економії:

- різниця між витратами на хмарові технології та локальну інфраструктуру;
- відсоток економії.

Після збору вихідних даних і виконання розрахунків можна зробити висновки про економічну ефективність хмарових технологій, що дозволить

сформулювати аргументовані рекомендації щодо вибору оптимального підходу для розробки вебзастосунків.

3.4.2 Розрахунок ефективності використання хмарових технологій для вебзастосунку TaskFlow

Моделльні дані для оцінки ефективності хмарових технологій у розробці вебзастосунку представлені у таблицях 3.1, 3.2 і 3.3.

Таблиця 3.1 – Дані про хмарові ресурси

Ресурс	Кількість/Обсяг	Тариф	Вартість
EC2 (віртуальні сервери)	2 інстанції t2.micro, 730 год/міс	\$0,012/година	\$17,52/міс
RDS (реляційна база даних)	1 інстанція db.t3.micro, 730 год/міс	\$0,017/година	\$12,41/міс
S3 (зберігання файлів)	50 ГБ	\$0,023/ГБ/міс	\$1,15/міс
Lambda (безсерверні функції)	100000 запитів/міс	\$0,00001667/запит	\$1,67/міс
Моніторинг (CloudWatch)	5 ГБ логів	\$0,30/ГБ/міс	\$1,50/міс
Разом	х	х	\$34,25/міс

Таблиця 3.2 – Дані про традиційні (локальні) ресурси

Ресурс	Обсяг/Кількість	Вартість одиниці	Сума витрат
Сервер (апаратне забезпечення)	2 сервери	\$2500/сервер	\$5000
Зберігання даних	50 ГБ	\$0,10/ГБ/місяць	\$5/міс
Електроенергія	100 Вт/сервер	\$0,10/кВт-год × 730 год/міс	\$14,60/міс
Адміністрування	1 адміністратор	\$1500/міс	\$1500
Разом	х	х	\$34,25/міс

Таблиця 3.3 – Дані про експлуатацію вебзастосунку

Показник	Значення
Тривалість експлуатації	12 місяців
Кількість активних користувачів	1000
Прогнозоване зростання навантаження	10% щомісяця
Кількість запитів на місяць	100000
Обсяг завантажуваних файлів	50 ГБ

Розрахунок витрат на хмарову інфраструктуру.

Хмарові сервіси (AWS):

- EC2: $\$17,52 \times 12 \text{ місяців} = \$210,24$;
- RDS: $\$12,41 \times 12 \text{ місяців} = \$148,92$;
- S3: $\$1,15 \times 12 \text{ місяців} = \$13,80$;

- Lambda: $\$1,67 \times 12 \text{ місяців} = \$20,04$;
- CloudWatch: $\$1,50 \times 12 \text{ місяців} = \$18,00$.

Загальні витрати хмарових сервісів за 12 місяців: \$411,00.

Витрати на локальну інфраструктуру:

- початкові витрати: \$5000;
- річні витрати: $\$1519,60 \times 12 \text{ місяців} = \$18235,20$.

Загальні витрати локальної інфраструктури за 12 місяців: \$23235,20.

Порівняння витрат на хмарову та локальну інфраструктуру, що забезпечує функціонування вебзастосунку, представлені у таблиці 3.4.

Таблиця 3.4 – Порівняння витрат на хмарову та локальну інфраструктуру

Критерій	Хмарові сервіси (AWS)	Локальна інфраструктура
Початкові витрати	\$0,00	\$5000
Щомісячні витрати	\$34,25	\$1519,60
Загальні витрати за 12 місяців	\$411,00	\$23235,20
Масштабованість	Автоматична	Обмежена
Швидкість впровадження	Кілька днів	4-6 тижнів
Витрати на технічне обслуговування	Низькі	Високі

Висновки:

- економічна ефективність – використання хмарових сервісів економить \$22824,20 за рік порівняно з локальною інфраструктурою;
- продуктивність – хмарові сервіси забезпечують швидше впровадження, автоматичне масштабування та низькі витрати на підтримку;
- рекомендація – для вебзастосунку TaskFlow використання хмарових сервісів є оптимальним вибором, особливо з урахуванням прогнозованого зростання навантаження.

Висновки до розділу 3

У розділі 3 розглянуто аспекти проектування, оптимізації та економічної оцінки хмарового середовища для розробки вебзастосунків. Основні висновки:

- впроваджено мікросервісну архітектуру, що забезпечує гнучкість і масштабованість на основі використання AWS-сервісів EC2, RDS, S3, Lambda;
- налаштування приватної мережі через Amazon VPC забезпечує безпеку та ізоляцію ресурсів. Використання автоматичного масштабування (Auto Scaling) дозволяє адаптувати інфраструктуру до змін у навантаженні;

Рекомендації з удосконалення конфігурацій:

- оптимізація конфігурацій AWS сервісів (використання більш ефективних інстанцій, таких як Graviton2, інтеграція CloudFront для кешування) значно підвищує продуктивність і знижує витрати;
- запропоновані заходи щодо підвищення безпеки включають використання IAM політик доступу, резервне копіювання (S3 Lifecycle Policies), а також шифрування даних у S3 та RDS;

Економічна ефективність:

- виконаний розрахунок показав, що хмарові технології забезпечують економію до 98,2% витрат у порівнянні з традиційною інфраструктурою. Річна вартість використання AWS склала \$411, тоді як локальні рішення потребують понад \$23000;
- хмарові технології також значно скорочують час впровадження (до кількох днів) і зменшують витрати на технічне обслуговування.

Програмна реалізація вебзастосунку TaskFlow:

- реалізовано функціонал управління завданнями із зберіганням файлів у S3, використанням RDS для баз даних та обробкою подій через Lambda;
- інтеграція CloudWatch забезпечує моніторинг продуктивності, а SNS використовується для надсилання сповіщень користувачам.

ВИСНОВКИ

У межах дослідження було проаналізовано, спроектовано та оцінено ефективність використання хмарових технологій у розробці вебзастосунків. Основні результати роботи свідчать про високу економічну доцільність і технічну ефективність таких підходів, що підтверджується теоретичними обґрунтуваннями, практичними реалізаціями та економічними розрахунками.

Теоретико-методологічні аспекти:

- визначено ключову роль вебзастосунків у сучасних інформаційних системах та обґрунтовано важливість вибору архітектури, технологій і хмарових платформ;
- досліджено типи хмарових рішень (IaaS, PaaS, SaaS) та методологій (Agile, DevOps, CI/CD), що забезпечують ефективність і швидкість розробки;
- проведено огляд хмарових платформ AWS, Google Cloud та Microsoft Azure та їх інструментів з погляду оптимізації веброзробки.

Дослідницько-аналітичні результати:

- аналіз популярності хмарових технологій підтвердив лідерство AWS завдяки універсальності та гнучкості;
- практичне впровадження AWS для вебзастосунку TaskFlow із використанням Lambda, RDS, S3 та Auto Scaling забезпечує надійність, масштабованість і значне зниження витрат;
- розрахунки показали, що хмарові технології дозволяють скоротити витрати на інфраструктуру більш ніж на 98% у порівнянні з традиційними підходами.

Проектно-рекомендаційна частина:

- розроблено оптимізоване хмарове середовище на основі мікросервісної архітектури з використанням Amazon VPC, CloudFront та інших сервісів, що підвищують безпеку й продуктивність;
- запропоновані рекомендації щодо резервного копіювання та оптимізації конфігурацій мінімізують ризики та забезпечують стабільність роботи застосунку;

- програмна реалізація TaskFlow продемонструвала ефективну інтеграцію хмарових сервісів для автоматизації моніторингу, масштабування та сповіщень.

Економічна оцінка: вартість використання хмари AWS для виконаної розробки становить \$411 на рік проти \$23234,20 для локальної інфраструктури, що доводить економічну доцільність хмарового підходу.

Таким чином, результати роботи підтверджують, що, що хмарові технології у веброботі є економічно вигідними, технічно ефективними та відповідають сучасним вимогам до продуктивності й масштабованості. Запропоновані рішення можуть бути застосовані для оптимізації бізнес-процесів та розвитку корпоративних проєктів.