Побудова багаторівневого веб-застосунку з високою доступністю на хмарній платформі

Науковий керівник: Старший викладач Черкасов Д.І.

Виконав: Іщенко Тіхон Олексійович ІПЗ-4

Мета та завдання роботи

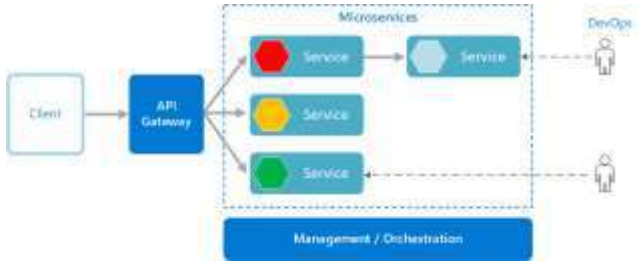
Мета: Вивчення та впровадження оптимальних архітектурних, інфраструктурних і технологічних рішень для забезпечення стабільної роботи багаторівневого веб-застосунку в умовах змінного навантаження.

Завдання роботи:

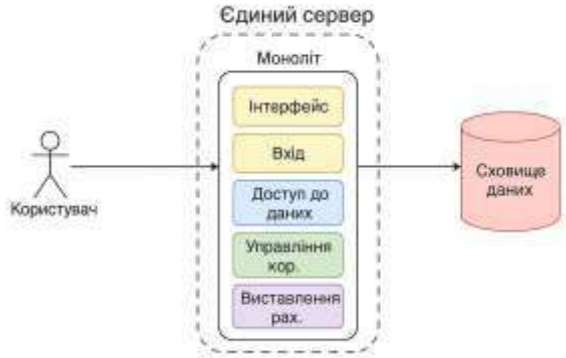
1. Провести аналіз сучасних технологій побудови високодоступних веб-застосунків.
2. Розробити архітектуру застосунку із забезпеченням відмовостійкості, масштабованості та безпеки.
3. Реалізувати застосунок для допомоги безпритульним тваринам на базі обраних технологій.
4. Налаштувати автоматизовані процеси CI/CD для спрощення розгортання та обслуговування системи.

Порівняльний аналіз архітектур

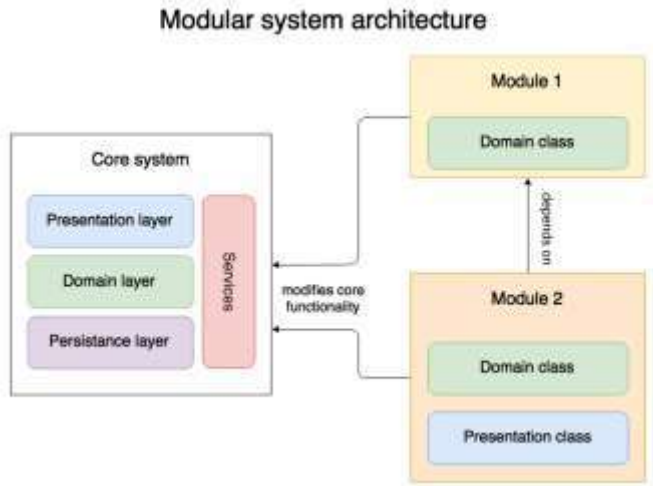
Мікросервісна архітектура



Монолітна архітектура



Модульна архітектура

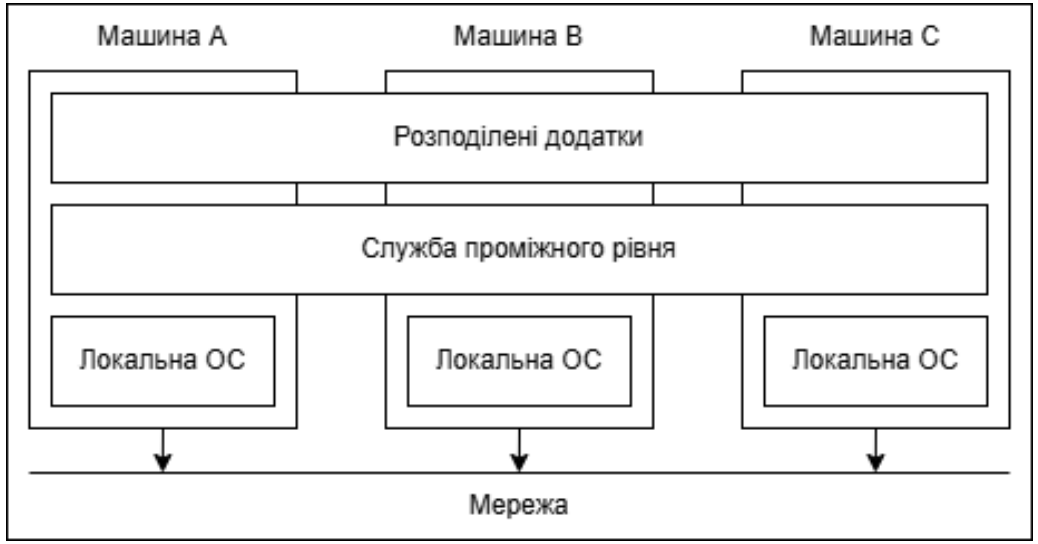


Архітектурні моделі розгортання

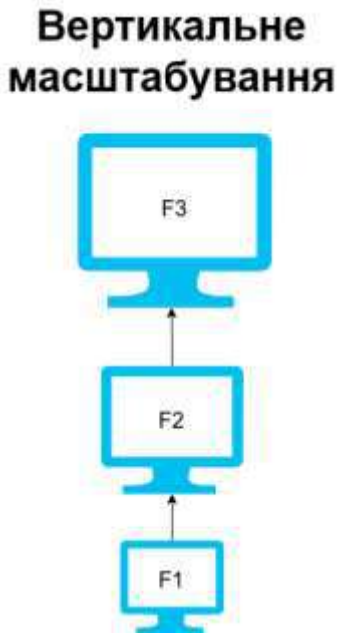
Централізована модель

Централізована архітектура

Розподілена модель



Масштабування та балансування навантаження



Порівняння хмарних платформ

Критерій	AWS	Microsoft Azure	Google Cloud Platform (GCP)
Масштабування	Автоматичне масштабування EC2, ECS, Lambda, гнучке вертикальне та горизонтальне масштабування.	Автоматичне масштабування для VM, App Services, AKS, інтеграція з Azure Monitor.	Автоматичне масштабування Compute Engine, GKE, Cloud Run, гнучке управління ресурсами.
Гео-доступність	33 регіони, 105 зон доступності, 600+ точок присутності по всьому світу.	64 регіони, 126 зон доступності, 192 точки присутності, активне розширення.	37 регіонів, 112 зон доступності, глобальна мережа Google.
Аварійне відновлення	AWS Elastic Disaster Recovery, підтримка мульти-регіональних стратегій, RTO/RPO від хвилин до годин.	Azure Site Recovery, інтеграція з Azure Backup, підтримка гібридних сценаріїв.	Вбудовані інструменти для резервного копіювання та відновлення, підтримка мульти-регіональних стратегій.
Ціноутворення	On-Demand, погодинна та похвилинна тарифікація.	Pay-as-you-go, Reserved Instances, погодинна тарифікація.	Pay-as-you-go, похвилинна тарифікація.

Порівняння фреймворків для розробки застосувань

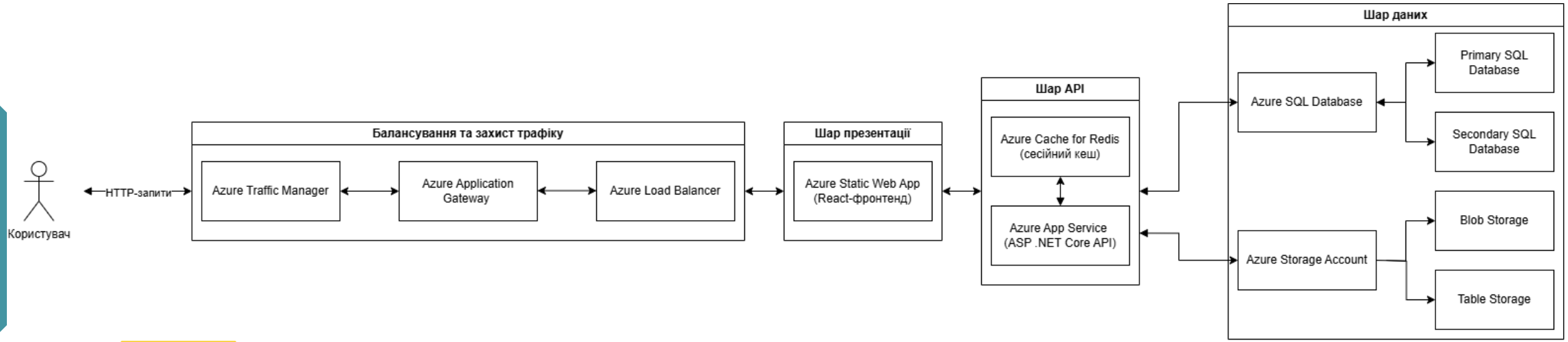
- .NET Core забезпечує високу швидкодію, гнучку інфраструктуру, глибоку інтеграцію з Azure.
- Node.js ефективний при I/O-навантаженнях, але гірше справляється з CPU-bound задачами.
- Spring Boot — стандарт для корпоративних Java-систем із широким стеком інтеграцій.
- Django — швидкий у розробці, але менш продуктивний при паралельній обробці запитів.

Платформа для пошуку та усиновлення бездомних тварин.
Основний функціонал:

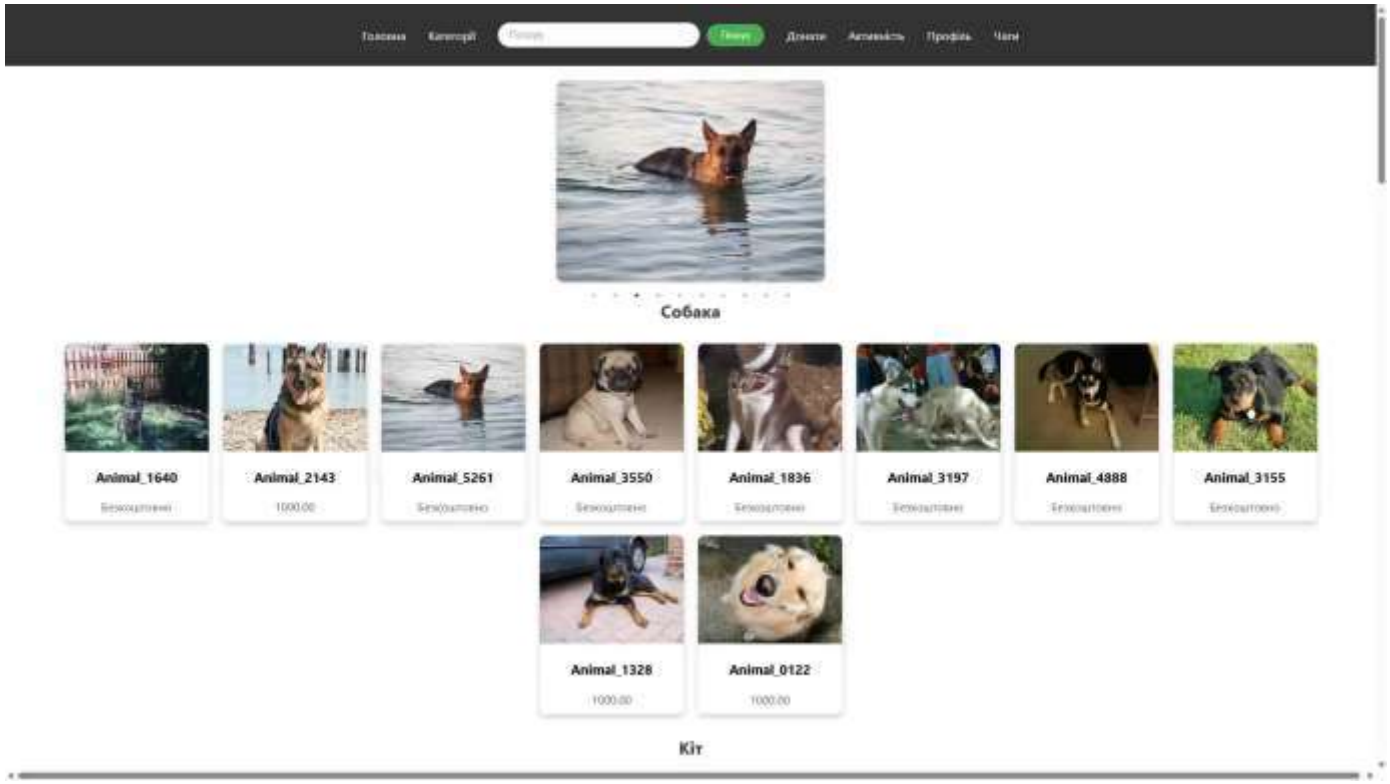
- Реєстрація та автентифікація користувачів із розподілом на ролі з використанням Cookie.
- Створення, редагування, видалення та перегляд оголошень про тварин з атрибутами: фото, вік, стан здоров’я, контактна інформація, теги.
- Пошук та фільтрація оголошень за віком та іншими тегами.
- Можливість спілкуватися з іншими користувачами за допомогою внутрішнього чату.

Функціональне призначення веб-застосунку

Фінальна архітектура застосунку

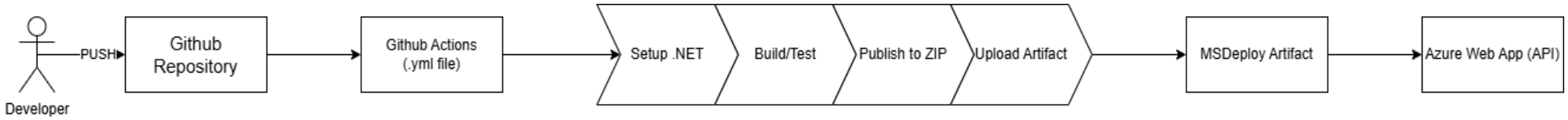


Реалізація веб-застосунку

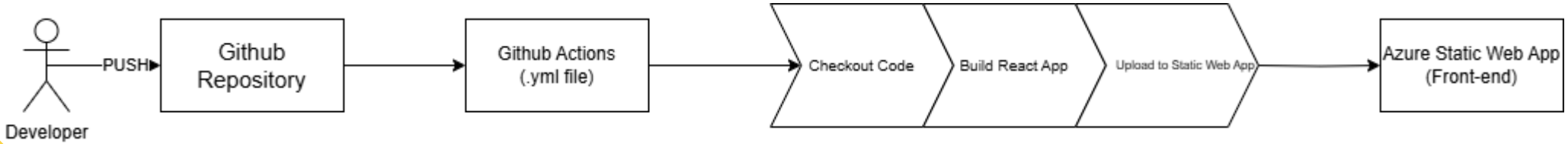


Процеси CI/CD

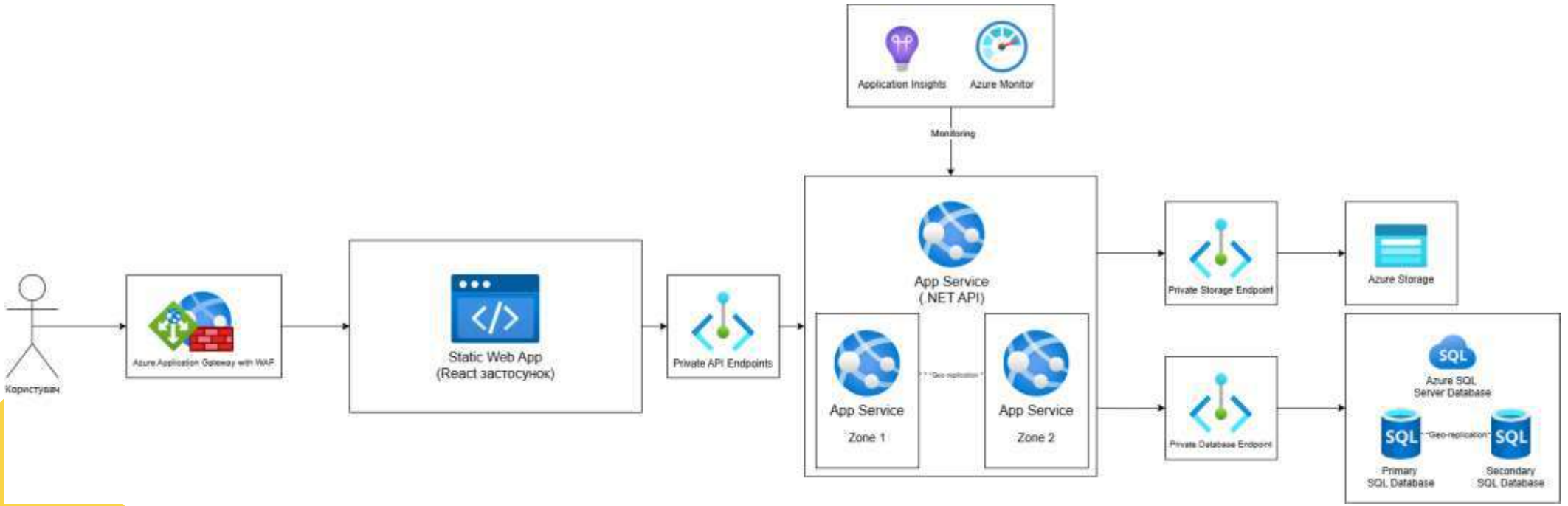
Azure Web App (API)



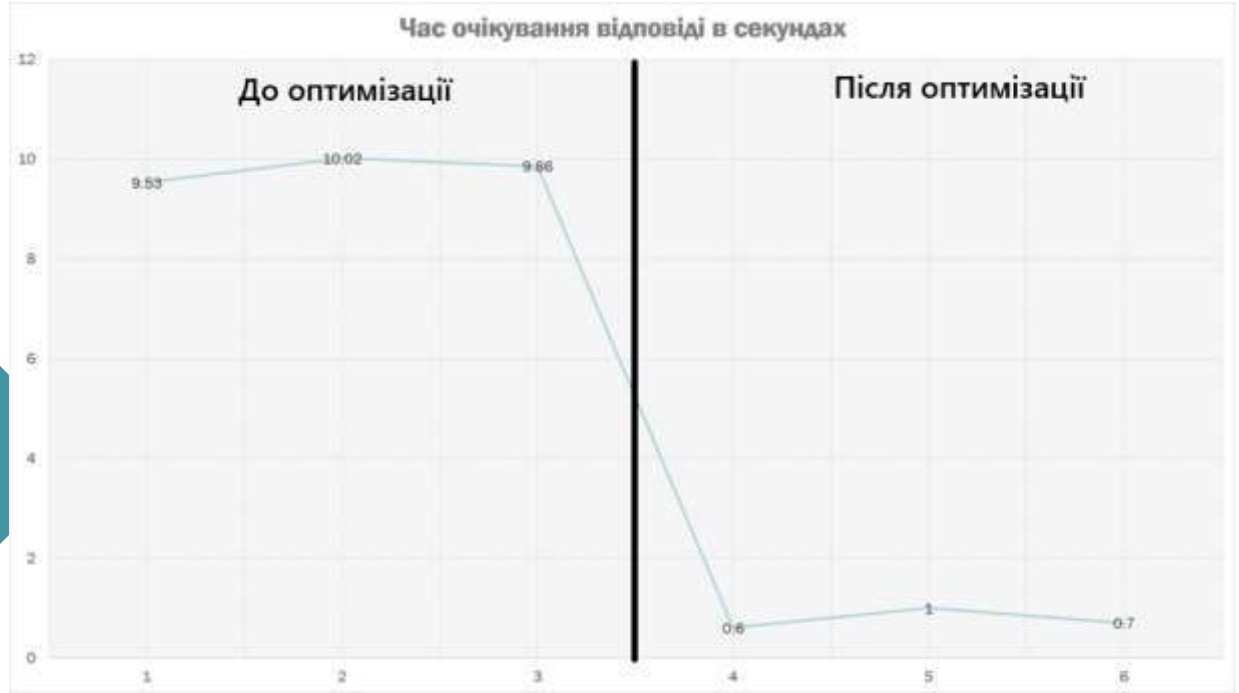
Azure Static Web App (React App)



Хмарна інфраструктура



Тестування продуктивності



Сценарії тестування:

- constant-arrival-rate: імітація тривалого навантаження (50 rps)
- ramping-arrival-rate: імітація пікових навантажень до 200 VUs

Результати:

- p95 ≈ 900 мс при 50 rps, throughput ≈ 49–50 rps
- При піковому навантаженні: p95 ≈ 2 с, до 200 rps, error rate < 1 %
- SLA досягнуто: затримка < 5 с навіть у пікові моменти

Оптимізація застосунку знизила затримку на 50%.

Результати роботи

- Проаналізовано існуючі підходи до побудови високо доступних вебзастосунків із використанням хмарних сервісів Azure.
- Запропоновано й реалізовано трирівневу архітектуру «PetFinder» з React-фронтендом, ASP .NET Core API та Azure SQL Database із Geo-реплікацією.
- Налагоджено CI/CD-процеси в GitHub Actions: автоматичне build→test→deploy фронтенду в Static Web Apps і бекенду в App Service.
- Впроваджено кешування сесій і даних через Azure Cache for Redis та зберігання медіаконтенту й бекапів у Azure Storage Account.
- Забезпечено моніторинг і алерти через Application Insights і Azure Monitor.

Дякую за увагу