Міністерство освіти і науки України

Харківський національний університет радіоелектроніки

Факультет комп’ютерної інженерії та управління
(повна назва)

Кафедра електронних обчислювальних машин
(повна назва)

КВАЛІФІКАЦІЙНА РОБОТА

Пояснювальна записка

Рівень вищої освіти другий (магістерський)

Методи тестування навантаження програмних систем
(тема)

Виконав:

студент II курсу, групи КСМм-23-1

Вакал В. В.
(прізвище, ініціали)

Спеціальність 123 «Комп’ютерна інженерія»
(код і повна назва спеціальності)

Тип програми освітньо-професійна
(освітньо-професійна або освітньо-наукова)

Освітня програма Комп’ютерні системи та мережі
(повна назва освітньої програми)

Керівник: доц. Федорченко В. М.
(посада, прізвище, ініціали)

Допускається до захисту

Зав. кафедри ЕОМ _____ Коваленко А. А.
(підпис) (прізвище, ініціали)

2025 р.

Харківський національний університет радіоелектроніки

Факультет комп'ютерної інженерії та управління

Кафедра електронних обчислювальних машин

Рівень вищої освіти другий (магістерський)

Спеціальність 123 «Комп'ютерна інженерія»
(код і повна назва)

Тип програми освітньо-професійна
(освітньо-професійна або освітньо-наукова)

Освітня програма Комп'ютерні системи та мережі
(повна назва)

ЗАТВЕРДЖУЮ:
Зав. кафедри _____
(підпис)
"____" _____ 20__ р.

ЗАВДАННЯ

НА КВАЛІФІКАЦІЙНУ РОБОТУ

студенту Вакалу Владиславу Валерійовичу
(прізвище, ім'я, по батькові)

1. Тема роботи Методи тестування навантаження програмних систем

затверджена наказом по університету від " 22 " листопада 2024 р. № 1237 Ст

2. Термін подання студентом роботи до екзаменаційної комісії 20 січня 2025 р.

3. Вхідні дані до роботи 1) методи тестування ПЗ;
2) хмарна платформа Azure;
3) науково-методичні розробки вітчизняних та зарубіжних авторів.

4. Перелік питань, що потрібно опрацювати у роботі
1) огляд найуживаніших підходів до тестування навантаження ПЗ та БД;
2) вибір та обґрунтування методики та засобів дослідження;
3) програмна реалізація методів тестування навантаження;
4) проведення експериментальних досліджень;
5) висновки.

5. Перелік графічного матеріалу із зазначенням креслеників, схем, плакатів, комп'ютерних ілюстрацій (слайдів)

Слайд-презентація – 18 слайдів

6. Консультанти розділів роботи (заповнюється за наявності консультантів згідно з наказом, зазначеним у п.1)

Найменування розділу	Консультант (посада, прізвище, ім'я, по батькові)	Позначка консультанта про виконання розділу	
		підпис	дата

КАЛЕНДАРНИЙ ПЛАН

№	Назва етапів роботи	Термін виконання етапів роботи	Примітка
1	Огляд методів тестування навантаження ПЗ та БД	26.11.24-30.11.24	
2	Вибір та обґрунтування методики дослідження	02.12.24-05.12.24	
3	Вибір інструментальних засобів	06.12.24-10.12.24	
4	Розробка методів тестування	11.12.24-21.12.24	
5	Проведення експериментів	23-12.24-03.01.25	
6	Оформлення матеріалів кваліфікаційної роботи	04.01.25-07.01.25	
7	Подання кваліфікаційної роботи керівникові та	08.01.25-11.01.25	
	її попередній захист		
8	Подання кваліфікаційної роботи на рецензування	13.01.25-17.01.25	

Дата видачі завдання 25 листопада 2024 р.

Студент _____
(підпис)

Керівник роботи _____ доц. Федорченко В. М.
(підпис) (посада, прізвище, ініціали)

РЕФЕРАТ

Пояснювальна записка кваліфікаційної роботи: 118 с., 91 рис., 1 табл., 3 дод., 27 джерел.

БАЗА ДАНИХ, ВІРТУАЛЬНА МАШИНА, ВЕБ-ЗАСТОСУНОК, МЕТРИКИ, МАРШРУТИЗАТОР.

Основною метою кваліфікаційної роботи є підвищення якості програмного забезпечення.

Об’єктом дослідження є методи тестування програмного забезпечення та баз даних.

Предметом дослідження є тестування навантаження веб-застосунку та бази даних.

У ході виконання кваліфікаційної роботи проведено дослідження методів тестування навантаження веб-застосунку та бази даних за допомогою засобів платформи Azure.

Результатами дослідження є методи, алгоритми та програмне забезпечення аналізу тестування навантаження. Алгоритми реалізовані на основі програмного забезпечення, розробленого на хмарній платформі Azure в середовищі Microsoft Visual Studio мовою C#.

Отримані результати можуть бути впроваджені в подальшому поліпшені сервісів, додатків.

ABSTRACT

Master’s thesis: 118 pages, 91 figures, 1 tables, 3 appendices, 27 sources.

BASIS OF DATA, VIRTUAL MACHINE, WEB-APPLICATION, METRICS.

The main goal of the qualification work is to improve the quality of the software.

The object of research is software and database testing methods.

The subject of the research is the load testing of the web application and the database.

In the course of the qualification work, a study of web application and database load testing methods was conducted using Azure platform tools.

The results of the research are load testing analysis methods, algorithms and software. The algorithms are implemented on the basis of software developed on the Azure cloud platform in the Microsoft Visual Studio environment in the C# language.

The results obtained can be implemented in further improved services and applications.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

БД – база даних

ПЗ – програмне забезпечення

СУБД – система управління базами даних

ALM – управління життєвим циклом програми (англ., Aplication Lifecycle Management)

CaaS – комунікації як послуга (англ., Communication-as-a-Service)

HaaS – апаратне забезпечення як послуга (англ., Hardware-as-a-Service)

HTTP – протокол передачі гіпертекстових документів (англ., Hypertext Transfer Protocol)

IaaS – інфраструктура як послуга (англ., Infrastructure as a Service)

ODBC – відкритий прикладний програмний інтерфейс доступу до баз даних (англ., Open DataBase Connectivity)

PaaS – платформа як послуга (англ., Platform-as-a-Service)

RTMP – пропрієтарний протокол потокового передавання даних (англ., Real Time Messaging Protocol)

SaaS – програмне забезпечення як послуга (англ., Software as a Service)

SQL – мова структурованих запитів (англ., Structured Query Language)

TFS – центр організації спільної роботи (англ., Team Foundation Server)

ВСТУП

Хмарні технології – це концепція обробки даних, в якій обчислення і зберігання даних здійснюється в хмарі – віддаленому дата-центрі. Всі функції з надання та управління низькорівневою інфраструктурою бере на себе «хмара», повністю приховуючи ці деталі від користувача. Серед плюсів організації таких обчислень можна відзначити наступні:

Доступність. Послуга надається через Інтернет і є доступною цілодобово за умови функціонування сервера-провайдера.

Використання технології віртуалізації, що дозволяє зробити обчислювальні ресурси автономними і взаємно незалежними.

Можливість розширення, масштабованість. Обробка і зберігання даних здійснюється на серверах віддалених дата-центрів і розподілена по фізичним машинам, надаючи рівно такі потужності, які необхідні користувачу.

Умови оплати за передплатою і за фактичний рівень спожитих ресурсів («pay as you go») – споживач платить рівно за ту кількість роботи обчислювальних потужностей, скільки їм було використано. Крім того, така організація оплати істотно скорочує рівень капітальних витрат, що критично важливо при виборі технології для стартапу.

Легкість у використанні. Користувачеві не потрібно піклуватися про інфраструктуру і супроводі процесів обробки і зберігання даних.

Надання в користування обчислювальних потужностей і баз даних дата-центру може здійснюватися в декількох варіантах: SaaS, PaaS, HaaS, IaaS, CaaS – as a Service – в якості Інтернет-сервісу. S, P, H, I, C – Software, Platform, Hardware, Infrastructure, Communication – відповідно, програмне забезпечення, платформа, апаратне забезпечення, інфраструктура, комунікації. Вибір варіанту використання обчислювальних потужностей безпосередньо залежить від цілей діяльності.

Один із постачальників хмарних технологій – компанія Microsoft, яка представляє на ринку хмарну платформу Windows Azure Platform. Ця платформа пропонує обчислювальні ресурси своїх дата-центрів, які керуються хмарною операційною системою Windows Azure, а також віддалену базу даних. Особливо важливо, що розробники .NET можуть розгорнути свої ASP.NET додатки в хмарі, використовуючи вже наявні знання та навички роботи з .NET.

Основною метою кваліфікаційної роботи є підвищення якості програмного забезпечення.

Об’єкт дослідження: тестування програмного забезпечення та БД.

Предмет дослідження: тестування навантаження веб-застосунку та БД з використанням хмарної платформи Azure.

Результат дослідження: інструменти та ПЗ тестування навантаження. Алгоритми реалізовані на основі ПЗ, розробленого засобами Azure в середовищі Visual Studio мовою C#.

1 АНАЛІЗ МЕТОДІВ ТА ІНСТРУМЕНТАЛЬНИХ ЗАСОБІВ ТЕСТУВАННЯ НАВАНТАЖЕННЯ ВЕБ-ЗАСТОСУНКІВ І БАЗ ДАНИХ

1.1 Поняття тестування навантаження веб-застосунків і баз даних

Навантажувальне тестування допомагає виявити проблеми з продуктивністю програмного забезпечення, включаючи бази даних або веб-застосунки, що можуть спричинити збої в роботі системи, а також збільшити ризик відмов з боку користувачів [1].

Цей вид тестування належить до автоматизованого тестування, завдяки якому моделюється навантаження на систему для оцінки її стабільності, працездатності та масштабованості.

Навантажувальне тестування охоплює такі напрями [2]:

- тестування продуктивності – визначення здатності системи до масштабування за умов певного навантаження;
- тестування стабільності – перевірка роботи веб-застосунку або програми за умов середнього навантаження протягом тривалого періоду експлуатації;
- стрес-тестування – оцінка надійності системи за пікових навантажень та визначення швидкості її повернення до нормальних параметрів після зняття стресу;
- об'ємне тестування – оцінка масштабованості системи при збільшенні обсягу даних у базі даних;
- тестування масштабованості – перевірка здатності системи стабільно функціонувати при зростанні навантаження без збільшення часу відгуку та витрат на проєкт.

Навантажувальне тестування проводять на різних етапах ALM: під час розробки перед запуском продукту, при випуску нових версій, для оптимізації роботи застосунку на будь-якому етапі його використання, а

також для вибору відповідної технічної інфраструктури (програмної платформи та серверної конфігурації), здатної витримувати задане навантаження.

Цілі проведення:

- для оцінки можливостей та масштабованості програмної системи;
- для виявлення багів програмної системи, що приводять до відмов під час експлуатації;
- для визначення засобів, необхідних для якісної роботи системи.

1.2 Огляд методів тестування навантаження веб-застосунків і баз даних

Функціонального та простого навантажувального тестування вже недостатньо для перевірки веб-застосунків. Воно не забезпечує повної впевненості в їхній працездатності за реальних умов експлуатації, що вимагає використання більш складних і точних методів.

З появою Web2.0 веб-застосунки стали інтерактивними: вони здатні розпізнавати користувачів і адаптуватися до їхніх очікувань. Водночас користувачі стали більш вимогливими, очікуючи миттєвих і точних результатів. Це спричинило значне зростання навантаження на веб-сервери та підвищення вимог до якості їхнього обслуговування. Затримки або помилки у роботі веб-застосунків тепер безпосередньо впливають на втрату користувачів.

Для більшості сучасних компаній працездатність веб-застосунків стала критично важливим фактором. Помилки у визначенні їхньої реальної продуктивності можуть мати серйозні наслідки. Тому навантажувальне тестування є вкрай актуальним і необхідним етапом у забезпеченні якості роботи веб-застосунків. [3].

Сьогодні застосовуються два основних напрямку до автоматизації навантажувального тестування та створення тестового навантаження.

Створення множинних копій браузерів.

У цьому підході дії користувача виконуються в автоматичному режимі за допомогою великої кількості одночасно запущених копій браузерів. Це дозволяє максимально точно відтворити реальний сеанс взаємодії користувача із системою. Однак такий метод потребує значних обчислювальних ресурсів. Зокрема, на комп'ютері стандартної архітектури можна одночасно запустити не більше сотні браузерів, що працюють паралельно. Для створення істотного навантаження доводиться використовувати цілу мережу комп'ютерів.

Емуляція HTTP-запитів.

У цьому випадку замість реальних дій користувачів емулюються лише їхні запити до системи. Такий підхід дозволяє створити тисячі віртуальних користувачів на одному комп'ютері, що значно зменшує потребу в ресурсах порівняно з використанням браузерів.

Ці напрями мають свої переваги та обмеження й вибір залежить від специфіки тестування та цілей, які необхідно досягти.

Продукти для навантажувального тестування можна поділити на три категорії:

Безкоштовні рішення.

До цієї групи належить, наприклад, Apache JMeter [4]. Хоча цей продукт поширюється безкоштовно, загальна вартість його використання може бути досить високою через необхідність залучення дорогих фахівців. До того ж JMeter не має офіційної технічної підтримки, а створення тестових сценаріїв займає багато часу.

Корпоративні продукти з широким функціоналом.

До цієї категорії входять такі інструменти, як HP LoadRunner, IBM Rational Performance Tester, Borland Silk Performer [5, 6]. Вони дозволяють автоматизувати процес навантажувального тестування, значно підвищуючи його ефективність і якість. Наприклад, LoadRunner пропонує практично всі необхідні функції для різних типів тестування, задовольняючи навіть

найвибагливіших користувачів. Проте висока вартість продукту і тривалий час, необхідний для вивчення його можливостей, можуть стати перешкодою.

Недорогі універсальні рішення.

Сюди входять інструменти, як-от PureLoad [7], NeoLoad, WAPT [7-9]. Ці продукти відрізняються простотою встановлення та налаштування, що робить їх зручними для виконання навантажувального тестування веб-застосунків незалежно від операційної системи, під якою вони працюють.

Кожна з цих категорій відповідає певним потребам і ресурсам, що дозволяє обирати рішення залежно від бюджету, функціональних вимог та рівня складності тестування.

Проаналізуємо методи тестування навантаження [3, 10].

Динамічні сесії.

Сучасні веб-застосунки є багаторівневими динамічними системами, що включають мережевий екран, балансувальник навантаження, кілька веб-серверів, сервери додатків і БД. Для забезпечення індивідуальної роботи з кожним користувачем такі застосунки підтримують персоналізовані сесії та обробляють постійно змінювані дані.

Ця гнучкість накладає особливі вимоги на тестування: тести для перевірки працездатності веб-застосунків також повинні адаптуватися до змін "на льоту". Це призвело до впровадження нового методу – тестування, керованого даними.

Вимоги до тестування, керованого даними:

- кожен віртуальний користувач повинен мати унікальний набір тестових даних, щоб для веб-застосунку він виглядав як унікальний відвідувач;

- тестовий сценарій повинен уміти обробляти індивідуальні відповіді сервера, які призначені конкретному користувачу;

- управління тестовими даними має бути динамічним і відповідати реальним умовам роботи системи.

Цей підхід дозволяє створити більш реалістичне навантаження, адекватно оцінюючи роботу веб-застосунку за умов взаємодії з численними унікальними користувачами..

Автопараметризація.

У сучасній розробці веб-застосунків дедалі частіше використовується методологія гнучкої розробки [11], яка передбачає поділ процесу розробки на серію коротких ітерацій (Agile). Кожна з них включає всі етапи – від формування вимог до отримання робочої версії продукту, яка реалізує ці вимоги.

Оскільки в рамках Agile зміни в систему вносяться регулярно, навантажувальні тести для веб-застосунку також доводиться постійно оновлювати та адаптувати. Це висуває такі вимоги до інструментів навантажувального тестування:

- швидке створення тестових сценаріїв, інструмент має забезпечувати можливість оперативної розробки та модифікації тестів для врахування змін у функціоналі системи;

- автопараметризація, тестові сценарії повинні автоматично адаптуватися до змін у даних, що використовуються, та динамічних параметрів, таких як ідентифікатори сесій або змінні запитів;

- гнучкість у налаштуванні, інструмент має дозволяти легко оновлювати існуючі сценарії без значних витрат часу й ресурсів.

Автопараметризація дозволяє автоматизувати процес підготовки тестових сценаріїв, знижуючи трудомісткість їхнього оновлення, що особливо актуально за умов швидких і частих змін у рамках Agile.

Валідація відповідей.

Сучасні веб-застосунки, у разі виникнення помилки, зазвичай не передають користувачеві стандартний код помилки. Натомість вони надають повідомлення про нештатну ситуацію, що вимагає перевірки коректності виконання ключових відповідей сервера.

Для цього інструменти навантажувального тестування повинні забезпечувати можливість:

- задавати критерії для перевірки правильності відповідей сервера, наприклад, шляхом пошуку очікуваних текстових фрагментів, кодів статусу HTTP, структури JSON або XML, а також інших ключових параметрів;
- автоматично ідентифікувати помилки у відповідях, які можуть бути критичними для користувацького досвіду;
- підтримувати гнучкі механізми валідації, щоб перевірки можна було легко налаштувати під специфіку конкретного веб-застосунку.

Валідація відповідей є необхідною для забезпечення точності тестування, оскільки дозволяє переконатися, що програмна система правильно реагує на дії користувачів навіть за умов високого навантаження.

Тестування на неадекватність системі.

Зазвичай навантажувальне тестування проводиться в програмному середовищі, яке може відрізнятися від реального робочого. Це може стосуватися різних аспектів, таких як ресурси, засоби балансування навантаження та інші параметри. Це створює поширену проблему невідповідності між тестовою і робочою системою.

Часто через обмеження в тестовому середовищі, зокрема через недостатні ресурси або обмеження самих інструментів для навантажувального тестування, тести проводять на менш потужному обладнанні, моделюючи меншу кількість користувачів, ніж це буде в реальній робочій системі. В таких випадках частина підсистем, наприклад, що відповідають за динамічне виділення ресурсів або балансування навантаження, може не брати участі в тестуванні. Це призводить до неадекватної оцінки продуктивності системи, і деякі проблеми можуть залишитися не виявленими.

Також важливо враховувати «географічний розподіл навантаження», оскільки реальні користувачі можуть мати значно різні швидкості з'єднання та часи відгуку в залежності від їхнього місця розташування. Ці фактори

можуть суттєво впливати на ресурси сервера, оскільки повільні користувачі займають з'єднання довше, що збільшує навантаження на сервер і вимагає більше пам'яті для збереження даних.

Щоб уникнути цієї проблеми, під час розробки тестів необхідно забезпечити, щоб:

- навантаження, яке створюється під час тестування, було максимально наближеним до реального, тобто еквівалентним робочій системі;

- тестова система повинна мати ту ж продуктивність і використовувати ті ж підсистеми, що й робоча система, для забезпечення коректних результатів.

Тільки в такому випадку можна отримати точну оцінку ефективності програмної системи в реальних умовах.

Тестування в хмарах.

Сучасні високонавантажені веб-застосунки зазвичай є розподіленими системами, які часто розміщуються в хмарних середовищах. Для ефективного тестування таких веб-сайтів необхідно використовувати порівнянні за потужністю ресурси, що робить «тестування навантаження в хмарах» доцільним і зручним варіантом [12].

Основні переваги тестування в хмарах:

- гнучкість у ресурсах, у хмарному середовищі легко масштабувати ресурси тестування, що дозволяє моделювати різні рівні навантаження, від низького до високого, відповідно до вимог;

- тестування з різних точок доступу, можна проводити як з-за меж периметра веб-сайту (імітуючи реальних користувачів з різних регіонів), так і зсередини системи (перевіряючи роботу компонентів всередині інфраструктури);

- аналіз впливу брандмауера та балансувальника навантаження, перебуваючи в хмарах, можна оцінити, як брандмауер і балансувальник

навантаження впливають на продуктивність системи, що є критично важливим для забезпечення стабільності при реальному навантаженні.

Тестування в хмарах дозволяє забезпечити більш точну і надійну перевірку веб-застосунків у реальних умовах, максимально наближених до їхньої роботи в робочому програмному середовищі.

Моніторинг продуктивності.

Тестування навантаження не обмежується лише визначенням загальної продуктивності системи, а також включає виявлення її «вузьких місць». Для цього необхідно проводити моніторинг продуктивності тестованої програмної системи, що дозволяє отримати точні дані про її роботу в умовах навантаження.

Основні аспекти моніторингу включають:

- завантаження процесора, вимірювання CPU дозволяє визначити, чи не перевантажується програмна система, що може свідчити про недостатність обчислювальних потужностей;

- використання пам'яті, моніторинг споживання пам'яті допомагає виявити потенційні проблеми з її оптимальним розподілом чи витоками пам'яті;

- завантаження дискової підсистеми, важливо стежити за швидкістю читання/запису на диск, оскільки низька швидкість або перевантаження дискової системи може стати причиною затримок у роботі програмної системи;

- утилізація мережевих інтерфейсів, аналіз мережевого трафіку дозволяє виявити проблеми з пропускною здатністю або з затримками в мережі;

- продуктивність веб-серверів і серверів баз даних, моніторинг цих компонентів дозволяє точно визначити, чи відповідають вони вимогам до навантаження та де виникають потенційні проблеми, що уповільнюють програмну систему.

Збір та аналіз цих даних під час тестування дозволяє не тільки оцінити загальну продуктивність, але й знайти вузькі місця, які можуть негативно впливати на ефективність роботи програмної системи в реальних умовах.

Сьогодні на ринку існує ряд універсальних та досить простих засобів навантажувального тестування, які дозволяють більшості розробників і тестувальників створювати надійні веб-застосунки. Одним із таких прикладів є сімейство продуктів WAPT, можливості яких типові для рішень цього класу.

Продукти WAPT мають низку ключових особливостей:

- локальне тестування для невеликих навантажень, до 2 тис. віртуальних користувачів. Це дозволяє проводити базові тести продуктивності на етапі розробки та перевірки;

- розподілене тестування для моделювання високих навантажень, що підходить для більш масштабних перевірок на великих обсягах трафіку;

- засоби для «тестування в хмарах і з хмар», це дає можливість провести тестування на реальних умовах, враховуючи можливості масштабування ресурсів;

- продукти підтримують тестування для будь-якого веб-застосунку, що працює з протоколом HTTP(S), зокрема для класичних веб-застосунків, хоча і не підтримують деякі пропрієтарні протоколи, наприклад RTMP.

Ці засоби дозволяють швидко і ефективно організувати навантажувальне тестування для більшості сучасних веб-застосунків без необхідності в складних налаштуваннях чи дорогих ресурсах.

Для створення профілю віртуального користувача в рамках навантажувального тестування достатньо виконати всі його дії в обраному браузері. Спеціальний рекордер автоматично фіксує послідовність запитів, що відправляються браузером, і параметризує динамічні дані, щоб ці запити були унікальними для кожного користувача.

Параметризація даних у запитах дозволяє обробляти такі елементи, як:

- заголовки запитів;

- файли cookie;
- параметри запитів в URL і в тілі запиту.

Це дає змогу створювати індивідуальні запити для кожного віртуального користувача, що дозволяє моделювати більш реалістичне навантаження на систему.

Параметризація проводиться за допомогою набору функцій, які дозволяють здійснювати ці операції без написання коду. У разі більш складних ситуацій, коли стандартних функцій недостатньо, можна використовувати скриптову мову для реалізації додаткової логіки.

Щоб забезпечити різноманітність поведінки віртуальних користувачів, використовуються різноманітні оператори для реакції на відповіді сервера та для зміни поведінки користувача. Це включає: цикли, розгалуження, рандомізацію. Ці механізми дозволяють створювати користувачів з різною поведінкою, що значно наближає навантажувальне тестування до реальних умов використання веб-застосунку.

Валідація відповідей здійснюється за двома основними критеріями: часом та вмістом відповіді.

Валідація за часом. Цей тип валідації застосовується в тих випадках, коли є жорстко визначені тимчасові характеристики відповіді. Наприклад, якщо відповідь від сервера повинна надходити протягом певного часу, то валідація перевіряє, чи відповідає цей час вказаним вимогам.

Валідація по вмісту. Для перевірки правильності виконання запиту сервером здійснюється перевірка вмісту відповіді. Це можна зробити, шукаючи певні ключові слова або фрази в тексті відповіді, які підтверджують, що запит був оброблений коректно. Крім того, завжди є можливість виконати перевірку вручну за допомогою мови програмування, щоб детальніше проаналізувати відповіді.

Розподілене навантаження забезпечується через систему, в якій є «робоче місце» для запуску тестів та агенти, що генерують навантаження.

Агенти можуть бути розміщені в будь-якому місці, а для їх роботи потрібні лише два основні ресурси:

- доступ до тестованого веб-застосунку;
- зв'язок з «робочим місцем» користувача.

Це дає можливість масштабувати навантаження, моделюючи необхідну кількість одночасних користувачів. Крім того, існує можливість налаштувати індивідуальну швидкість віртуальних користувачів, що дозволяє імітувати, наприклад, роботу користувачів з повільними з'єднаннями (наприклад, через сотову мережу 3G), щоб вивчити вплив таких умов на роботу додатка.

Продукти для навантажувального тестування повинні підтримувати можливість розміщення в хмарах, таких як Amazon Web Services (AWS), що дозволяє забезпечити географічно розподілене навантаження. Оскільки центри обробки даних Amazon розташовані по всьому світу, можна вибрати найбільш зручне місце для проведення тестування, щоб моделювати різні умови доступу до сервера з різних географічних точок. Це дає змогу протестувати, як веб-застосунок буде реагувати на запити з різних регіонів і континентів.

Для покращення якості важливо мати можливість моніторингу продуктивності веб-серверів. Для цього можна використовувати такі протоколи, як Windows Management Instrumentation та Simple Network Management Protocol. Вбудовані лічильники продуктивності дозволяють в режимі реального часу відслідковувати різні показники:

- завантаження процесора;
- використання пам'яті;
- дискова підсистема;
- мережеві інтерфейси.

Це дає змогу виявляти вузькі місця в роботі серверів, які входять до складу веб-застосунку. Крім того, через Open Database Connectivity можна отримувати специфічні лічильники продуктивності для баз даних, що

дозволяє оцінити навантаження саме на бази даних, які часто є критичним елементом при обробці запитів.

Сучасні інструменти для тестування навантаження враховують всі ключові аспекти роботи сучасних веб-застосунків. Якщо є обмеження у бюджеті або достатній час на тестування, можна обирати безоплатні продукти, які дозволяють провести ефективне тестування навіть при обмежених ресурсах. Однак для складніших завдань і більш масштабованих систем, особливо в корпоративному середовищі, можуть знадобитися інструменти корпоративного класу. Проте для більшості стандартних завдань універсальних продуктів цілком достатньо для проведення якісного навантажувального тестування.

1.3 Огляд інструментальних засобів тестування навантаження веб-застосунків і баз даних

Навантажувальні тести часто проводяться для визначення кількості користувачів що є передбачається, як максимальна, яку може обслуговувати сервер перед запуском продукту в продакшен. Такий підхід дозволяє оцінити, чи здатен сервер витримати очікувані навантаження, однак він не дає відповіді на важливе питання: чи можна збільшити кількість користувачів, яких може обслуговувати сервер, за рахунок оптимізації програмного коду або налаштувань сервера.

Хоча можна спробувати оптимізувати код перед запуском продукту, такий підхід часто не є оптимальним, оскільки він не має чітких метрик для оцінки ефективності змін. Для кращої оптимізації необхідно проводити навантажувальні тести постійно, не чекаючи кінця розробки. В ідеалі, тестування має розпочинатися з моменту готовності прототипу застосунку.

Проведення тесту на еталонному сервері в момент завершення етапу розробки каркаса дозволяє встановити базові метрики продуктивності. Ці дані будуть служити еталоном для подальших порівнянь. Якщо в процесі

розробки зміни в коді чи конфігурації призводять до значних відхилень від еталону, це є сигналом, що потрібно провести додаткові роботи для оптимізації. Шукати проблеми з продуктивністю на ранніх етапах значно зручніше, ніж намагатися виправити їх у кінці проекту.

Однією з ключових проблем проведення навантажувальних тестів є необхідність наявності відповідного обладнання для генерації навантаження. Щоб отримати коректні результати тестування, потрібно мати сервери, здатні відтворити навантаження, яке максимально наближене до реальних умов експлуатації. Для цього необхідно використовувати сервери з характеристиками, що відповідають або перевищують параметри еталонного сервера.

У реальності ж, часто виникають складнощі. Якщо сервер, який тестується, доступний тільки через зовнішній порт (наприклад, порт 80 для HTTP), а тестування проводиться через локальну мережу організації, то це може призвести до неадекватних результатів. Використання внутрішніх ресурсів для генерування навантаження, наприклад, старих серверів або серверів, що зазвичай не використовуються, також має свої мінуси.

До них відноситься:

- застарілі сервери можуть бути відключені для економії енергоспоживання, або вони можуть бути «непідтримуваними». Якщо сервер не є в робочому стані або більше не підтримується, це значно знижує точність навантажувальних тестів;
- витрачання внутрішніх ресурсів для тестування, наприклад, шляхом вимкнення застосунків або переведення серверів у режим, що не використовується, може призвести до помилкових результатів, оскільки це змінює реальні умови роботи організації та системи в цілому;
- різниця в середовищах може також призвести до того, що сервер, що тестується, працює в різних умовах, ніж в реальному робочому середовищі, що впливає на результати тестів.

Тому для отримання більш точних і репрезентативних результатів важливо використовувати платформи для навантажувального тестування, які відповідають реальним умовам і мають відповідні ресурси для створення точного навантаження.

Загалом, треба використовувати хмарні ресурси з оплатою по годинах. На сервери потрібно залити і налаштувати необхідне ПЗ для генерації навантаження.

Тестування засобами Visual Studio Team Services (VSTS) [14, 15]

Після створення безкоштовного акаунту в VSTS, зараз відомому як Azure DevOps, можемо створити новий проект, виконавши такі кроки:

Крок 1. Увійти в Azure DevOps.

Перейдіть на сайт Azure DevOps (https://dev.azure.com) і увійдіть за допомогою вашого облікового запису.

Крок 2. Створення нового проекту:

- після входу на панель управління Azure DevOps, натискайте «New Project» або «Create Project»;
- введіть назву вашого проекту в полі «Project Name».

Крок 3. Вибір системи контролю версій.

На цьому етапі вам потрібно вибрати, яку систему контролю версій ви будете використовувати:

- Git: Це розподілена система контролю версій. Виберіть Git, якщо ви плануєте працювати з Git-репозиторіями;
- Team Foundation Version Control: Це централізована система контролю версій. Виберіть її, якщо ви хочете використовувати централізовану модель контролю версій, яка доступна в Azure DevOps.

Крок 4. Інші налаштування:

- можна вказати доступність проекту (приватний або публічний);
- можна також вибрати область, де буде розміщений ваш проект (географічне розташування серверів).

Крок 5. Завершення створення проекту.

Натискаєте «Create», і ваш новий проект буде створено.

Після цього буде доступ до різних інструментів Azure DevOps для управління кодом, тестуванням, CI/CD та іншими аспектами вашого проекту (рисунок 1.1):

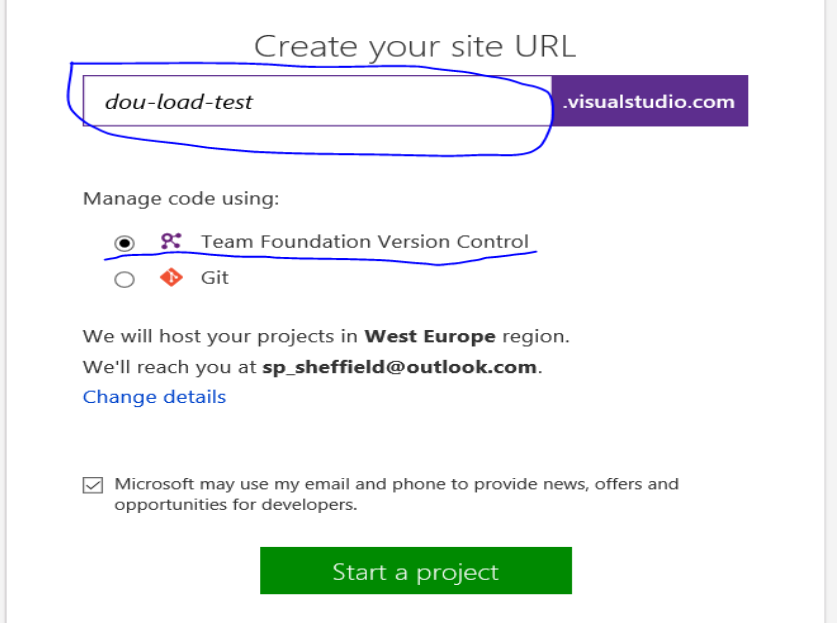

Рисунок 1.1 – Новий проект VSTS

Щоб прив'язати створений акаунт VSTS (Azure DevOps) до Visual Studio (VS) та з'єднати проект через Team Explorer, виконуються наступні кроки: відкрийте VS, запустіть її на вашому комп'ютері, відкрийте Team Explorer, увійдіть в акаунт Azure DevOps, підключиться до вашого проекту (виберіть ваш проект із списку, натисніть «Connect» або «Clone», якщо ви хочете клонувати репозиторій на локальну машину.

Після цього можна побачити проект у Team Explorer, отримувати і відправляти зміни в репозиторій, працювати з робочими елементами (work items), використовувати CI/CD та інші інструменти, доступні через Azure DevOps. Тепер ваш акаунт Azure DevOps прив'язаний до VS (рисунок 1.2-1.5):

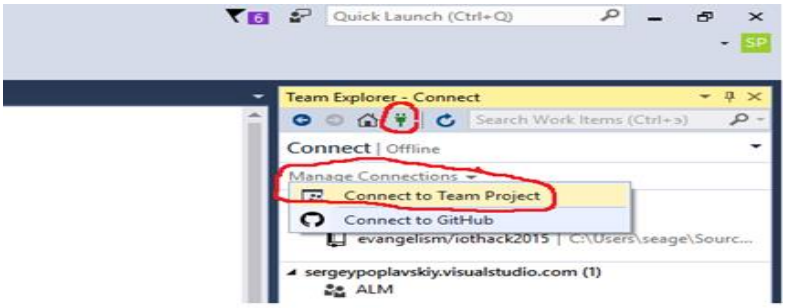

Рисунок 1.2 – Вікно з’єднання VS

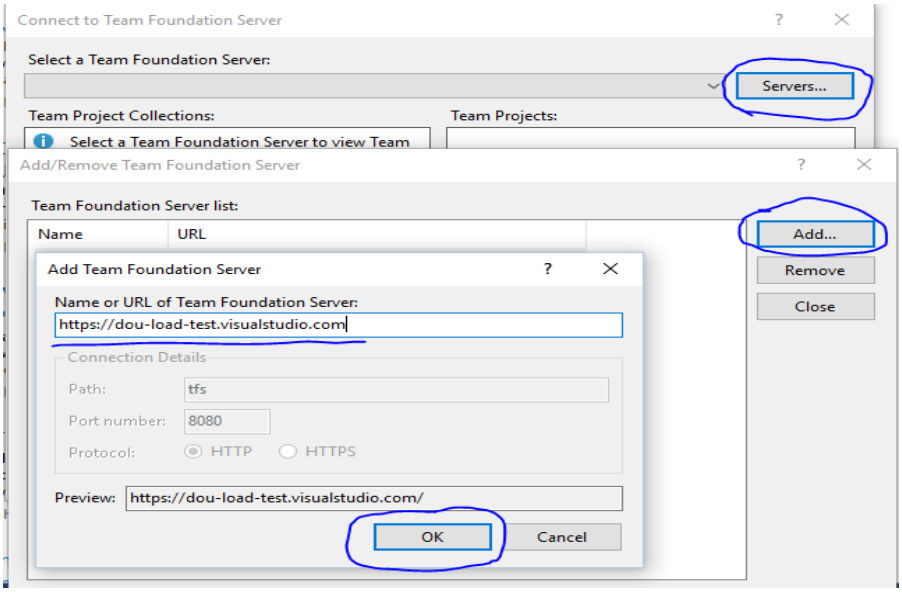

Рисунок 1.3 – Вікно налаштування з’єднання VS

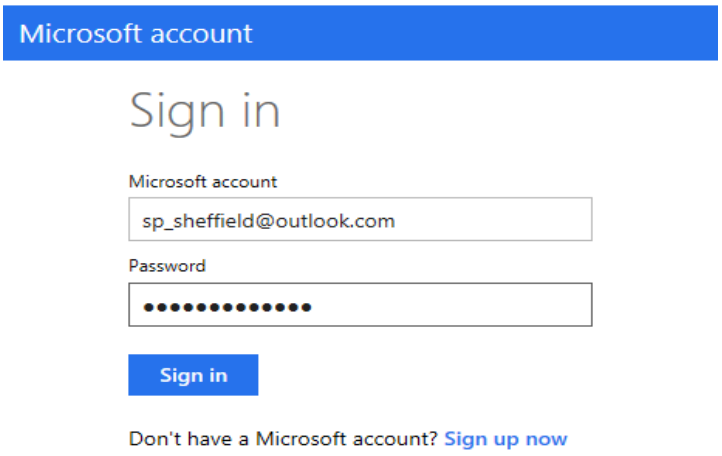

Рисунок 1.4 – Вікно підключення користувача в VS

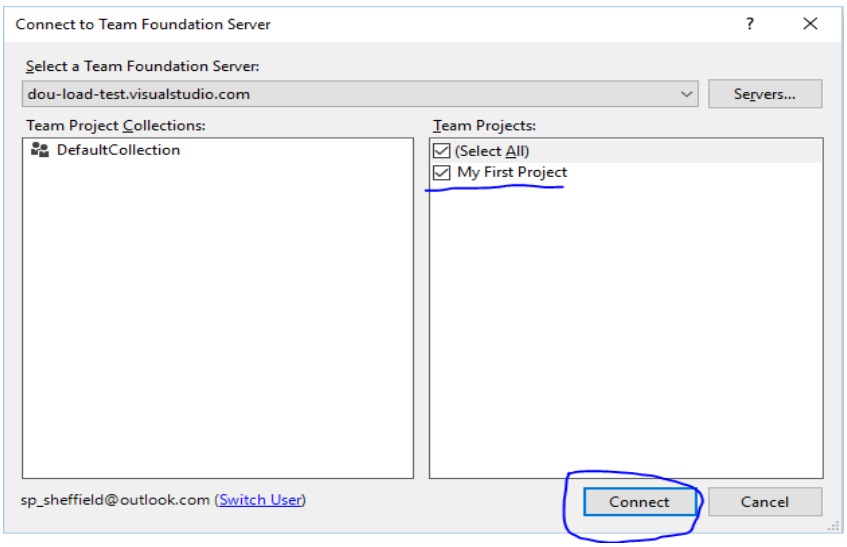

Рисунок 1.5 – Вікно вибору проєкту VS

Щоб створити новий проект для навантажувального тестування в Visual Studio, виконайте наступні кроки: створіть новий проект, виберіть шаблон для тестування «Web performance and Load Test Project», виберіть Web Performance and Load Test Project у розділі Test (Тестування), назвіть проект, виберіть шлях, де буде збережений проект, натисніть Create (Створити).

Налаштування проекту.

Після створення проекту у вас з'явиться спеціальне середовище для налаштування та виконання навантажувальних тестів.

У проекті будуть автоматично створені необхідні файли для тестування веб-застосунків:

- Web Performance Test для запису сценаріїв взаємодії з веб-застосунком;
- Load Test для налаштування навантаження та параметрів тестування.

Налаштування тестів.

Відкриється Test Explorer і Web Performance вікно, де ви зможете налаштувати тести за допомогою інтерфейсу або додати відповідні дії для запису користувацьких сесій.

Для тестування продуктивності, вам необхідно налаштувати параметри навантаження, такі як кількість віртуальних користувачів, час виконання тесту, а також моніторинг ресурсів сервера.

Тепер ваш проект для навантажувального тестування налаштовано, і ви готові створювати конкретні сценарії та виконувати тестування вашого веб-застосунку! (рисунок 1.6):

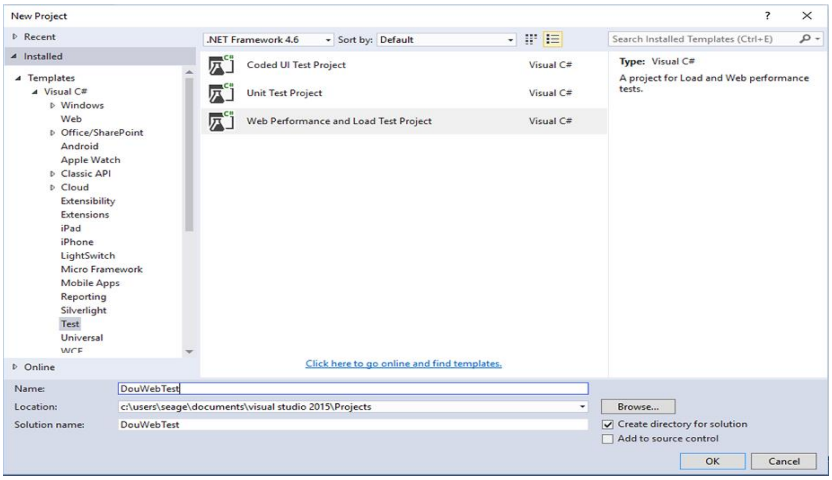

Рисунок 1.6 – Вікно створення тесту проєкта в VS

Після натискання на кнопку «ОК» з'явиться новий проєкт веб-тесту. Необхідно записати цей веб-тест. В веб-тесті використаємо кнопку «Add recording» (рисунок 1.7):

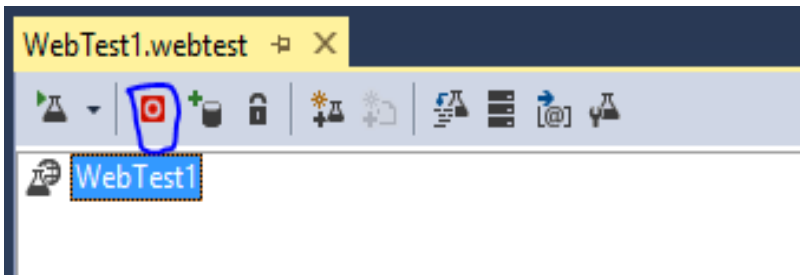

Рисунок 1.7 – Вікно запису тесту

У браузері завантажується веб-сайт (рисунок 1.8).

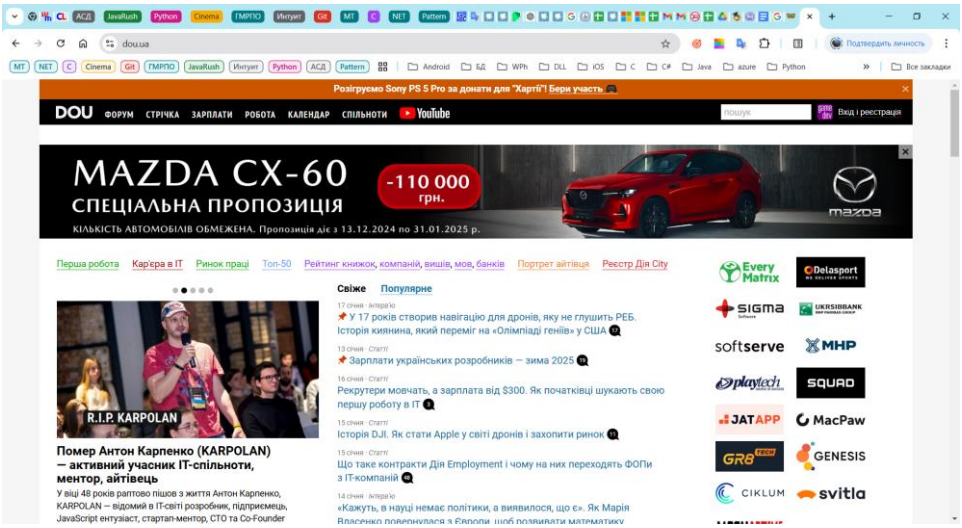

Рисунок 1.8 – Веб-сайт

Тиснемо «Stop recording» в VS.

Якщо нічого не змінилося (в веб-тесті не з'явився записаний url з набором атрибутів), необхідно включити аддон в веб-браузері «Microsoft Web Test Recorder Helper» (рисунок 1.9):

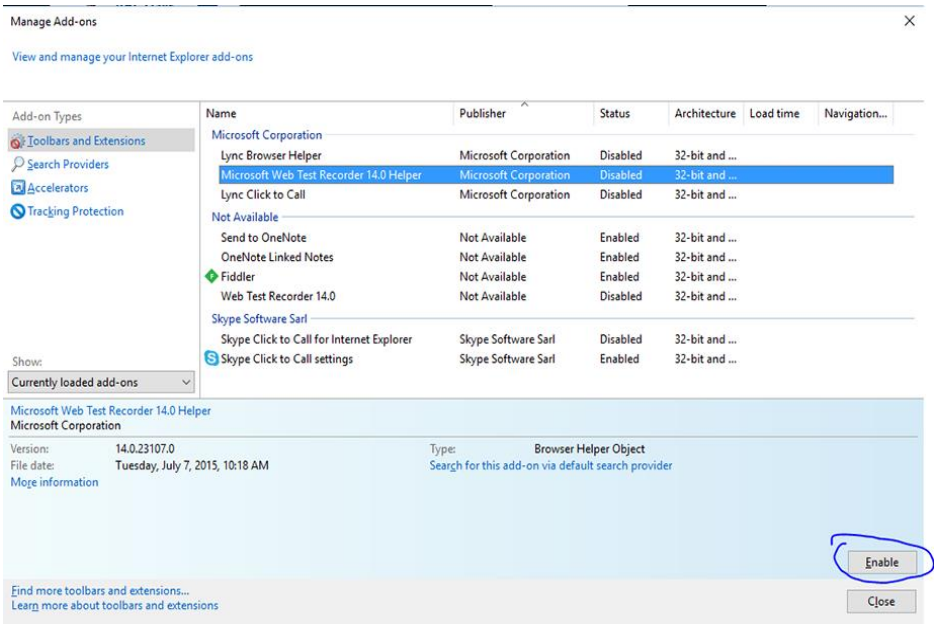

Рисунок 1.9 – Вікно включення аддону

У підсумку маємо (рисунок 1.10):

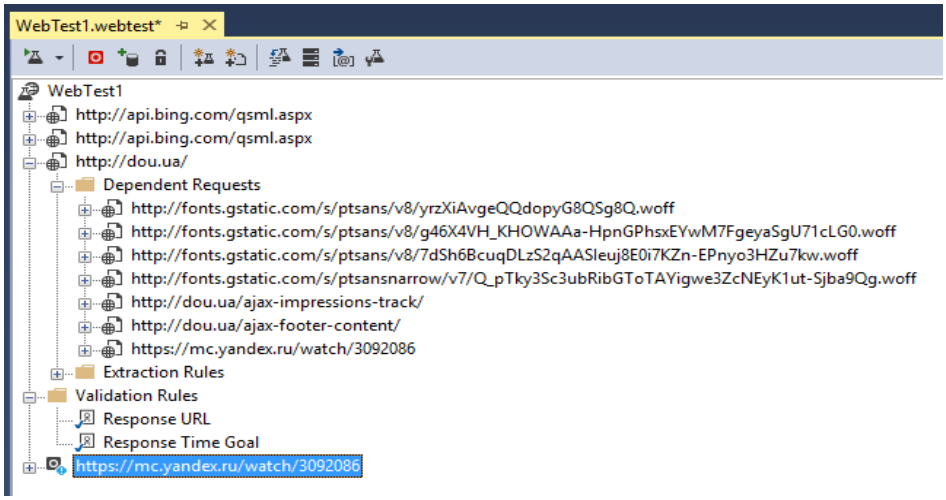

Рисунок 1.10 – Закінчення веб-тесту VS

Видалимо правила валідації VS (рисунок 1.11):

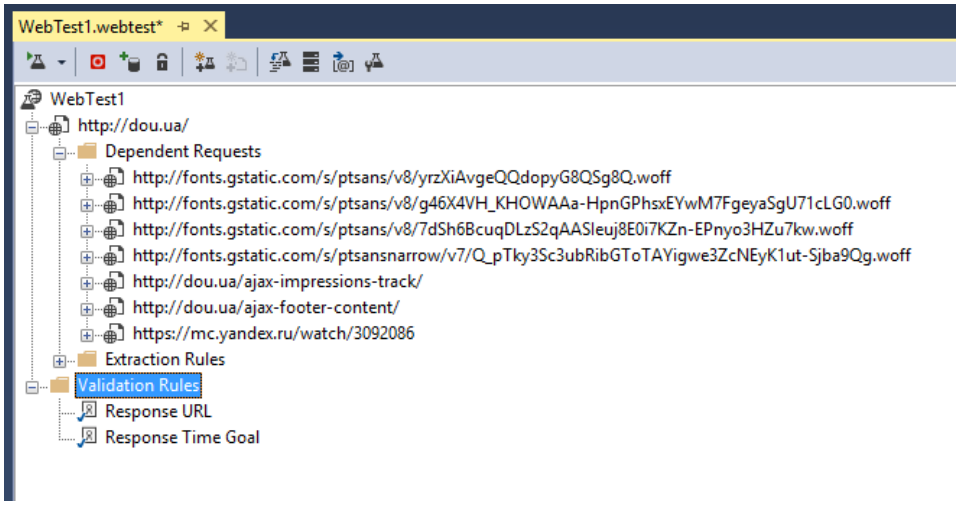

Рисунок 1.11 – Видалення правила валідації

Так, після того як веб-тест записано, додайте навантажувальний тест у вашому проекті VS.

Крок 1. Додайте Load Test:

- у Solution Explorer знайдіть проект;

- клацніть правою кнопкою миші на проект веб-тесту (Web Performance Test), щоб відкрити контекстне меню;

- виберіть Add → Load Test (Додати → Навантажувальний тест).

Крок 2. Налаштування Load Test.

Після того як додано Load Test до проекту, відкриється діалогове вікно для налаштування тесту:

- Select the Web Performance Test – виберіть веб-тест, який вже створений, як частина навантажувального тесту;

- Define Load Pattern – визначається, скільки віртуальних користувачів повинні бути змоделювані під час тестування;

- Load Pattern Options – виберіть потрібну кількість віртуальних користувачів, час їх запуску та час проведення тесту. Можна додавати різні варіанти, наприклад: Number of Virtual Users (Кількість віртуальних користувачів), Duration (Тривалість тесту).

Крок 3. Налаштування моніторингу.

У вікні Load Test можна додатково налаштувати моніторинг сервера:

- для моніторингу серверних ресурсів (CPU, пам'ять, диск, мережа) додайте Counter Set;

- можна підключити сервери через WMI, SNMP або інші монітори, якщо потрібно відстежувати продуктивність у реальному часі під час тестування.

Крок 4. Запуск тесту: натискайте Run у меню тестування, щоб запустити навантажувальний тест. Тест буде виконуватися з визначеною кількістю віртуальних користувачів, і можна побачити результати тестування в вікні Test Results.

Крок 5. Аналіз результатів: після завершення тесту ви можете переглянути різноманітні метрики та графіки, щоб оцінити продуктивність вашого веб-застосунку. Це включає середній час відповіді, кількість успішних запитів, час відгуку, кількість помилок тощо.

Таким чином, додавши Load Test до вашого проекту, ви зможете симулювати потрібне навантаження і проаналізувати ефективність вашого веб-застосунку за різних умов. (рисунок 1.12):

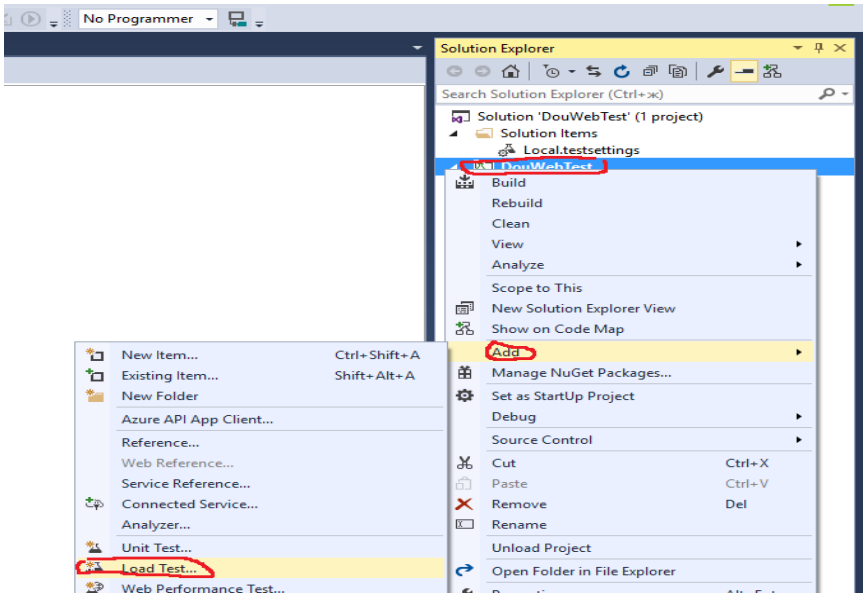

Рисунок 1.12 – Вікно додавання тесту з певною кількість користувачів

Після додавання Load Test у VS, відкриється Load Test Wizard, де потрібно виконати крокі для налаштування.

Крок 1: Вибір типу навантажувального тесту.

На першому кроці Load Test Wizard обираємо опцію:

Cloud-based Load Test with Visual Studio Team Services (Хмарне навантажувальне тестування з Visual Studio Team Services).

Цей варіант дозволяє запустити тест у хмарі, використовуючи ресурси Visual Studio Team Services (VSTS), що дає можливість тестувати з великою кількістю віртуальних користувачів без необхідності налаштовувати додаткові сервери для генерації навантаження.

Крок 2: Підключення до VSTS.

Після вибору варіанту хмарного тесту потрібно підключити проект до Visual Studio Team Services.

Обираємо відповідний акаунт та проект у VSTS, щоб зберегти дані тесту та відправити його на хмарний сервер для виконання.

Крок 3: Налаштування параметрів навантаження:

- Number of virtual users (Кількість віртуальних користувачів);
- Duration of test (Тривалість тесту).

Можна налаштувати також інші опції, наприклад, кроки навантаження чи різні варіанти трафіку.

Крок 4: Завершення налаштувань

Після завершення налаштувань необхідно натиснути Finish, щоб створити конфігурацію для тесту.

Тест буде відправлений у хмару для виконання, і можна слідкувати за його прогресом.

Цей підхід дозволяє використовувати потужності хмари для проведення тесту, що значно полегшує процес налаштування і дає можливість моделювати великі навантаження без потреби в локальному апаратному забезпеченні (рисунок 1.13-1.18):

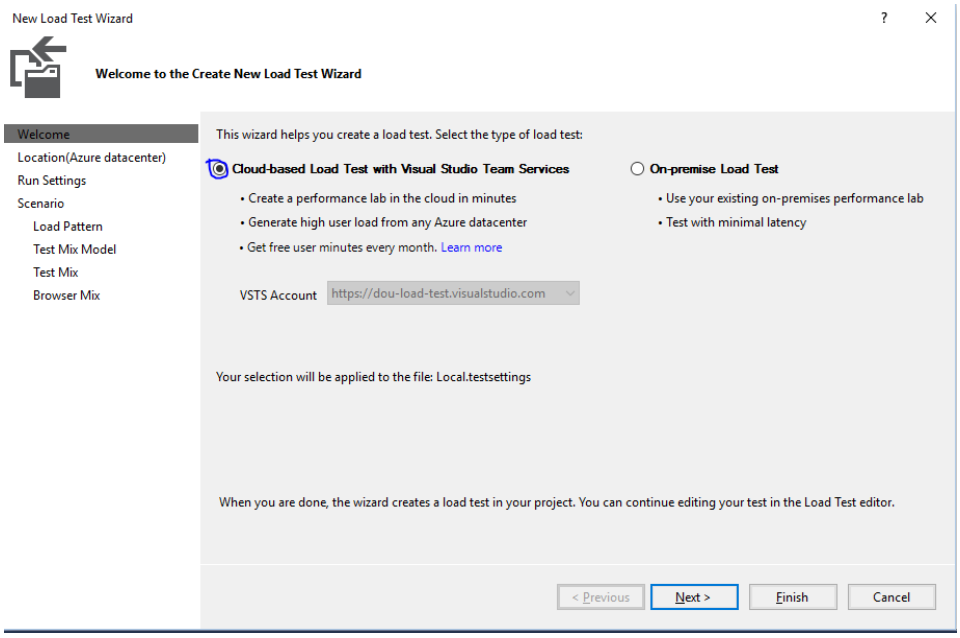

Рисунок 1.13 – Вибираємо «Cloud-based Load Test with Visual Studio Team Services»

Далі вибираємо локацію дата-центру, звідки буде генеруватися навантаження (рисунок 1.14 - 1.16):

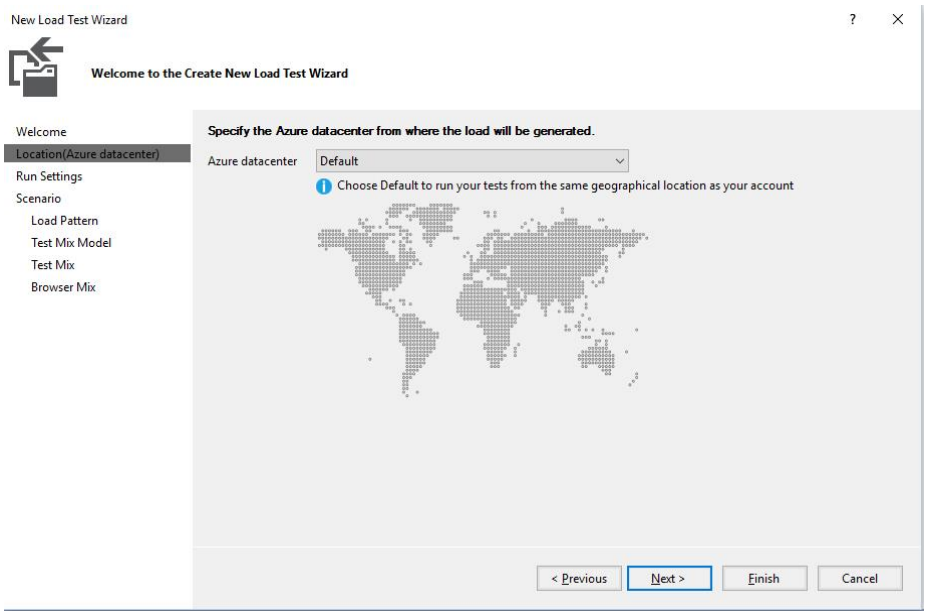

Рисунок 1.14 – Вікно локації дата-центру VS

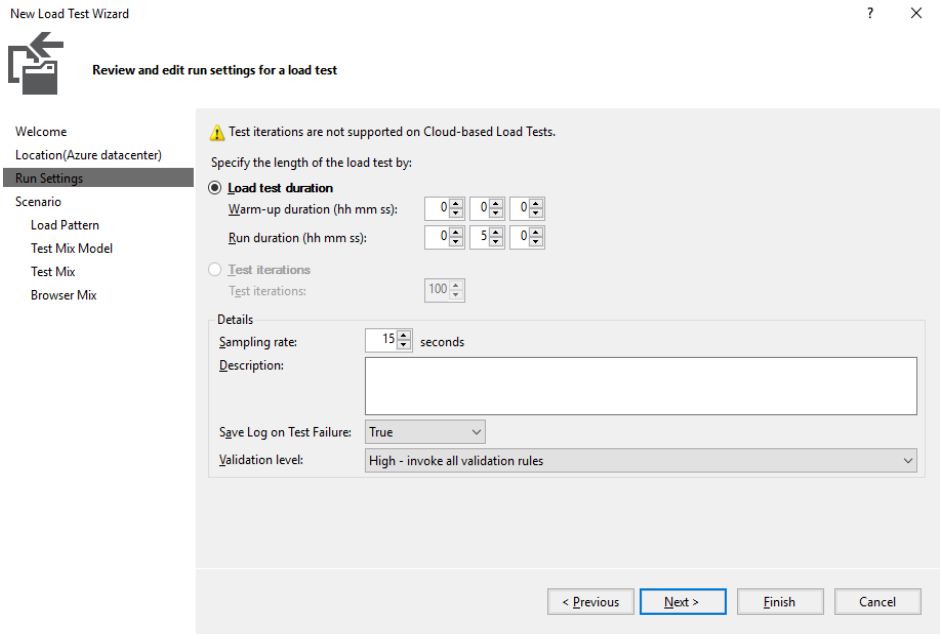

Рисунок 1.15 – Встановлення часу тесту VS

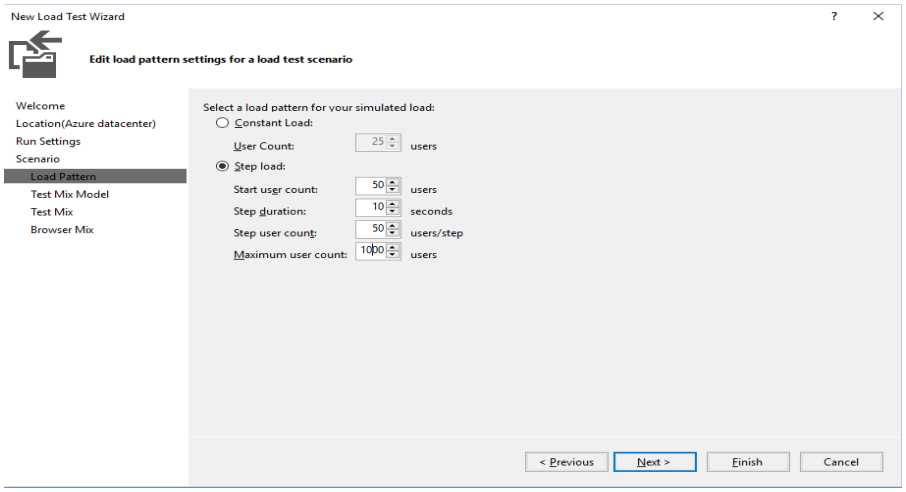

Рисунок 1.16 – Вікно налаштування зростання користувачів за заданим шаблоном VS

Навантажимо DOU тисячею одночасних користувачів. Далі в Test Mix додаємо веб-тест (рисунок 1.17 - 1.18):

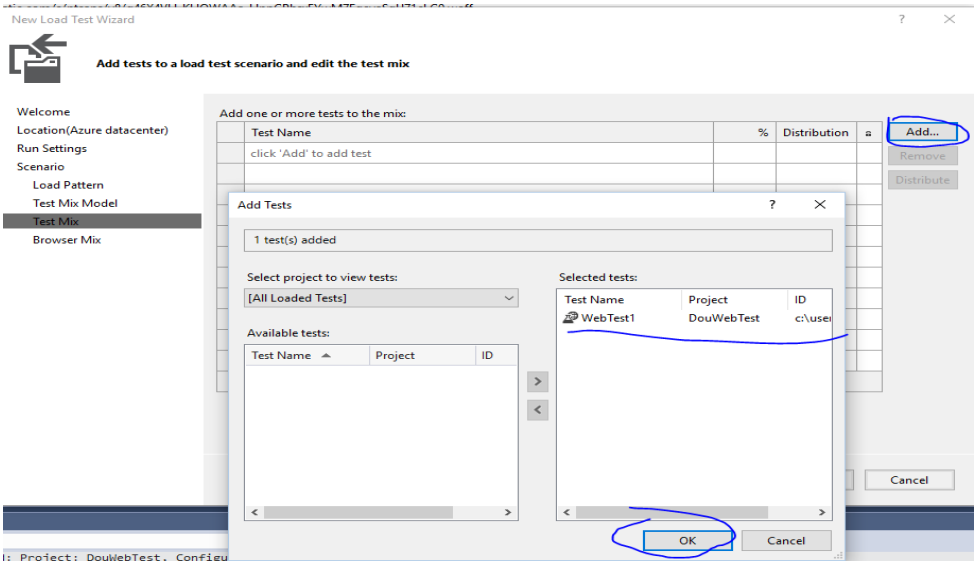

Рисунок 1.17 – Додавання веб-тесту

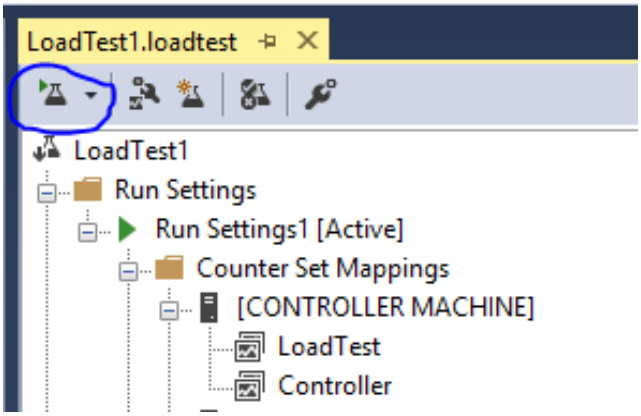

Рисунок 1.18 – Запуск веб-тесту у VS

Через певний час можна бачити графіки залежності часу відповіді від кількості одночасних користувачів (рисунок 1.19):

Рисунок 1.19 – Результати тестування веб-застосунку

По закінченню тесту можна завантажити в VS детальний звіт і подивитися, що відбувається. В даному конкретному випадку майже відразу почалися помилки. Ось статистика по всіх помилок тесту (рисунок 1.20):

Errors

Type	Subtype	Count	Last Message
Total		4,005	
Extraction Rule Error	ExtractHiddenFields	2,000	No hidden fields were found in the response.
Http Error	503 - ServiceUnavailable	2,000	503 - ServiceUnavailable
Exception	LoadTestErrorLimitExceededExce...	4	More than 1000 errors of type 'ExtractionRuleError' and sub t
Exception	WebException	1	The request was aborted: The request was canceled.

Рисунок 1.20 – Помилки тестування

Важливі помилки №503 (рисунок 1.21):

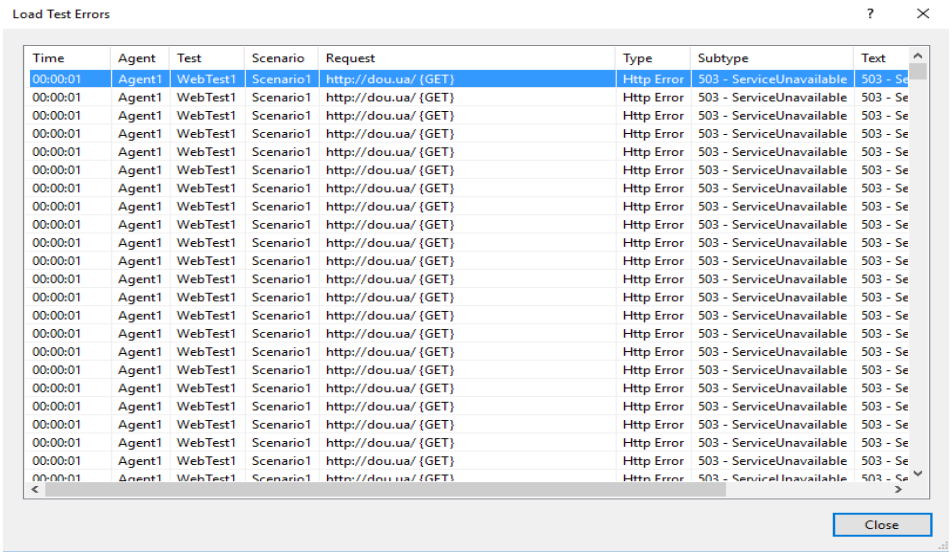

Рисунок 1.21 – Вікно помилок №503 в VS

Можна проаналізувати кожен запис окремо. Окрім того, детальні звіти можна вивантажити і проаналізувати в Excel.

Розглянемо альтернативні інструменти навантажувального тестування.

Apache HTTP server benchmarking tool [16]

Самий часто використовуваний, т. к. входить до складу Apache.

ab [options] [http [s]: //] hostname [: port] / path

де основні необхідні options :

-c concurrency – кількість одночасних запитів до сервера (за замовчуванням 1);

-n requests – загальна кількість запитів (за замовчуванням 1).

В результаті команди отримуємо звіт, який вказано в додатку Б:

Плюси:

- є скрізь, де є Apache;
- не потребує ніякої додаткової настройки;
- дуже простий інструмент.

Мінуси:

- дуже простий інструмент;
- тестує тільки продуктивність веб-сервера: опитує тільки один URL, не підтримує сценарії навантаження, неможливо імітувати призначену для користувача навантаження і оцінити працездатність проекту з усіх боків – як з точки зору інфраструктури, так і з точки зору розробки.

Joe Dog Siege [17]

Трохи складніше ab і необхідні завдання виконує набагато краще.

У файлі-сценарії задаються URL-и та запити тестування. Якщо сценарій великий за обсягом, то можна винести всі запити в окремий файл і в команді вказати цей файл при тестуванні:

```
 # Cat urls.txt
# URLS file for siege
# -
http://www.bitrix24.ru/
http://www.bitrix24.ru/support/forum/forum1/topic3469/?PAGEN_1=2
http://www.bitrix24.ru/register/reg.php POST domain = test & login = login
http://www.bitrix24.ru/search/ POST </ home / www / siege_post_data
# Siege -f urls.txt -c 10 -r 10 -d 3
```

У команді вказується кількість користувачів -c , кількість повторень -r і затримку між хітами -d .

Результат можна виводити в log-файл або відразу в консоль в режимі реального часу (див. додаток Б)

Також можна взяти з access-log веб-сервера URL-и, по яких ходили реальні користувачі і емулювати навантаження реальних користувачів.

Плюси:

- багатопотоковий;
- можна задавати як кількість запитів, так і тривалість (час) тестування – тобто можна емулювати призначену для користувача навантаження;
- підтримує найпростіші сценарії.

Мінуси:

- ресурсномісткий;
- мало статистичних даних і не дуже добре емулює такі призначені для користувача сценарії, як обмеження швидкості запитів користувача;
- не підходить для серйозного і масштабного тестування в сотні і тисячі потоків, та він сам по собі ресурсномісткий, а на великій кількості запитів і потоків дуже сильно навантажує систему.

Apache JMeter [4]

Основні можливості:

- написаний на Java;
- HTTP, HTTPS, SOAP, Database via JDBC, LDAP, SMTP (S), POP3 (S), IMAP (S);
- консоль і GUI;
- розподілене тестування;
- план тестування – XML-файл;
- може обробляти лог веб-сервера як план тестування;
- візуалізація результатів в GUI.

Результати виводяться в графічному вигляді (рисунок 1.22):

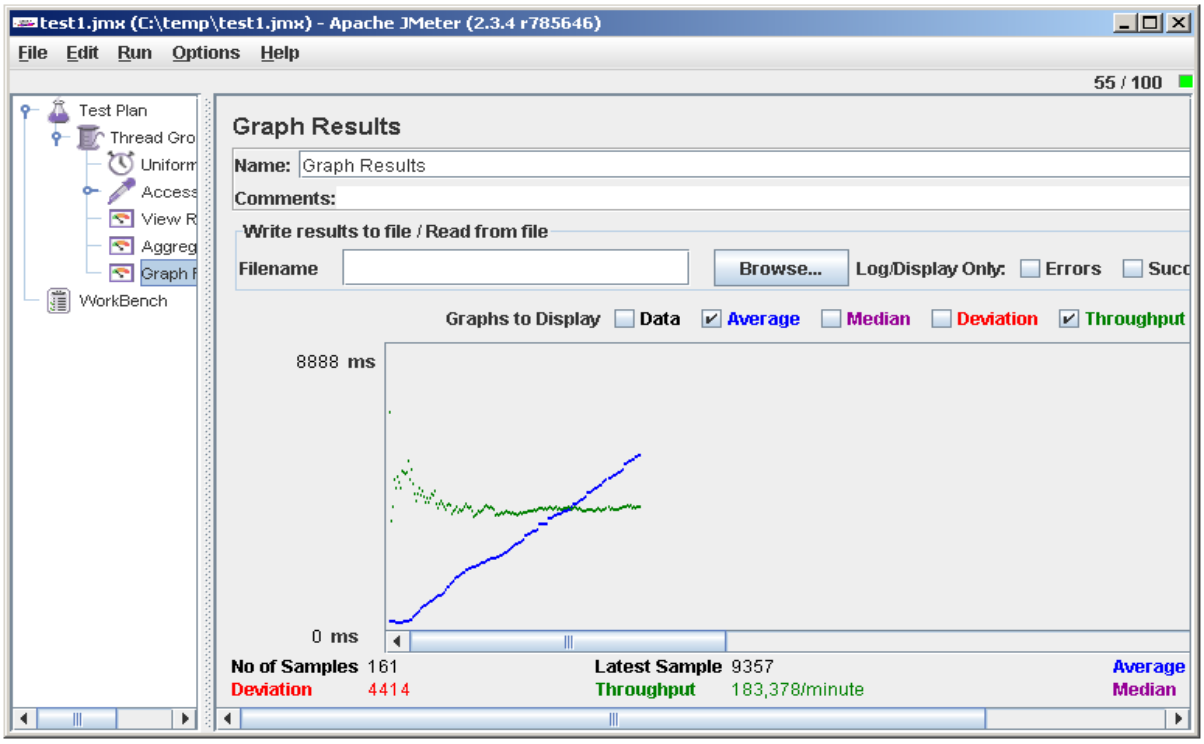

Рисунок 1.22 – Вікно Apache JMeter

Плюси:

- багатоплатформовий, т.я написаний на Java;
- дуже гнучкий, використовується багато протоколів, не тільки веб-сервер, але і бази;
- управляється через консоль і gui інтерфейс;
- використання безпосередньо логів веб-сервера Apache і Nginx в якості сценарію с можливістю варіювання навантаження за цими профілями;
- досить зручний і потужний інструмент.

Мінуси:

- ресурсномісткий;
- на тривалих і важких тестах часто падає з різних причин;
- стабільна робота залежить від оточення і конфігурації сервера.

Tsung [18]

Основні можливості:

- написаний на Erlang;

- HTTP, WebDAV, SOAP, PostgreSQL, MySQL, LDAP, Jabber / XMPP;
- консоль (GUI через сторонній плагін);
- розподілене тестування (мільйони користувачів);
- фази тестування;
- план тестування - XML;
- запис плану за допомогою Tsung recorder'a;
- моніторинг тестованих серверів (Erlang, munin, SNMP);
- інструменти для генерації статистики і графіків з логів роботи.

За допомогою власних скриптів, які обробляють логи роботи, можна виводити різні звіти по тестуванню (рисунок 1.23 - 1.24):

- tsung_stats.pl:

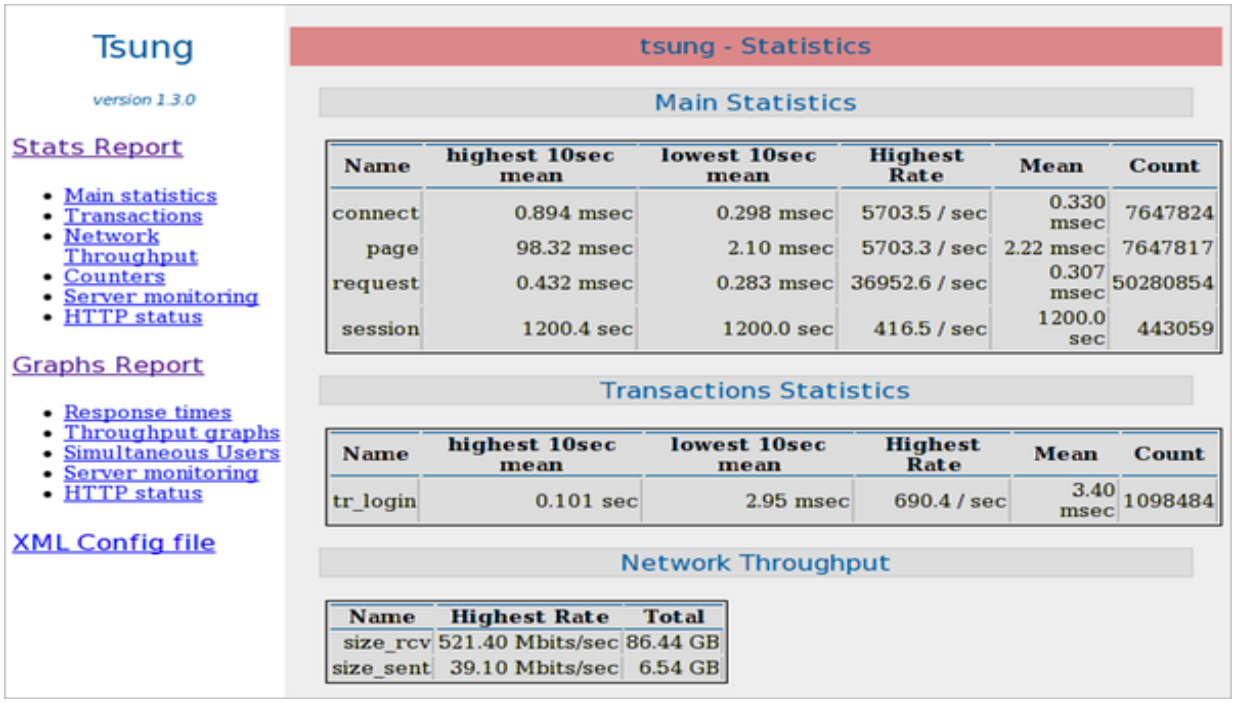

Рисунок 1.23 – Вікно tsung_stats.pl

- tsung_stats.pl + Gnuplot:

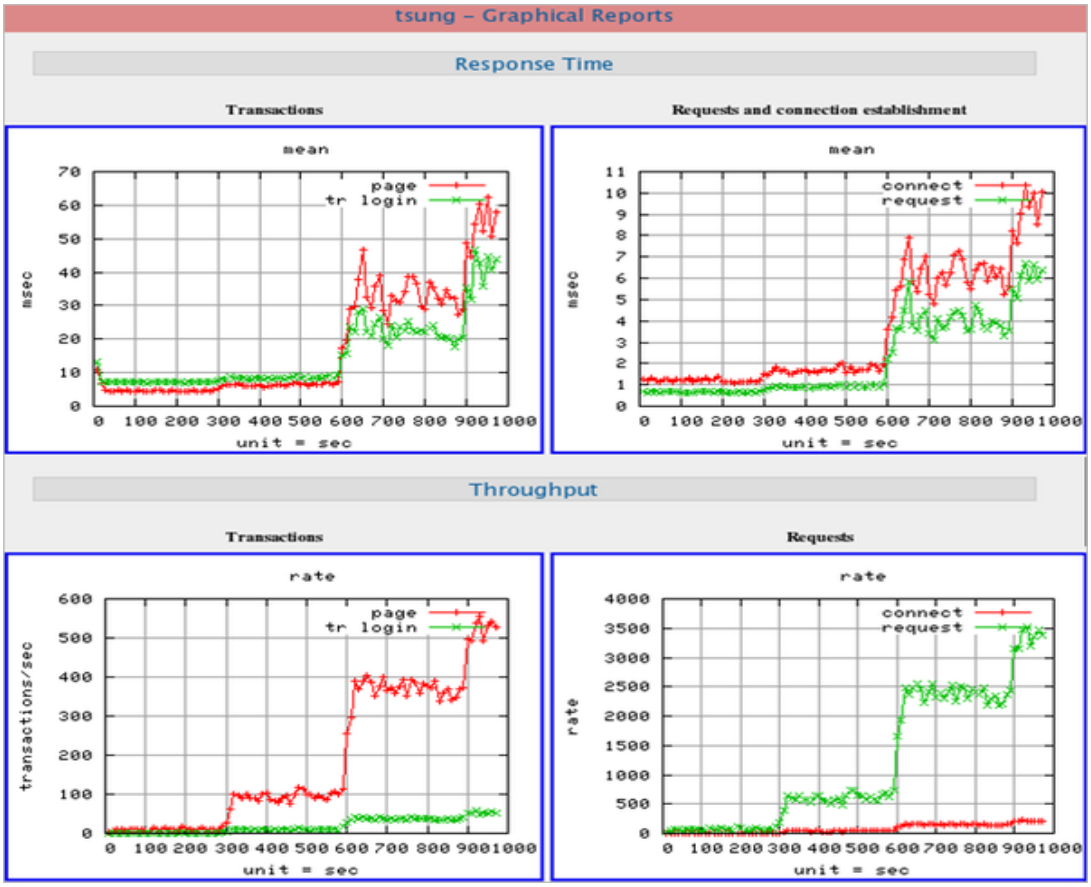

Рисунок 1.24 – Вікно tsung_stats.pl

Плюси:

- дозволяє проводити аналіз різних параметрів продуктивності, що допомагає виявити вузькі місця системи;

- є проектом з відкритим кодом, у нього є активна спільнота користувачів і розробників, які регулярно додають нові функції, виправляють помилки і підтримують інструмент;

- ефективно використовує доступні ресурси на сервері, і навіть одна інстанція може генерувати значні навантаження. Це дає можливість тестувати без потреби в додаткових дорогих серверах;

- дозволяє розподіляти навантаження між кількома машинами, що робить його особливо корисним для великих тестів з великою кількістю віртуальних користувачів;

- велика кількість тестованих систем – не тільки веб-сервери і БД, а й, наприклад, XMPP-сервер: може відправляти повідомлення, тести з авторизацією;

- можна інтегрувати в процеси автоматизованого тестування, що дозволяє запускати навантажувальні тести як частину CI/CD пайплайнів;

- наявність в комплекті інструменту Tsung Recorder – свого роду, proxy-сервер, через який можна ходити по тестованому сайту і записувати відразу як профіль навантаження;

- надає детальні звіти з результатами тестування, що дозволяє зібрати метрики про час відгуку, пропускну здатність і помилки на сервері. Звіти можна експортувати у різні формати (XML, CSV, HTML).

Мінуси:

- немає gui-інтерфейсу;

- тільки * nix системи.

WAPT [9]

Основні можливості:

- Windows

- платний (є тріал на 30 днів / 20 віртуальних користувачів);

- запис плану тестування з десктопних і мобільних браузерів;

- залежності в планах тестування (наступний URL в залежності від відповіді сервера);

- імітації реальних користувачів (затримки між сполуками, обмеження швидкості з'єднань).

Звіт можна вивести як таблицею, так і графіком (рисунок 1.25):

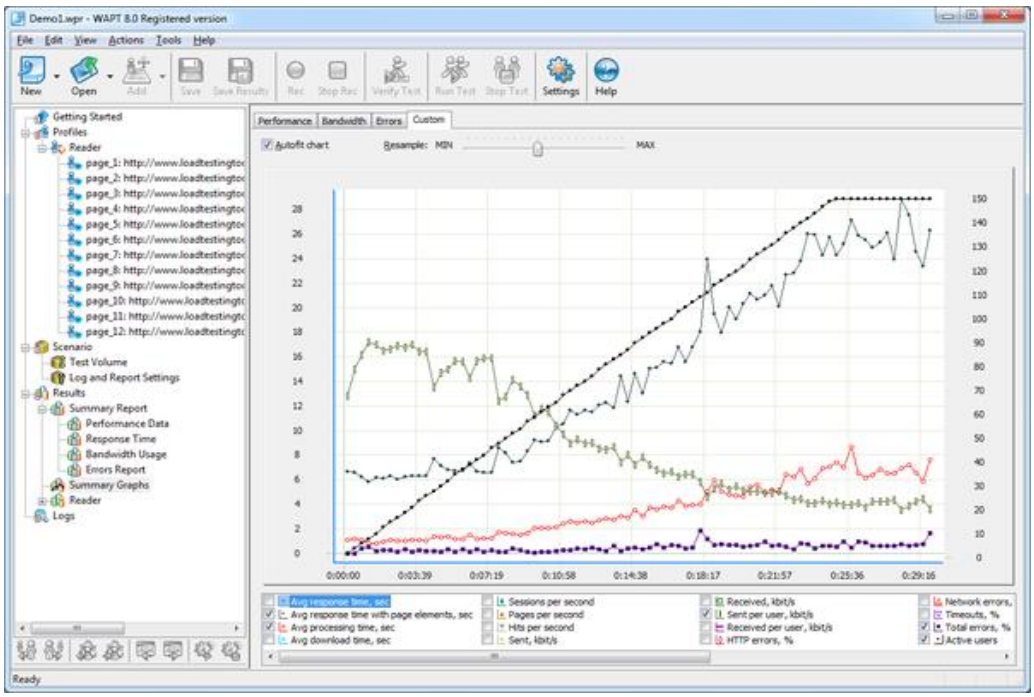

Рисунок 1.25 – Вікно звіту tsung_stats.pl

Плюси:

- дуже гнучкий, велика кількість налаштувань і тестів;
- емуляція повільних каналів з'єднань користувачів;
- підключення модулів;
- запис сценаріїв тестування прямо з браузера, як з десктопного, так і з мобільного;
- генерація різних графіків тестування за допомогою скриптів.

Мінуси:

- доступний тільки для Windows;
- платний.

2 ДОСЛІДЖЕННЯ МЕТОДІВ ТЕСТУВАННЯ НАВАНТАЖЕННЯ ЗА ДОПОМОГОЮ ЗАСОБІВ ПЛАТФОРМИ AZURE

2.1 Розгортання бази даних з допомогою засобів AZURE

Структура бази даних, яка буде використовуватись під час тестування має наступний вигляд (рисунок 2.1-2.12).

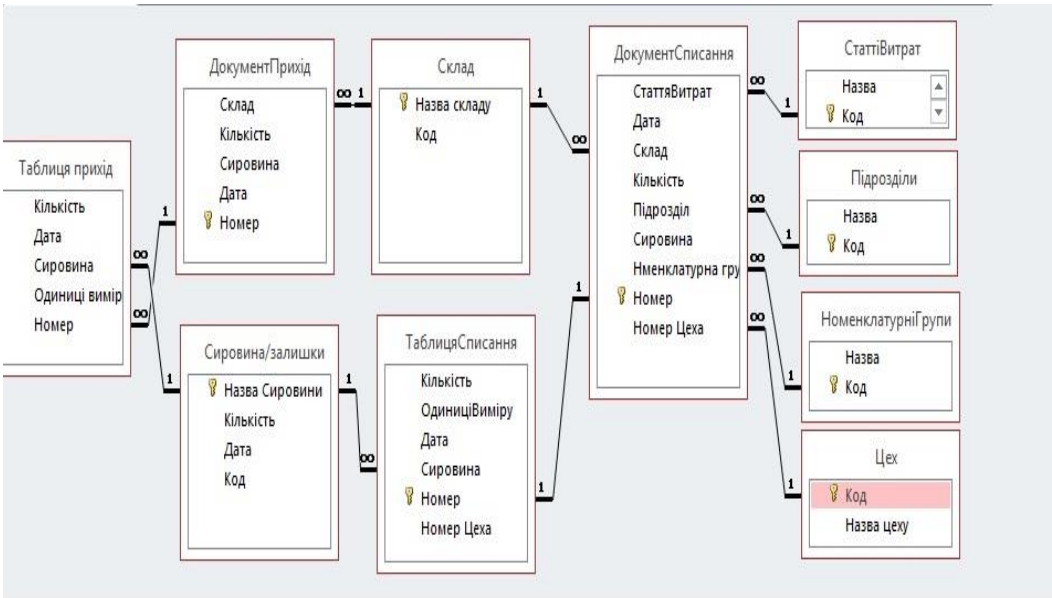

Рисунок 2.1 – Структура БД

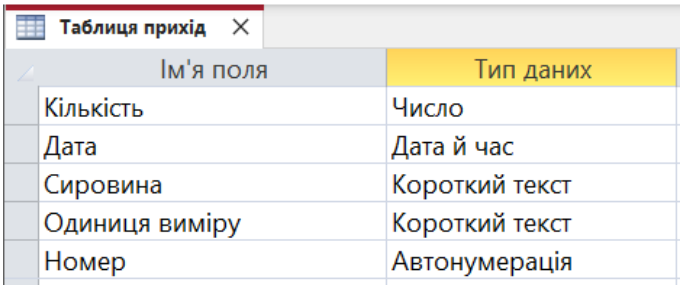

Рисунок 2.2 – Структура таблиці «Таблиця прихід»

Таблиця сировина/залишки

Ім'я поля	Тип даних
Назва сировини	Короткий текст
Кількість	Короткий текст
Дата	Дата й час
Код	Автонумерація

Рисунок 2.3 – Структура таблиці «Сировина/залишки»

Таблиця документ прихід

Ім'я поля	Тип даних
Склад	Короткий текст
Кількість	Число
Сировина	Короткий текст
Дата	Дата й час
Номер	Автонумерація

Рисунок 2.4 – Структура таблиці «Документ прихід»

Таблиця склад

Ім'я поля	Тип даних
Назва складу	Короткий текст
Код	Автонумерація

Рисунок 2.5 – Структура таблиці «Склад»

Таблиця списання

Ім'я поля	Тип даних
Кількість	Число
Одиниці виміру	Короткий текст
Дата	Дата й час
Сировина	Короткий текст
Номер	Автонумерація
Номер цеху	Число

Рисунок 2.6 – Структура таблиці «Таблиця списання»

Таблиця документ списання

Ім'я поля	Тип даних
Стаття витрат	Число
Дата	Дата й час
Склад	Короткий текст
Кількість	Число
Підрозділ	Число
Сировина	Короткий текст
Номенклатурна група	Число
Номер	Автонумерація
Номер цеху	Число

Рисунок 2.7 – Структура таблиці «Документ списання»

Таблиця статті витрат

Ім'я поля	Тип даних
Назва	Короткий текст
Код	Автонумерація

Рисунок 2.8 – Структура таблиці «Статті витрат»

Таблиця Підрозділи

Ім'я поля	Тип даних
Назва	Короткий текст
Код	Автонумерація

Рисунок 2.9 – Структура таблиці «Підрозділи»

Таблиця Номенклатурні групии

Ім'я поля	Тип даних
Назва	Короткий текст
Код	Автонумерація

Рисунок 2.10 – Структура таблиці «Номенклатурні групи»

Таблиця Цех

Ім'я поля	Тип даних
Код	Автонумерація
Назва Цеху	Короткий текст

Рисунок 2.11 – Структура таблиці «Цех»

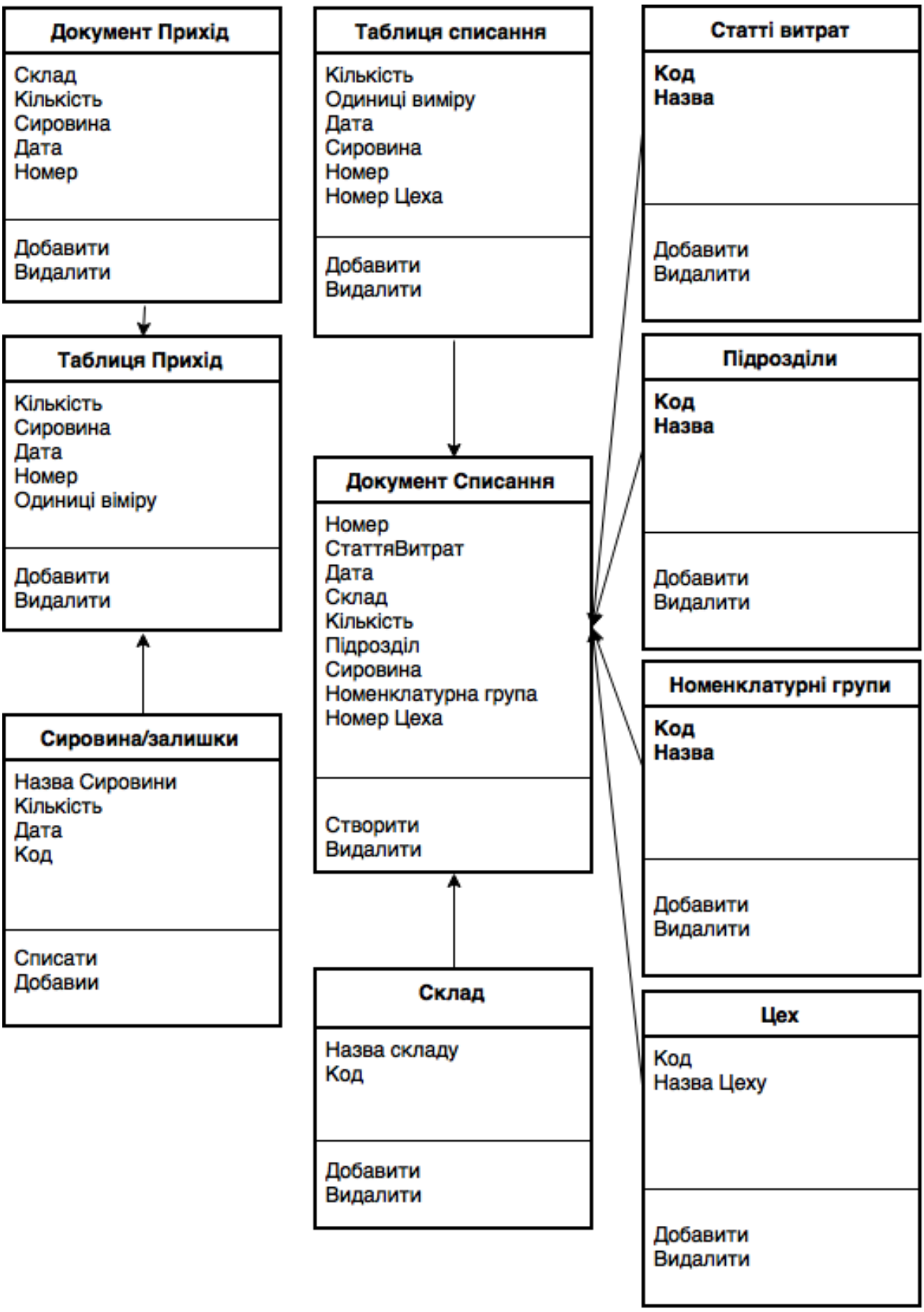

Рисунок 2.12 – Діаграма класів

В індустрії сервісів і виробництва ПЗ технології та інфраструктура Інтернету досягли рівня, достатнього для того, щоб серйозно говорити про можливість надання додатків у вигляді сервісів (Software as a Service, SaaS). Дана модель стала найбільш поширеною при роботі з хмарами [19]. Як приклад реалізації рішення SaaS для публічної хмари можна назвати сервіс Team Foundation Services на платформі Windows Azure [20], призначений для контролю версій, збірки та виконання іншої функціональності спільної роботи над проектами з розробки програмного забезпечення. Разом з тим перехід від традиційної моделі створення та поширення програм до моделі SaaS пов'язаний зі зміною процесів розробки ПЗ, що вимагають тепер більш короткого циклу випуску оновлень і релізів при більш високому, ніж при традиційній розробці, як продукту, що випускається. Інакше кажучи, модель розробки програм змінюється з каскадної (waterfall) моделі на модель швидкої (agile) розробки [21]. Якщо для першої важливо повне розуміння вимог клієнта ще до початку дизайну або розробки, то agile-модель спрямована на облік змін уже в процесі розробки, що дозволяє реагувати на динамізм бізнес-середовища і запити клієнтів. Все це означає, що розробники повинні не тільки швидше публікувати додатки, але і більш оперативно вносити зміни протягом усього життєвого циклу хмарного додатка.

Крім цього, на ринку хмарних сервісів сьогодні спостерігається посилення конкуренції та збільшення числа гравців, наприклад, компанії, які раніше фокусувалися на ПО для корпоративного сегмента, через хмари виходять тепер на малі і середні компанії. Мало того, стираються географічні кордони – для кінцевого клієнта в більшості випадків вже не має значення, чи надається сервіс місцевої або зарубіжною компанією, а з огляду на, що зрілість процесів багатьох західних підприємств в середньому вище, то дана проблема для російського ринку може стати досить гострою.

Таким чином, якість обслуговування і якість наданого продукту можуть стати визначальними в боротьбі за місце на ринку, де враховується, зокрема, такий показник, як коефіцієнт «плинності» клієнтської бази (churn

rate) – відсоток передплатників, які вирішили відмовитися від сервісу. Підвищення і підтримання якості сервісу є одним із заходів, спрямованих на зниження churn rate. Якість стало сьогодні фактором підвищення темпів розробки, що обумовлено наявністю платформ, здатних адаптуватися до нових вимог, наприклад, за рахунок додавання нової можливості без повернення назад і зміни існуючого коду. Фундаментом для досягнення необхідної якості рішення є впровадження повноцінного процесу управління ALM, відсутність якого зазвичай веде до додаткових витрат, зниження споживчих характеристик рішення і появі негативних відгуків клієнтів

Без ALM складності можуть виникати як безпосередньо на етапі розробки, так і на етапі експлуатації. Відсутність інтегрованих інструментів планування і контролю за ходом розробки може призвести до того, що частина розподілених завдань втрачається внаслідок розсинхронізації журналів невиконаних робіт або при передачі завдань по електронній пошті. Через відсутність коштів прототипування інтерфейсу на ранніх етапах і відповідних сценаріїв використання може виявитися, вже на фінальних стадіях проекту, що кінцева система не відповідає вимогам і очікуванням користувачів. На етапі експлуатації поширеною ситуацією є неможливість відтворення помилок в тестовому середовищі і відсутність відомостей про причини і показниках, при яких виникла помилка в роботі системи.

Для управління життєвим циклом додатків корпорація Microsoft пропонує TFS – центр організації спільної роботи, в якому реалізовані ключові служби управління життєвим циклом:

- випуск програмного забезпечення і управління невиконаною роботою;
- відстеження робочих елементів і інтеграція з SharePoint Server для спрощення спільної роботи групи;
- управління версіями;
- інтеграція і розгортання;
- інтегровані служби звітів і аналітики.

До засобів групової розробки Visual Studio, що інтегруються з TFS, відносяться: Visual Studio Ultimate, Visual Studio Premium, Visual Studio Professional, Visual Studio Test Professional і Visual Studio Team Explorer Everywhere. Слід зауважити, що TFS – відкрита платформа, функції якої доступні в якості стандартизованих Web-сервісів, тому аналогічну інтеграцію можна провести і для середовищ і засобів інших виробників. Інструменти, що входять до складу Visual Studio і TFS, надають кошти і функції, необхідні для управління життєвим циклом додатка.

1. Планування і відстеження проектів. Visual Studio Application Lifecycle Management дозволяє управляти потребами клієнтів, наприклад, можна створити високорівневий план, що дає можливість розбити проект на відрізки (ітерації), які можуть бути деталізовані і виконуватися і контролюватися окремо.

2. Проектування і моделювання додатків. У Visual Studio Ultimate можна створювати моделі з різними рівнями деталізації і пов'язувати їх один з одним, з тестами і планом розробки. Створення та розробка моделей протягом життєвого циклу програми може бути складовою частиною процесу розробки. Visual Studio підтримує UML, доменну мову (DSL), схему шарів, граф залежностей і схему послідовностей, засновану на коді.

3. Збірка програми. Team Foundation Builde Server дозволяє реалізувати автоматичний процес перевірки та збірки додатку, забезпечити відповідність коду і додатки до встановлених правил і вимог якості.

4. Тестування програми. Виконується за допомогою ручних або автоматичних тестів, включаючи тести продуктивності і навантажувальні тести. Visual Studio Ultimate і Visual Studio Test Professional дозволяють підвищити якість системи за рахунок повноцінного етапу тестування, який включає планування, проведення тестів і відстеження ходу виконання робіт. Інструмент Microsoft Test Manager допомагає визначати роботи з тестування і управляти ними за допомогою плану тестування, який містить всі необхідні набори тестів, тестові випадки і конфігурації. Створений план дозволяє

вимірювати хід тестування при виконанні тестів і формувати звіти по залишилася роботі.

5. Розгортання в віртуальних середовищах. Дозволяє реалізувати складні сценарії розробки і тестування. Visual Studio Lab Management представляє собою розширення Microsoft Test Manager, що дозволяє оптимізувати роботу з технологією Hyper-V при тестуванні, створенні та розробці додатків в Visual Studio. Рішення Visual Studio Lab Management інтегровано з System Center Virtual Machine Manager, завдяки чому можна керувати кількома фізичними комп'ютерами і віртуальними машинами.

Важливою частиною процесу розробки є тестування, яке зазвичай проходить паралельно з розробкою, іноді – відомою тестуванням.

Тестування сервісів.

Тестування хмарних сервісів і додатків на платформі Windows Azure має певну специфіку в порівнянні з традиційним тестуванням, особливо у контексті використання різних середовищ та інструментів. Основні відмінності можна поділити на кілька етапів, кожен з яких має свої особливості.

1. Локальне тестування (за допомогою локального емулятора Windows Azure). Локальний емулятор дозволяє запускати додатки та сервіси в середовищі, схожому на Windows Azure, на локальній машині. Це дозволяє проводити попереднє тестування без необхідності публікувати додаток у хмарі.

Переваги:

- швидкість розробки і тестування: можна проводити модульні та інтеграційні тести швидко, без підключення до хмари;
- тестування може бути проведено навіть при відсутності підключення до інтернету.

Обмеження:

- локальний емулятор не завжди точно відтворює поведінку сервісів в реальному середовищі Azure;

- для складніших сценаріїв (наприклад, при використанні великих обсягів даних або специфічних компонентів Azure) точність емулятора може бути недостатньою.

2. Тестування в реальному хмарному середовищі.

Це більш складний етап, який включає тестування реального функціонування додатка в умовах реального навантаження і розподілених ресурсів.

Переваги:

- тестування в реальному середовищі дає точну картину продуктивності додатка, адже він працює в умовах реальної інфраструктури, з урахуванням всіх факторів, таких як доступність ресурсів, мережеві затримки і масштабування;

- можливість перевірити вплив різних компонентів хмарного середовища на ефективність додатку, таких як балансування навантаження, зберігання даних, різні варіанти зберігання даних тощо.

Обмеження:

- тестування в реальному середовищі потребує реальних хмарних ресурсів, що може бути дорогим, особливо при великих навантаженнях;

- іноді важко змоделювати в реальному середовищі такі умови, як аварійні ситуації чи різні варіанти навантаження.

3. Автоматизація процесу збірки та тестування.

Важливим кроком є автоматизація тестування на кожному з етапів для забезпечення безперервної інтеграції (CI) і доставки (CD). Це дозволяє мінімізувати помилки, пов'язані з вручну виконуваними тестами, і гарантує, що кожна зміна коду буде протестована на всіх етапах.

Team Foundation Server (TFS) – це одна з платформ для автоматизації цього процесу. З TFS можна автоматизувати всі етапи розробки, від збірки до тестування і релізу.

Переваги TFS:

- автоматизація тестів, що дозволяє запускати їх на різних етапах розробки;
- легка інтеграція з іншими інструментами Microsoft, такими як Visual Studio, Azure DevOps;
- можливість моніторингу і збору результатів тестування, що дає змогу швидко виявляти помилки і недоліки.

Проблеми:

- для ефективної роботи необхідно налаштувати інфраструктуру та інструменти, що вимагає часу і знань;
- у випадку з хмарними сервісами можуть виникнути проблеми з обмеженнями щодо ресурсів чи підключень.

4. Постійне тестування в реальному середовищі.

Тестування в реальному середовищі повинно проводитись не тільки під час розробки, а й після впровадження в експлуатацію, щоб виявляти потенційні проблеми при підвищених навантаженнях.

Навантажувальне тестування та тестування на відмовостійкість є важливими етапами для перевірки, як додаток поводиться під високим навантаженням і в умовах аварійних ситуацій. Це також дозволяє оптимізувати код і інфраструктуру для кращої роботи в хмарі.

Тестування хмарних додатків в Windows Azure має свої унікальні вимоги і виклики. Крім традиційного тестування (модульне, функціональне, інтеграційне), необхідно враховувати додаткові етапи, як локальне тестування за допомогою емулятора і тестування в реальному хмарному середовищі. Автоматизація процесів збору і тестування через інструменти, як Team Foundation Server, дозволяє створити безперервний процес розробки і тестування, що покращує якість і продуктивність хмарних додатків [22].

Реальне хмарне середовище.

Тестування хмарних додатків вимагає уважного підходу, особливо при використанні емуляторів для локального тестування. Основні емулятори, такі

як емулятор обчислень та емулятор сховища, мають свої переваги та обмеження.

Емулятори для тестування.

Емулятор обчислень. Цей емулятор управляє життєвим циклом екземплярів ролей (виробничі контейнери для додатків в Windows Azure), забезпечуючи запуск, виконання та завершення роботи примірників.

Перевага: дозволяє швидко тестувати функціональність в локальному середовищі, не потребуючи постійного підключення до хмари.

Недоліки:

- не моделює повністю поведінку в реальному хмарному середовищі, особливо в аспектах, пов'язаних із балансуванням навантаження і безпекою;

- для належної роботи потрібні права адміністратора, тоді як в реальній хмарній інфраструктурі додатки часто працюють з обмеженими правами доступу.

Емулятор сховища. Емулятор сховища надає можливість локально тестувати роботу з даними, використовуючи SQL Server [23], який емулює як реляційне, так і нереляційне сховище.

Перевага: дозволяє працювати з даними на ранніх етапах розробки, не вимагаючи доступу до хмарних сховищ.

Недоліки:

- не призначений для роботи з великими файлами чи даними більше 2 ГБ, що обмежує його використання для певних сценаріїв;

- оскільки емулюється SQL Server, він не відображає всі особливості реального розподіленого сховища в Windows Azure (наприклад, зберігання даних у Blob Storage чи Table Storage).

Особливості тестування з використанням емуляторів.

Обмеження емуляторів:

- емулятор не є масштабованим і не здатен моделювати складні навантаження, тому не підходить для проведення навантажувальних тестів

або для тестування продуктивності, які мають критичне значення для хмарних додатків;

- при роботі з ним можуть виникати відмінності в поведінці додатків через різницю в правам доступу і в можливостях середовища (наприклад, робота з БД через SQL Server в локальному середовищі не гарантує однакову поведінку при використанні реального хмарного сховища, такого як SQL Azure).

Перехід від локального тестування до тестування в хмарі.

Кроки для ефективного тестування:

Крок 1. Локальне тестування на емулюторах: на початкових етапах розробки, коли додаток ще не готовий для публікації в реальному середовищі, емулятори є корисними для тестування базових функцій і перевірки логіки.

Крок 2. Врахування обмежень емулятора: важливо враховувати, що емулятори мають обмеження і можуть не відображати всі нюанси, що виникають у реальному хмарному середовищі. Тому тестування навантаження і продуктивності повинно бути проведено тільки в реальній хмарі.

Крок 3. Фінальні тести в реальному середовищі: після того як додаток буде завершений і пройде локальні тести, можна запускати публікацію та тестування в хмарному середовищі (Azure). Важливо враховувати, що таке тестування може бути витратним, тому воно повинно бути заплановане на фінальні етапи проекту.

Крок 4. Мінімізація витрат: запуск публікації і тестування в хмарі має сенс лише після того, як додаток продемонструє стабільну роботу в локальному середовищі. Це дозволить знизити витрати на тестування, оскільки тестування в хмарному середовищі може вимагати реальних хмарних ресурсів і коштувати значних грошей.

Тестування хмарних додатків вимагає поєднання локальних емуляторів і реальних хмарних тестів, причому ключовим є перехід від емуляції до

реального середовища. Використання емуляторів на ранніх етапах допомагає знизити витрати на тестування, але важливо враховувати їх обмеження та відмінності від реальної хмари, щоб забезпечити надійне тестування додатка перед фінальним розгортанням у хмарі.

Середовища тестування.

Розгортання додатків у Windows Azure передбачає використання двох середовищ: проміжного (staging) та експлуатаційного (production). Це дозволяє організувати ефективне тестування нових версій додатків перед їхнім остаточним запуском у реальне середовище, забезпечуючи високий рівень надійності і мінімізацію ризиків.

Проміжне середовище (Staging).

Проміжне середовище використовується для первинного розгортання додатка перед публікацією в експлуатаційне середовище.

Це середовище має особливий URL-адрес, наприклад, `http://bb7ea70c3be24eb08a08b6d39f801985.cloudapp.net`, який відрізняється від експлуатаційного, що дозволяє ізолювати тести та нові функціональні можливості від кінцевих користувачів.

Додаток у проміжному середовищі можна тестувати на відповідність вимогам, перевірити нові функції, провести навантажувальні тести та валідацію перед тим, як перенести його в експлуатаційну середу.

Експлуатаційне середовище (Production).

Це середовище є основним для користувачів. Коли додаток тестується і стабільний в проміжному середовищі, його можна перенести в експлуатаційне середовище.

Тут застосовується звичайна адреса, типу `http://example.cloudapp.net`.

Після успішного тестування в проміжному середовищі додаток можна перемістити в експлуатаційне середовище за допомогою функції Swap VIP (зміна віртуальних IP-адрес), що дозволяє легко обміняти місцями проміжне та експлуатаційне середовище.

Swap VIP (Зміна віртуальних IP-адрес).

Swap VIP дозволяє швидко і безболісно перенести додаток з проміжного середовища в експлуатаційне і навпаки. Заміна віртуальних IP-адрес відбувається практично без простоїв і дозволяє мінімізувати ризики при розгортанні нових версій додатків.

Якщо після перенесення додатка в експлуатаційне середовище виникають проблеми, можна виконати відкат, перенісши додаток назад в проміжне середовище, використовуючи ту ж функцію Swap VIP.

Рекомендації при розгортанні додатків.

Спочатку треба публікувати нові версії додатка в проміжному середовищі. Це дозволяє перевірити нові функції та провести їх тестування в реальному середовищі, але без впливу на кінцевих користувачів.

Після успішного перенесення додатка в експлуатаційне середовище, варто провести додаткові вибіркові тести для перевірки роботи в реальному масштабі. Це допоможе виявити можливі проблеми, які можуть не бути помічені в проміжному середовищі.

Завжди треба передбачати можливість швидкого відкату на попередню версію додатка, якщо виникнуть критичні проблеми після перенесення в експлуатаційне середовище.

Переваги:

- перенесення додатка спочатку в проміжне середовище для тестування дозволяє знизити ризик помилок в реальному середовищі;
- можливість миттєвого відкату і заміни середовищ дозволяє швидко реагувати на несподівані проблеми і забезпечує стабільність експлуатаційного середовища;
- Swap VIP дозволяє мінімізувати час простоїв при оновленні додатка, що важливо для високодоступних сервісів.

Цей підхід є оптимальним для хмарних додатків, оскільки дозволяє організувати плавний процес оновлення і тестування, при цьому зберігаючи стабільність експлуатаційного середовища для кінцевих користувачів..

IntelliTrace.

IntelliTrace – це потужний інструмент для налагодження та відстеження, який доступний у Visual Studio Ultimate і може значно покращити процес тестування та налагодження для додатків, що працюють у хмарному середовищі Windows Azure.

Запис розширених налагоджувальних відомостей.

IntelliTrace дозволяє записувати детальні відомості про виконання програми під час її роботи в Windows Azure. Це включає ключові дані про середовище, стан змінних, стеки викликів, виняткові ситуації та інші дані, які можуть бути корисними для діагностики помилок.

Якщо при тестуванні в Azure виникає проблема, IntelliTrace допомагає відновити послідовність подій та точно відтворити хід виконання програми. Це дає змогу не тільки побачити, де саме відбулася помилка, а й зрозуміти, чому вона сталася.

Безпосереднє налагодження в середовищі Azure.

IntelliTrace дозволяє користувачам виконувати налагодження на сервері Windows Azure так, як якщо б код працював локально. Це забезпечує максимально точне відтворення сценаріїв роботи додатка в реальному середовищі.

Після запису інформації за допомогою IntelliTrace, користувач може покроково виконувати код у VS, що дозволяє виявити проблеми на будь-якому етапі виконання програми, навіть якщо вона працює в хмарі.

Налагодження без необхідності встановлення VS на сервер.

Однією з важливих переваг IntelliTrace є те, що його можна встановити на сервер Azure без необхідності встановлення VS на самому сервері. Це дозволяє зберегти ресурси сервера та зменшити навантаження на систему.

Можна отримати повні дампи поведінки програми, стан стека, значення змінних та інші деталі для аналізу та виявлення причин виникнення помилок.

Аналіз причин помилок.

IntelliTrace фіксує інформацію про всі виняткові ситуації, що виникли в процесі виконання додатка, а також про інші критичні моменти, які можуть вказувати на проблеми з продуктивністю або помилки в коді.

Записані дані можуть бути відтворені в VS, що дає змогу тестувальникам та розробникам точно відтворити умови, при яких виникла помилка, і з'ясувати, чому це сталося.

Переваги для тестування та оптимізації в Azure.

Завдяки IntelliTrace можна тестувати додаток не тільки в локальному середовищі, але й у реальному хмарному середовищі Windows Azure, що дозволяє точніше оцінити його поведінку.

Використання IntelliTrace для виявлення помилок та проблем з продуктивністю дозволяє ефективно оптимізувати додатки ще на етапі тестування, що значно знижує ймовірність помилок в експлуатаційному середовищі.

Безкоштовне використання та можливості для безперервного тестування.

IntelliTrace може бути інтегровано з іншими інструментами Visual Studio, такими як TFS, що дозволяє створити безперервний процес тестування та налагодження в розробці.

За допомогою IntelliTrace розробники та тестувальники можуть значно зменшити витрати на тестування та налагодження, оскільки це потужний інструмент, який дозволяє виявити навіть найскладніші помилки без додаткових витрат на ресурси або ліцензії.

IntelliTrace є незамінним інструментом для розробників, які працюють з Windows Azure, оскільки він дозволяє ефективно відслідковувати, записувати і відтворювати відомості про виконання програми в хмарному середовищі. Це значно полегшує процес налагодження та тестування, особливо коли додаток працює у складних розподілених середовищах, як у Windows Azure.

Тестування навантаження.

Навантажувальне тестування є критично важливим етапом у розробці та експлуатації хмарних додатків, оскільки воно допомагає визначити здатність системи обробляти великі обсяги запитів та користувачів без втрати

продуктивності. Однак через специфіку хмарних платформ цей процес має певні складнощі, які необхідно враховувати:

Динамічна природа хмарних додатків.

Хмарні додатки часто мають змінювану інфраструктуру, оскільки ресурси можуть бути масштабовані залежно від навантаження. Це створює труднощі в тестуванні, оскільки важко визначити постійне навантаження або ресурсні вимоги, що відрізняються від традиційних монолітних систем.

Для реалістичного тестування важливо імітувати не тільки кількість користувачів, а й характер їхніх запитів, швидкість підключення, затримки тощо.

Георозподіленість користувачів.

Хмарні додатки часто обслуговують користувачів з різних частин світу. Це означає, що для реалістичного тестування необхідно враховувати різні мережеві умови, швидкість підключення, затримки при доступі до серверів. Для цього використовується географічно розподілене навантаження, де навантажувальні тести симулюють користувачів з різних регіонів. Важливо враховувати, як ці фактори впливають на продуктивність та споживання ресурсів.

Імітація великої кількості віртуальних користувачів.

Одним з основних завдань навантажувального тестування є перевірка системи під високим навантаженням. Для цього використовуються віртуальні користувачі, що імітують реальну поведінку. Важливо коректно налаштувати кількість віртуальних користувачів, час їхньої взаємодії з системою та інші параметри.

При великому навантаженні важливо не тільки протестувати кількість одночасних користувачів, але й звернути увагу на вплив затримок між запитами, можливі «вузькі місця» в обробці запитів та їх взаємодії з базою даних і сховищем.

Навантаження на ресурси та оптимізація.

Тестування навантаження дозволяє оцінити, як система реагує на зростаюче навантаження, і чи призводить це до зростання споживання ресурсів, таких як пам'ять, процесор, мережеві ресурси. Якщо навантаження збільшується, це може призвести до зниження числа одночасних користувачів, на яке система розрахована.

Один із результатів тестування – визначення оптимальності розбиття сховища на розділи бази даних. Неоптимальне налаштування індексів, розділів або самих запитів може стати причиною суттєвого зниження швидкості роботи при високому навантаженні.

Зміна продуктивності під час тестування.

Навантажувальні тести також допомагають виявити нестабільність системи при високому навантаженні. Наприклад, зростання затримок при обробці запитів або падіння швидкості відповіді можуть свідчити про те, що ресурси системи перевантажені. Оцінка цих змін є важливою для коригування налаштувань або масштабування ресурсів у реальному середовищі.

Визначення меж системи.

Однією з основних цілей тестування є визначення максимальної кількості користувачів або запитів, які система може обробити без збоїв. Навантажувальні тести дозволяють встановити ці межі, а також з'ясувати, як система поводиться після досягнення цієї межі – чи зберігається працездатність, чи починаються помилки, як відновлюється працездатність.

Навантажувальне тестування є ключовим етапом для забезпечення стабільності хмарних додатків при великому навантаженні. Однак для того, щоб результати були точними, важливо враховувати не тільки кількість користувачів, а й географічне розташування, типи запитів, пропускну здатність мережі та інші фактори. Крім того, важливо аналізувати ефективність роботи з базами даних та сховищами для виявлення можливих слабких місць, що можуть обмежувати масштабованість системи.

Visual Studio Ultimate [24] надає потужні інструменти для тестування продуктивності, включаючи можливість створення, виконання та аналізу результатів навантажувальних тестів. Крім того, вона забезпечує зручні засоби для збору і представлення даних тестування у вигляді звітів, що дозволяє командам відслідковувати продуктивність додатків і виявляти можливі проблеми в їх роботі.

Аналітика та звітність. Після виконання тестів VS надає детальні звіти про результати тестування. Ці звіти можуть містити інформацію про навантаження на систему, час відповіді, використання ресурсів і багато іншого. Звіти можуть бути збережені в сховищі, що дозволяє створювати історію тестування та відслідковувати тенденції продуктивності з часом.

Інтеграція з хмарними середовищами. Однією з ключових можливостей VS є підтримка навантажувального тестування в хмарі. Замість того щоб виконувати тестування на локальних серверах, можна генерувати навантаження безпосередньо в хмарному середовищі.

Віртуальні машини в Windows Azure можна використовувати для розгортання агентів, які генерують навантаження, і контролера, який керує цими агентами. Це дозволяє ефективно масштабувати тести та обробляти навантаження, яке важко відтворити в локальних умовах.

Масштабованість. Використання хмарних ресурсів дозволяє масштабувати кількість віртуальних користувачів для тестування продуктивності додатка, що дуже важливо для реалістичних навантажувальних тестів. Хмара дозволяє розгортати стільки агентів, скільки потрібно для симуляції різних умов навантаження, а також забезпечує високий рівень доступності та масштабованості інфраструктури.

Автоматизація процесів. За допомогою VS можна автоматизувати виконання тестів, що є важливим для постійного моніторингу продуктивності під час розробки. Тести можуть виконуватись автоматично в рамках CI/CD процесів, що дозволяє швидко виявляти проблеми з продуктивністю на ранніх етапах розробки.

Основні елементи (рисунок 2.13):

- агенти навантаження – віртуальні машини, розгорнуті в Windows Azure, які генерують навантаження, виконуючи імітацію роботи великої кількості користувачів, роблячи запити до тестованого додатка;

- контролер – компонент, який управляє агентами, координує їх роботу і збирає дані про результати тестування. Контролер може бути також розгорнутий на віртуальній машині в Azure, що забезпечує зручне управління навантажувальними тестами;

- інтерфейс Visual Studio – налаштування і контроль через VS, де зібрані звіти й аналітика допомагають зрозуміти, як система поводиться під різними навантаженнями.

Переваги хмарного навантажувального тестування:

- можливість здійснювати тестування в умовах, які близькі до реальних, за допомогою глобально розподілених віртуальних користувачів;

- зниження витрат на необхідне для тестування фізичне обладнання та інфраструктуру;

- більш гнучке масштабування ресурсів під час виконання тестів.

Отже, використання VS в поєднанні з хмарними можливостями Windows Azure дозволяє виконувати масштабоване, автоматизоване та ефективне навантажувальне тестування для хмарних додатків, що забезпечує високу надійність і продуктивність додатків під великими навантаженнями.

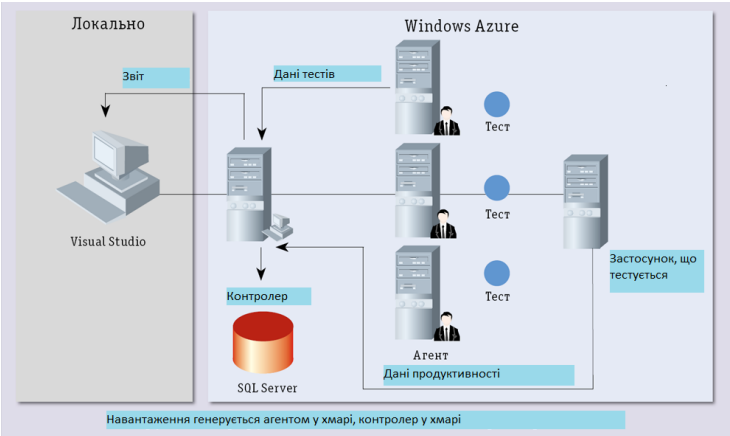

Рисунок 2.13 – Основні елементи навантажувального тестування в VS

Вибір місця для генерації навантаження під час тестування хмарних сервісів має суттєвий вплив на точність і ефективність тестування. У межах цього процесу необхідно враховувати низьку аспектів.

Генерація навантаження в хмарі.

Переваги:

- відсутність мережевої затримки – трафік генерується і обробляється всередині хмарного середовища, що виключає затримки, пов'язані з передачею даних через інтернет. Це дозволяє тестувати реальну продуктивність додатка, не спотворену мережею;

- масштабування за потребою – хмарні ресурси дозволяють гнучко збільшувати або зменшувати кількість агентів для навантаження залежно від сценарію тестування;

- низька вартість трафіку – трафік, згенерований у хмарі, зазвичай має мінімальну вартість у межах хмарного середовища;

- відсутність залежності від локальної інфраструктури – використання хмарних ресурсів знімає потребу в потужних локальних серверах і мережевому обладнанні.

Недоліки:

можливість моделювання лише ідеальних умов – генерація навантаження в хмарі не враховує мережеві затримки чи специфіку роботи користувачів із різних регіонів.

обмеження в налаштуванні специфічних сценаріїв – у деяких випадках

Генерація навантаження локально.

Переваги:

- реалістичність мережевих умов – локальні агенти дозволяють врахувати мережеві затримки, втрати пакетів та інші фактори, з якими стикаються користувачі в реальних умовах;

- повний контроль над середовищем – адміністратори мають змогу налаштовувати мережеве та серверне обладнання відповідно до специфіки тестування;

- можливість моделювання складних сценаріїв – локальне середовище дає змогу точно налаштувати параметри генерації трафіку.

Недоліки:

- Високі витрати на трафік – передача великих обсягів даних до хмарного середовища може бути дорогою;

- потреба в обладнанні – для тестування потрібні потужні локальні сервери, здатні генерувати значне навантаження;

- потенційні обмеження з боку проксі-серверів – корпоративні мережі можуть обмежувати кількість одночасних запитів, що вимагає створення окремого сегмента мережі з відповідними налаштуваннями.

Особливості тестування Windows Azure.

Хмарна платформа Windows Azure додає свої специфічні аспекти до процесу тестування:

- балансувальник Azure відіграє ключову роль у маршрутизації запитів і може стати "вузьким місцем", якщо генерація навантаження виконується через нього;

- контролер віртуальних машин (Fabric Controller) також бере участь у розподілі запитів і управлінні ресурсами. Перенесення генерації навантаження в хмару допомагає уникнути перевантаження цих компонентів;

- сервіси в Azure можуть мати специфічні обмеження щодо кількості одночасних з'єднань або використання ресурсів, які слід враховувати при налаштуванні тестів.

Таким чином для локального тестування використовується окремий мережевий сегмент із спеціальними налаштуваннями для обробки великої кількості одночасних запитів та плануються додаткові витрати на обладнання та трафік. Для тестування в хмарі використовуються хмарні агенти для моделювання великого навантаження без впливу мережевих факторів та інтегруються тести у хмарну інфраструктуру Azure для повної симуляції умов роботи додатка.

Комбінований підхід:

- початкові тести виконуються локально, щоб оцінити вплив мережевих факторів;

- тестування завершується в хмарному середовищі, щоб перевірити масштабованість і стабільність сервісу.

Використовуються розподілені агенти в різних регіонах для перевірки продуктивності додатка з урахуванням географічних затримок.

Правильний вибір підходу до генерації навантаження дозволяє виявити потенційні проблеми та забезпечити стабільну роботу додатків навіть у пікових умовах.

Автоматизація тестування.

Тестування хмарних додатків вимагає широкого використання автоматизації, адже ручне тестування стає практично неможливим через високу складність, масштабність та специфіку хмарних середовищ. Розглянемо ключові аспекти автоматизації тестування хмарних додатків за допомогою TFS та інших інструментів.

Автоматизація тестування локального емулятора. Для публікації хмарного додатка в локальне середовище використовуються емулятори, які моделюють обчислювальні та сховищні ресурси Windows Azure.

Інструмент CSRun використовується для розгортання додатка в емулятор середовища Windows Azure, управління запущеними службами, що дозволяє швидко перевіряти функціональність додатка перед його публікацією в хмару.

Переваги автоматизації для локального емулятора:

- скорочення часу на ручну перевірку кожного етапу;
- зменшення людського фактора при розгортанні;
- можливість швидко виявляти та усувати помилки до публікації в реальне середовище.

Автоматизація публікації в хмарне середовище застосовує інструменти та API: Windows Azure Service Management API – дозволяє взаємодіяти з хмарною інфраструктурою Azure для керування додатками, включно з їх

публікацією, Certificate API – забезпечує безпечну автентифікацію під час роботи з хмарними сервісами.

При автоматизації визначають певні етапи: збірка пакета додатка Windows Azure – використовуються інструменти для створення готового до розгортання пакета, що включає всі необхідні компоненти додатка, тестування – запуск тестів на різних етапах, включаючи функціональне, регресійне та навантажувальне тестування, публікація – пакет автоматично розгортається в хмарному середовищі, де відбувається фінальна перевірка працездатності.

TFS надає потужні інструменти для управління автоматизацією процесів розробки, тестування та публікації: безперервна інтеграція (Continuous Integration) – автоматичний запуск збірки та тестування після кожного коміту коду, режим попередньої збірки (Gated Check-In) – код тестується в ізольованому середовищі до його інтеграції в основне сховище, що запобігає поширенню помилок у спільну гілку розробки, гнучкі тригери – налаштування різних умов запуску збірки (за розкладом, вручну, при коміті тощо).

Підтримка робочих процесів Windows Workflow у TFS дозволяє:

- ефективно управляти чергами завдань;
- налаштовувати середовище збірки відповідно до потреб проекту;
- масштабувати автоматизацію для великих команд і складних проектів.

Автоматизація за допомогою TFS забезпечує надійний процес тестування та розгортання, а також дозволяє командам зосередитися на вдосконаленні функціональності, мінімізуючи ризики помилок у готовому продукті.

2.2 Тестування навантаження веб-застосунків у хмарі

Методика тестування навантаження веб-застосунку складається з кількох етапів. Розглянемо їх детально.

Налаштування середовища для тестування

Спочатку необхідно створити в Azure віртуальну машину, на якій будуть проводитися тести. Для цього виконуються наступні дії (рисунок 2.14):

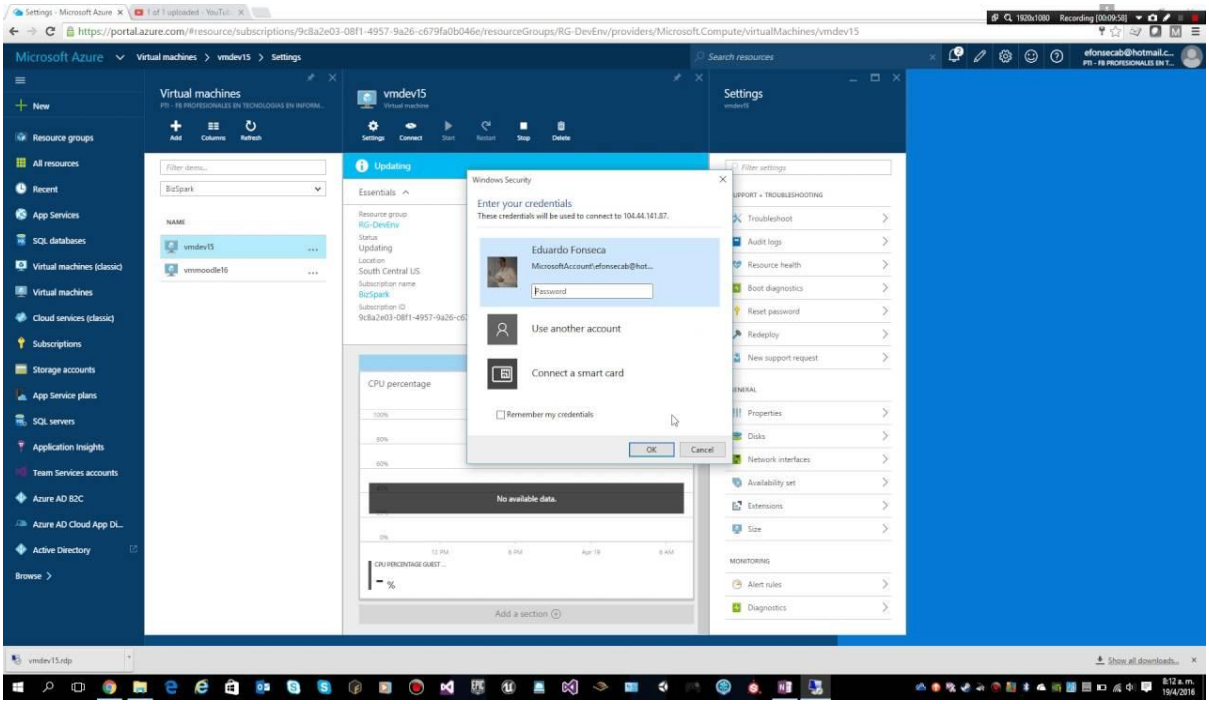

Рисунок 2.14 – Створення віртуальної машини засобами Azure

Залежно від типу випробувань, який потрібно запустити, може знадобитися досить потужна машина. Для проведення серйозних тестів вибираємо формат A4 (рисунок 2.15 - 2.16).

Рисунок 2.15 – Налаштування конфігурації віртуальної машини

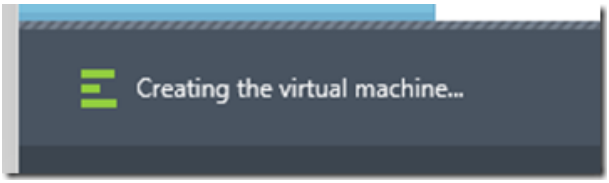

Рисунок 2.16 – Створення віртуальної машини засобами Azure

Після створення з'явиться сервіс, схожий на цей. Підключаємося до нього віддалено (рисунок 2.17 – 2.18):

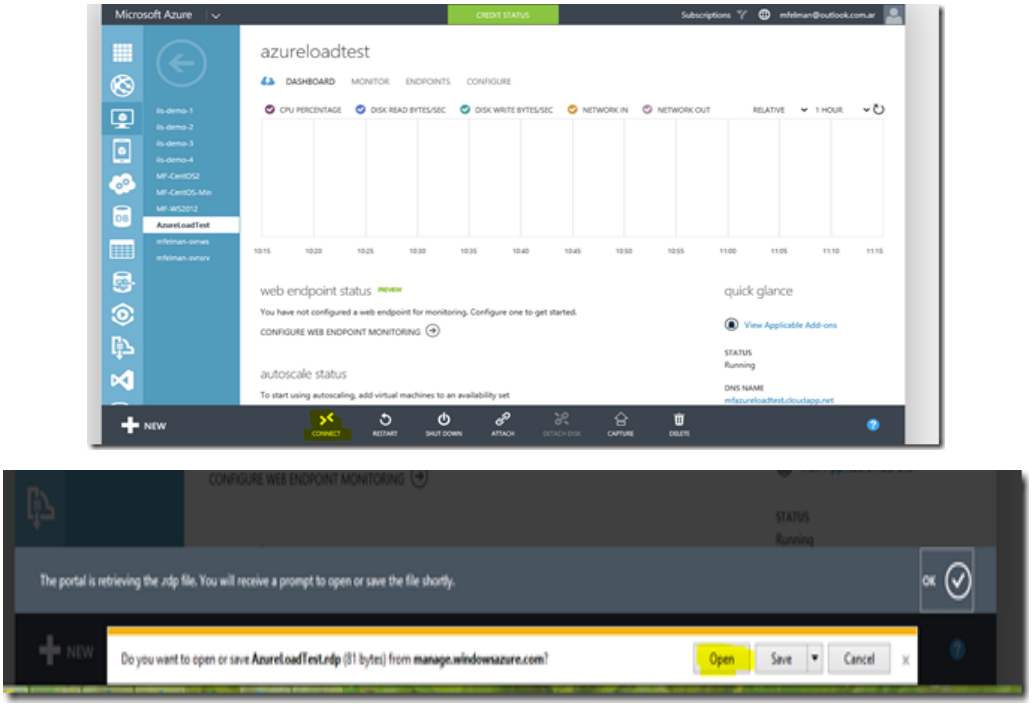

Рисунок 2.17 – Створений сервіс засобами Azure

Рисунок 2.18 – Створена віртуальна машина засобами Azure

Тепер ми будемо працювати у віртуальній машині, що працює в Azure. Запускаємо Visual Studio (рисунок 2.19):

Рисунок 2.19 – Запуск Visual Studio

Тепер необхідно перейти File> New> Project і вибрати Web Performance і Load Test Project (рисунок 2.20):

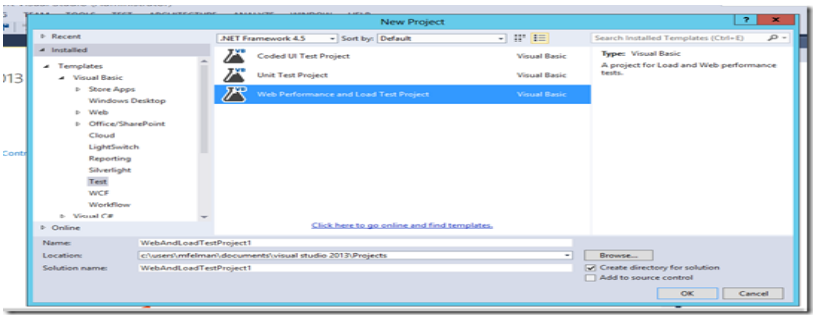

Рисунок 2.20 – Створення Web Performance і Load Test Project

Почнемо зі створення спільного тесту веб продуктивності (Web Performance Test) (рисунок 2.21):

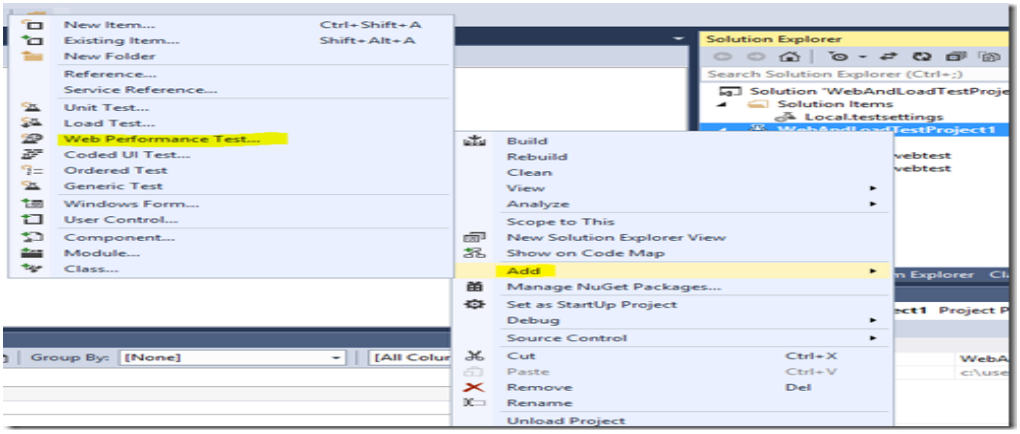

Рисунок 2.21 – Додавання Web Performance Test

Тут записуються сценарії тесту. В даному випадку ми здійснюємо запис сценарію відвідування веб-застосунку в Visual Studio, а потім одночасно повторюємо його десятки або навіть тисячі разів. Діями можуть бути аутентифікація, генерування запитів, транзакції, і т. д.

Приступаємо до запису сесії (рисунок 2.22 - 2.23):

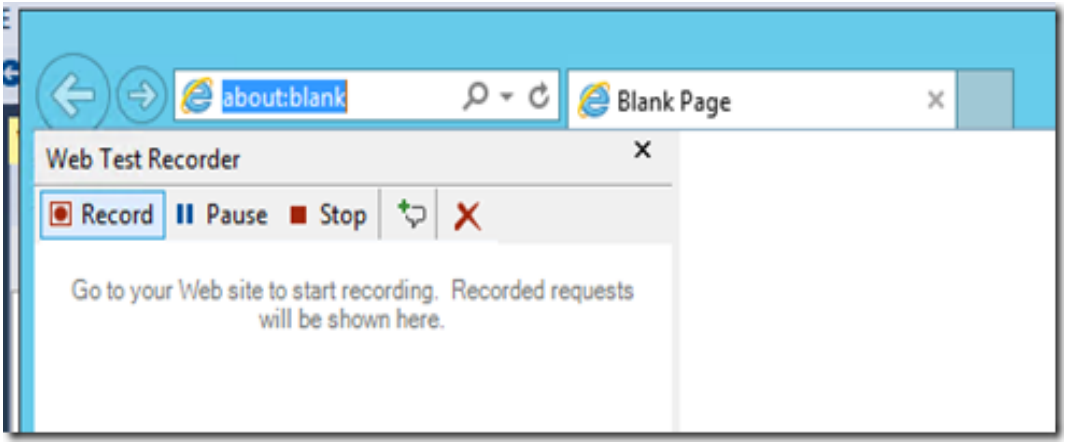

Рисунок 2.22 – Запуск тесту

Тут простий приклад тільки з одним запитом.

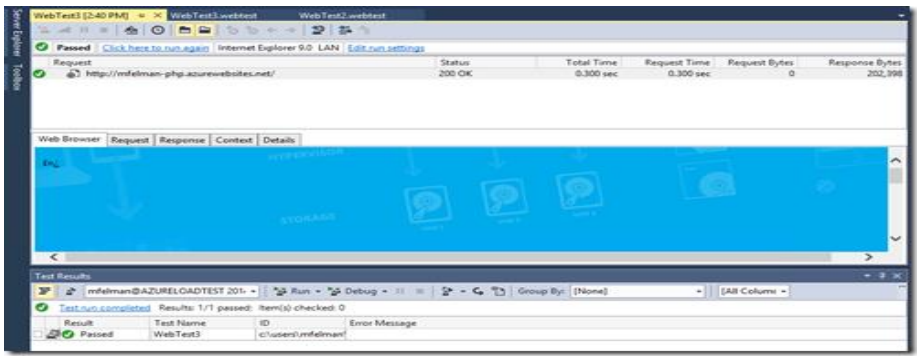

Рисунок 2.23 – Приклад з одним запитом

Далі створюємо автоматизований випадок відвідування веб-застосунку користувачем. Потім можна змоделювати сотні одночасних відвідувань з входом на веб-застосунок (рисунок 2.24 - 2.26):

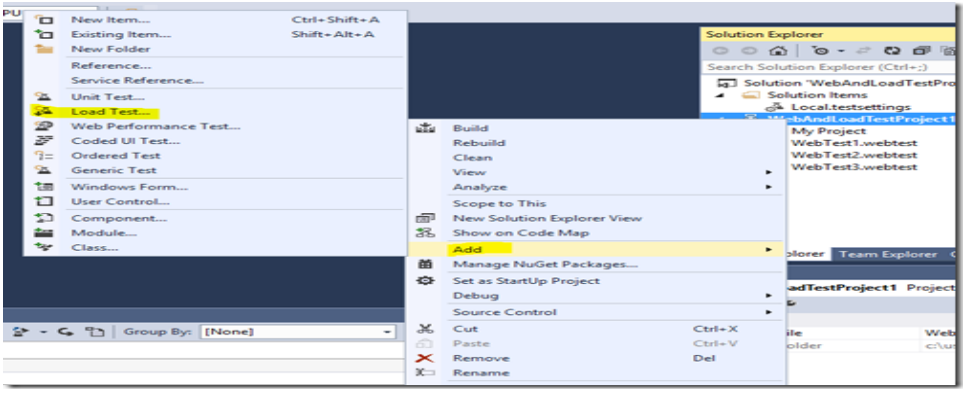

Рисунок 2.24 – Створюємо автоматизований випадок відвідування веб-застосунку користувачем

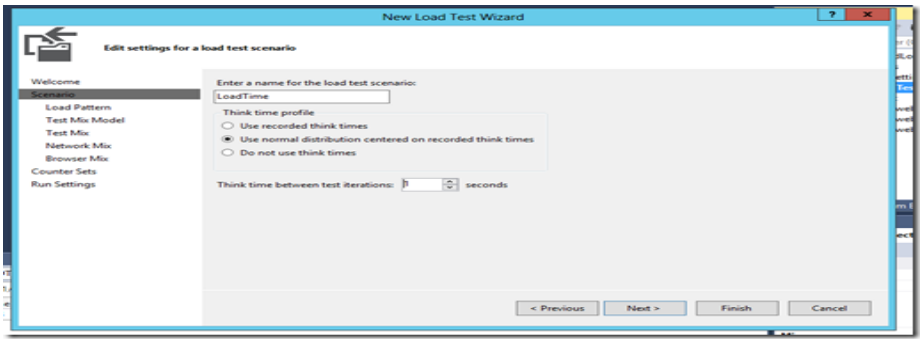

Рисунок 2.25 – Створення тесту

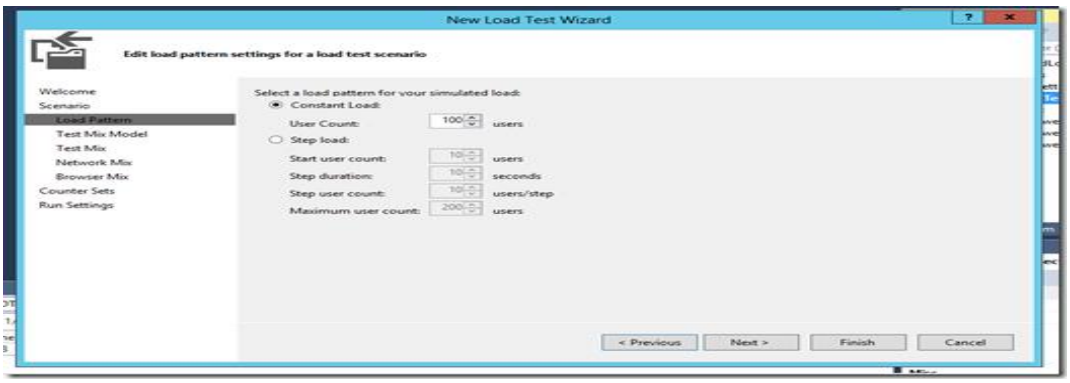

Рисунок 2.26 – Налаштування тесту

Додаємо створений раніше веб-тест (рисунок 2.27 - 2.28):

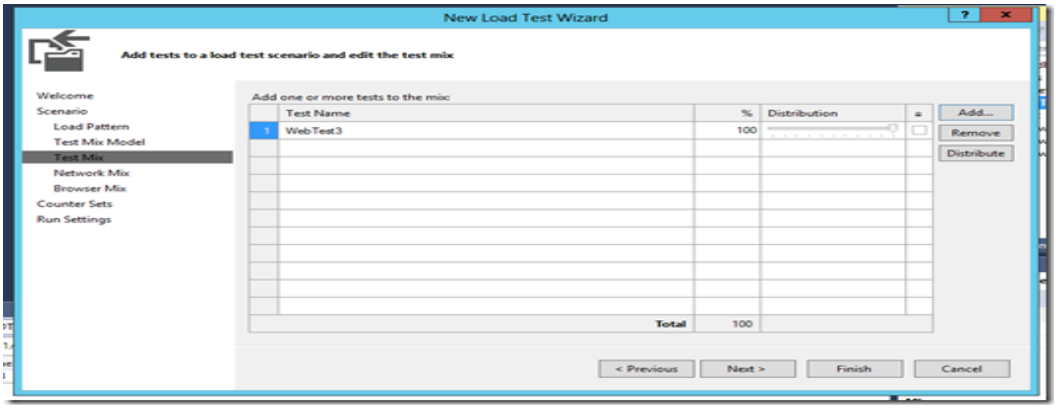

Рисунок 2.27 – Додавання раніше створеного тесту

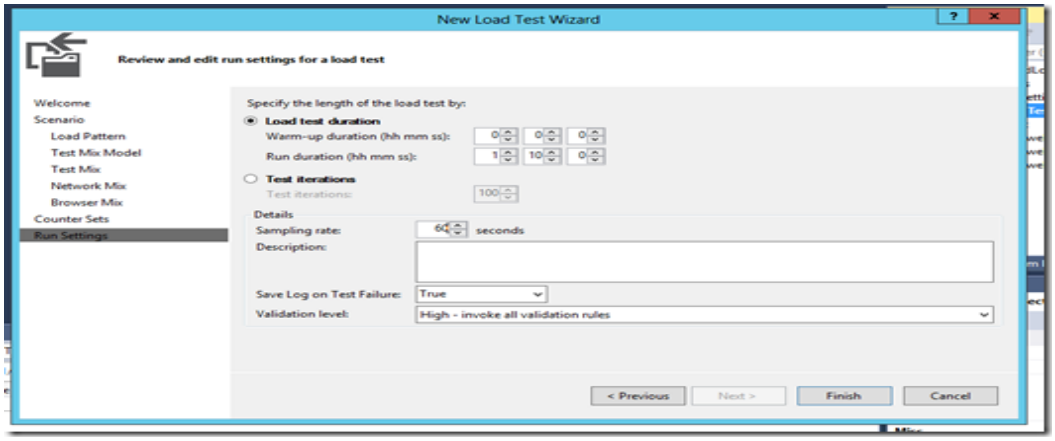

Рисунок 2.28 – Налаштування часу тестування

Як видно, майстер надає безліч варіантів.

Отже, тест вже налаштований і щоб запустити його, натискаємо Run (Виконати) (рисунок 2.29 - 2.30).

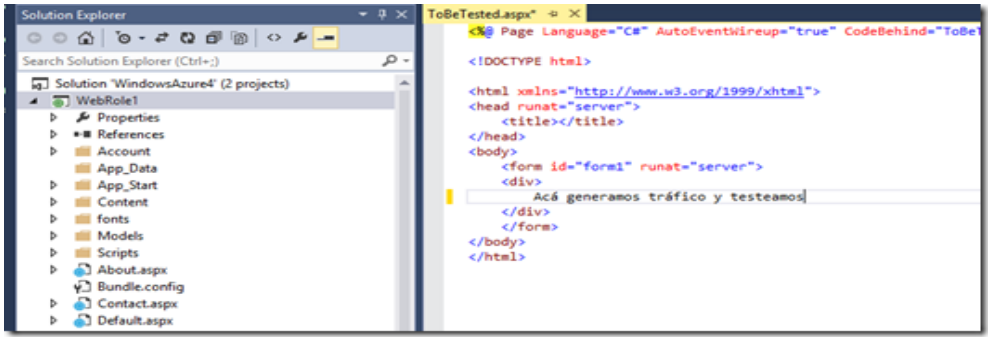

Рисунок 2.29 – Вікно запуску тесту

```
using System.Web;
using System.Web.UI;
using System.Web.UI.WebControls;

namespace WebRole1
{
    public partial class ToBeTested : System.Web.UI.Page
    {
        protected void Page_Load(object sender, EventArgs e)
        {
            IList<string> bolsaDeGatos = new List<string>();

            for (int i = 0; i < 1000; i++)
            {
                bolsaDeGatos.Add(i.ToString());
            }

            IList<string> bolsaDePerros = new List<string>();
            foreach(string gato in bolsaDeGatos)
            {
                int num = int.Parse(gato);
                if (num % 2 == 0)
                {
                    bolsaDePerros.Add(bolsaDeGatos[num]);
                    bolsaDeGatos.OrderBy(x => x.ToUpper());
                }
            }
        }
    }
}
```

Рисунок 2.30 – Код тесту навантаження

Далі програма виконує власне саме тестування навантаження, результати якого в п.3.1.

2.3 Тестування навантаження бази даних з допомогою засобів Azure

Щоб додати БД на порталі Microsoft Azure потрібно перейти в меню створення хмарних ресурсів і перейти в розділ «Database». BD SQL знаходиться у списку [25] (рисунок 2.31).

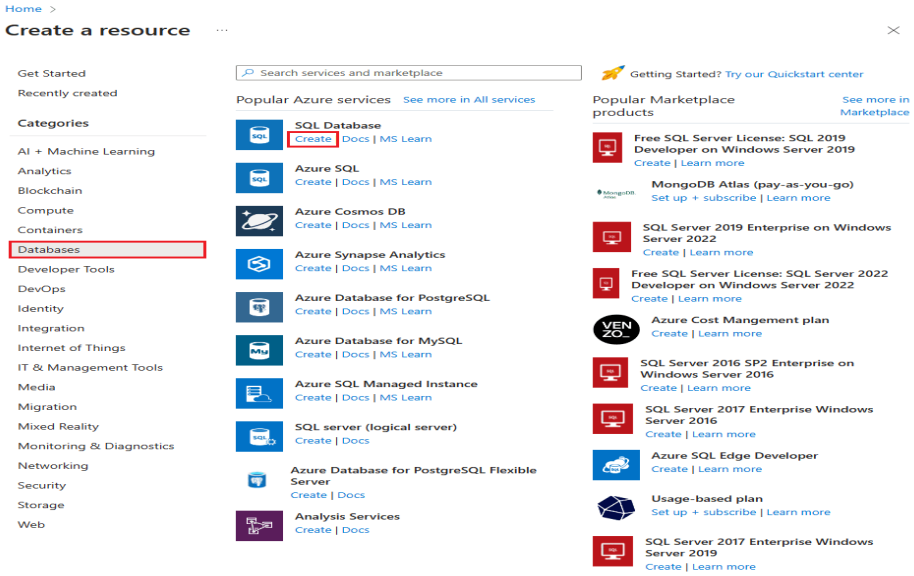

Рисунок 2.31 – Створення БД засобами Azure

Далі необхідно задати ряд нескладних налаштувань (рисунок 2.32).

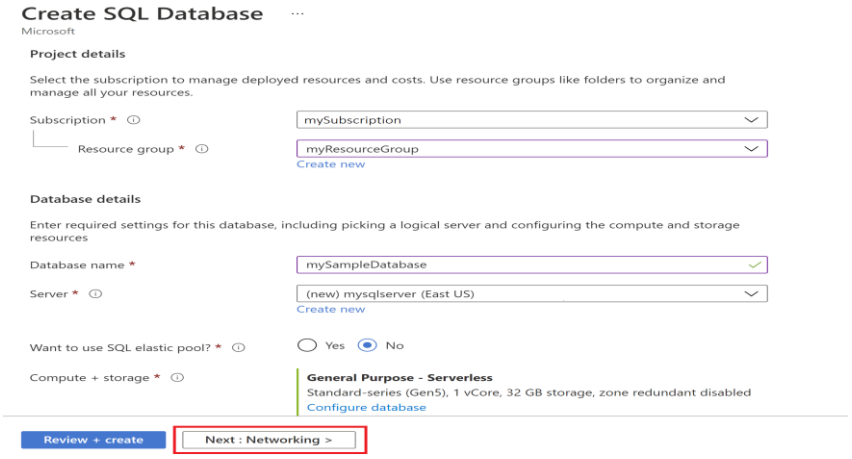

Рисунок 2.32 – Налаштування БД

На вкладці Мережа в розділі Метод підключення виберіть пункт Загальна доступна кінцева точка.

У розділі Правила брандмауера встановіть перемикач Додати поточну IP-адресу клієнта в положення Так. Залишить значення Ні для параметра Дозволити доступ до сервера служб та ресурсів Azure (рисунок 2.33).

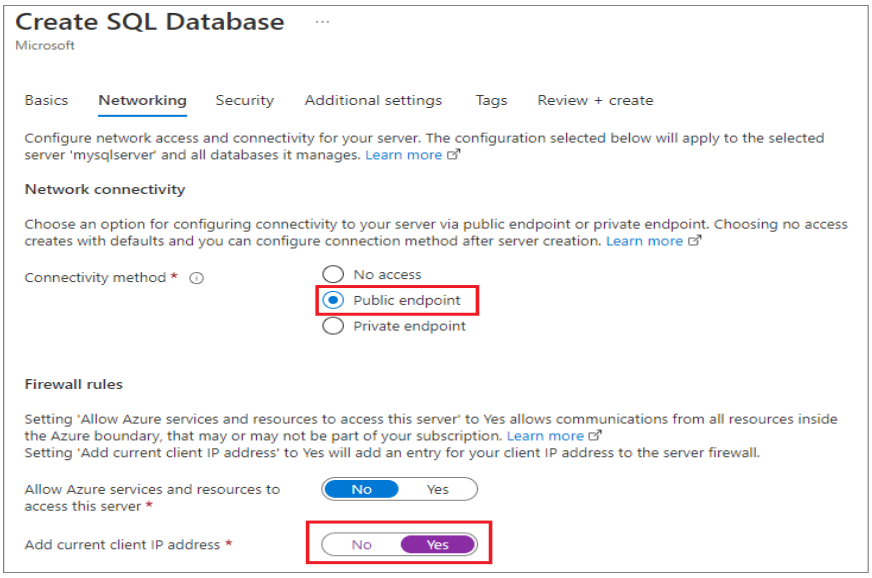

Рисунок 2.33 – Створення групи ресурсів

Можемо створити нашу базу даних (рисунок 2.34):

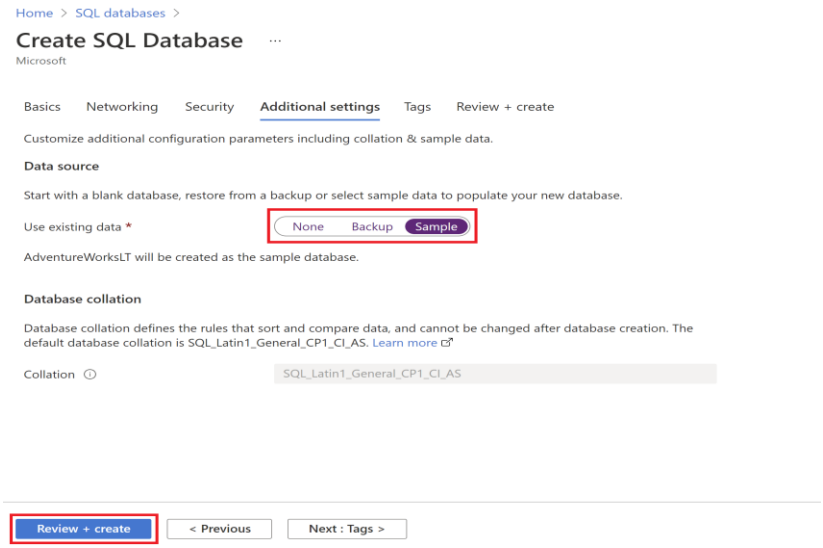

Рисунок 2.34 – Налаштування серверу для БД

Процес створення займає певний час, і поки база даних розгортається – бачимо анімовану іконку. Після того, як база даних створиться – побачимо повідомлення.

За допомогою плитки на головній сторінці зайдемо в властивості бази даних (рисунок 2.35):

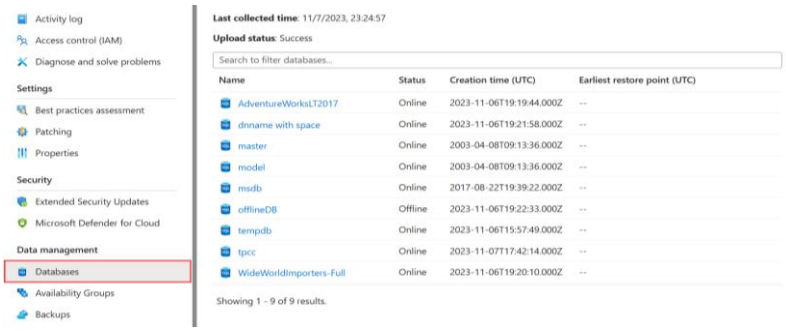

Рисунок 2.35 – Розгорнута БД

І перейдемо до повного списку параметрів. Нас цікавлять «Ім'я вузла», «Порт», «Ім'я користувача» і «Пароль». Вони знадобляться для доступу до БД.

По завершенню установки запускаємо MySQL Workbench (рисунок 2.36):

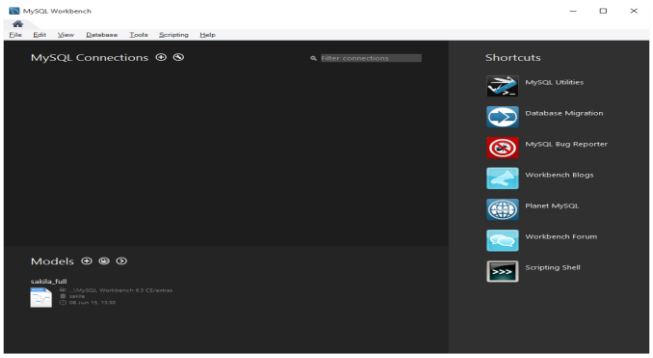

Рисунок 2.36 – Запуск MySQL Workbench

Так як доведеться підключатися до MySQL не один раз, створимо шаблон підключення (рисунок 2. 37):

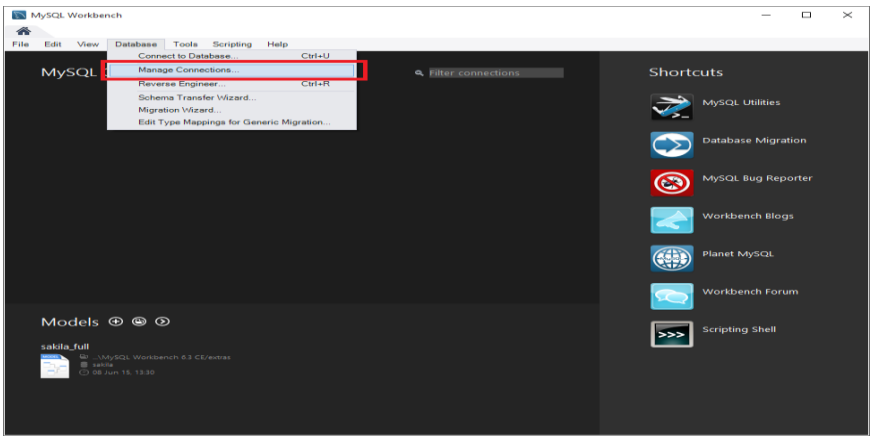

Рисунок 2.37 – Шаблон підключення

Шаблон підключення (рисунок 2.38 - 2.39):

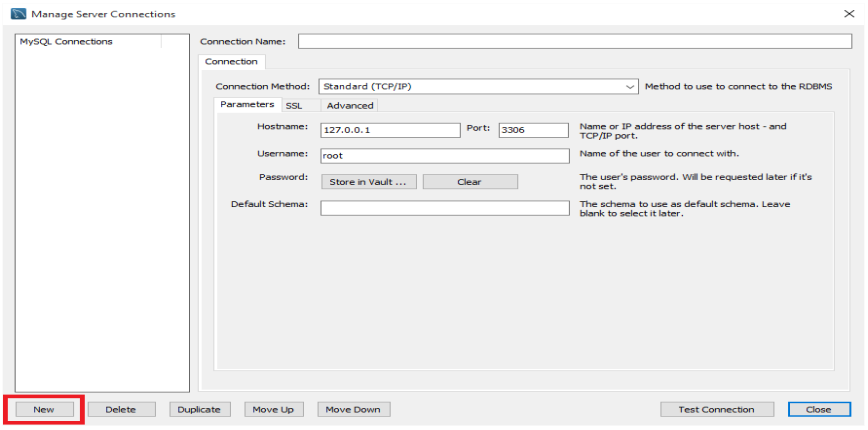

Рисунок 2.38 – Налаштування шаблону

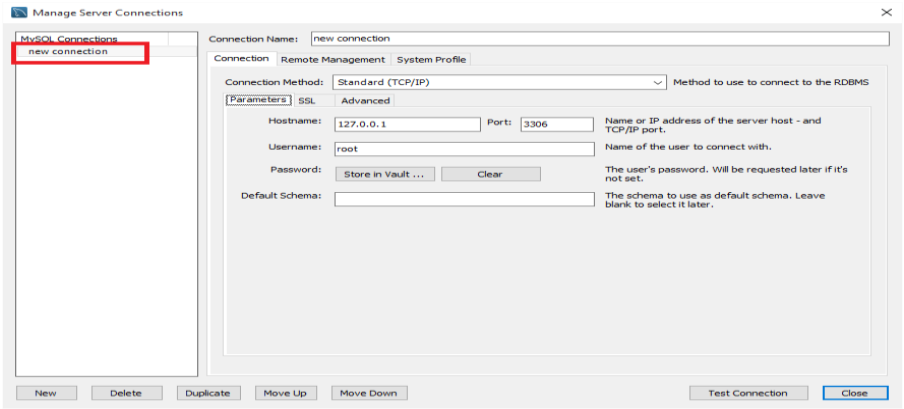

Рисунок 2.39 – Параметри підключення

Тепер введемо дані, які бачили на консолі управління Azure. А для того, щоб кожен раз при підключенні не вводити пароль, збережемо його в сховище паролів Workbench (рисунок 2.40):

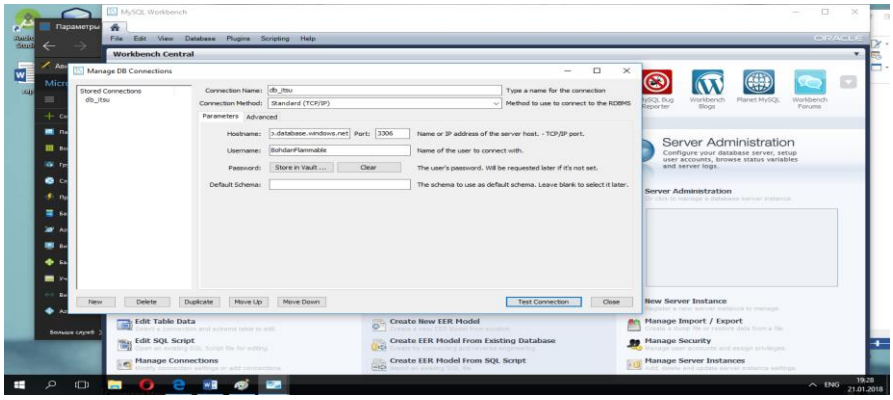

Рисунок 2.40 – Зберігання паролю в сховище паролів Workbench

Введемо пароль (рисунок 2.41):

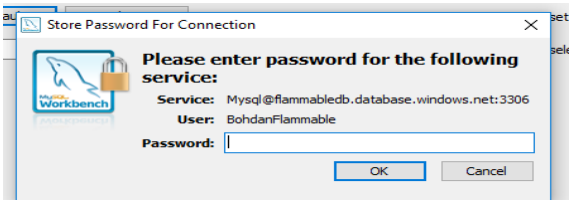

Рисунок 2.41 – Вікно введення паролю

Тепер можемо перевірити, чи все ввели вірно (рисунок 2.42):

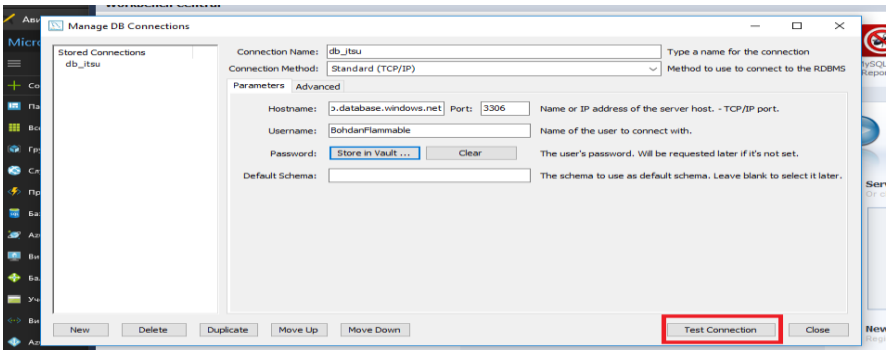

Рисунок 2.42 – Запуск тесту підключення

Підключення встановлено (рисунок 2.43):

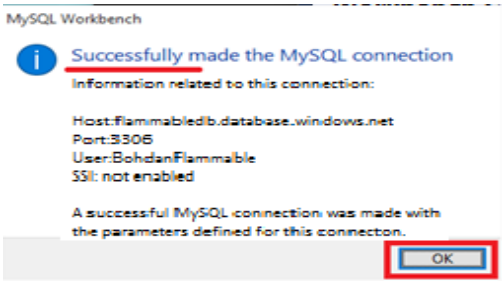

Рисунок 2.43 – Вікно підключення до БД

Тепер при запуску Workbench ми можемо в один клік підключитися до БД (рисунок 2.44 - 2.45):

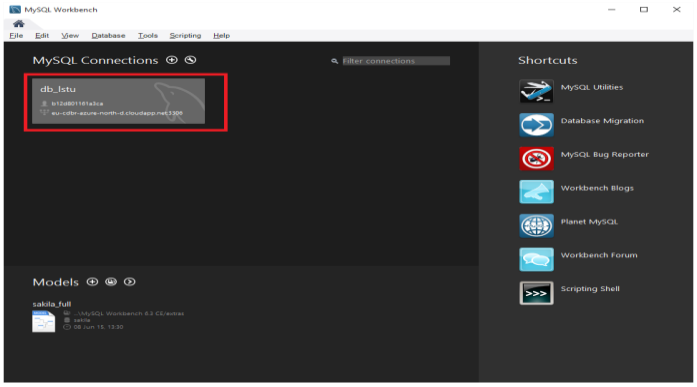

Рисунок 2.44 – Підключення до БД, яка розгорнута в хмарі

Трохи почекаємо:

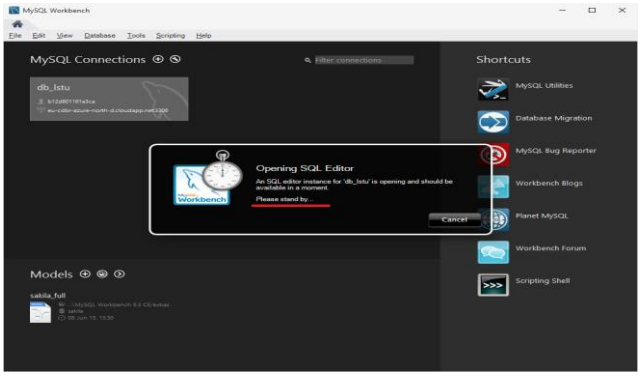

Рисунок 2.45 – Запуск сесії підключення до БД

Відкриється класичне середовище управління базами даних. Перейдемо до нашої БД (рисунок 2.46):

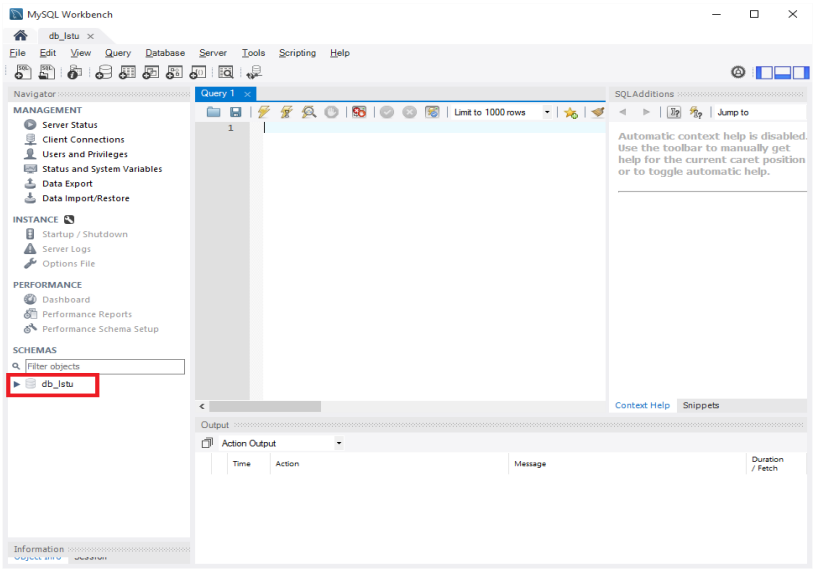

Рисунок 2.46 – Середовище управління базами даних

Як бачимо, БД порожня. Можете створити першу таблицю за допомогою SQL запиту CREATE TABLE у вікні Query (рисунок 2.47):

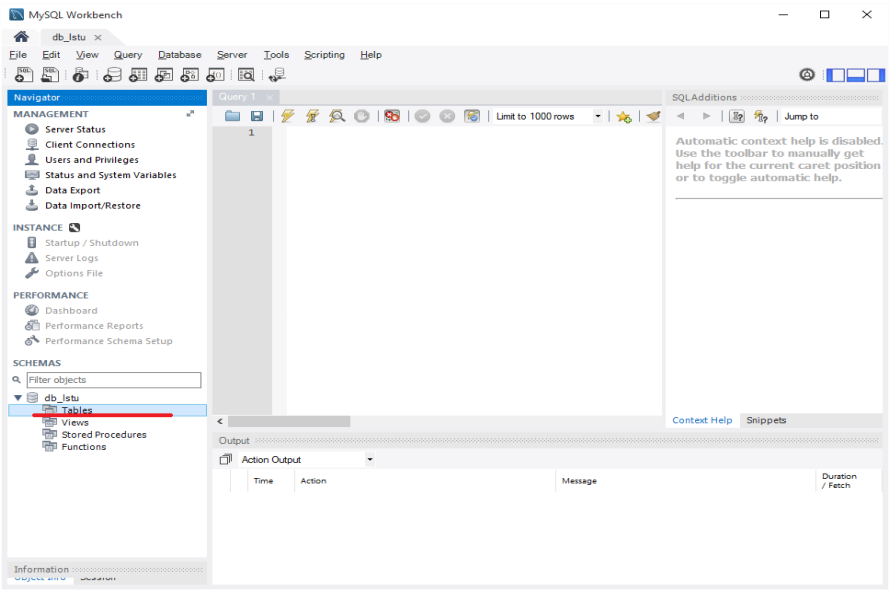

Рисунок 2.47 – Вікно для створення таблиць БД

Запустимо VS та створимо найпростіший Веб-проєкт. Це буде пусте ASP.NET рішення (рисунок 2.48):

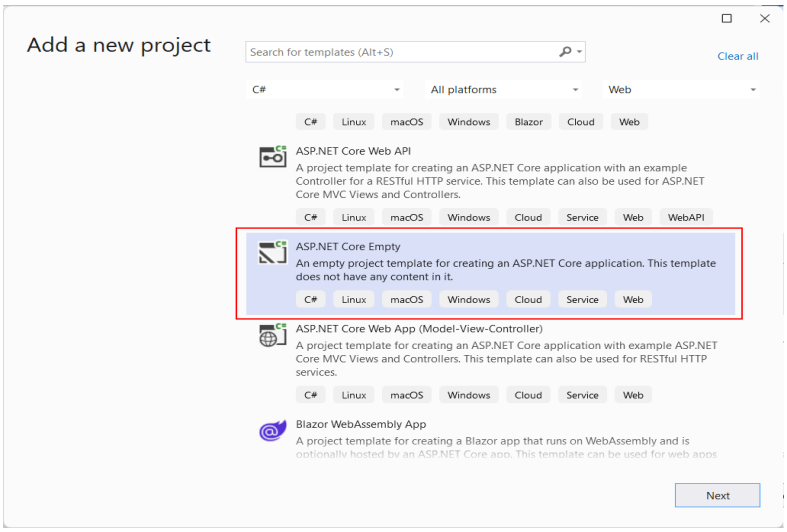

Рисунок 2.48 – Вибираємо пустий проект

Тепер перейдемо до «Оглядача серверів», щоб встановити підключення. Бачимо підписку (рисунок 2.49):

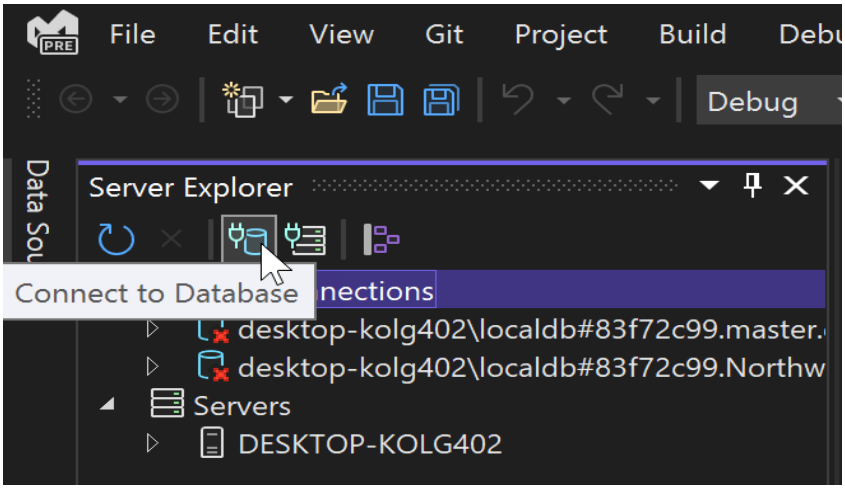

Рисунок 2.49 – Моя підписка

Кількома по іконці «Підключитися до бази даних».

Тепер в списку доступних провайдерів нам доступний MySQL.

Введемо облікові дані з консолі управління Azure і перевіримо доступність (рисунок 2.50):

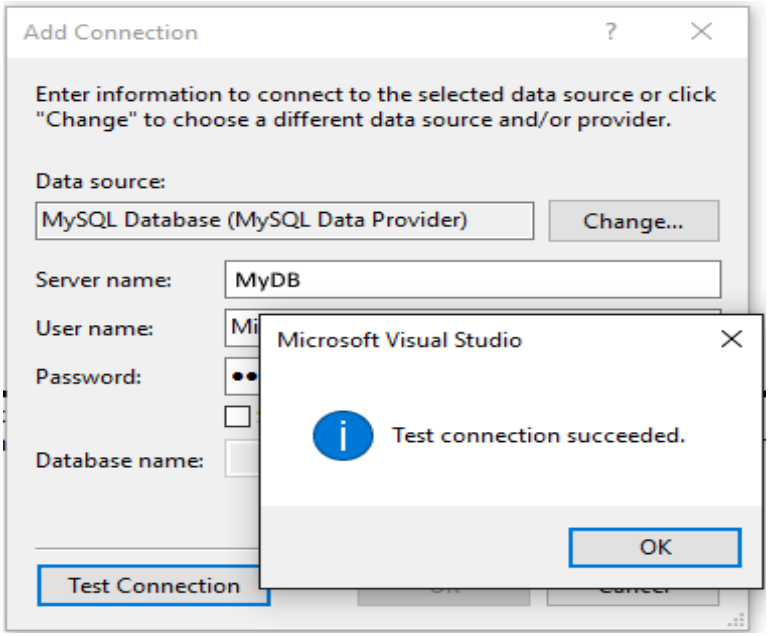

Рисунок 2.50 – Перевірка підключення до БД

Тепер зі списку бачимо БД (рисунок 2.51):

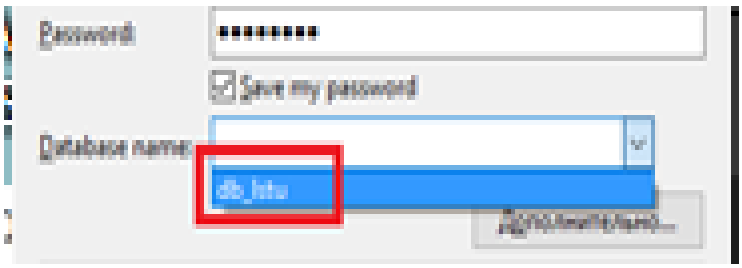

Рисунок 2.51 – Вибираємо нашу БД зі списку

Маємо повний доступ до БД MySQL, розміщеної в хмарі Azure, з VS.

Майстер створення тестового випадку.

Запуск майстра тестового випадку:

- вибір нового тестового випадку – відкриваємо меню «Test-інженер» і вибираємо опцію «Новий тестовий випадок (Test Case)»;

- аналіз БД – під час запуску майстер аналізує БД ssmatester2005db або ssmatester2008db на вихідному сервері Sybase, ці бази даних є розширенням схеми тест-інженера, де зберігаються допоміжні об'єкти;

- створення БД розширення – якщо майстер не знаходить потрібні бази даних (особливо при першому запуску), з’являється діалогове вікно для їх створення, натисніть «Так», щоб створити БД Sybase для тест-інженера, у новому діалоговому вікні натисніть «Додати», щоб вибрати пристрій для бази даних, вкажіть розмір бази даних і, за необхідності, налаштуйте окремий пристрій для журналів бази даних.

Процес створення тестового випадку складається з п'яти основних кроків:

Крок 1. Ініціалізація тестових випадків (SybaseToSQL) – підготовка середовища для створення тестових сценаріїв.

Крок 2. Вибір і настройка об'єктів для тестування (SybaseToSQL) – визначення об'єктів, які необхідно протестувати, алаштування параметрів тестування для кожного об'єкта.

Крок 3. Вибір і настройка порушених об'єктів (SybaseToSQL) – виявлення залежних чи змінених об'єктів, які можуть вплинути на результати тестування, коригування параметрів для забезпечення точності.

Крок 4. Налаштування порядку викликів (SybaseToSQL) – встановлення послідовності викликів об'єктів, щоб уникнути конфліктів чи збоїв під час виконання тестів.

Крок 5. Завершення підготовки тестового випадку (SybaseToSQL) – остаточна перевірка тестових параметрів, збереження тестового випадку.

Виконання тестових випадків (SybaseToSQL):

- запуск тесту – виконання збереженого тестового випадка, нагляд за процесом і результатами виконання через інтерфейс майстра;

- звітність – перегляд результатів тестування, зокрема успішність виконання кожного етапу, збереження звіту для аналізу або передачі в команду розробників;

- очищення середовища – після завершення видалення допоміжних об'єктів, що створені під час тестування.

Переваги використання майстра:

- спрощення процесу створення та виконання тестових випадків;
- інтуїтивний покроковий підхід для налаштування тестів;
- ефективне управління ресурсами та об'єктами бази даних;
- можливість швидко виявляти й усувати проблеми в об'єктах, перенесених із Sybase у SQL Server.

Процес виконання тестового випадку в SSMA Test Engineer.

Для успішного завершення перевірки необхідно:

- перетворити та завантажити всі об'єкти Sybase у цільову базу даних SQL Server;
- синхронізувати вміст таблиць між Sybase і SQL Server;
- забезпечити відповідність об'єктів відповідно до параметрів схеми для поточного проєкту SSMA.

Етапи виконання тестового випадку.

Запуск тесту: натисніть кнопку «Запуск» у Test Engineer, у діалоговому вікні «Connect to Sybase» введіть дані для підключення та натисніть «Connect».

Ініціалізація: SSMA Test Engineer створює допоміжні об'єкти для обох баз даних (Sybase і SQL Server), ці об'єкти дозволяють відстежувати зміни в таблицях, вибраних для перевірки.

Наприклад, для таблиці USER_TABLE створюються такі допоміжні об'єкти:

- у базі Sybase – таблиці, тригери, подання;
- у базі SQL – аналогічні об'єкти в базі даних «ssmatesterdb_syb».

Перевірка відповідності об'єктів: SSMA Test Engineer порівнює результати виконання об'єктів між Sybase і SQL Server, якщо результати збігаються, перевірка вважається успішною.

Після завершення тесту генерується «звіт про результати тестування»:

- натисніть кнопку «Reports», щоб переглянути звіт у Test Engineer;
- звіт автоматично зберігається в репозиторії для подальшого використання.

SSMA Test Engineer створює допоміжні об'єкти для трасування змін: таблиці (для зберігання змінених даних), тригери (для моніторингу змін у таблицях), подання (для спрощеного доступу до змінених даних).

Усі створені об'єкти зберігаються у відповідних базах. Sybase: ssmatester2005db або ssmatester2008db. SQL Server: ssmatesterdb_syb.

Цей процес забезпечує надійну перевірку даних і об'єктів між платформами, гарантуючи правильність міграції та відповідність функціоналу. Наступні об'єкти створюються в БД (таблиця 2.1), перевірених на Sybase і SQL Server.

Таблиця 2.1 – Ssmatesterdb_syb

Ім'я	Тип	Description
USER_TABLE $ Trg	тригер	Аудит змін в таблиці.
USER_TABLE $ Aud	Table	Таблиця збереження рядків, що видаляються і перезаписані.
USER_TABLE $ AudID	Table	Таблиця збереження нових і змінених рядків.
USER_TABLE	Перегляд	Спрощене уявлення зміни таблиць.
$ USER_TABLE new	Перегляд	Спрощене уявлення про вставлені і перезаписані рядки.
USER_TABLE $ new_id	Перегляд	Ідентифікація: вставлені і змінені рядки.
$ USER_TABLE old	Перегляд	Спрощене уявлення: рядки, що видаляються і перезаписані.

Це завершальний етап тестування: SSMA та управління об'єктами.

Дії тест-інженера SSMA. Виклик об'єктів для тестування: кожен об'єкт, вибраний для тестування, проходить через етап виклику SSMA (SQL Server Migration Assistant), для кожного об'єкта виконуються порівняльні аналізи результатів роботи. Порівняння результатів: виявлення відмінностей між очікуваними та фактичними результатами, оцінка відповідності результатів вимогам, включаючи час відгуку, коректність обробки даних, масштабованість тощо. Після завершення аналізу формується звіт із такими ключовими розділами: результати тестування (оцінка успішності тестових сценаріїв), відхилення (перелік проблем чи невідповідностей із зазначенням причин), рекомендації (пропозиції щодо вдосконалення чи оптимізації).

Завершення тестування:

- очищення допоміжних об'єктів – усі тимчасові об'єкти, створені під час ініціалізації, видаляються;

- забезпечення чистоти середовища – після очищення допоміжних об'єктів середовище повертається до початкового стану, готового для подальшого тестування або розгортання;

- документація процесу – усі дії, виконані під час тестування, фіксуються для створення документації, допомагає створити чіткий запис процесу тестування для аудиту чи повторного виконання.

Цей підхід забезпечує завершеність і якість тестування, а також готує базу для подальших етапів розробки та тестування.

3 ЕКСПЕРИМЕНТАЛЬНЕ ДОСЛІДЖЕННЯ ТЕСТУВАННЯ НАВАНТАЖЕННЯ ВЕБ-ЗАСТОСУНКІВ І БАЗ ДАНИХ

3.1 Результати експериментального тестування навантаження веб-застосунків

Спочатку подивимося на панель з VS. Відображаються значення ключових показників, часу відгуку сторінки і інші корисні для аналізу роботи застосунку параметри (рисунок 3.1).

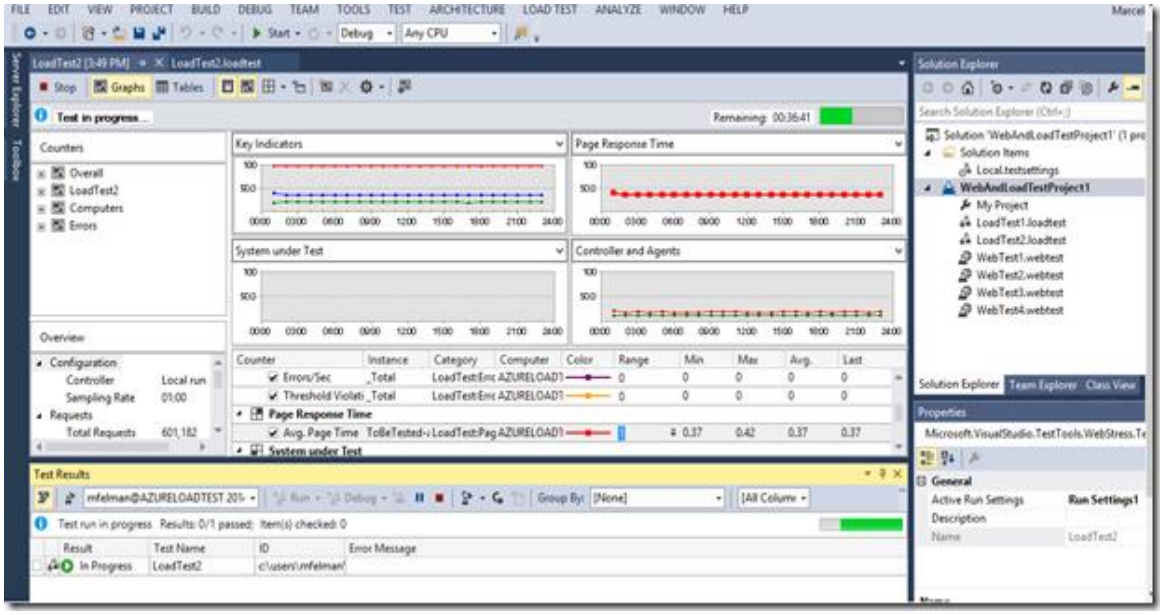

Рисунок 3.1 – Значення ключових показників

Тепер приступимо до найцікавішої частини – результат "кінцевого користувача" (end user). Як показано нижче, застосунок починає навантажуватися користувачами приблизно о 12:45. Стрибків в 100 створюваних кожну секунду запитів користувачів збільшує навантаження процесора до 63,58% (рисунок 3.2).

Рисунок 3.2 – Навантаження процесора

Попередній тест із помірним навантаженням показав прийнятний час відгуку сторінок – сторінка відповідала нормально, а значення часу відгуку відповідали встановленим вимогам.

Симуляція навантаження дозволила оцінити необхідну кількість примірників для стабільної роботи, Можливість масштабування для забезпечення ефективності.

Тепер мета – створити стресову ситуацію, щоб оцінити, наскільки система здатна витримувати високі пікові навантаження, а саме визначення порогу, за яким починається зниження продуктивності (збільшення часу відгуку, падіння запитів), перевірка використання процесора, пам’яті та інших ресурсів, виявлення компонентів системи (веб-сервери, балансувальники навантаження, база даних), які стають обмежуючим фактором, Чи зможе система витримати таке навантаження без аварій.

Умови тестування:

- кількість одночасних користувачів: 5000;
- частота запитів: 1 запит на користувача щосекунди (рисунок 3.3).

Запуск подібного сценарію дозволить отримати глибше розуміння про стійкість системи та її готовність до роботи в умовах значних навантажень.

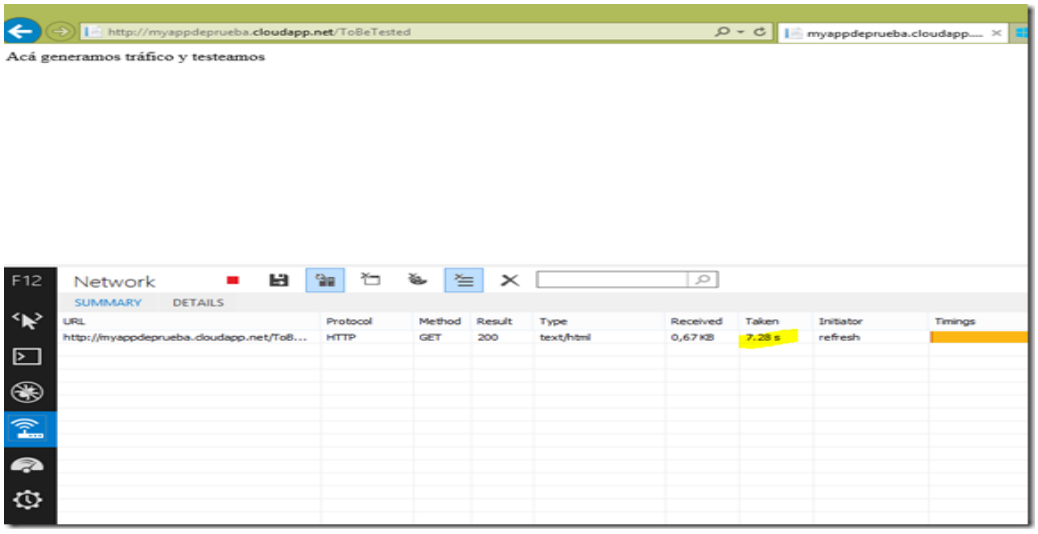

Рисунок 3.3 – Збільшення часу відгуку

Час відгуку збільшився. Це погано. Час повертається до нормального стану після закінчення стрибка (рисунок 3.4):

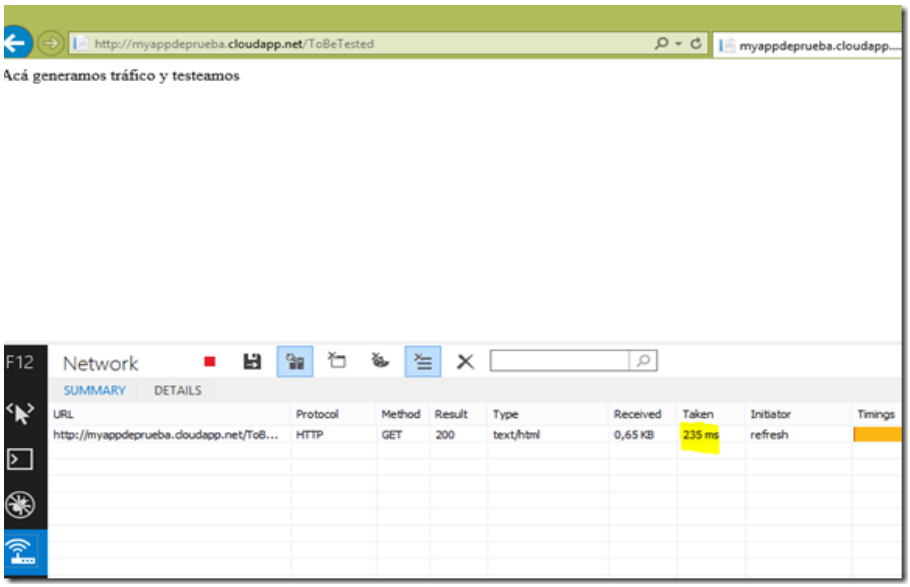

Рисунок 3.4 – Час відгуку після пікового навантаження

Завантаження процесора показана нижче (рисунок 3.5)

Рисунок 3.5 – Навантаження процесора

Результати тесту (рисунок 3.6):

Warm-up duration	00:00:00
Duration	00:07:00
Controller	Local run
Number of agents	1
Run settings used	Run Settings1

Overall Results

Max User Load	5,000
Tests/Sec	384
Tests Failed	5,882
Avg. Test Time (sec)	12.7
Transactions/Sec	0
Avg. Transaction Time (sec)	0
Pages/Sec	384
Avg. Page Time (sec)	12.7
Requests/Sec	766
Requests Failed	5,882
Requests Cached Percentage	0
Avg. Response Time (sec)	6.41
Avg. Content Length (bytes)	295

Рисунок 3.6 – Результати тестування навантаження засобами Azure

Збільшимо кількість примірників до двох (рисунок 3.7 - 3.8):

Рисунок 3.7 – Масштабування веб-застосунку в хмарі

Рисунок 3.8 – Результат масштабування веб-застосунку в хмарі

Запустимо повний тест з двома екземплярами. Балансування навантаження налаштовується на використання алгоритму Round Robin (тобто відправку запиту чергуючи екземпляри).

Перед початком тесту (рисунок 3.9):

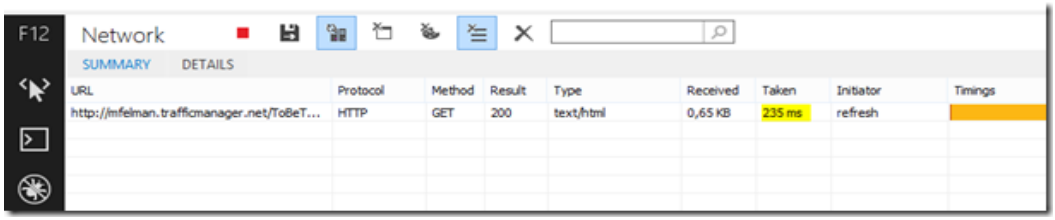

Рисунок 3.9 – Результати першого тесту

Під час тесту (рисунок 3.10):

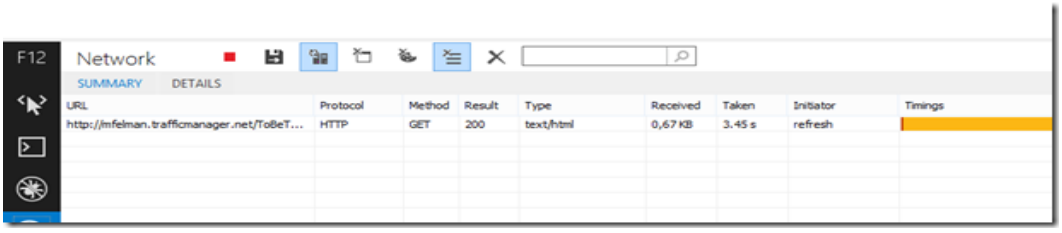

Рисунок 3.10 – Тест при збільшенні примірників веб-застосунку

Продуктивність знижується. Проте, час відгуку скорочується на вдвічі у випадку подвоєння кількості примірників (рисунок 3.11).

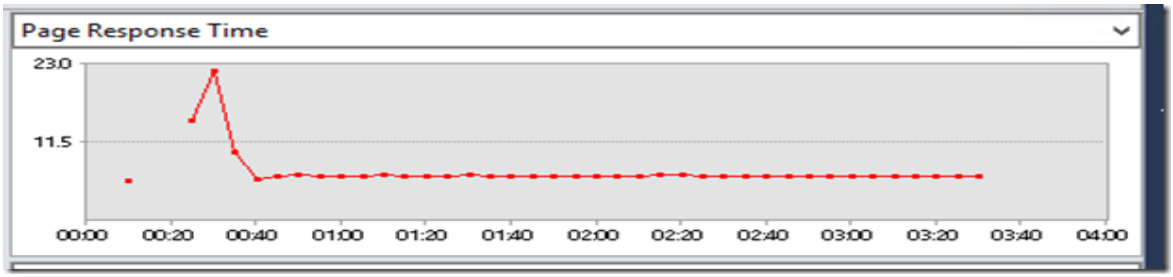

Рисунок 3.11 – Час відгуку скорочується

Залежність навантаження процесора показана на рисунку 3.12:

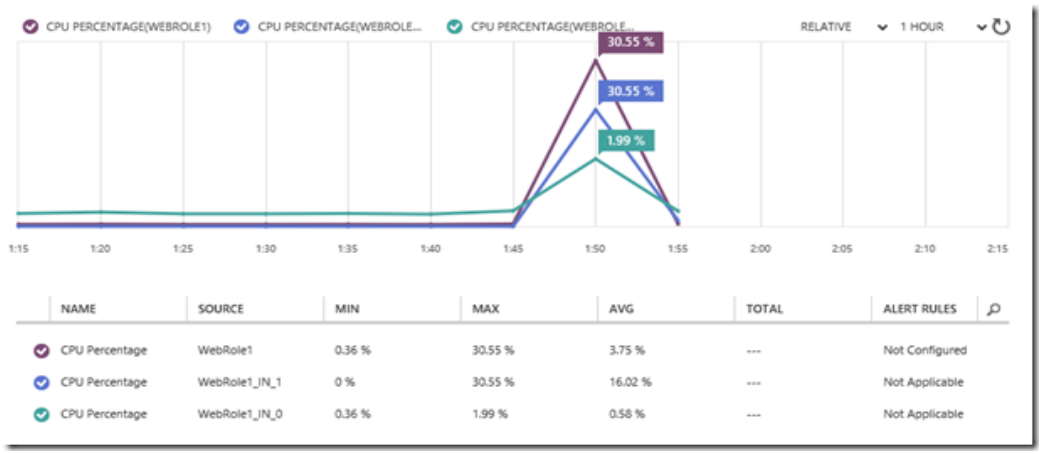

Рисунок 3.12 – Залежність навантаження процесора від кількості примірників веб-застосунку

За допомогою методів і інструментів, доступних у хмарі Azure, можна оцінити стійкість веб-застосунку до пікових навантажень. Таке тестування дозволяє визначити:

- межі продуктивності застосунку – зрозуміти, який обсяг одночасних запитів здатен обробляти;

- необхідну кількість ресурсів – провести точну оцінку кількості примірників (віртуальних машин, веб-ролей або інших компонентів), які необхідні для забезпечення потрібного рівня продуктивності та доступності;

- оптимізацію витрат – завдяки гнучкості масштабування Azure, уникнути перевитрат на інфраструктуру.

Всі необхідні інструменти для тестування та моніторингу доступні в середовищі Azure без додаткового програмного забезпечення. Налаштування займає мінімум часу (менше ніж година). Використання сервісів Azure для тестування дозволяє уникнути витрат на покупку дорогих інструментів або налаштування локальної інфраструктури. Навантажувальні тести в Azure забезпечують точні дані про продуктивність у реальних умовах роботи хмарної інфраструктури. Azure дозволяє оперативно збільшувати або зменшувати кількість примірників додатка, забезпечуючи високий рівень доступності під час пікових навантажень. Можна дізнатися, яка кількість користувачів або запитів може обробляти додаток при одночасному підключенні. Отримані результати дозволяють заздалегідь налаштувати автоматичне або ручне масштабування, що гарантує стабільну роботу навіть у періоди пікового навантаження. Завдяки автоматизації процесів та інтегрованим рішенням Azure, можна швидко провести аналіз та впровадити оптимізації.

3.2 Результати експериментального тестування навантаження бази даних

Для визначення максимального числа одночасно працюючих користувачів були визначені наступні параметри сценарію навантаження: навантаження плавно подається протягом 10 годин для фіксованого числа користувачів, починаючи зі значення 500 користувачів і збільшується кожні 30 хвилин на 500 користувачів до значення 10 000 одночасно працюючих користувачів (рисунок 3.13).

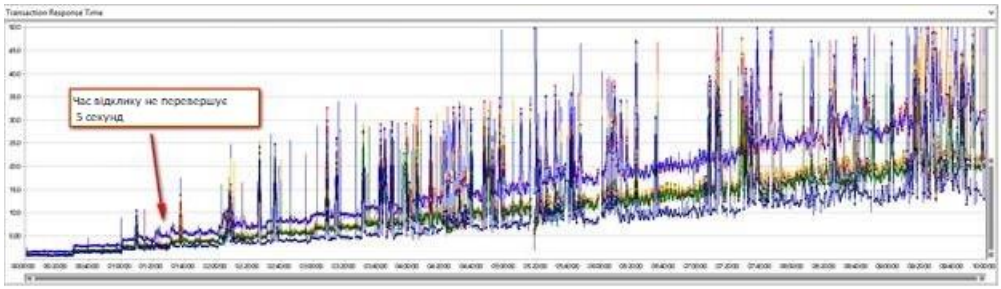

Рисунок 3.13 – Результати тестування навантаження БД, тест 1

За підсумками цього тесту можна зробити висновок, що після закінчення 1 години 30 хвилин додаток починає деградувати. Зіставивши ці дані з графіком кількості активних віртуальних користувачів, робимо висновок, що часи відгуків починають перевищувати значення 5 секунд при одночасній використання 1 500 користувачів. Однак протягом усього тесту, тривалістю 10 годин, не було зафіксовано таймаутів, що свідчить про можливість застосування витримувати навантаження до 10 000 одночасно працюючих користувачів зі збільшенням часу відгуку.

Провівши другий тест з умовами одночасної роботи 1 500 користувачів (за підсумками попереднього тесту) протягом 3 годин, ми отримали наступний графік часів відгуку, який говорить про те, що БД витримує навантаження в 1 500 віртуальних користувачів без деградації (рисунок 3.14).

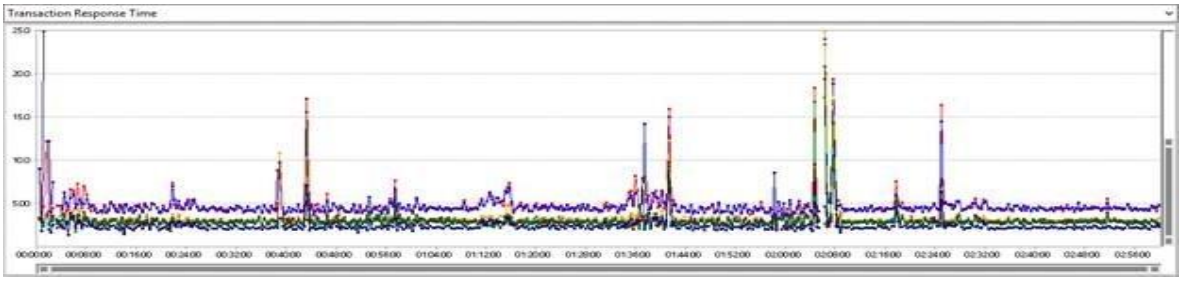

Рисунок 3.14 – Результати тестування навантаження БД, тест 2

Моніторинг утилізації ресурсів дозволяє зробити висновок, що має сенс переглянути конфігурацію сервера БД з метою зниження CPU, однак і

поточна конфігурація забезпечує одночасну безперебійно працювати з 1 500 користувачів (рисунок 3.15).

Counter	Instance	Category	Computer	Min	Max	Avg.
System under Test						
% Processor Time	_Total	Processor	ADV5IIS	40.1	80.3	55.5
Available MBytes	-	Memory	ADV5IIS	1,938	2,058	2,036
% Processor Time	_Total	Processor	ADV5SQL	43.9	94.7	80.9
Available MBytes	-	Memory	ADV5SQL	2,388	2,444	2,432

Рисунок 3.15 – Результати тестування

Засобами хмарної служби Microsoft Azure було проведено тестування навантаження БД, так само розгорнутого в хмарі. А завдяки тому, що інфраструктура навантажувального тестування була розгорнута в хмарі, ми забезпечили собі можливість в будь-який момент повторити навантажувальні випробування з мінімальним часом на підготовку і розгортанням стенду з новою конфігурацією – досить переналаштувати і запустити VM у хмарі.

3.3 Аналіз отриманих результатів

Розгортання хмарних застосунків у дво- або трирівневій моделі є популярним підходом для забезпечення гнучкості, масштабованості та управління в середовищі Azure. У цій моделі використовуються комбінації служб PaaS (хмарні служби Azure) та IaaS (віртуальні машини Azure).

Рівень представлення (Presentation Layer) відповідає за взаємодію користувача з застосунком та включає інструменти та сервіси: Web Roles (PaaS) у хмарних службах Azure, призначені для обробки HTTP/HTTPS-запитів від користувачів і Azure App Services, використовується для розміщення веб-застосувань, мобільних бекендів або API. Забезпечує автоматичне масштабування на основі навантаження, спрощене управління та розгортання.

Бізнес-рівень (Business Layer) відповідає за обробку бізнес-логіки застосунку та включає інструменти та сервіси Worker Roles (PaaS) у хмарних службах Azure які виконують фонові завдання, такі як обробка черг, розрахунки або інтеграція з іншими системами, функції Azure (Azure Functions), що використовуються для подій, заснованих на мікросервісній архітектурі.

Забезпечує легкість у реалізації складних бізнес-правил, масштабування для підвищення продуктивності.

Рівень даних (Data Layer) відповідає за зберігання та обробку даних. Містить SQL Server на віртуальних машинах Azure (IaaS), що використовується для розгортання реляційних БД, де потрібно повний контроль, Azure SQL Database (PaaS), підходить для застосунків, яким не потрібен повний доступ до операційної системи БД, Azure Storage або Cosmos DB – для нереляційного зберігання даних або великих обсягів інформації.

Переваги комбінації PaaS та IaaS:

- PaaS спрощує управління службами завдяки вбудованим інструментам моніторингу, масштабування та автоматичних оновлень;

- IaaS надає повний контроль над операційною системою та конфігураціями серверів;

- можливість горизонтального масштабування Web і Worker Roles для обробки великих обсягів навантаження;

- використання хмарних служб Azure для представлення та бізнес-рівнів спрощує моніторинг і автоматизацію розгортань;

- PaaS усуває потребу в ручному обслуговуванні інфраструктури, зменшуючи операційні витрати.

Дана модель використовує Azure Resource Manager (ARM) для управління ресурсами, розбиття БД на шарди або реплікація даних для покращення продуктивності, Azure Monitor, Application Insights або Log Analytics для збору телеметрії з усіх рівнів.

Типові сценарії використання:

- застосунки, що потребують інтеграції з наявними БД на SQL Server;
- системи з високими вимогами до масштабованості та розподілу навантаження;
- бізнес-додатки зі складною бізнес-логікою та великою кількістю фонових обробок.

Ця модель дозволяє ефективно використовувати ресурси Azure, адаптуючи хмарні служби та віртуальні машини для кожного рівня застосунку відповідно до їх специфіки та вимог.

Хмарні служби Azure надають потужну платформу для розгортання застосунків, виконуючи значну частину адміністративних завдань, таких як обслуговування інфраструктури, застосування оновлень, усунення збоїв та забезпечення високої доступності.

В частині інфраструктурного обслуговування Azure автоматично підтримує працездатність інфраструктури, надаючи такі послуги як планове обслуговування – автоматичне застосування оновлень до операційних систем без втручання користувачів, відновлення після збоїв – перезапуск примірників або перенесення їх на інші сервери у разі апаратного збою.

В частині горизонтального масштабування застосунків можливе ручне масштабування – збільшення або зменшення кількості примірників вручну через портал Azure або CLI, автоматичне масштабування – налаштовується на основі метрик, таких як використання CPU, трафік або затримка відповіді, що дозволяє оптимізувати продуктивність і витрати.

Visual Studio підтримує розробку, тестування та розгортання застосунків у хмарі безпосередньо з локального середовища. Інструменти інтеграції дозволяють створювати пакети для хмарних служб і керувати їх розгортанням у середовищі Azure.

Використання хмарних служб Azure (PaaS) підходить, якщо не потрібно виконувати складні адміністративні завдання, застосунок не потребує спеціальних конфігурацій програмного забезпечення або

операційної системи. Планується активне використання автоматичного масштабування.

Використання віртуальних машин Azure (IaaS) підходить, якщо потрібен повний контроль над конфігурацією операційної системи, використовуються спеціалізовані функції бази даних SQL Server, які недоступні в Azure SQL Database.

Переваги комбінованого підходу:

- хмарні служби забезпечують легке масштабування для рівня уявлення та бізнес-логіки;
- віртуальні машини дозволяють використовувати специфічні можливості для зберігання та обробки даних;
- використання хмарних служб позбавляє від необхідності виконувати рутинне адміністрування;
- автоматичне горизонтальне масштабування допомагає ефективно керувати ресурсами, зменшуючи витрати в періоди низького навантаження;
- Azure гарантує автоматичне відновлення після збоїв.

Сценарій розгортання веб-застосунку з інтенсивним навантаженням:

- рівень уявлення та бізнес-логіки – використовуйте Web Roles та Worker Roles для автоматичного масштабування та обслуговування;
- рівень даних – використовуйте SQL Server на віртуальних машинах, якщо потрібна підтримка складних запитів, що не доступні у Azure SQL Database.

Цей підхід дозволяє максимально використати переваги PaaS та IaaS, поєднуючи гнучкість, автоматизацію та контроль.

У моделі застосунку із трирівневою архітектурою в Azure (рисунок 2.13) визначено рівень представлення (Presentation Layer), який використовує веб-роль (компонент хмарних служб Azure) що налаштована для підтримки IIS та ASP.NET для розробки веб-додатків. На цьому рівні виконується обробка запитів користувачів і передача їх на бізнес-рівень, підтримує горизонтальне масштабування з використанням балансування навантаження.

Бізнес-рівень (Business Logic Layer), який використовує веб-роль (компонент хмарних служб Azure) що налаштована для виконання фонової обробки даних або складних обчислень. Може спільно розроблятися із рівнем представлення, забезпечує обробку запитів між рівнем представлення та рівнем даних.

Рівень даних (Data Layer) використовує віртуальні машини Azure із встановленим SQL Server використовується для застосунків, де Azure SQL Database не підтримує всі необхідні функції. Забезпечує високий рівень доступності та аварійного відновлення.

Рекомендації для використання моделі:

- міграція корпоративних додатків у хмару – переміщення застосунків із локальної інфраструктури до Azure з використанням можливостей SQL Server для високої доступності;

- динамічне масштабування – потреба в інфраструктурі, яка може автоматично збільшувати чи зменшувати обчислювальні ресурси залежно від навантаження;

- обмеження Azure SQL Database – якщо програми вимагають функціоналу, який недоступний у базах даних Azure SQL;

- тестування навантаження – використовується при необхідності масштабування ресурсів для обробки змінних робочих навантажень.

При налаштуванні та інтеграції моделі бажано застосувати *Azure Load Balancer, що розподіляє запити між кількома екземплярами веб-ролей та робочих ролей і забезпечує високу доступність за рахунок резервування, віртуальну мережа Azure, що забезпечує підключення компонентів застосунку (веб-роль, робоча роль, віртуальна машина SQL Server) через приватні IP-адреси та низькі затримки при обміні даними.

Шляхи оптимізації трирівневої моделі:

- консолідована веб-роль – для додатків із вимогою збереження стану об'єднує рівень представлення та бізнес-рівень в одній веб-ролі;

- зберігання стану сеансів – Azure Cache для швидкого доступу до даних, Azure Table Storage для економного зберігання великих обсягів даних, Azure SQL Database для більш складної обробки.

Трирівнева архітектура з використанням веб-ролей, робочих ролей та віртуальних машин Azure є оптимальним рішенням для додатків із високими вимогами до масштабування, доступності та складної обробки даних. Вона дозволяє ефективно використовувати ресурси Azure, забезпечуючи стабільну роботу і гнучке управління інфраструктурою.

ВИСНОВКИ

В результаті виконання дослідження проведено:

- аналіз підходів до тестування навантаження веб-застосунків і баз даних;
- огляд методів тестування навантаження веб-застосунків і баз даних;
- огляд інструментальних засобів тестування навантаження веб-застосунків і баз даних;
- аналіз методики тестування навантаження веб-застосунків;
- аналіз методики тестування навантаження баз даних;
- аналіз методики тестування навантаження веб-застосунків і баз даних з допомогою засобів Azure;
- створено базу даних засобами Azure;
- наведено результати експериментального тестування навантаження веб-застосунків;
- наведено результати експериментального тестування навантаження БД.

Модель із віртуальними машинами Azure, SQL Azure та веб-додатками Azure забезпечує баланс між швидкістю розгортання, гнучкістю налаштувань та масштабованістю. Вона підходить як для короткострокових задач (тестування чи розробка), так і для довготривалого використання в продуктивному середовищі.

Отримані результати можуть бути впроваджені в процесах тестування ПЗ та БД для поліпшення якості та надійності веб-сервісів, застосунків.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Wiegers K., Hokanson C. Software Requirements Essentials: Core Practices for Successful Business Analysis: 1st Edition. AddisonWesley Professional, 2022. 208 p.

2. Крепич С. Я, Співа І. Я. Якість програмного забезпечення та тестування: базовий курс : навч. посіб. для бакалаврів галузі знань 12 «Інформаційні технології» спеціальності 121 «Інженерія програмного забезпечення». / за ред. С.Я. Крепич, І.Я. Співа. Тернопіл : ФОП Паляниця В.А., 2020. 478с.

3. Тестування продуктивності [Electronic resource] – URL: https://training.qatestlab.com/blog/technical-articles/performance-testing/

4. Apache JMeter™ [Electronic resource] – URL: https://jmeter.apache.org/index.html

5. Open Text Professional Performance Engineering (Load Runner Professional) [Electronic resource] – URL : https://www.opentext.com/products/professional-performance-engineering

6. IBM DevOps Test Performance [Electronic resource] – URL : https://www.ibm.com/products/devops-test/performance

7. Pure Load: Powerful functional and performance testing for applications and networks [Electronic resource] – URL: https://emblasoft.com/product/pureload

8. Tricentis Neo Load: Continuous performance testing [Electronic resource] – URL : https://www.tricentis.com/products/performance-testing-neoload

9. Test the Performance of Web Applications Under Load [Electronic resource] – URL : https://www.loadtestingtool.com/

10. Top 20 Performance Testing Tools in 2024 [Electronic resource] – URL : https://www.browserstack.com/guide/performance-testing-tools

11. Pressman Roger, Maxim Bruce. Software Engineering: A Practitioner's Approach. NY : McGraw-Hill Education., 2019. 705 p.

12. Biswas Sourya. How Can Cloud Computing Help In Education? / Sourya Biswas. [Electronic resource] – URL : http://www.cloudtweaks.com/2011/02/how-can-cloud-computing-help-in- education/

13. Britto Marwin. Cloud Computing in Higher Education / Marwin Britto. Library Student Journal. [Electronic Resourse] – URL : http://www.librarystudentjournal.org/index.php/lsj/article/view/289/321

14. Introduction to Visual Studio Team Services [Electronic Resourse] – URL : https://learn.microsoft.com/en-us/shows/level-up/introduction-to-visual-studio-team-services

15. Get started with Azure DevOps documentation [Electronic Resourse] – URL : https://learn.microsoft.com/en-us/azure/devops/get-started/?view=azure-devops

16. Apache HTTP Server Documentation [Electronic Resourse] – URL : https://httpd.apache.org/docs/

17. JoeDog/siege [Electronic Resourse] – URL : https://github.com/joedog/siege/issues

18. Welcome to Tsung’s documentation! [Electronic Resourse] – URL : http://tsung.erlang-projects.org/user_manual/

19. Моделі хмарних сервісів: різниця між IaaS, SaaS, PaaS та приклади [Electronic Resourse] – URL : https://www.sim-networks.com/ukr/blog/cloud-computing-service-models

20. Browse Azure products [Electronic Resourse] – URL : https://learn.microsoft.com/en-us/azure/?view=azure-devops&product=popular

21. Trish Isaacs. Agile Teaching and the Agile Manifesto. Pedagogicon Conference Proceedings, 2. 2002. [Electronic Resourse] – URL : https://encompass.eku.edu/pedagogicon/2021/buckleup/2

22. Install and configure Team Foundation Server [Electronic Resourse] – URL : https://learn.microsoft.com/en-gb/azure/devops/server/admin/backup/tut-single-svr-install-config-tfs?view=azure-devops-2022

23. SQL Server technical documentation [Electronic Resourse] – URL : https://learn.microsoft.com/en-gb/sql/sql-server/?view=sql-server-ver16

24. Visual Studio 2022: Powerhouse of AI [Electronic Resourse] – URL : https://visualstudio.microsoft.com/vs/

25. Azure SQL Database documentation [Electronic Resourse] – URL : https://learn.microsoft.com/en-us/azure/azure-sql/database/?view=azuresql

26. Зінченко О.В., Прокопов С.В., Сєрих С.О., Василенко В.В., Березівський М.Ю. Хмарні технології : навч. посіб. – Режим доступу : https://duikt.edu.ua/uploads/l_2048_32915773.pdf.

27. Vakal V. APPROACHES TO BUILDING DATA ANALYSIS SYSTEMS / V. Vakal, V. Fedorchenko. Computer and information systems and technologies : Proceedings of Seventh International Scientific and Technical Conference, Kharkiv, 26-27 September, 2024. Kharkiv NURE, 2024. P. 50−52.