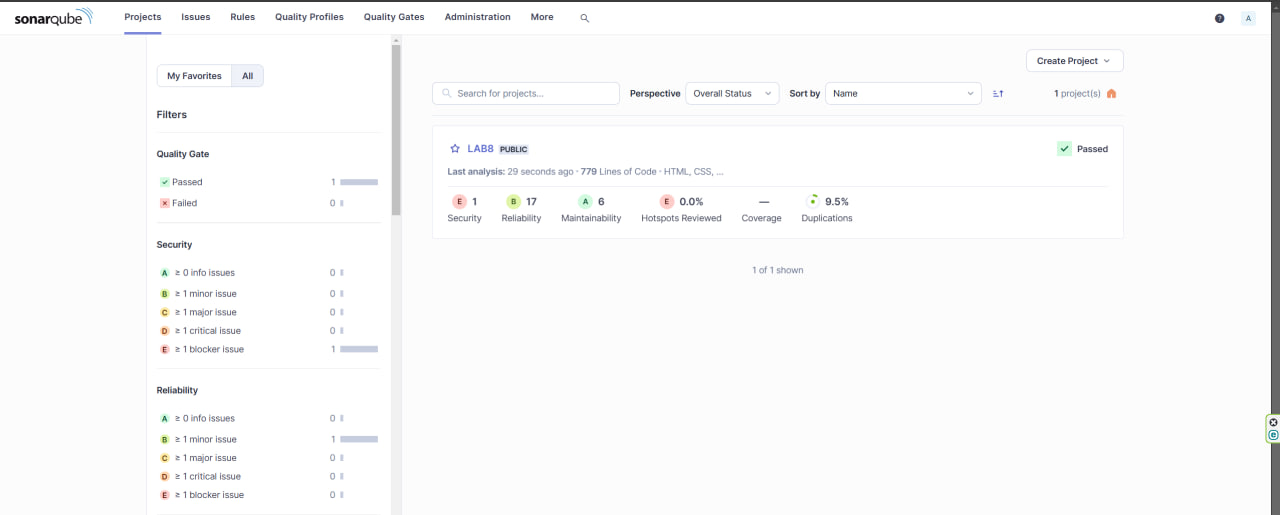
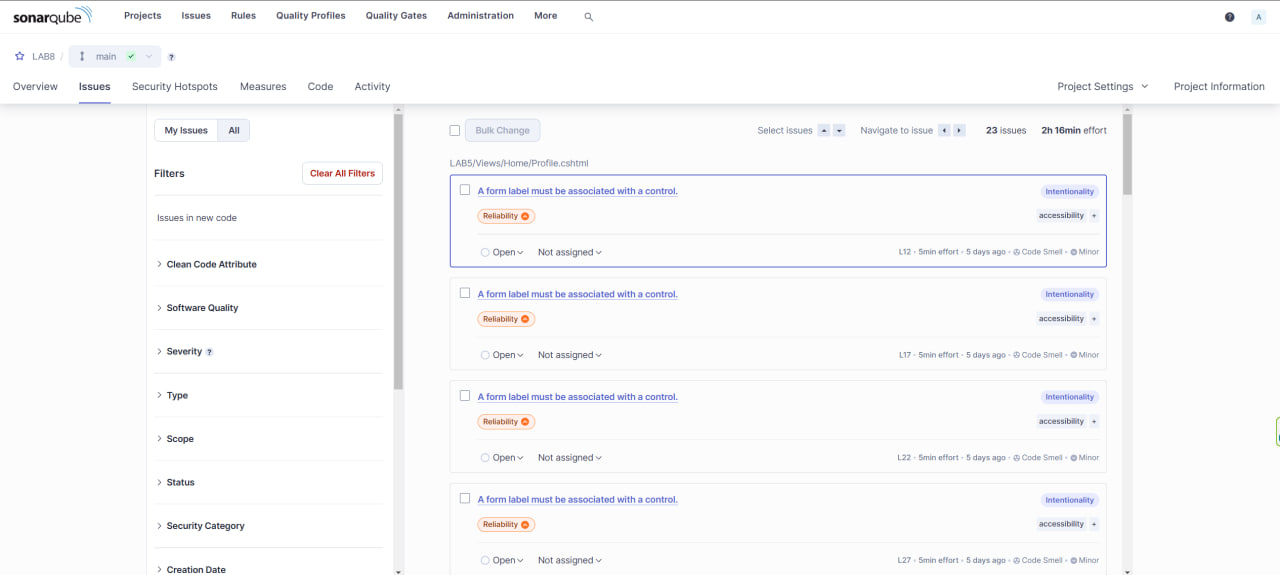
**Завдання 1: Оцінка якості коду за допомогою SonarQube**

1. Проведено аналіз коду програмного забезпечення з використанням SonarQube. Оцінено різні метрики якості, включаючи складність коду, дублікацію, покриття тестами та вразливості.
2. Підготовлено звіт з результатами аналізу, що містить візуалізацію метрик та рекомендації щодо покращення якості коду.



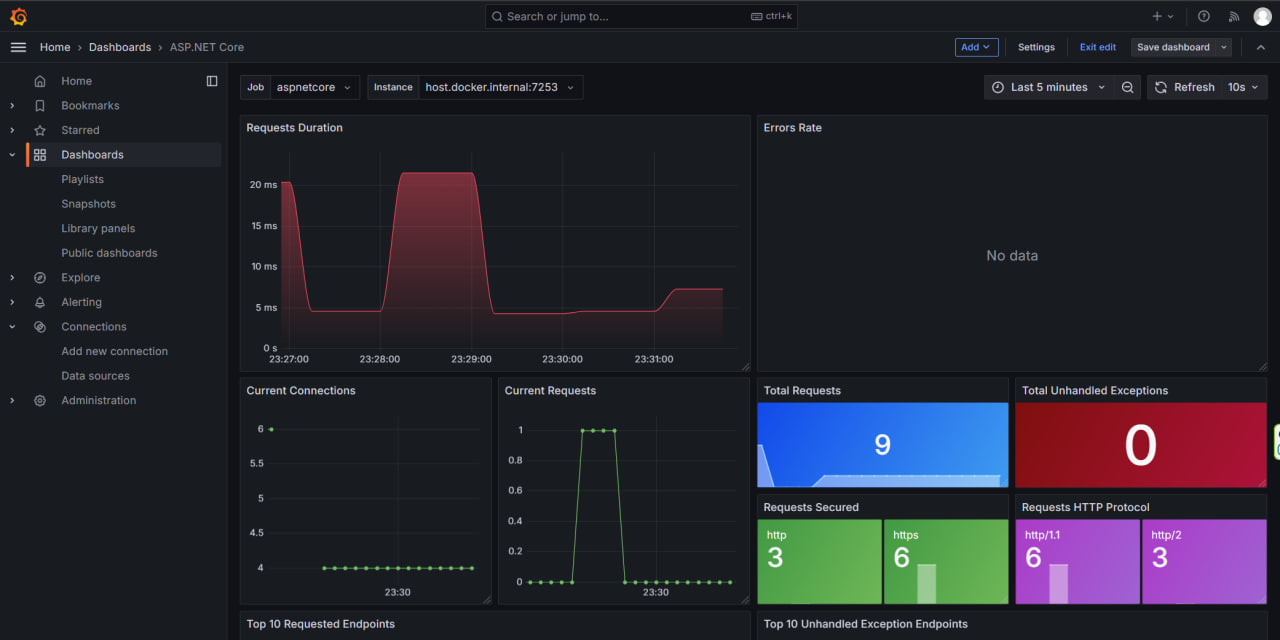


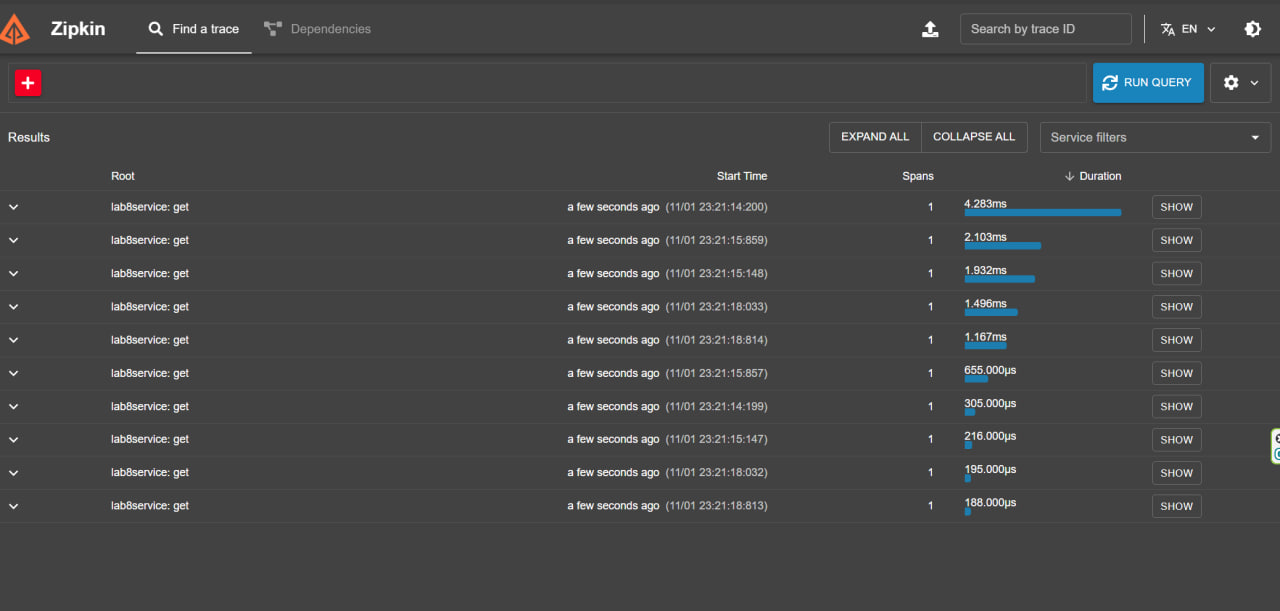
### Завдання 2: Інтеграція OpenTelemetry

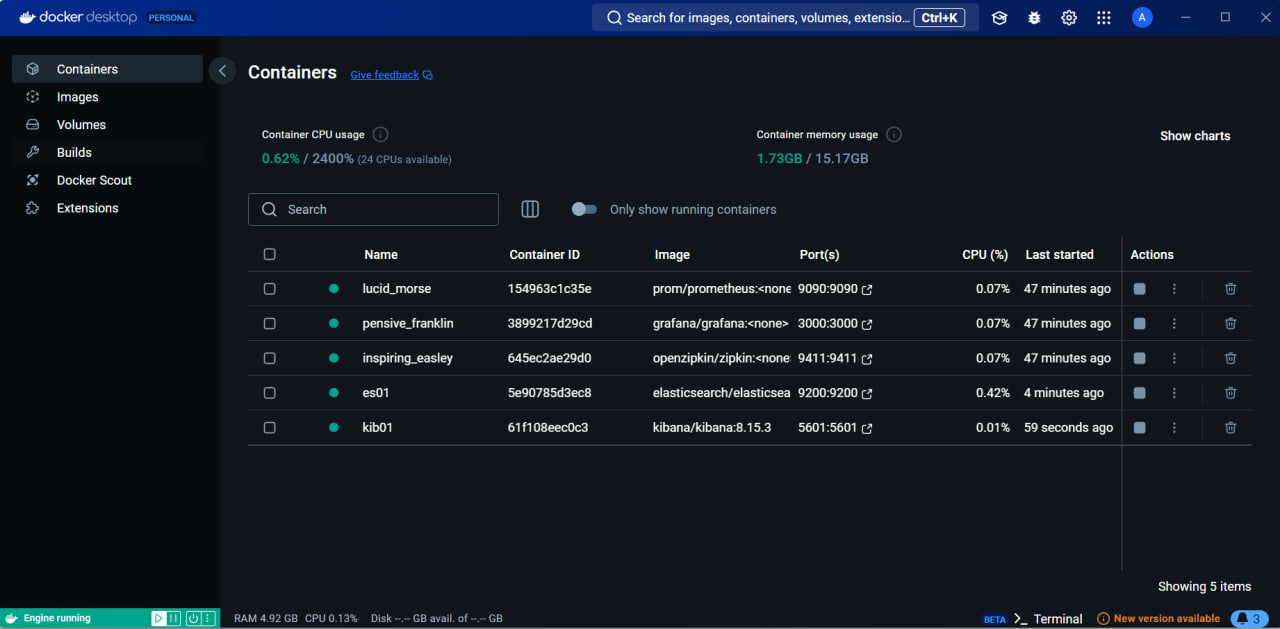
1. Інтегровано OpenTelemetry в застосунки ПР5 та ПР6. Зібрано метрики, такі як:
   * Кількість об'єктів в поколіннях GC
   * Розмір Heap
   * Використані системні ресурси
2. Зібрані метрики продемонстровані в Grafana та ElasticSearch. Реалізовано розподілений трейс з Zipkin для аналізу продуктивності, а також візуалізовано граф залежностей та логи в ElasticSearch.

### Завдання 3: Розширення можливостей трейсингу

1. Додано нові поля до системи трейсингу для збору додаткової інформації.
2. Створено додатковий SPAN для імітації тривалого процесу, що дозволяє детальніше аналізувати затримки в обробці запитів.





Контейнер на якому були розгорнути відповідні застосунки   


### Завдання 4: Навантажувальне тестування

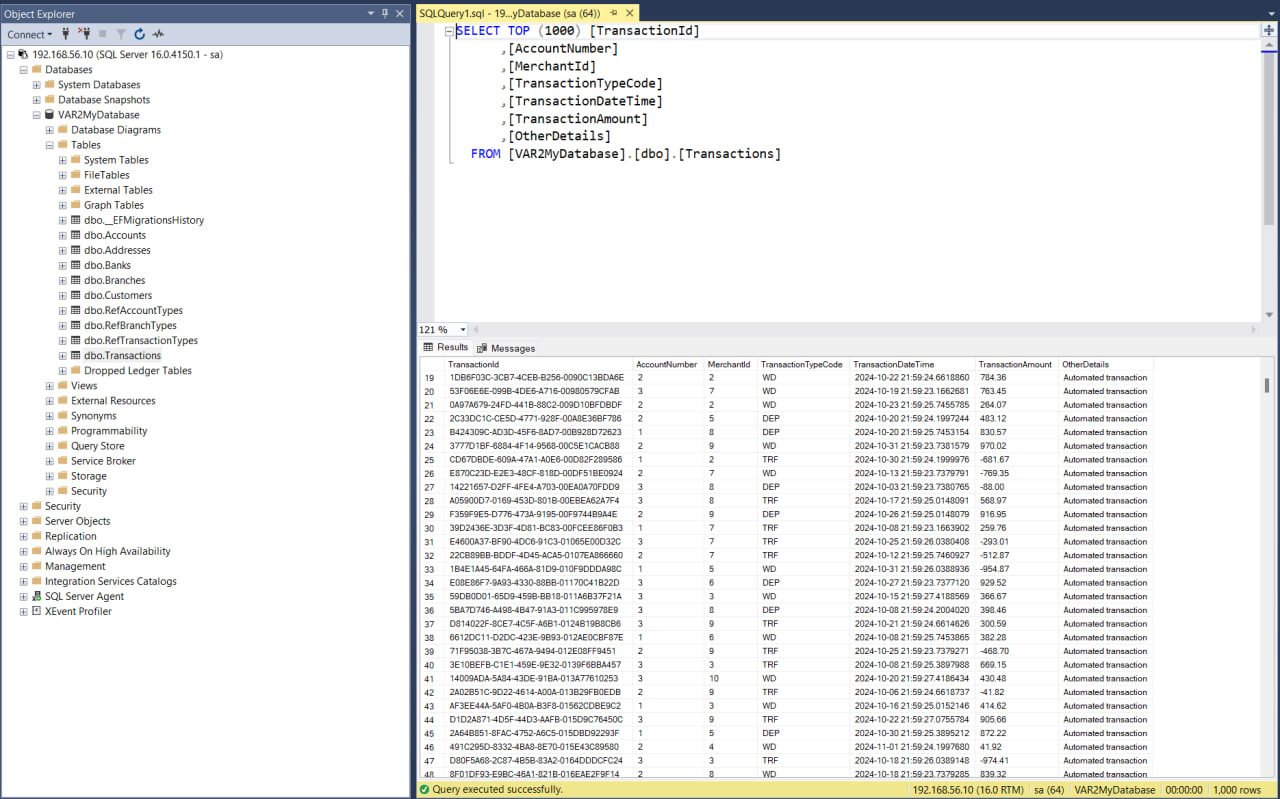
Наповнено базу даних понад 10,000 записами в центральні таблиці СУБД MsSQL шляхом реалізації консольного застосунку для генерації тестових даних. З метою тестування API, яке повертає всі ці 10,000 записів, було проведено навантажувальне тестування Asp.Net MVC застосунку за допомогою jMeter.

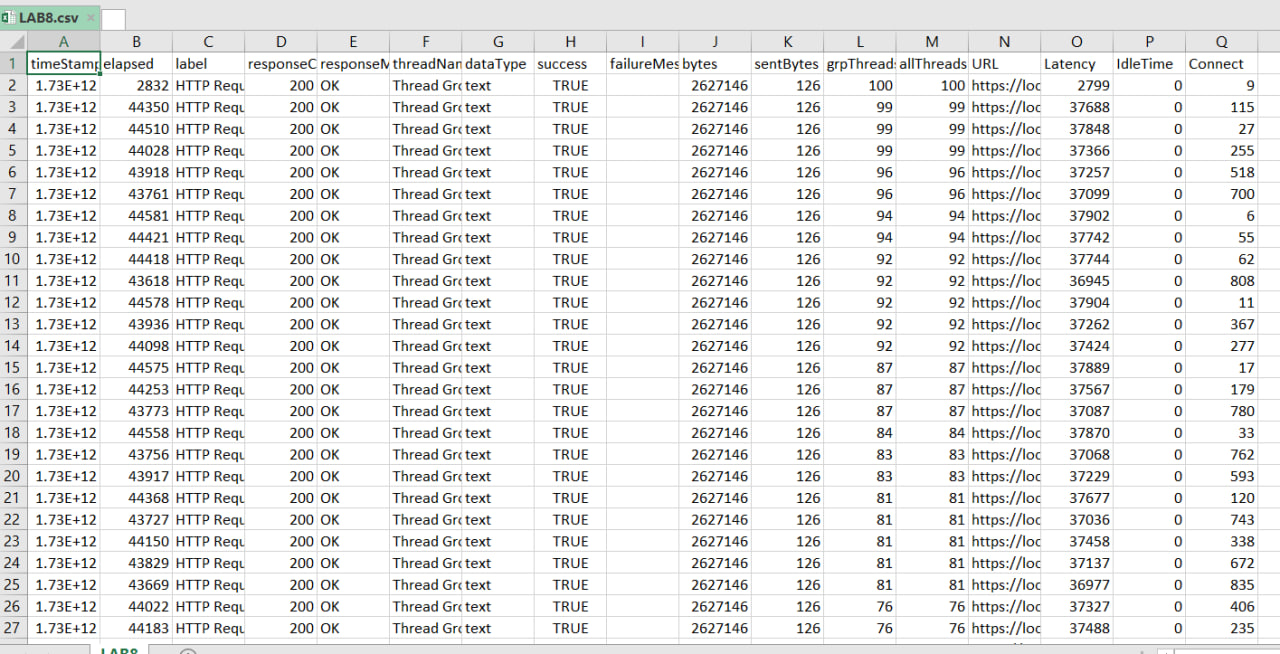
Під час тестування виміряно продуктивність при різній кількості одночасних користувачів (1, 5, 20, 50, 100, 300). Визначено, що зростання кількості користувачів пропорційно збільшує навантаження на сервер. Зафіксовано, що функція залежності між кількістю користувачів та часом відповіді була схожа на логарифмічну. Це свідчить про те, що з часом продуктивність починає знижуватись, але не в такій же мірі, як у лінійних або квадратичних функціях. Важливо зазначити, що під час тестування error rate не було виявлено, що вказує на стабільність та надійність системи при заданих навантаженнях.

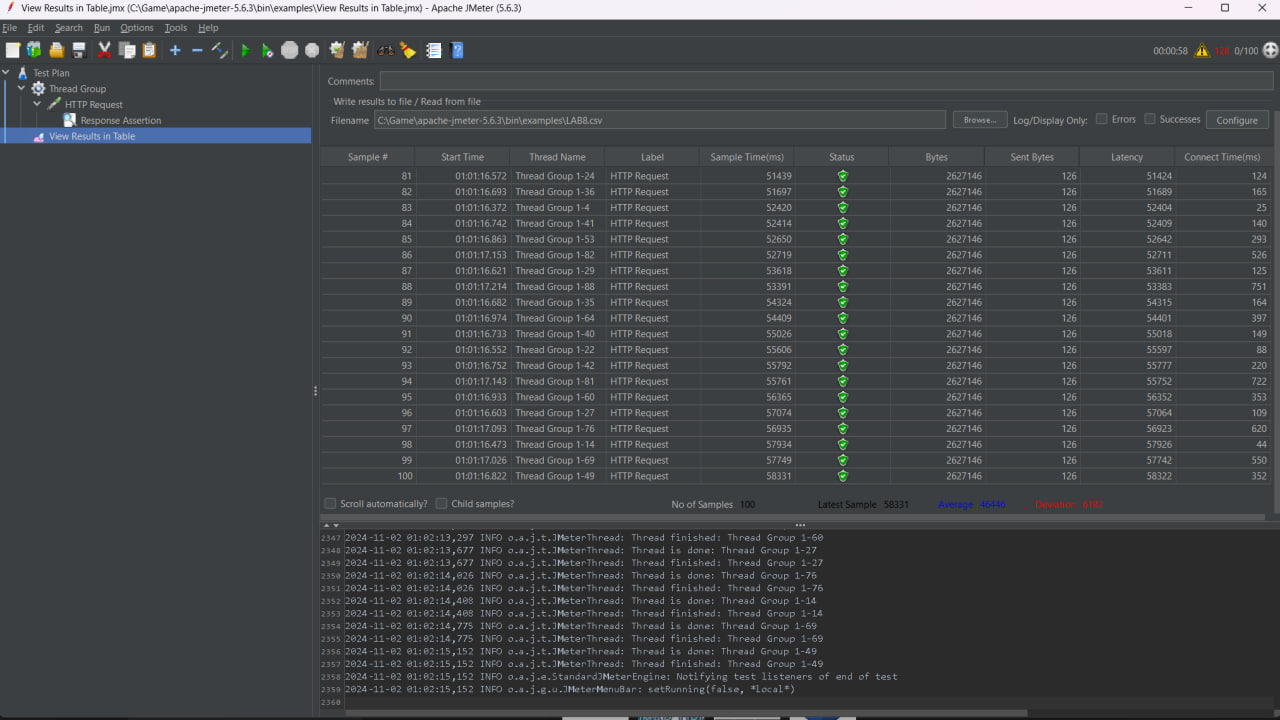
Для подальшої оптимізації Asp.Net MVC застосунку можуть бути запропоновані наступні заходи:

1. **Кешування даних:** Використання кешування на рівні контролера або в додатку може зменшити кількість запитів до бази даних, що підвищить швидкість відповіді. Зокрема, розглянути застосування MemoryCache або DistributedCache для зберігання часто запитуваних даних.
2. **Оптимізація запитів до бази даних:** Проаналізувати та оптимізувати SQL-запити, що використовуються для отримання даних. Використання індексів, уникнення неефективних JOIN-ів та підготовлених запитів можуть значно підвищити продуктивність.
3. **Асинхронні запити:** Застосування асинхронних методів у контролерах може покращити масштабованість програми. Це дозволить обробляти більше запитів одночасно, не блокуючи потоки.
4. **Використання пагінації:** Для API, що повертає великі набори даних, доцільно реалізувати пагінацію. Це дозволить зменшити обсяг даних, що передаються за один запит, а також зменшить навантаження на сервер.
5. **Аналіз та моніторинг:** Впровадження систем моніторингу та профілювання (наприклад, Application Insights, ELK Stack) для збору даних про продуктивність. Це дозволить виявити "вузькі місця" у застосунку та вчасно реагувати на проблеми.
6. **Зменшення розміру відповіді:** Використання таких технік, як стиснення (gzip), для зменшення обсягу передаваних даних.

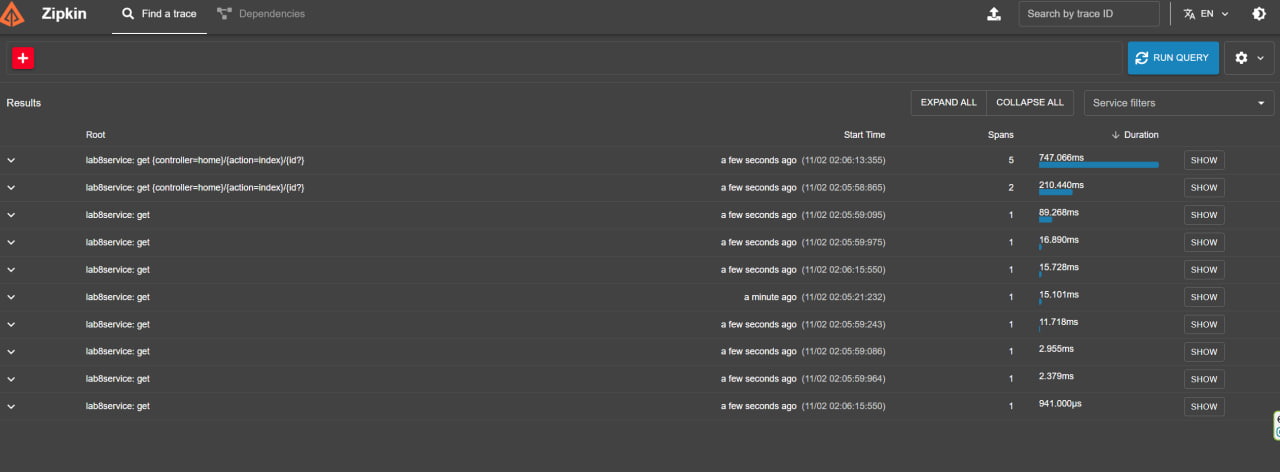
Реалізація зазначених заходів може суттєво покращити продуктивність та ефективність Asp.Net MVC застосунку під час навантажувального тестування та в реальних умовах експлуатації.

Штучні дані 10к у суБД MSSql  


JMetric  




Поведінка застосунку при навантажені у Graphana+InfluxDB, Zipkin







### Завдання 5: Тестування на безпеку

1. Проведено тестування безпеки застосунку з OWASP ZAP.
2. Підготовлено звіт з виявленими вразливостями та рекомендаціями для їх усунення.