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 перший (бакалаврський)

Розробка веб-застосунку для візуалізації даних з використанням баз даних часових рядів

(тема)

Виконав:
здобувач 4 року навчання,
групи КІУКІ-21-5
Костянтин УТОЧКІН
(власне ім'я, прізвище)

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

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

Освітня програма
Комп'ютерна інженерія
(повна назва освітньої програми)

Керівник: ст. викл. Яна Ні
(посада, власне ім'я, прізвище)

Допускається до захисту

Завідувач кафедри ЕОМ _____ Андрій КОВАЛЕНКО
(підпис) (власне ім'я, прізвище)

2025 р.

Харківський національний університет радіоелектроніки

Факультет комп’ютерної інженерії та управління

Кафедра електронних обчислювальних машин

Рівень вищої освіти перший (бакалаврський)

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

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

Освітня програма Комп’ютерна інженерія
(повна назва)

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

ЗАВДАННЯ

НА КВАЛІФІКАЦІЙНУ РОБОТУ

здобувачеві Уточкіну Костянтину Максимовичу
(прізвище, ім’я, по батькові)

1. Тема роботи Розробка веб-застосунку для візуалізації даних з використанням баз даних часових рядів

затверджена наказом по університету від “ 26 ” травня 2025 р. № 424 Ст

2. Термін подання здобувачем роботи до екзаменаційної комісії 18 червня 2025 р.

3. Вхідні дані до роботи 1) Часові ряди 2)Метрики 3) .NET Web API 4) Моніторинг 5) Система управління базою даних 6) Дашборди

4. Перелік питань, що потрібно опрацювати у роботі
1) аналіз теоретичних питань;
2) огляд технологій та підходів розробки;
3) реалізація візуалізацій;
4) опис системи моніторингу;
5) використання метрик;
6) висновки.

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

Слайд-презентація – 10 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	Строк / терміни виконання етапів роботи	Примітка
1	Теоретичний аналіз проблем та переваг	27.05.25-31.05.25	
	часових рядів		
2	Вибір інструментів для розробки	01.06.25-02.06.25	
3	Розробка веб застосунку	03.06.25-05.06.25	
4	Тестування роботи та інтеграцій	06.06.25-09.06.25	
5	Формування інтерфейсу для використання	10.06.25-13.06.25	
6	Подання кваліфікаційної роботи керівникові	15.06.25	
7	Подання кваліфікаційної роботи на рецензу-	16.06.25-17.06.25	
	вання		

Дата видачі завдання " 26 " травня 2025 р.

Здобувач _____
(підпис)

Керівник роботи _____ ст. викл. Яна Ні
(підпис) (посада, власне ім'я, прізвище)

РЕФЕРАТ

Пояснювальна записка кваліфікаційної роботи: 54 с., 21 рис.,1 дод., 17 джерел.

МЕТРИКИ, МОНІТОРИНГ, ЧАСОВІ РЯДИ, ВІЗУАЛІЗАЦІЯ, АНАЛІЗ, АГРЕГАЦІЯ ДАНИХ, БАЗИ ДАННИХ.

Метою кваліфікаційної роботи є дослідження використання баз даних часових рядів у сучасних веб застосунках, візуалізувати метрики та дані що будуються на основі часового ряду та продемонструвати необхідність впровадження баз даних часових рядів у сучасних системах

У ході виконання кваліфікаційної роботи було проаналізовано теоретичні відомості про такий тип даних як часовий ряд, де його доцільніше використовувати, створено панель для візуалізації даних та метрик, створено веб застосунок для роботи з базою даних часових рядів та обробки даних для подальшої візуалізації. Використано найбільш популярній інструментарій відкритого доступу для досягнення цілей – Grafana, налаштовано і інтегровано у веб застосунок систему моніторингу Prometheus визначено важливість та зручність використання часових рядів та спеціалізованих баз даних що дозволяють взаємодіяти з ними і вивчено можливість та способи обробки даних відповідно до різних типів баз даних.

ABSTRACT

Bachelor’s thesis: 54 pages, 21 figures , 1 appendices, 17 sources.

METRICS, MONITORING, TIME SERIES, VISUALIZATION, ANALYSIS, DATA AGGREGATION, DATABASES

The major goal of this thesis is to explore the use of time series databases in modern web applications, visualize metrics and data based on time series, and demonstrate the necessity of integrating time series databases into contemporary systems.

During the development of this thesis, the theoretical foundations of time series data were analyzed, including where and when their use is most appropriate. A dashboard was built for visualizing data and metrics, and a web application was developed to interact with a time series database and process the data for further visualization. To achieve these goals, widely adopted open-source tools were utilized, such as Grafana. Additionally, the Prometheus monitoring system was configured and integrated into the web application. The research highlights the importance and convenience of using time series data and specialized databases designed to work with them, as well as the various methods of data processing suitable for different types of databases.

ЗМІСТ

СКОРОЧЕННЯ ТА УМОВНІ ПОЗНАКИ

БД – база даних

ЗЗЧІ – запитів за час інтервалу

API – набір правил та інструментів, що дозволяють різним програмним забезпеченням взаємодіяти між собою (англ. Application Programming Interface)

DevOps – взаємодія і розробка, також поняття про людину що відповідає за процеси моніторингу систем (англ., development and operations)

DDoS – вид кібератаки що атакує сервер намагаючись перевантажити їх трафіком (англ. Distributed Denial of Service)

IoT – інтернет речей (англ., Internet of Things)

REST – підхід до архітектури мережевих протоколів, які надають доступ до інформаційних ресурсів (англ. Representational State Transfer)

SQL – мова програмування, призначена для управління та роботи з базами даних, зокрема реляційними (англ. Structured Query Language)

TSDB – база даних часових рядів (англ., Time Series Data Base)

ВСТУП

У сучасному світі веб-застосунки генерують величезні обсяги даних, значна частина яких має часову прив’язку. Відстеження продуктивності серверів, активності користувачів, кількості запитів, помилок, змін у бізнес показниках – усе це приклади метрик, значення яких змінюються з плином часу. Для ефективного збору, зберігання, аналізу та візуалізації таких даних все частіше використовуються спеціалізовані бази даних часових рядів.

На відміну від традиційних реляційних або документно-орієнтованих баз даних, TSDB (TSDB) оптимізовані саме для роботи з послідовними часовими даними. Вони забезпечують високу швидкодію при записі великої кількості точок даних, зручні механізми агрегації, компресію, а також підтримують інтеграцію з системами моніторингу та візуалізації, такими як Grafana або Prometheus.

Застосування TSDB у веб-розробці особливо актуальне для моніторингу стану системи та аналізу поведінки користувачів у реальному часі. Вони дозволяють розробникам і адміністраторам вчасно виявляти аномалії, аналізувати навантаження на систему, приймати обґрунтовані рішення для масштабування та оптимізації.

Використання баз даних часових рядів у сучасних веб-застосунках не лише покращує якість обслуговування, а й сприяє підвищенню надійності, стабільності та адаптивності цифрових сервісів завдяки високій продуктивності, масштабованості та точності обробки часових даних тому важливо підтримувати їх застосування у сучасній розробці

1 ТЕОРЕТИЧНА ЧАСТИНА

1.1 Поняття та особливості часових рядів

Часовий ряд – це впорядкована послідовність даних, кожне значення якої пов’язане з певною міткою часу. Іншими словами, це набір спостережень, зроблених або зафіксованих через регулярні або нерегулярні проміжки часу [3]. Типовий приклад часового ряду – температура повітря, виміряна щогодини, або кількість запитів у будь-який проміжок часу.

Основними характеристиками часових рядів є наявність часової мітки, регулярність даних та хронологічний напрямок аналізу. Кожне значення в часовому ряді має унікальну часову прив’язку – це означає, що дані фіксуються у певний момент часу. Така мітка може бути фіксованою, коли значення записуються з постійним інтервалом (наприклад, щосекунди або щохвилини), або змінною – коли події виникають нерегулярно, як от дії користувача чи спрацювання сенсора. Залежно від цього часові ряди поділяються на регулярні та нерегулярні. Аналіз таких даних завжди враховує хронологічну послідовність, оскільки саме зміна значень у часі дозволяє виявляти тренди, аномалії або сезонні коливання. Час у цьому випадку виступає ключовим фактором, який визначає порядок та зв’язки між окремими спостереженнями.

1.2 Типи даних для приладного застосування часових рядів

У прикладних задачах часові ряди використовуються для відстеження, аналізу та прогнозування змін у даних, що залежать від часу. Залежно від сфери застосування, часові ряди можуть містити різні типи даних. Розуміння цих типів є важливим для ефективного зберігання [1], обробки та візуалізації інформації.

Часові ряди можуть містити числові (кількісні) дані, які є найпоширенішим типом. Вони дозволяють відстежувати зміни величин у часі та аналізувати їх математично. Наприклад, це може бути температура повітря, що вимірюється кожні 10 секунд, кількість відвідувачів сайту щохвилини, рівень заряду акумулятора пристрою раз на годину або денний обсяг продажів в інтернет-магазині. Такі дані є основою для аналізу кліматичних умов, управління енергоспоживанням, аналітики в реальному часі та фінансового моніторингу.

Окрім того, часові ряди можуть включати дискретні дані, що фіксують зміни стану об’єктів у певні моменти часу. Вони не змінюються поступово, як числові, а переходять між заздалегідь визначеними значеннями. Наприклад, це можуть бути стани сервера на кшталт “активний”, “зупинено” чи “перевантаження”, або режими роботи пристрою – “вимкнено”, “робота”, “очікування”. У медицині це може бути стан пацієнта, позначений як “нормальний” чи “критичний”. Такі дані є критично важливими для контролю технічного стану систем, логістичних процесів

Бінарні (логічні) дані є ще одним типом інформації у часових рядах. Вони мають лише два можливих значення – істина або хиба, наприклад true/false або 1/0. Вони надзвичайно корисні для відстеження подій, які або відбулися, або ні. Типовими прикладами є спрацювання датчика руху (рух виявлено / не виявлено), виявлення помилки в системі або фіксація натискання кнопки користувачем. Такі дані часто використовуються в системах сигналізації, охороні, безпеці та моніторингу подій.

Часові ряди можуть також включати текстові або анотаційні дані, що слугують для фіксації додаткової інформації, яка не є числовою, але важлива для інтерпретації подій. Це можуть бути повідомлення в логах, наприклад: “Користувач увійшов у систему”, або описи оновлень: “Перезапуск системи завершено”. Подібні мітки полегшують аудит подій, аналіз помилок, ведення журналів змін і дозволяють краще розуміти контекст, у якому змінилися числові значення.

Комбіновані або структуровані дані є комплексним типом, у якому в межах одного часового запису поєднується кілька різних типів інформації. Наприклад, один запис може містити числове значення, статус і опис події. Такий підхід є характерним для IoT-платформ, розширених систем моніторингу чи діагностики обладнання, де потрібна повна картина змін: як з боку параметрів, так і з боку подій, які супроводжують ці зміни та є зазвичай найчастіше використовуваним через можливу реалізацію будь яких інших типів даних для логіки роботи.

```
{timestamp: "2025-05-01T10:00", temperature: 22.5, status: "OK", note: "Normal"}

{time: "12:00", cpu_load: 85, error: true, error_message: "Overload"}
```

Рисунок 1.1 Вигляд комбінованих даних часових рядів

1.3 Моніторинг та візуалізація даних у веб-застосунках

Моніторинг та візуалізація даних у веб-застосунках є надзвичайно важливими складовими сучасної архітектури програмного забезпечення, оскільки вони забезпечують постійне спостереження за станом системи, ефективне виявлення проблем і прийняття обґрунтованих рішень на основі реальних даних. У світі, де цифрові продукти працюють безперервно, а збої в роботі навіть на кілька хвилин можуть призвести до фінансових втрат або втрати довіри користувачів, якісний моніторинг є ключем до стабільної роботи сервісу.

Веб-застосунки генерують величезну кількість інформації – від рівня навантаження на сервери до поведінки користувачів на сторінках. Ці дані можуть надходити щосекунди або ще частіше, тому збереження і аналіз у реальному часі можливий лише за наявності відповідних інструментів моніторингу. Без належного контролю можуть залишитися непоміченими критичні події, як приклад зниження продуктивності, помилки в API, перевищення допустимого часу відповіді чи відмова сторонніх сервісів.

Візуалізація даних у вигляді графіків, діаграм і дашбордів значно спрощує процес аналізу. Людина значно краще сприймає візуальну інформацію, ніж сирі числові таблиці чи логи. Завдяки візуалізації можна миттєво виявити аномалії, наприклад, різке зростання навантаження або падіння кількості активних сесій. Крім того, графічне представлення дозволяє простіше відстежувати довгострокові тренди, сезонність і ефективність впроваджених змін.

Інтеграція систем моніторингу у веб-застосунки також сприяє покращенню розробки та супроводу проєктів. Розробники можуть швидше тестувати нові функції, аналізуючи їхній вплив на продуктивність, а DevOps-фахівці – оперативно реагувати на інциденти завдяки попереджувальним сповіщенням та історичним даним. Моніторинг також допомагає у прийнятті бізнес-рішень: наприклад, аналіз пікових навантажень може вплинути на вибір часу для проведення рекламних кампаній або оновлень системи.

Таким чином, моніторинг та візуалізація не лише допомагають виявляти і вирішувати технічні проблеми, а й є інструментами стратегічного управління веб-застосунком. Їх важливість постійно зростає разом зі складністю систем, кількістю користувачів та обсягом оброблюваних даних.

Окрім технічної користі, моніторинг і візуалізація даних у веб-застосунках відіграють важливу роль у забезпеченні надійності, прозорості та довіри до сервісу. У світі, де конкуренція між онлайн-продуктами є дуже високою, здатність демонструвати стабільність і контроль над інфраструктурою – це перевага, яка напряму впливає на лояльність користувачів і партнерів. Користувач очікує, що веб-застосунок буде доступним і швидкодіючим у будь-який момент, незалежно від часу доби чи навантаження на систему.

З погляду управління, візуалізовані дані дозволяють не лише реагувати на події, а й прогнозувати їх. Завдяки часовим рядам, можна будувати моделі для передбачення навантаження у святкові дні, пікові години користування або навіть ймовірність відмови певних компонентів системи. Такий підхід,

орієнтований на прогнозування, дає змогу не просто "гасити пожежі", а діяти на випередження – масштабувати інфраструктуру завчасно, оптимізувати використання ресурсів або навіть автоматично вмикати резервні потужності при підозрі на перевантаження.

Ще однією перевагою систем моніторингу є можливість автоматизованої реакції на інциденти. У разі виявлення критичних відхилень система може автоматично надсилати сповіщення відповідальним особам, запускати скрипти для перезапуску сервісів, або тимчасово обмежувати доступ до окремих функцій для збереження працездатності загальної системи.

У сфері бізнес-аналітики візуалізація (рисунок 1.2) є незамінним інструментом для прийняття стратегічних рішень. Наприклад, дані про поведінку користувачів у часі дозволяють визначати найбільш популярні функції, часи активності, або рівень утримання клієнтів. Ці знання можуть лягти в основу рішень щодо розвитку функціоналу, маркетингових кампаній чи зміни тарифної політики та інших рішень

Рисунок 1.2 Приклад візуалізації даних про користувачів сервісу

У процесі командної розробки системи моніторингу і дашборди також виконують роль комунікаційного засобу. Вони дозволяють різним учасникам

команди – розробникам, адміністраторам, бізнес-аналітикам – працювати з єдиним джерелом даних, але в зручному для кожного вигляді. Це сприяє швидшому прийняттю рішень та підвищує загальну ефективність розробки що є також важливим для бізнесу.

1.4 Адміністрування систем на основі метрик

Адміністрування систем на основі метрик у розподілених системах – це критично важливий підхід до керування сучасною інфраструктурою, в якій окремі компоненти взаємодіють через мережу, функціонують незалежно, масштабуються горизонтально та можуть працювати у різних середовищах (наприклад, у хмарі, контейнерах або на фізичних серверах). У таких системах класичні методи адміністрування – на основі логів або ручної діагностики – стають неефективними, оскільки вони не дозволяють швидко виявляти проблеми або отримувати цілісну картину стану системи. Саме тому метрики стають основою для ефективного адміністрування.

Метрики – це кількісні показники, які відображають технічний або бізнесовий стан компонентів системи. У розподілених середовищах адміністратор працює не з окремими серверами чи процесами, а з великою кількістю взаємозалежних елементів, таких як мікросервіси, бази даних, брокери повідомлень, розподілювачі навантаження тощо. Метрики дозволяють отримувати дані про кожен із цих елементів у реальному часі: час відповіді, кількість запитів, кількість помилок, завантаження CPU, використання пам’яті, обсяг трафіку, статус контейнерів і багато іншого.

Однією з ключових особливостей адміністрування на основі метрик у розподілених системах є побудова централізованої системи моніторингу (рисунок 1.3), які збирають метрики з усіх вузлів, агрегують їх і дозволяють будувати динамічні дашборди. У поєднанні з системами алертингу, адміністратори можуть автоматично отримувати сповіщення у разі перевищення встановлених порогових значень.

Time	CPU Temp	System Load	Memory Usage %
2019-10-31T03:48:05+00:00	37	0.85	92
2019-10-31T03:48:10+00:00	42	0.87	90
2019-10-31T03:48:15+00:00	33	0.74	87
2019-10-31T03:48:20+00:00	34	0.72	77
2019-10-31T03:48:25+00:00	40	0.88	81
2019-10-31T03:48:30+00:00	42	0.89	82
2019-10-31T03:48:35+00:00	41	0.88	82

Рисунок 1.3 Приклад даних для прийняття рішення про роботу системи

У розподілених середовищах важливо не тільки мати доступ до метрик, а й контекстуалізувати їх. Наприклад, зниження швидкості відповіді одного мікросервісу може бути спричинене проблемами з мережею, високим навантаженням на інші мікросервіси або недостатнім масштабуванням у Kubernetes-кластері. Тому адміністратор має не просто бачити значення метрики, а розуміти їх взаємозв'язки. Для цього використовуються трасування запитів (tracing) та агрегація логів, що доповнюють метрики, але саме метрики дають основу для дій у реальному часі.

Крім технічних показників, адміністрування на основі метрик дозволяє інтегрувати бізнес-метрики: кількість замовлень, транзакцій, рівень конверсії тощо. Це відкриває можливості для спільної роботи DevOps і бізнес-команд, коли прийняття рішень ґрунтується на загальній картині – як технічній, так і продуктовій.

Ще однією важливою перевагою є можливість автоматизованого масштабування та самовідновлення. На основі метрик платформи оркестрації, такі як Kubernetes, можуть автоматично запускати додаткові екземпляри сервісів, коли навантаження зростає, або перезапускати контейнери, які "падають" або працюють з аномальними показниками.

У сфері безпеки також застосовуються метрики: наприклад, аномальне зростання запитів може свідчити про DDoS-атаку, зміна кількості невдалих спроб аутентифікації – про спробу злому. Метрики дозволяють не тільки виявити такі події, а й вжити автоматизованих заходів захисту.

На завершення, адміністрування систем на основі метрик у розподілених середовищах дає змогу перейти від реактивного до

превентивного керування: не чекати, поки система вийде з ладу, а прогнозувати потенційні проблеми й вирішувати їх ще до того, як вони вплинуть на користувачів. Це забезпечує високу доступність, надійність, масштабованість та контрольованість сучасних веб-систем. У світі DevOps і Site Reliability Engineering (SRE) це – обов'язковий стандарт, без якого якісна експлуатація складних систем неможлива.

1.5 Бази даних часових рядів: загальна характеристика

Бази даних часових рядів – це спеціалізовані системи зберігання даних, призначені для ефективного опрацювання, зберігання й аналізу великих обсягів інформації, яка має часову прив'язку (рисунок 1.4). Їх ключова особливість полягає у тому, що кожне збережене значення в таких базах обов'язково супроводжується часовою міткою, яка виступає первинним ключем у контексті подальших операцій фільтрації, агрегації чи порівняння. Цей тип баз даних зосереджений не лише на точковому зберіганні значень, а й на оптимізації роботи з послідовностями даних, що змінюються з часом – так званими часовими рядами.

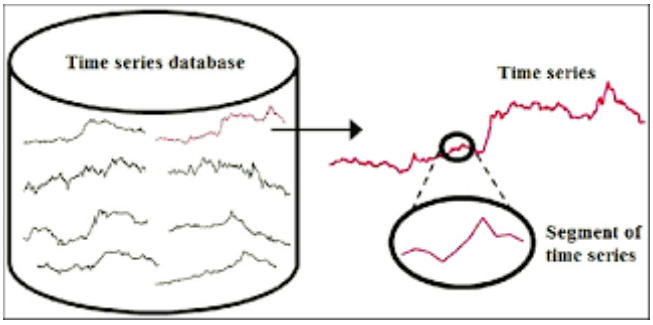


Рисунок 1.4 Узагальнений вигляд бази даних часових рядів

У порівнянні з традиційними реляційними або документоорієнтованими базами, системи для роботи з часовими рядами з самого початку розроблялась на специфічні запити: наприклад, отримання значень за останню годину, побудова ковзного середнього за певний період,

агрегація даних по днях чи хвилинах, або обробка великих масивів телеметрії, які надходять у реальному часі. Для цього такі бази оптимізовані як у структурі зберігання так і у механізмах індексації, стискання та кешування. Це дозволяє досягати високої швидкодії при виконанні аналітичних запитів навіть на масштабах мільйонів записів.

Ще однією важливою особливістю є підтримка стратегії зберігання з урахуванням обмеження часу життя даних. Бази даних часових рядів часто застосовуються у системах моніторингу, де нові значення мають пріоритет, а старі поступово видаляються або агрегуються до меншої деталізації. Таким чином, підтримується баланс між точністю, актуальністю та обсягом збереженої інформації.

Завдяки своїй спеціалізації ці бази активно застосовуються у сферах, де критичним є саме фактор часу: в індустріальних IoT-системах, фінансовому аналізі, телекомунікаціях, системах енергоменеджменту, мережевому моніторингу, наукових дослідженнях та аналітиці поведінки користувачів у веб-застосунках. Їхнє впровадження дозволяє зменшити навантаження на загальну інфраструктуру, знизити витрати на обчислення та забезпечити масштабування систем збору та обробки метрик у режимі реального часу.

1.6 Опис запропонованого рішення

З огляду на теоретичні аспекти, розглянуті в попередніх підрозділах, стає очевидним, що в умовах сучасних розподілених систем необхідною є гнучка, масштабована та надійна система моніторингу, яка дозволяє не лише збирати й зберігати часові ряди, а й візуалізувати їх, реагувати на аномалії за допомогою алертингу та інтегрувати з існуючими веб-застосунками. Така система повинна містити компоненти для збору метрик, API, яке генерує або отримує реальні або симульовані дані, базу даних часових рядів для зберігання цих даних, а також візуалізаційну платформу, що дає змогу аналізувати динаміку показників.

Майбутнє рішення – повноцінний веб-застосунок, який включатиме:

- модуль збору та обробки даних у форматі часових рядів;
- API для взаємодії з даними;
- зберігання інформації одночасно в класичній реляційній базі даних (SQL Server) та TSDB (InfluxDB) ;
- панель Grafana для візуалізації показників;
- інтеграцію з Prometheus для збору технічних метрик .NET API;
- систему алертингу для оперативного сповіщення про відхилення в роботі системи.

Перед реалізацією було проаналізовано низку наявних інструментів і платформ, таких як Kibana [13] (рисунок 1.5), Datadog [14] (рисунок 1.6), з метою оцінити їхню ефективність, гнучкість та доступність для інтеграції.

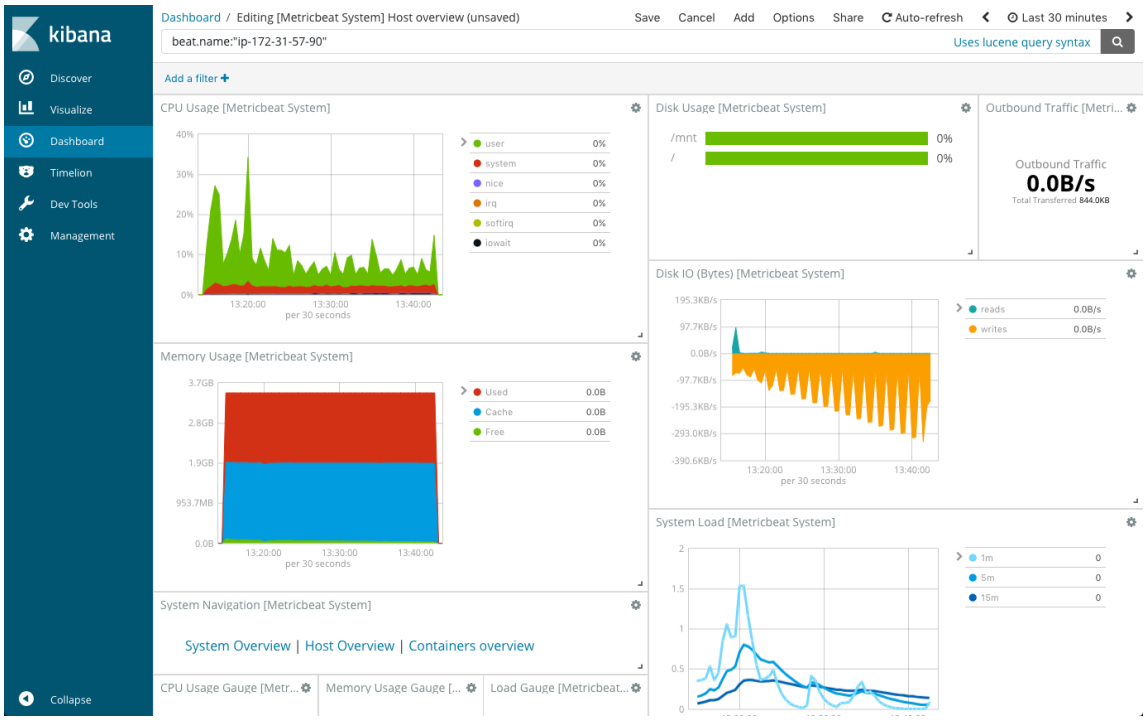


Рисунок 1.5 – Вигляд системи візуалізації та моніторингу Kibana

Порівняння цих рішень продемонструвало доцільність обрання Grafana як основного інструменту візуалізації завдяки її відкритості, підтримці великої кількості джерел даних, простому інтерфейсу та широким можливостям кастомізації панелей. Також у порівнянні з іншими системами

Grafana має вбудовану систему алертигу на відміну від аналогів, що спрощує роботу з алертами В якості системи збору метрик було обрано Prometheus через його нативну підтримку .NET-експортерів та зручність у конфігурації. Для зберігання даних використовується InfluxDB, яка оптимізована саме для зберігання часових рядів, а також SQL Server для порівняння з класичним типом СКБД.

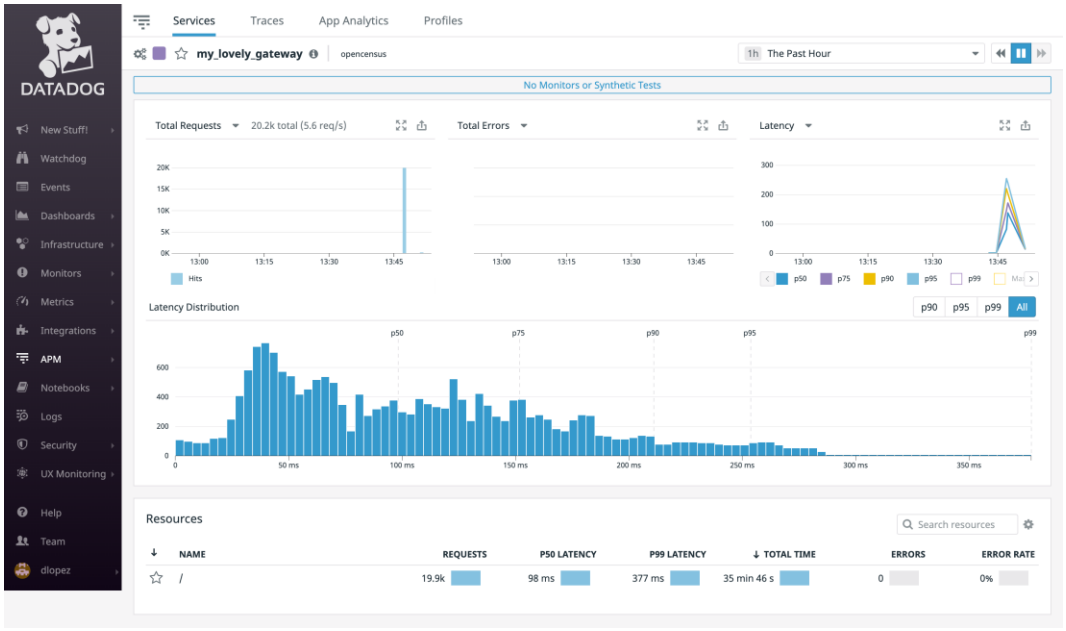


Рисунок 1.6 – Вигляд системи візуалізації та моніторингу Datadog

І хоча системи моніторингу мають дуже схожий інтерфейс, але з технічної точки зору мають багато нюансів реалізації, у наступному розділі буде детальніше проаналізовано вибрані технології, інструменти та бібліотеки, які ляжуть в основу майбутньої реалізації системи. Буде наведено технічні обґрунтування, що підтверджують доцільність їх застосування саме для цілі моніторингу роботи застосунку.

2 АНАЛІЗ ОБРАНИХ ТЕХНОЛОГІЙ

2.1 Дизайн API для роботи з даними в реальному часі

Дизайн API для роботи з даними в реальному часі є критично важливою складовою сучасних веб-застосунків, особливо коли йдеться про системи моніторингу, аналітики або відображення змінних метрик на графіках. Такий API має забезпечувати мінімальну затримку передачі даних, масштабованість, стабільність та зручність інтеграції з іншими сервісами або клієнтськими інтерфейсами.

Серед можливих підходів до реалізації API [8] найпоширенішими є REST, WebSocket, Server-Sent Events (SSE) і GraphQL з підписками. REST API залишається популярним через свою простоту та сумісність, однак він менш ефективний для безперервної передачі даних, оскільки потребує періодичних опитувань (polling), що може створювати зайве навантаження на сервер. У випадках, коли потрібне миттєве оновлення інформації, перевага надається WebSocket – двосторонньому з’єднанню між клієнтом і сервером, яке дозволяє передавати дані практично без затримок. Це особливо зручно для дашбордів, де інформація постійно змінюється, як в системах моніторингу чи біржових платформах.

Ще однією альтернативою є SSE – технологія, яка дозволяє серверу надсилати оновлення у реальному часі через HTTP, підтримуючи односпрямовану передачу. SSE простіші в реалізації, ніж WebSocket, і добре підходять для потокових оновлень у випадках, коли зворотний зв’язок від клієнта не є обов’язковим.

Ключовим критерієм при виборі архітектури API для реального часу є баланс між продуктивністю, простотою інтеграції та обсягом переданих даних. Важливо враховувати характер застосунку: чи потрібне постійне з’єднання, наскільки критична затримка, скільки клієнтів одночасно буде

підключено, і які саме дані потрібно передавати. API має бути також спроєктоване з урахуванням безпеки, керування навантаженням та масштабованості – адже у реальному часі навіть незначна втрата продуктивності або нестабільність з'єднання можуть вплинути на точність аналітики та користувацький досвід, у нашому випадку обрано REST API і частково реалізовано WebSocket.

2.2 Інструменти візуалізації Grafana

Grafana [7] – це зручний інструмент для перегляду та аналізу даних у вигляді графіків і таблиць. Його основна задача – допомогти людям бачити, як змінюються різні показники з часом. Наприклад, скільки користувачів відвідало сайт, скільки запитів на сервер було за годину, чи наскільки завантажений комп'ютер або сервер. Усе це можна побачити на одному екрані, який називається дешбордом.

У Grafana (рисунок 2.1), можна створювати власні дашборди додаючи до них різні елементи: графіки, діаграми, числа, таблиці. Кожен такий елемент можна налаштувати окремо. Це дозволяє легко зрозуміти, коли відбулися якісь зміни, помилки або перевищення норми.

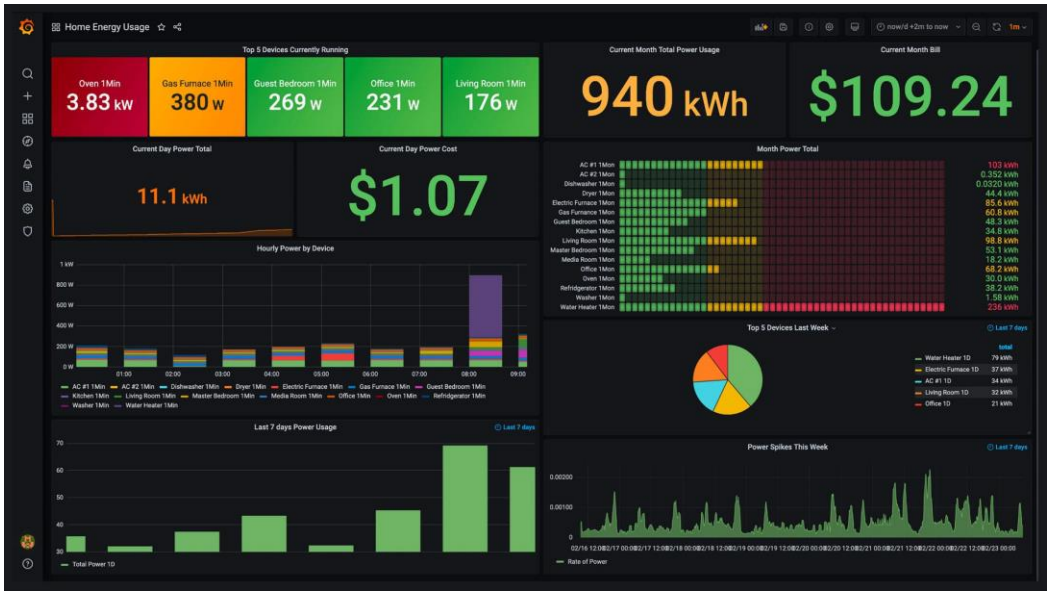


Рисунок 2.1 Можливий вигляд дешборду

Однією з переваг Grafana є можливість працювати з багатьма джерелами даних. Це можуть бути спеціальні бази, які зберігають дані по часу (наприклад, InfluxDB [5] чи Prometheus [4]), або навіть звичайні бази, де зберігаються числа з датами. Grafana [7] "підтягує" ці дані й одразу показує їх у зрозумілому вигляді.

Дуже зручно, що в можна змінювати час, за який показуються дані – наприклад, дивитися, що було за останні 5 хвилин, годину або цілий тиждень. А ще можна зробити так, щоб інформація оновлювалася автоматично кожні кілька секунд.

Коли стається щось важливе – наприклад, навантаження на сервер стає дуже високим – Grafana може надіслати повідомлення. Вона може надсилати сповіщення, на email або у чат команди, щоб усі швидко дізналися про проблему.

Grafana підходить як для розробників і адміністраторів, так і для звичайних користувачів, яким потрібно просто бачити інформацію зручно і зрозуміло. Вона дозволяє будувати власні панелі управління й легко слідкувати за роботою будь-якої системи – сайту, додатка чи навіть домашньої мережі так як вона з відкритим кодом і безкоштовна для власного користування

2.3 Централізована система логування запитів

Даний підхід є важливим у сучасній експлуатації різноманітних систем, основна ідея полягає в тому, щоб усі сервіси, які генерують логи, передавали їх до централізованого сховища або платформи логування через стандартизовані канали. Це може бути реалізовано за допомогою агентів, які встановлені на кожному сервері або контейнері, або через бібліотеки, інтегровані безпосередньо в код сервісів, що відправляють логи у централізовану систему. Важливо, щоб логи мали уніфікований формат, який містить ключові поля: часову мітку, ідентифікатор запиту, тип події, рівень

важливості, текст повідомлення та додаткові метадані (наприклад, ім’я сервісу, користувача, IP-адресу).

Переваги такого підходу включають можливість швидкого виявлення помилок і проблем, аналіз продуктивності окремих сервісів, кореляцію подій у масштабних системах, а також зручний аудит та ведення історії. Водночас необхідно враховувати виклики, пов’язані з обробкою великих обсягів даних, забезпеченням безпеки доступу до логів та підтримкою високої доступності централізованої системи. дизайн централізованої системи логування у розподілених системах – це комплексний процес, що вимагає правильного вибору інструментів, стандартизації форматів даних, ефективної маршрутизації логів та забезпечення масштабованості і надійності для підтримки стабільної роботи всієї інфраструктури.

2.4 Засоби моніторингу Prometheus

Prometheus [4] – це одна з найпопулярніших систем моніторингу та збору метрик, яка активно використовується у сучасних веб-застосунках, хмарних інфраструктурах та розподілених системах. Його основне завдання – автоматично збирати дані з різних джерел, зберігати їх у вигляді часових рядів та надавати можливість виконувати запити до цих даних для аналізу та візуалізації.

Однією з головних особливостей Prometheus [4] є модель “pull”, тобто система сама періодично звертається до заданих цілей (наприклад, серверів, додатків або контейнерів) і запитує у них метрики. Дані зазвичай надсилаються у спеціальному форматі через HTTP

Prometheus [4] підтримує інструментальні бібліотеки (client libraries), які вбудовуються безпосередньо у код додатку. Наприклад, у веб-застосунок можна підключити відповідну бібліотеку і самостійно визначити, які метрики збирати – наприклад, час обробки запиту, кількість помилок або кількість запитів на кожну сторінку. Це дає максимальну гнучкість у моніторингу

власних сервісів та оптимізує ресурси самостійно обираючи які метрики потрібно збирати

2.5 Огляд баз даних часових рядів

Бази даних часових рядів (Time Series Databases, TSDB) – це спеціалізовані системи зберігання (рисунок 2.2), які оптимізовані для роботи з послідовними даними, прив'язаними до часу у сучасних системах

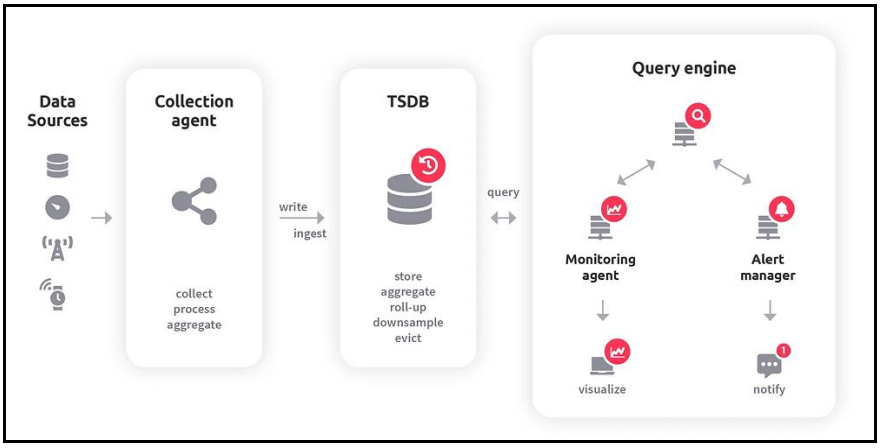


Рисунок 2.2 Структура та місце використання баз даних часових рядів

InfluxDB [5] є спеціалізованою базою даних часових рядів із власною мовою запитів Flux або спрощеною InfluxQL. Вона вирізняється високою продуктивністю, гнучкістю у формулюванні запитів, підтримкою візуалізації та легко інтегрується з Grafana [7]. Водночас при розгортанні у великих кластерах потребує значних ресурсів, а мова Flux має доволі складний синтаксис. Найчастіше InfluxDB [5] використовують у моніторингу інфраструктури, проєктах IoT та DevOps, при цьому є як безкоштовна так і платна версія, що має більші можливості для великих проєктів, а безкоштовна версія призначена для некомерційних додатків

Prometheus має власну модель зберігання даних та використовує pull-модель збору метрик. Основною мовою запитів є PromQL. Його перевагою є просте розгортання, хороша інтеграція з Kubernetes та наявність вбудованого

Alertmanager для оповіщень. Проте Prometheus [4] менш ефективний при довготривалому зберіганні даних і має обмеження щодо масштабування. Його часто застосовують для моніторингу веб-застосунків, серверів і мікросервісної інфраструктури.

TimescaleDB [6] створений як розширення для PostgreSQL і використовує стандартну мову SQL. Це забезпечує легку інтеграцію зі звичайними інструментами, знайомий синтаксис для запитів і підтримку горизонтального масштабування. Проте складність налаштування і проблеми з ефективністю при високому навантаженні можуть створювати труднощі. Найчастіше використовується для аналітики, обробки великих масивів даних і фінансових часових рядів.

OpenTSDB [9], побудований на HBase, працює через HTTP API або CLI. Він добре масштабується і підходить для роботи з великими обсягами історичних даних. Його головні недоліки – залежність від інших систем і складність у розгортанні та обслуговуванні. Цей інструмент переважно використовують для зберігання телеметрії та історичних метрик.

Graphite [10] базується на файловій структурі (whisper) і має просту власну мову запитів. Його легко налаштувати, він швидко запускається, проте має застарілу архітектуру, погано масштабується і не підтримує складні запити. Graphite застосовується в системах моніторингу для візуалізації метрик.

2.6 Проблеми та виклики при роботі з часовими рядами

Робота з часовими рядами має багато переваг [2], особливо коли йдеться про моніторинг, аналіз змін у часі чи візуалізацію активності. Але водночас цей тип даних приносить і чимало складностей, які потрібно враховувати під час проєктування систем, зберігання даних та побудови API. Одна з головних проблем – це велика кількість даних, що постійно накопичуються. Якщо система фіксує значення щосекунди або навіть

частіше, обсяг інформації швидко зростає до гігабайтів чи терабайтів. Зберігати такі дані довго і при цьому забезпечити швидкий доступ до них – важке завдання, яке потребує спеціалізованих баз даних та механізмів оптимізації.

Ще однією складністю є підтримка точності та регулярності. Якщо дані приходять із затримкою або з пропущеними значеннями, аналіз стає менш надійним. Такі "дірки" в часових рядах потрібно або заповнювати штучно (інтерполяцією), або враховувати під час обчислень. У нерегулярних часових рядах (наприклад, події, що відбуваються випадково) важко будувати порівняльну статистику або агрегати без попередньої обробки.

Важливу роль відіграє і правильне обчислення агрегованих значень: середніх, мінімальних, пікових. Якщо зробити це неправильно, графіки можуть показувати спотворену картину. Особливо це важливо при зменшенні масштабу часу (наприклад, показ значень за тиждень замість хвилин).

У розподілених системах ще складніше – часові мітки можуть приходити з різних серверів, і потрібно синхронізувати час, щоб уникнути плутанини. Навіть незначні розбіжності в годинах або часових зонах можуть призвести до неправильних висновків у звітах і візуалізаціях.

Крім цього, викликом є побудова ефективних запитів до баз даних. Якщо користувач хоче переглянути дані за довгий період, наприклад за рік, система має швидко обробити великий обсяг інформації. Потрібні індексація, збереження попередньо обчислених агрегатів, а також кешування, щоб уникнути перевантаження серверів.

Проблема зберігання великих обсягів даних у часових рядах є однією з ключових і найскладніших. Через постійний потік нових значень, які надходять щосекунди або навіть частіше, обсяг зібраної інформації дуже швидко зростає. Це створює серйозне навантаження на системи зберігання: як на жорсткі диски, так і на оперативну пам’ять, якщо йдеться про обробку в реальному часі.

Навіть прості метрики – наприклад, температура в серверній або кількість запитів на сайт – можуть у масштабах тижня чи місяця перетворитися на мільйони записів. У складніших випадках, коли фіксується багато параметрів з різних пристроїв або компонентів системи, обсяг даних може сягати десятків чи сотень терабайтів.

Це ставить виклик не лише щодо фізичного зберігання, а й щодо ефективного доступу до цих даних. Звичайні підходи до збереження, як у реляційних базах, часто не підходять, оскільки не можуть забезпечити необхідну швидкість читання та запису. Тому доводиться використовувати стиснення, видалення старих даних, зберігання лише агрегатів (наприклад, середніх значень за годину) або спеціальні механізми ротації – автоматичне очищення найстаріших записів.

Також варто враховувати витрати: зберігання великих обсягів даних у хмарних системах або на локальних серверах коштує чимало, особливо якщо потрібно забезпечити швидкий доступ і надійність. Пошук компромісу між повнотою історії, точністю та розміром – це ще одне завдання, яке має вирішувати розробник або адміністратор.

Також слід враховувати безпеку й конфіденційність. У багатьох випадках часові ряди містять чутливі дані – про активність користувачів, бізнес-показники чи внутрішні процеси. Забезпечити контроль доступу до окремих серій, ролей користувачів та історії змін – ще один важливий аспект роботи з такими даними. Усе це показує, що робота з часовими рядами потребує не лише спеціалізованих технічних рішень, а й глибокого розуміння контексту, у якому використовуються дані. Від вибору бази та структури зберігання до підходів до обробки – кожен етап має свої виклики, і від якості їх вирішення залежить точність аналізу, швидкість реагування та надійність усього застосунку.

3 АРХІТЕКТУРНЕ РІШЕННЯ СИСТЕМИ

3.1 Архітектура розробленого рішення

У розробленій архітектурі (рисунок 3.1) кожен елемент виконує важливу роль у забезпеченні повного циклу обробки та візуалізації даних, і разом вони формують єдину цілісну систему. Центральною ланкою є .NET Web API, який відповідає за отримання, обробку та передачу інформації між усіма іншими компонентами.

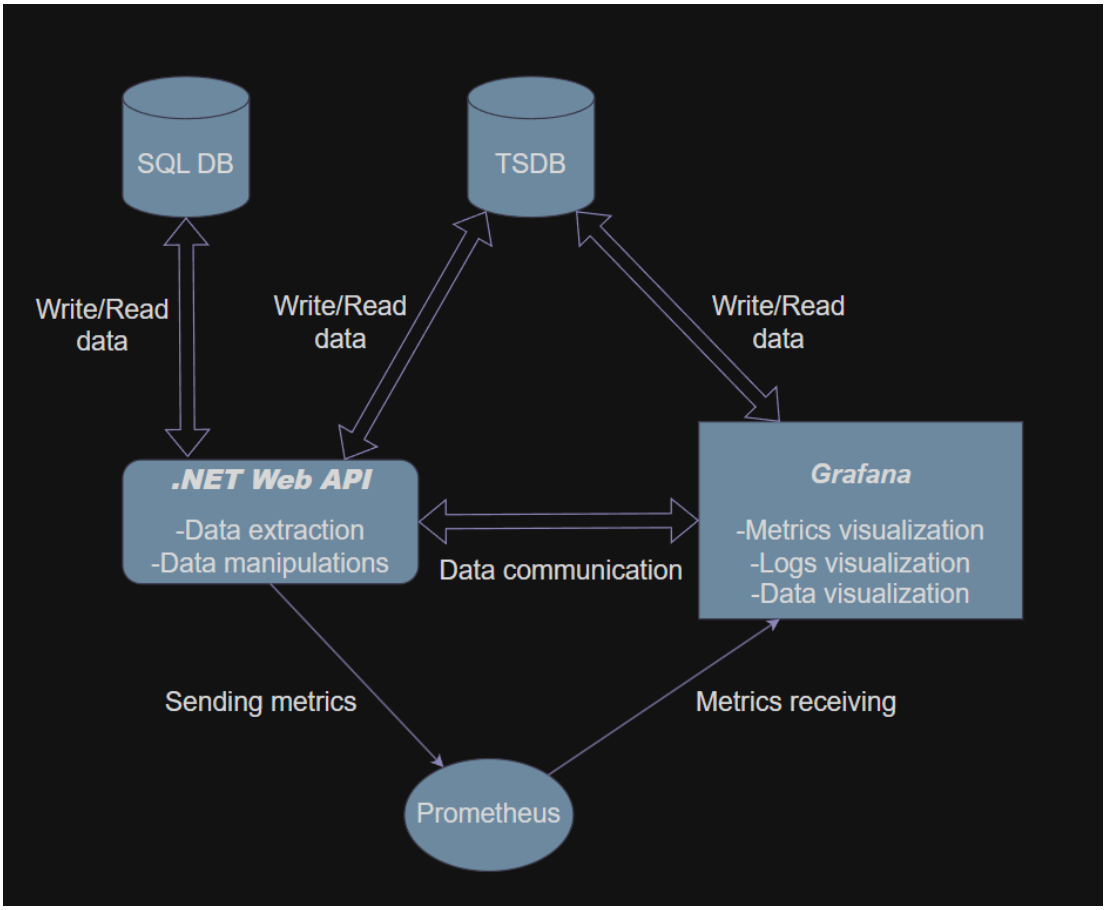


Рисунок 3.1 Зображення архітектури побудованої системи

Саме через API відбувається зчитування даних із джерел, їх попередня фільтрація та трансформація у формат, зручний для зберігання або аналізу. API також реалізує логіку роботи з базами даних, забезпечує їхню

ініціалізацію, виконання запитів на запис або вибірку та передачу результатів у систему візуалізації або через базу даних.

Винесення коду на інтерфейси та сервіси (рисунок 3.2) є важливою практикою у сучасній розробці програмного забезпечення, оскільки забезпечує чисту архітектуру, підвищує масштабованість і спрощує супровід коду. Інтерфейси дозволяють відокремити визначення поведінки (контракту) від конкретної реалізації, що дає можливість легко змінювати реалізацію сервісу без необхідності змінювати інші частини системи.

Сервіси ж відділяють бізнес-логіку, відділяючи її від контролерів або компонентів інтерфейсу. Завдяки цьому контролери зосереджуються лише на обробці HTTP-запитів, делегуючи всю обробку даних відповідним сервісам. Такий підхід полегшує підтримку. Крім того, структура проєкту стає більш читабельною, логічно впорядкованою та гнучкою до змін у майбутньому.

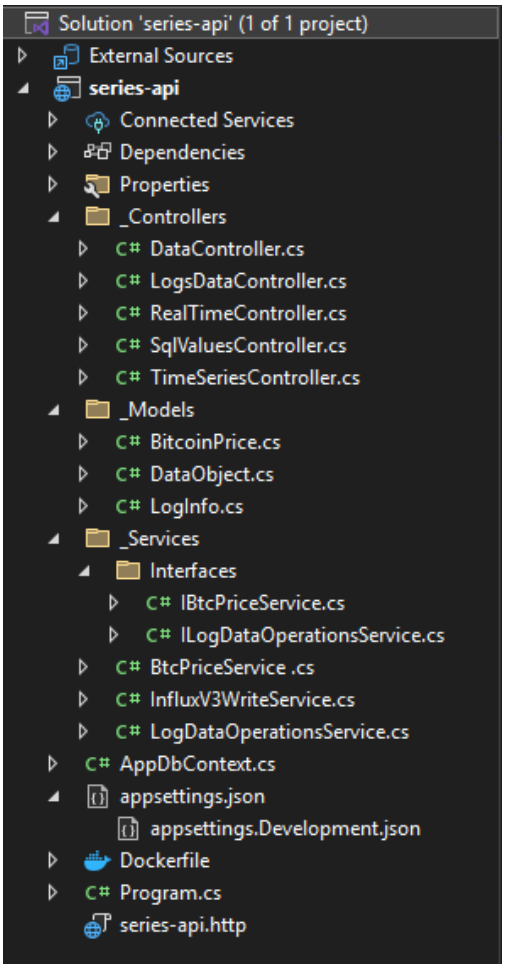


Рисунок 3.2 Структура розробленого WEB API роботи з данимим

SQL база даних використовується як сховище структурованих даних (рисунок 3.3), де зберігаються результати запитів, історичні дані, а також зібрані лог-записи. Вона зручна для складного аналітичного запиту, реляційних залежностей між сутностями та роботи з великими масивами табличної інформації. Запис і зчитування відбуваються через API, який виконує базові-операції та обробку запитів.

TSDB (Time Series Database) використовується для зберігання даних, що мають часову прив'язку, зокрема – метрик або даних із сенсорів. Завдяки оптимізації під часові ряди, вона дозволяє ефективно працювати з великими обсягами даних у реальному часі. TSDB інтегрована з API, через який вона отримує дані, а також напряму з Grafana, яка може зчитувати їх для візуалізації. Саме TSDB є ключовим джерелом метрик, що оновлюються постійно, і підходить для агрегацій, фільтрацій та аналізу трендів.

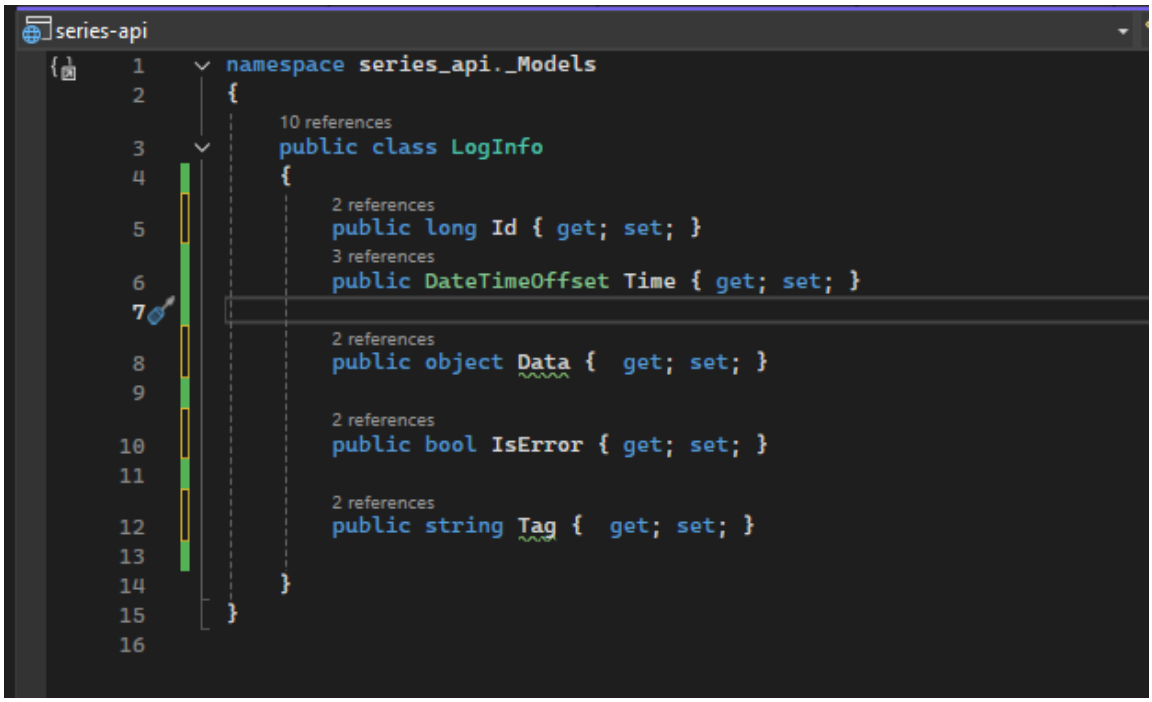


Рисунок 3.3 Приклад структури для представлення даних у застосунку

Grafana виконує роль основного інтерфейсу (рисунок 3.4) для користувача – в ній відображаються всі ключові дані: метрики продуктивності, логи, інформаційні панелі. Вона працює як із TSDB, так і з

SQL базою, витягуючи інформацію через запити та формуючи зрозумілі графіки, діаграми або таблиці. Важливо, що Grafana також підтримує візуалізацію логів, отриманих із API чи Prometheus, і дозволяє налаштовувати сповіщення (alerting) на основі заданих умов. Також має систему адміністрування користувачами і має вбудовану вікі про те як працюють і правильно налаштовувати всі елементи Grafana що полегшує розробку.

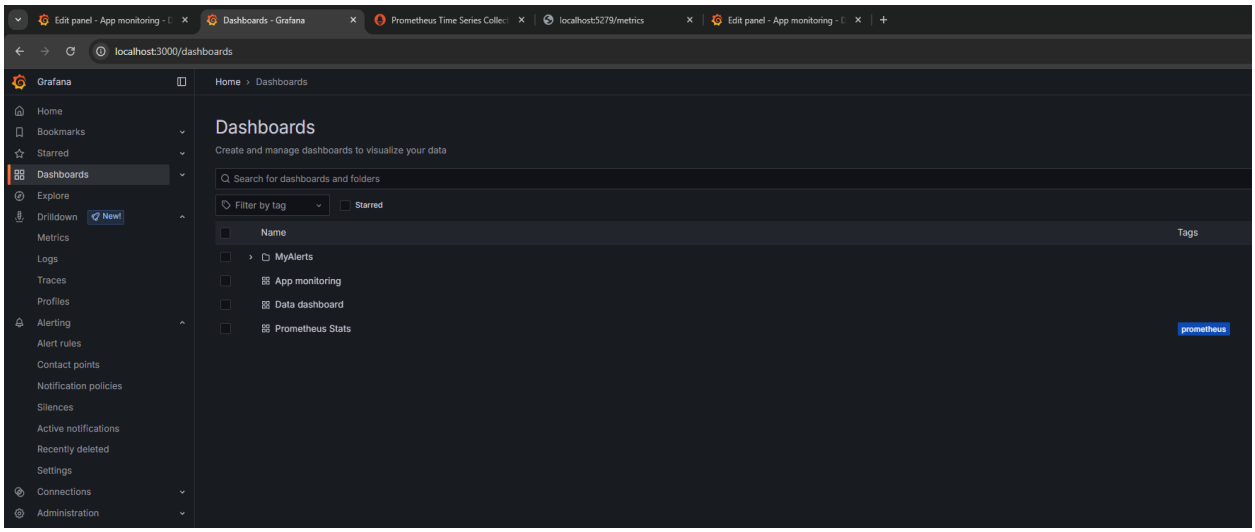


Рисунок 3.4 Успішно інтегрована Grafana

Prometheus реалізує функцію збору метрик. Його завдання –регулярно опитувати API або інші джерела, збирати значення показників (наприклад, продуктивності, завантаження, кількості запитів), зберігати їх у власній базі та передавати до Grafana для побудови графіків. API, своєю чергою, готує ці метрики у форматі, зручному для читання. Prometheus виступає також буфером – він дозволяє масштабувати збір даних та робити його незалежним від навантаження на основні системи.

3.2 Особливості розробленої панелі

Розроблена панель (рисунок 3.5) в Grafana є інструментом для оперативного моніторингу стану системи та її ключових показників. Однією з головних метрик, представлених на панелі, є Memory utilization – вона

демонструє, скільки оперативної пам’яті наразі використовується від загального доступного обсягу. Завдяки цьому можна оцінити, наскільки ефективно працює система, чи не виникає перевантаження або витік пам’яті. Показник реалізовано у вигляді що надає в реальному часі інформацію про обсяг задіяної пам’яті у гігабайтах, із кольоровими індикаторами для виявлення перевищення встановлених порогів., що дозволяє візуально оцінити ситуацію без потреби аналізу числових даних.

Панелі використовуються для візуалізації та моніторингу [11] різноманітних даних у зручному та інтуїтивно зрозумілому вигляді. Вони дозволяють оперативно відстежувати стан систем, аналізувати показники продуктивності, виявляти аномалії та приймати обґрунтовані рішення на основі візуалізованої інформації. Панелі допомагають консолідувати великі обсяги даних у компактні графіки, діаграми чи інші форми відображення, що значно полегшує роботу адміністраторів, аналітиків і розробників. Завдяки їм можна швидко реагувати на проблеми, покращувати ефективність систем і планувати подальший розвиток на основі реальних даних.

Іншою важливою метрикою є ЗЗЧІ, яка відображає кількість запитів, що надходили до застосунку протягом фіксованих інтервалів часу Це дає змогу побачити динаміку звернень до API.

Рисунок 3.5 Вигляд деяких розроблених елементів панелі

Загалом, панель не лише допомагає контролювати роботу системи, а й слугує зручним інструментом для виявлення та реагування на проблеми ще до того, як вони стають критичними. Її використання значно підвищує прозорість процесів у веб-застосунку та полегшує адміністрування.

3.3 Алертинг на основі метрик

Алертинг на основі метрик – це механізм автоматичного сповіщення про зміни або відхилення в роботі системи, який базується на аналізі показників, зібраних у реальному часі. Його головною метою є виявлення проблем ще до того, як вони вплинуть на кінцевого користувача або призведуть до простою системи. Система моніторингу постійно збирає дані, наприклад, про навантаження на процесор, обсяг вільної пам’яті, кількість запитів чи помилок, і порівнює їх із заздалегідь визначеними пороговими значеннями.

Панелями іноді складно користуватися через велику кількість налаштувань і складність інтеграції різних джерел даних, що може вимагати технічних знань. Користувачам, особливо без досвіду, буває важко розібратися у великій кількості типів візуалізацій, метрик і фільтрів, а також у правильному налаштуванні алертів і звітів. Інтерфейс панелей може бути перевантаженим або неінтуїтивним, що ускладнює швидкий доступ до потрібної інформації. Крім того, для точного аналізу даних часто потрібне розуміння специфіки самої системи моніторингу, що підвищує поріг входу для новачків і робить використання панелей менш зручним для широкого кола користувачів.

Важливо сповіщувати адміністраторів систем, оскільки своєчасне отримання інформації(рисунок 3.6) про критичні події, помилки чи аномалії дозволяє оперативно реагувати на проблеми, запобігати збоям і мінімізувати час простою системи. Це допомагає підтримувати стабільність та надійність роботи сервісів, знижує ризик втрати даних або порушення безпеки, а також

покращує загальний рівень обслуговування користувачів. Крім того, автоматизовані сповіщення сприяють більш ефективному управлінню ресурсами і дозволяють адміністраторам вирішувати проблеми

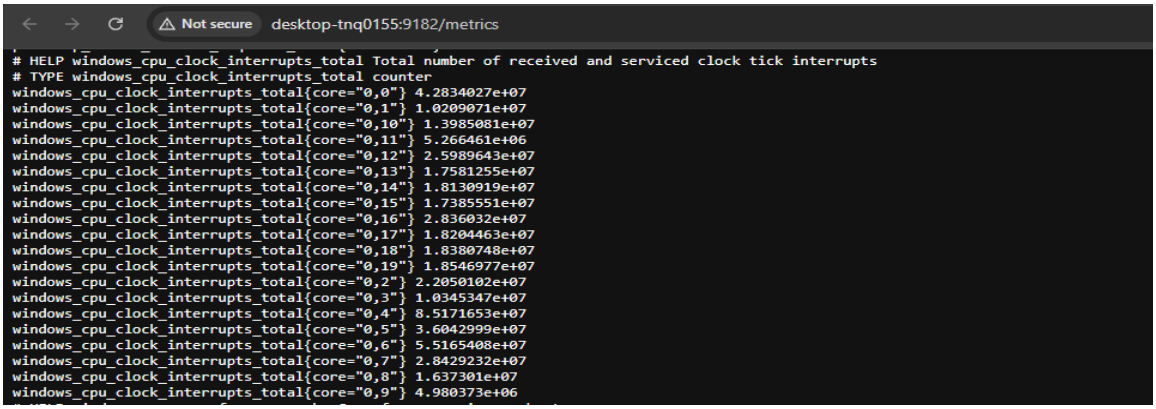


Рисунок 3.6 Метрики з системи

Цей підхід дозволяє скоротити час реагування на збої та полегшує роботу адміністраторів, які можуть не відстежувати всі показники вручну. Алертинг може бути простим – наприклад, коли кількість запитів досягає екстремальних значень (рисунок 3.7), або складним, із врахуванням кількох умов чи затримки спрацьовування, щоб уникнути хибних спрацювань.

Алертинг є критично важливим елементом сучасної інфраструктури спостереження особливо у розподілених системах, де контроль за всіма сервісами вручну є неможливим. Використовуються з системами моніторингу [12], такими як Prometheus,

Рисунок 3.7 Вигляд демонстраційного розробленого алерту

Коли умови алерту будуть виконані (рисунок 3.8) ми побачимо зміну статусу та спробу повідомити користувача за допомогою пошти, найважливішим для системи є критичний алерт [15]. Він сигналізує про серйозні збої або відмови у роботі, що можуть призвести до повного припинення роботи сервісу або суттєвого погіршення його функціональності.

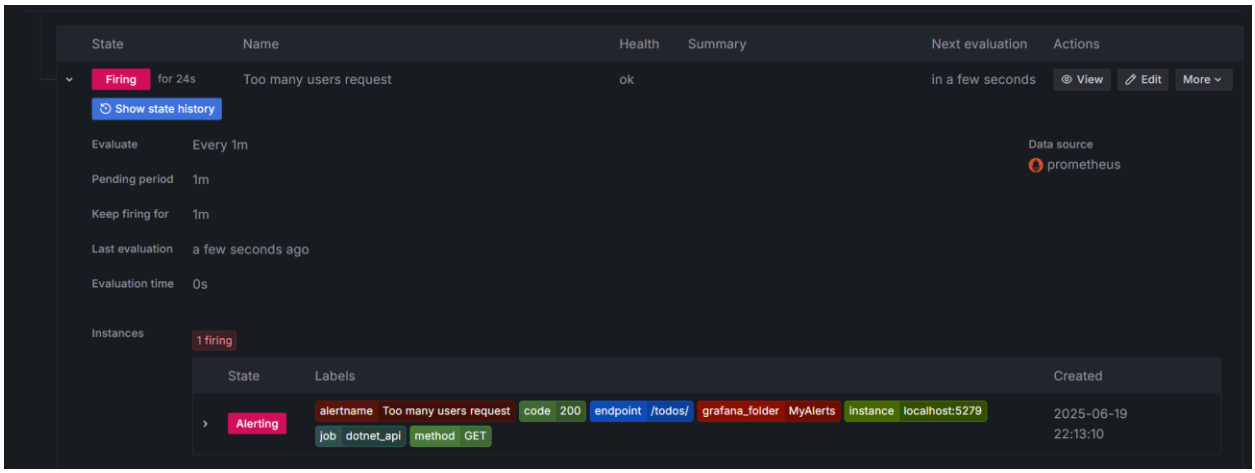


Рисунок 3.8 Вигляд алерту при досягненні заданої умови

4 ОПИС ВІЗУАЛІЗОВАНИХ ДАНИХ

4.1 Історичні дані ціни

Історичні дані ціни є надзвичайно важливими, оскільки вони дозволяють аналізувати минулі тенденції та поведінку ринку, що дає змогу приймати обґрунтовані рішення щодо інвестицій, прогнозування і управління ризиками. Завдяки накопиченню великого обсягу таких даних можна виявляти закономірності, сезонні коливання та аномалії, що сприяє більш точному моделюванню майбутніх змін цін. Історичні ціни також використовуються для тестування торгових стратегій, оптимізації алгоритмів та покращення систем автоматичних торгів. Вони є базою для побудови технічного аналізу та розробки складних фінансових моделей, що підвищує ефективність і надійність управління активами. Таким чином, наявність і якість історичних даних є ключовим фактором успіху у фінансовій аналітиці та прийнятті стратегічних рішень.

Реалізовано запис до бази даних часових рядів(Лістинг 4.1) та візуалізацію ціни віртуального активу (Лістинг 4.2).

Лістинг 4.1 Робота з клієнтом бази даних часових рядів

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;
using InfluxDB3.Client;
using InfluxDB3.Client.Write;
using series_api._Models;

public class InfluxV3WriteService
{
public async Task WritePricesAsync(List<BitcoinPrice> prices)
{
var host = "http://localhost:8181";
var token =
"apiv3_4yGsk6X18Gr7Ne3c_OhUgNwkYXVtbvoKbcKqWyKUCUb6sQUfbtxXeHqNR
dNXb3wybd328hfXmyFZp8VDj3asOQ";
```

```
var db = "test";

using var client = new InfluxDBClient(host, token, db);

var points = prices.Select(p =>
PointData
.Measurement("bitcoin_price")
.SetTag("symbol", "BTC")
.SetField("value", (double)p.Value)
.SetTimestamp(((DateTimeOffset)p.Time).ToUnixTimeSeconds(),
WritePrecision.S)
).ToList();

await client.WritePointsAsync(points, database: db);
}
}
```

Лістинг 4.2 Приклад розробленої панелі для візуалізації

```
{
  "id": 1,
  "type": "timeseries",
  "title": "BTC Price",
  "gridPos": {
    "x": 0,
    "y": 0,
    "h": 18,
    "w": 24
  },
  "fieldConfig": {
    "defaults": {
      "custom": {
        "drawStyle": "line",
        "lineInterpolation": "linear",
        "barAlignment": 0,
        "barWidthFactor": 0.6,
        "lineWidth": 3,
        "fillOpacity": 13,
        "gradientMode": "none",
        "spanNulls": false,
        "insertNulls": false,
        "showPoints": "auto",
        "pointSize": 1,
        "stacking": {
          "mode": "none",
          "group": "A"
        },
        "axisPlacement": "auto",
        "axisLabel": "",
        "axisColorMode": "text",
        "axisBorderShow": false,
        "scaleDistribution": {
          "type": "linear"
```

```
        },
        "axisCenteredZero": false,
        "hideFrom": {
          "tooltip": false,
          "viz": false,
          "legend": false
        },
  "datasource": {
    "type": "influxdb",
    "uid": "eepg4vtr9uscgc"
  },
  "options": {
    "tooltip": {
      "mode": "single",
      "sort": "none",
      "hideZeros": false
    },
    "legend": {
      "showLegend": false,
      "displayMode": "list",
      "placement": "bottom",
      "calcs": []
    }
  }
}
```

Панель(рисунок 4.1) з даними має такий вигляд у системі візуалізації

Рисунок 4.1 Вигляд візуалізації даних з наведеного коду

4.3 Можливості інших типів візуалізації

У Grafana доступний широкий вибір типів візуалізацій [17], які дають змогу максимально наочно подати дані, залежно від їх природи та цільового призначення. Один з найпоширеніших варіантів – це лінійний графік (time

series chart), який дозволяє відслідковувати зміну метрик у часі. Така візуалізація ідеально підходить для моніторингу ресурсів, наприклад, навантаження на процесор або обсяг використаної оперативної пам’яті. Завдяки можливості масштабування за часовими інтервалами користувач може як швидко побачити загальну тенденцію, так і заглибитись у конкретні моменти часу.

Іншим корисним типом є gauge(рисунок 4.2) (шкала), яка показує поточне значення метрики у вигляді стрілки або заповненої шкали. Цей тип ідеально підходить для контролю рівня завантаження, температури, тиску або будь-якого іншого показника, що має граничні допустимі значення. Наприклад, можна налаштувати шкалу для відображення використаної пам’яті, де зелена зона позначатиме норму, а червона – критичне перевищення.

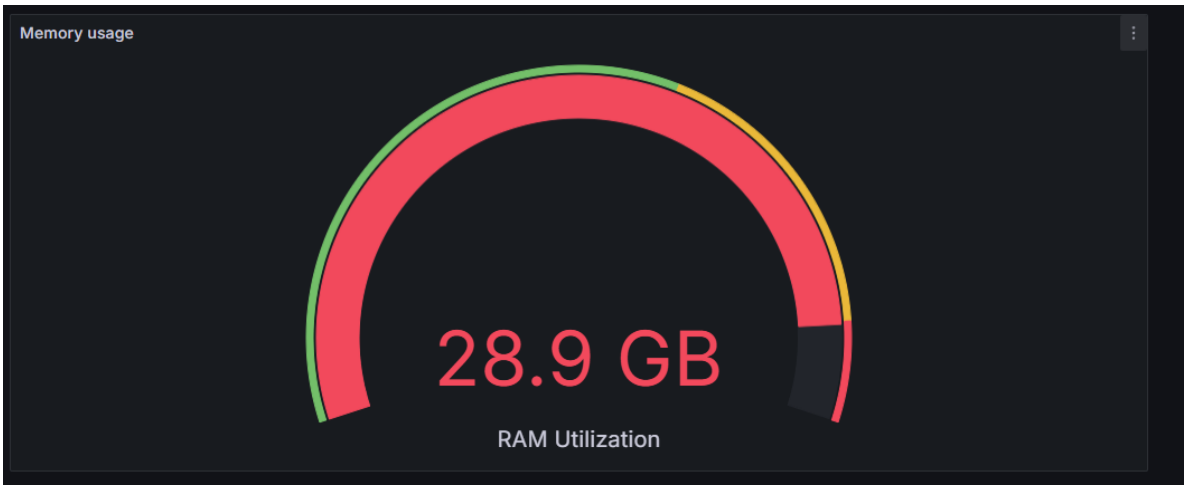


Рисунок 4.2 вигляд типу візуалізації шкали

Bar chart (стовпчикові діаграми) дозволяють порівнювати кілька значень між собою, наприклад кількість запитів до різних сервісів або обсяг трафіку по серверах. Вони зручні тоді, коли потрібно швидко оцінити відносні показники. Також для таких цілей часто використовують pie chart (секторну діаграму), яка дає змогу візуально уявити частку кожного елемента в загальній структурі.

Table (таблиці) у Grafana корисні для представлення точних значень, списків або логів. Вони зручно поєднуються з фільтрами та форматуванням, що робить їх ідеальними для виведення списків алертів, станів систем або переліку подій з часовими мітками.

Pie chart (секторна діаграма) є одним із класичних типів візуалізації даних, (рисунок 4.3) який широко застосовується для представлення відносної частки елементів у межах цілого. У контексті платформи Grafana, цей тип візуалізації дозволяє легко і наочно відображати пропорції категорій, значень або подій на основі зібраних метрик.

Особливістю pie chart у Grafana є те, що вона працює на основі агрегації даних, які мають категоріальні характеристики або мітки (labels), що дозволяє розбити сукупні значення на сегменти. Кожен сегмент кола відображає частку, яку займає певна категорія від загального обсягу. Наприклад, це можуть бути HTTP-коди відповідей (200, 404, 500), різні типи подій у системі моніторингу, або використання ресурсів за окремими службами чи вузлами.

У Grafana pie chart налаштовується через вибір джерела даних (наприклад, Prometheus), запит метрики, і вказівку групування значень за мітками. Наприклад, можна створити графік, який показує розподіл використання CPU між різними процесами або сервісами, де кожен сектор буде відповідати окремому job.

Користувач також може налаштувати вигляд діаграми: відображення абсолютних значень, відсотків або обох варіантів одночасно, змінити кольори сегментів, визначити пороги, які впливають на колірну палітру, та застосувати легенду для зручності інтерпретації. Також підтримуються анімації оновлення, що робить зміни більш помітними при моніторингу в реальному часі.

Однак варто враховувати, що pie chart менш ефективна при великій кількості категорій або незначних відмінностях між значеннями – у таких випадках рекомендується використовувати bar chart або table. Тим не менш, у

випадках, коли потрібно швидко оцінити загальну картину або домінування однієї категорії над іншими, секторна діаграма залишається потужним і інтуїтивно зрозумілим інструментом.

У систем моніторингу, таких як Grafana, pie chart часто використовується для створення дашбордів, які надають миттєве уявлення про стан системи [16] – наприклад, частка здорових інстансів сервісів, розподіл пам’яті між процесами, або розподіл запитів по HTTP-кодах. Саме тому вона є важливим елементом візуального аналізу метрик у реальному часі.

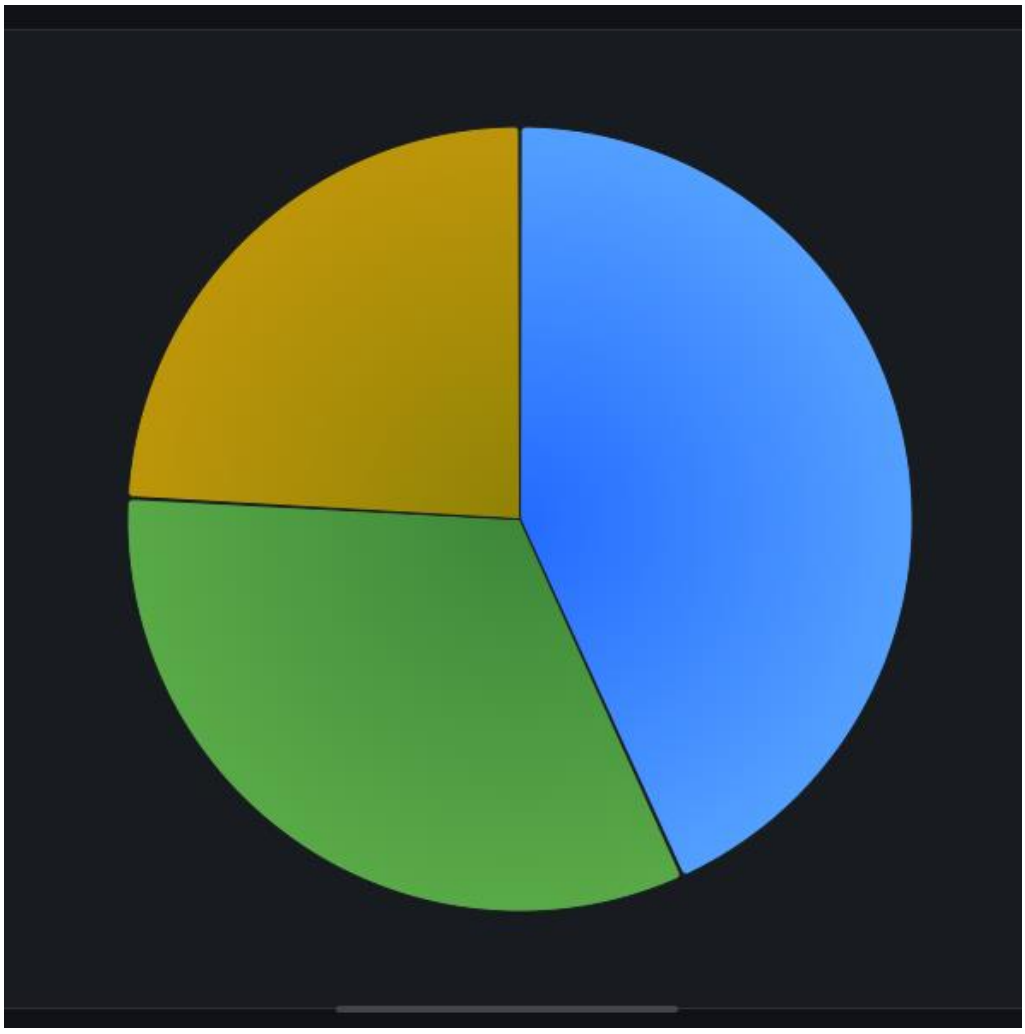


Рисунок 4.3 Вигляд візуалізації кругової діаграми у Grafana

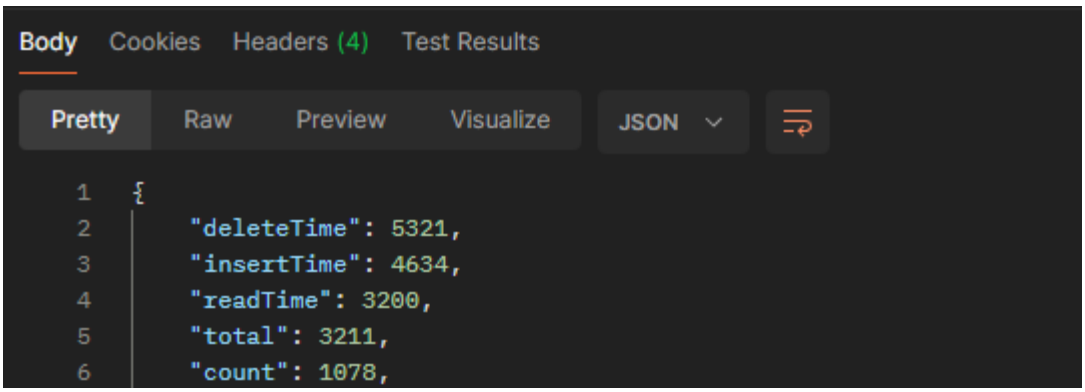
Різні типи візуалізацій даних виникали як відповідь на зростаючі обсяги інформації та потребу в зручному й ефективному способі її аналізу.

Спочатку візуалізація даних зводилася до простих таблиць та текстових звітів, однак зі збільшенням складності систем, кількості параметрів і швидкості зміни даних, виникла необхідність у більш наочних і динамічних інструментах. Так, з розвитком інформатики, обчислювальної техніки та моніторингових систем, почали формуватися візуальні засоби подання даних. Подальший розвиток систем візуалізації був зумовлений не лише обсягом даних, а й потребою у зручному виявленні аномалій, порівнянні історичних значень, аналізі трендів.

4.3 Зчитування та замір продуктивності різних типів БД

Зчитування та замір продуктивності різних типів баз даних є важливим етапом у дослідженні ефективності обробки та зберігання великих обсягів даних, особливо коли йдеться про роботу з часовими рядами. У процесі реалізації було реалізовано механізм, який дає змогу в автоматичному режимі виміряти продуктивність запису, читання та попереднього очищення бази даних, як для класичної реляційної системи управління базами даних (наприклад, MSSQL), так і для спеціалізованої системи зберігання часового ряду – InfluxDB.

У рамках дослідження було реалізовано механізм зчитування та заміру продуктивності (рисунок 4.4) для двох типів баз даних: класичної реляційної БД Microsoft SQL Server та спеціалізованої бази даних часових рядів InfluxDB v3. Для обох систем були реалізовані окремі контролери, кожен з яких забезпечував процес видалення старих даних, завантаження нових значень цін Bitcoin із зовнішнього API з інтервалом у 4 години, збереження їх у відповідну базу даних та подальше зчитування даних. У процесі кожної операції фіксувався час виконання з використанням засобів вимірювання продуктивності (Stopwatch), що дозволило точно оцінити, скільки часу витрачається на видалення, вставку та читання інформації. Це дозволило порівняти швидкодію обох типів БД у контексті однакових операцій.



Риснок 4.4 Результат вимірювання продуктивності

Як показали результати(рисунок 4.5), InfluxDB v3 продемонструвала високу ефективність при збереженні та доступі до великого обсягу даних із часовими позначками, що підтверджує її доцільність у задачах з обробки часових рядів. З іншого боку, SQL Server добре впорався зі збереженням структурованих даних, однак при роботі з великим обсягом часових даних поступається TSDB за швидкодією при виконанні агрегацій та серійного доступу. Отримані значення часу операцій дають змогу аналітично оцінити продуктивність і допомагають обрати найбільш ефективне сховище в залежності від характеру даних і задачі незважаючи на недолік у вигляді точності округлення до цілого.

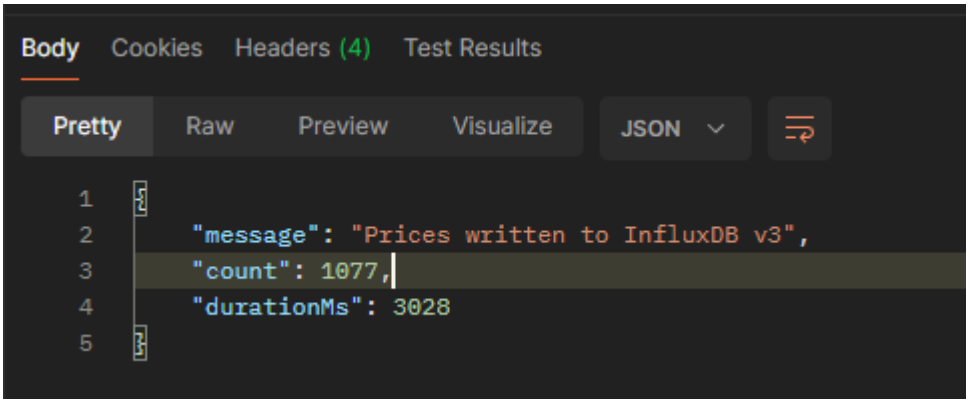


Рисунок 4.5 Результат вимірювання продуктивності для TSDB

У випадку з реляційною базою даних (SQL), перед записом нових значень реалізовано повне очищення таблиці з історичними даними, що дає

змогу точно виміряти час виконання кожного окремого етапу: видалення старих записів, вставки нових та подальшого зчитування. Для вимірювання часу було використано системний секундомір (stopwatch), що надає точну інформацію про тривалість кожної операції в мілісекундах. Це дозволяє побачити, наскільки швидко система здатна обробляти запити в умовах повного оновлення даних, що є типовим підходом для сценаріїв з обробкою історичних значень, які постійно змінюються.

У випадку з InfluxDB, яка є базою даних часового ряду, основну увагу було приділено вимірюванню швидкості вставки даних (Лістинг 4.3), оскільки ця система зберігає інформацію у вигляді точок із часовими мітками. Перевагою такого підходу є оптимізація саме під великі обсяги послідовних записів, що ілюструється високою швидкістю запису порівняно з SQL. Дані вставляються у вигляді окремих точок значеннями та міткою часу, після чого заміряється загальна тривалість процесу запису.

Лістинг 4.3 Запис у базу даних часових рядів

```
using Microsoft.AspNetCore.Mvc;
using series_api._Services;
using System.Diagnostics;

[ApiController]
[Route("api/[controller]")]
public class TimeSeriesController : ControllerBase
{
private readonly IBtcPriceService _btcService;
private readonly InfluxV3WriteService _influxWriter;

public TimeSeriesController(IBtcPriceService btcService,
InfluxV3WriteService influxWriter)
{
_btcService = btcService;
_influxWriter = influxWriter;
}

[HttpPost("load")]
public async Task<IActionResult> Load([FromQuery] DateTime
start)
{
var btcPrices = await _btcService.GetPricesAsync(start);
var sw = Stopwatch.StartNew();
await _influxWriter.WritePricesAsync(btcPrices);
```

```
sw.Stop();

return Ok(new
{
message = "Prices written to InfluxDB v3",
count = btcPrices.Count,
durationMs = sw.ElapsedMilliseconds
});
}
}
```

Отримані результати дозволяють порівняти ефективність роботи двох типів баз даних при роботі з ідентичним масивом даних. Таким чином, реалізований підхід забезпечує прозору та наочну оцінку продуктивності кожної системи, дозволяючи зробити висновки про доцільність їх використання в залежності від типу задачі – зберігання історичних значень або обробка реального часу.

ВИСНОВКИ

Застосування баз даних часових рядів у сучасних веб-застосунках є необхідною складовою якісного управління системами, що працюють у режимі реального часу. В умовах постійного зростання обсягів даних з часовою прив’язкою та дедалі більшої складності інфраструктури, такі бази даних забезпечують ефективний механізм збирання, зберігання та обробки метрик. Їх використання дозволяє не лише своєчасно реагувати на технічні або поведінкові аномалії, а й прогнозувати майбутні стани системи на основі аналізу динаміки показників.

Досягнення цієї мети було реалізовано шляхом створення повноцінного веб-застосунку, що демонструє можливості інтеграції бази даних часових рядів у сучасну інфраструктуру. У роботі було реалізовано -систему для збору та обробки даних , з подальшим збереженням інформації в реляційну (SQL) та часову (InfluxDB) бази даних. Окрему увагу приділено порівнянню продуктивності цих двох підходів шляхом вимірювання часу запису та читання з обох БД.

Було також реалізовано сучасну панель моніторингу з використанням Grafana, яка забезпечує візуалізацію ключових метрик, таких як використання оперативної пам’яті, кількість запитів у певному інтервалі часу, та дані про ціну. Для демонстрації можливостей системи візуалізації додано графіки типу gauge, bar, pie та таблиці, що дозволяють аналізувати інформацію в реальному часі. Крім того, налаштовано алертинг на основі показників, що дозволяє відстежувати критичні ситуації та реагувати на них .

Завдяки високій продуктивності, здатності до масштабування, спеціалізованим алгоритмам агрегації й стискання, бази даних часових рядів забезпечують стійкість до навантажень і гнучкість у роботі з великими потоками даних. У поєднанні з системами візуалізації та алертингу вони дають можливість побудувати повноцінну систему моніторингу та

адміністрування, яка підтримує безперервну роботу складних, розподілених веб-систем. Тип даних у часовому ряді визначає як методи його зберігання, так і способи аналізу. Для числових даних доречно використовувати математичну обробку та агрегацію, для бінарних – виявлення подій, а для текстових – логічний аналіз та контекст. У реальних системах часто застосовуються комбіновані формати даних, що вимагають потужних систем зберігання, таких як TSDB, здатних ефективно обробляти великі обсяги змішаних даних.

Таким чином, інтеграція баз даних часових рядів у архітектуру веб-застосунку є не лише технологічною перевагою, а й стратегічною необхідністю для підвищення якості обслуговування, надійності й довготривалої підтримки цифрових продуктів що підтверджується їх популяризацією і інтеграцію у великих технологічних компаніях

Таким чином, практична частина роботи підтверджує теоретичну доцільність використання баз даних часових рядів і демонструє реальне впровадження всіх етапів: від збору й обробки даних до візуалізації й моніторингу, що робить запропоноване рішення релевантним для сучасних застосунків, орієнтованих на стабільність, масштабованість і аналітику в реальному часі.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Ted Dunning, Ellen Friedman. Time Series Databases: New Ways to Store and Access Data. 2014. 1st Edition. P. 53-78 ISBN: 9781491914724

2. Уточкін К. М. , Ні Я. С. USAGE OF THE TIME SERIES DATABASES IN THE MODERN WEB APPLICATIONS : тези доп. 15-ї міжнар. наук.-техн. конф. (24–25 квіт. 2025 р., Баку – Харків – Жиліна). – Т. 2. – Харків ; Баку ; Жиліна, 2025. – С. 45.

3. Часові ряди, основні поняття, методи аналізу часових рядів . URL: https://web.posibnyky.vntu.edu.ua/fksa/12kocubynsky,kyslycia_osn_model_rynk_sytuac/p4.html(дата звернення: 16.06.2025).

4. Prometheus Developers Documentation. URL: https://prometheus.io/docs/introduction/overview/ (дата звернення: 15.06.2025).

5. InfluxData. InfluxDB Time series Database Docs. URL: https://docs.influxdata.com/influxdb/v2.0/ (дата звернення: 13.06.2025).

6. TimescaleDB Documentation. URL: https://docs.timescale.com/latest/ (дата звернення: 22.05.2025).

7. Grafana Labs. Grafana Documentation. URL: https://grafana.com/docs/grafana/latest/ (дата звернення: 12.06.2025).

8. Official Microsoft Documentation. ASP.NET Core Web API. URL: https://learn.microsoft.com/en-us/aspnet/core/web-api/?view=aspnetcore-9.0&WT.mc_id=dotnet-35129-website (дата звернення: 15.06.2025).

9. OpenTSDB Documentation. URL: http://opentsdb.net/docs/build/html/ (дата звернення: 22.05.2025).

10. Graphite file based whisper DB documentation. URL: https://graphite.readthedocs.io/en/latest/ (дата звернення: 10.06.2025).

11. Rob Chapman, Peter Holmes Observability with Grafana: Monitor, control, and visualize your Kubernetes and cloud platforms using the LGTM stack.

2014. 1st Edition P. 256 – 312 ISBN: 1803248009

12. Brian Brazil. Prometheus: Up & Running: Infrastructure and Application Performance Monitoring 2018 1st Edition P.114-198 ISBN: 9781492034094

13. Pranav Shukla, Sharath Kumar M N Learning Elastic Stack 7.0: Distributed search, analytics, and visualization using Elasticsearch, Logstash, Beats, and Kibana, 2nd Edition 2019 P. 24-41 ISBN: 1789954398

14. Thomas Kurian Theakanath (Author) Datadog Cloud Monitoring Quick Start Guide: Proactively create dashboards, write scripts, manage alerts, and monitor containers using Datadog 2021 P.218-276 ISBN: 1800568738

15. Slawek Ligus Effective Monitoring and Alerting: For Web Operations 1st Edition 2012 P. 14-88 ISBN: 1449333524

16. Product-Focused Reliability for SRE and Maintaners URL: https://sre.google/resources/practices-and-processes/product-focused-reliability-for-sre/ (дата звернення: 17.06.2025)

17. Grafana visualization types with examples URL: https://grafana.com/docs/grafana/latest/panels-visualizations/visualizations/(дата звернення: 18.06.2025)