НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»

Факультет прикладної математики

Кафедра програмного забезпечення комп'ютерних систем

«До захисту допущено»
Науковий керівник кафедри
_____ Іван ДИЧКА
«___» _____ 2020 р.

Дипломний проєкт

на здобуття ступеня бакалавра

за освітньо-професійною програмою «Інженерія програмного забезпечення комп'ютерних та інформаційно-пошукових систем»

спеціальності 121 Інженерія програмного забезпечення

на тему: «Програмна система моніторингу веб-додатків в реальному часі»

Виконав:
студент IV курсу, групи КП-61
Щербина Вадим Олегович _____

Керівник:
Ст. викладач кафедри ПЗКС, к.т.н.,
Люшенко Леся Анатоліївн _____

Консультант з нормоконтролю:
Доцент кафедри ПЗКС, к.т.н., доцент,
Онай Микола Володимирович _____

Рецензент:
Доц. каф. ММСА ІПСА, к.ф.-м.н., доц.,
Шубенкова Ірина Анатоліївна _____

Засвідчую, що у цьому дипломному проєкті немає запозичень з праць інших авторів без відповідних посилань.
Студент _____

Київ – 2020 року

Національний технічний університет України

«Київський політехнічний інститут імені Ігоря Сікорського»

Факультет прикладної математики

Кафедра програмного забезпечення комп'ютерних систем

Рівень вищої освіти – перший (бакалаврський)

Спеціальність – 121 «Інженерія програмного забезпечення»

Освітньо-професійна програма «Інженерія програмного забезпечення комп'ютерних та інформаційно-пошукових систем»

ЗАТВЕРДЖУЮ

Науковий керівник кафедри

_____ Іван ДИЧКА

«___»_____2019 р.

ЗАВДАННЯ

на дипломний проєкт студенту

Щербині Вадим Олегович

1. Тема проєкту «Програмна система моніторингу веб-додатків в реальному часі», керівник проєкту Люшенко Леся Анатоліївна, к.т.н., старший викладач, затверджені наказом по університету від «___»_____ 2020 р. №_____

2. Термін подання студентом проєкту «___» червня 2020 р.

3. Вихідні дані до проєкту: див. Технічне завдання.

4. Зміст пояснювальної записки:
   - аналіз існуючих рішень поставленої задачі;
   - обґрунтування вибору засобів реалізації;
   - розроблення моніторингової системи;
   - аналіз розробленої моніторингової системи.

5. Перелік обов'язкового графічного матеріалу:
   - схема бази даних (креслення);
   - схема додавання метрики (креслення);
   - діаграма використання системи (плакат);
   - структурна схема системи (плакат).

6. Консультанти розділів проєкту

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Нормоконтроль	Онай М.В., доцент		

7. Дата видачі завдання «31» жовтня 2019 р.

Календарний план

№ з/п	Назва етапів виконання дипломного проєкту	Термін виконання етапів проєкту	Примітка
1.	Вивчення літератури за тематикою проєкту	14.11.2019	
2.	Розроблення та узгодження технічного завдання	25.11.2019	
3.	Розроблення архітектури системи	15.12.2019	
4.	Розроблення дизайну сторінок та графічних елементів	03.02.2020	
5.	Програмна реалізація серверної частини системи	17.03.2020	
6.	Програмна реалізація клієнтської частини системи	23.02.2020	
7.	Тестування системи	12.04.2020	
8.	Підготовка матеріалів текстової частини проєкту	25.04.2020	
9.	Підготовка графічної частини дипломної роботи	10.05.2020	
10.	Оформлення технічної документації проєкту	25.05.2020	

Студент Вадим ЩЕРБИНА

Керівник проєкту Леся ЛЮШЕНКО

АНОТАЦІЯ

Даний дипломний проєкт присвячений розробці програмної моніторингової системи веб-додатків в реальному часі для відслідковування користувацьких метрик, підрахунку статистичних показників по них і візуалізації цих даних.

У даному дипломному проєкті було проведено порівняння з існуючими аналогами моніторингової системи. Було проаналізовано функціонал, який надають ці системи і створено відповідні функціональні і нефункціональні вимоги до системи.

Дана моніторингова система розроблена у вигляді веб-додатка і клієнтської бібліотеки. Веб-додаток містить сторінку для перегляду користувацьких додатків над якими ведеться моніторинг з можливістю перегляду графіків з користувацькими метриками. Клієнтська бібліотека являє собою модуль, який використовується для збору і відправки необхідних метрик на серверну частину, для подальшої їх обробки. Серверна частина даної системи містить спеціально розроблену структуру даних для обрахунку статистичних показників, яка була протестована на продуктивність.

У даному дипломному проєкті розроблено: архітектуру для клієнтської і серверної частин системи, модуль для обрахунку статистичних показників, модуль для відображення користувацьких метрик, модуль відправки сповіщень, модуль для відправки користувацьких метрик та інтерфейс користувача.

ABSTRACT

The current project is dedicated to the monitoring system research for web-extensions in real time to observe users' metrics, examine the resulting statistics and visualize gathered data.

For comparison, this project used the existing analogues of the monitoring systems. The functionalities of said systems were analyzed, and functional, as well as, non-functional requirements for the system were created accordingly.

The current monitoring system was developed in a form of a web-extension and a client library. The web-extension contain a page to view user extensions, which are being monitored, along with the possibility of observing the graphs containing user metrics. The client library is in itself a module, that is used to collect and distribute the necessary metrics to the server part for their further processing. The server part of this given system contains a specially designed data structure for calculating statistical measurements that had been tested on their productivity.

During this project, the following were developed: the architecture of the client and server parts of the system, the calculation module for statistical measurements, the display module for user metrics, the notification delivery module, the module for exporting user metrics, and the user interface.

АННОТАЦИЯ

Данный дипломный проект посвящен разработке программной мониторинговой системы веб-приложений в реальном времени для отслеживания пользовательских метрик, подсчета статистических показателей по ним и визуализации этих данных.

В данном дипломном проекте было проведено сравнение с существующими аналогами мониторинговой системы. Были проанализированы функционал, который предоставляют эти системы и созданы соответствующие функциональные и нефункциональные требования.

Данная мониторинговая система разработана в виде веб-приложения и клиентской библиотеки. Веб-приложение содержит страницу для просмотра пользовательских приложений над которыми ведется мониторинг с возможностью просмотра графиков с пользовательскими метриками. Клиентская библиотека представляет собой модуль, который используется для сбора и отправки необходимых метрик на серверную часть для дальнейшей их обработки. Серверная часть данной системы содержит специально разработанную структуру данных для расчета статистических показателей, которая была протестирована на производительность.

В данном дипломном проекте разработаны: архитектура для клиентской и серверной частей системы, модуль для расчета статистических показателей, модуль для отображения пользовательских метрик, модуль отправки уведомлений, модуль для отправки пользовательских метрик и интерфейс пользователя.

ДП.045440-01-90 Програмна система моніторингу веб-додатків в реальному часі.

Відомість проєкту

Позначення	Найменування	Кіл-ть	Примітка
	Документація проєкту		
ДП.045440-02-91	Програмна система	5	
	моніторингу веб-додатків		
	в реальному часі.		
	Технічне завдання		
ДП.045440-03-81	Програмна система	63	
	моніторингу веб-додатків		
	в реальному часі.		
	Пояснювальна записка		
ДП.045440-04-51	Програмна система	4	
	моніторингу веб-додатків		
	в реальному часі.		
	Програма та методика		
	тестування		
ДП.045440-05-34	Програмна система	7	
	моніторингу веб-додатків		
	в реальному часі.		
	Керівництво користувача		
ДП.045440-06-99	Програмна система	1	
	моніторингу веб-додатків		
	в реальному часі.		
	Додавання нової метрики		
	Схема додавання метрики		

Позначення	Найменування	Кіл-ть	Примітка
ДП.045440-07-99	Програмна система	1	
	моніторингу веб-додатків		
	в реальному часі.		
	Структура бази даних		
	ERD-діаграма		
ДП.045440-08-98	Програмна система	1	
	моніторингу веб-додатків		
	в реальному часі.		
	Компакт-диск		

Факультет прикладної математики

Кафедра програмного забезпечення комп'ютерних систем

«ЗАТВЕРДЖЕНО»
Науковий керівник кафедри
_____ Іван ДИЧКА
"___" _____ 2019 р.

ПРОГРАМНА СИСТЕМА МОНІТОРИНГУ ВЕБ-ДОДАТКІВ В РЕАЛЬНОМУ ЧАСІ

Технічне завдання

ДП.045440-02-91

«ПОГОДЖЕНО»

Керівник проєкту:

_____ Леся ЛЮШЕНКО

Нормоконтроль:

_____ Микола ОНАЙ

Виконавець:

_____ Вадим ЩЕРБИНА

2019

ЗМІСТ

1. НАЙМЕНУВАННЯ ТА ГАЛУЗЬ ЗАСТОСУВАННЯ

Назва розробки: Програмна система моніторингу веб-додатків в реальному часі.

Галузь застосування: інформаційні технології.

2. ПІДСТАВА ДЛЯ РОЗРОБЛЕННЯ

Підставою для розроблення є завдання на дипломне проєктування, затверджене кафедрою програмного забезпечення комп'ютерних систем Національного технічного університету України «Київський політехнічний інститут імені Ігоря Сікорського» (КПІ ім. Ігоря Сікорського).

3. ПРИЗНАЧЕННЯ РОЗРОБКИ

Розроблювальна моніторингова система призначена для моніторингу метрик (системних і користувацьких), обрахунку статистичних показників і візуалізації цих даних на графіках, що дозволяє користувачу прийняти рішення стосовно подальшого управління додатком.

4. ВИМОГИ ДО ПРОГРАМНОГО ПРОДУКТУ

Моніторингова система має відповідати таким вимогам:

- забезпечення обрахунку та відображення необхідних статистичних даних:
    - мінімальне/максимальне значення для метрики;
    - середнього значення;
    - середнє квадратичне відхилення.
- збереження, фільтрування і сортування даних за часом;
- сповіщення про некоректну роботу:
    - обробка і аналіз метрик, для визначення системних збоїв і перевищення заданих значень;
    - відправка сповіщень про збої в мессенджери.

- система має відображати показники використання системних ресурсів (CPU, RAM і ROM) фізичної машини на якій запущено веб-додаток;
- система має надавати можливість переглядати графіки за певний проміжок часу;
- захист від збоїв:
    - основні статистичні дані зберігаються у файл через певний проміжок часу;
    - відновлення системи на випадок збою.
- система має надавати змогу користувачам налаштовувати графіки і сповіщення:
    - можливість вибирати тип графіку і вид і кількість графіків на сторінці;
    - можливість налаштовувати метрики для, яких потрібно відправляти сповіщення.
- система повинна перевіряти доступність веб-додатку.

5. ВИМОГИ ДО ПРОЄКТНОЇ ДОКУМЕНТАЦІЇ

У процесі виконання проєкту повинна бути розроблена наступна документація:

- пояснювальна записка;
- програма та методика тестування;
- керівництво користувача;
- креслення:
    - «Моніторингова система веб-додатків в реальному часі. Схема використання»
    - «Моніторингова система веб-додатків в реальному часі. Структура бази даних. ERD-діаграма».

6. ЕТАПИ ПРОЄКТУВАННЯ

Вивчення літератури за тематикою роботи……..14.11.2019

Розроблення та узгодження технічного завдання……....25.11.2019

Розроблення архітектури системи…….....15.12.2019

Розроблення дизайну сторінок та графічних елементів……..03.02.2020

Програмна реалізація серверної частини системи…..……....17.03.2020

Програмна реалізація клієнтської частини системи……....23.02.2020

Тестування системи…….....12.04.2020

Підготовка матеріалів текстової частини проєкту……..….25.04.2020

Підготовка матеріалів графічної частини проєкту……..…10.05.2020

Оформлення технічної документації проєкту……..…25.05.2020

7. ПОРЯДОК ТЕСТУВАННЯ РОЗРОБКИ

Тестування розробленого програмного продукту виконується відповідно до “Програми та методики тестування”.

Факультет прикладної математики

Кафедра програмного забезпечення комп'ютерних систем

«ЗАТВЕРДЖЕНО»
Науковий керівник кафедри
_____ І.А. Дичка
"___" _____ 2020 р.

ПРОГРАМНА СИСТЕМА МОНІТОРИНГУ ВЕБ-ДОДАТКІВ В РЕАЛЬНОМУ ЧАСІ

Пояснювальна записка

ДП.045440-03-81

«ПОГОДЖЕНО»

Керівник проєкту:
_____ Леся ЛЮШЕНКО

Нормоконтроль:
_____ Милола ОНАЙ

Виконавець:
_____ Вадим ЩЕРБИНА

2020

ЗМІСТ

СПИСОК ТЕРМІНІВ, СКОРОЧЕНЬ ТА ПОЗНАЧЕНЬ

Моніторинг – це процес збору і обробки даних для подальшого їх аналізу з метою співставлення отриманих результатів з очікуваними [1].

Метрика – міра для представлення деякої властивості програмного забезпечення у вигляді числового значення [2].

CPU – центральний процесор, системна метрика, яка визначає відсоток навантаження центрального процесору [3].

RAM – оперативна пам’ять, системна метрика, що показує використання опертивної пам’яті на пристрої [4].

ROM – пам’ять постійного зберігання даних, системна метрика, яка показує кількість байтів зчитаної і записаної на диск інформації [5].

MVC – Model-View-Controller схема, що дозволяє розділити додаток на 3 частини: модель, інтерфейс корисувача і контролер, який дозволяє взаємодіяти моделі і інтерфейсу корисувача.

Node.js – платформа для створення серверних додатків за допомогою мови програмування Javascript.

SQL (англ. Structured query language) – декларативна мова запитів до бази даних, яка використовується реляційнимм базами даних. За допомогою цієї мови можна створювати, оновлювати, шукати і видаляти дані з бази даних.

Angular – фреймворк для створення браузерної частиги веб-додатку, використовує мову програмування Typescript, яка є надбудовою над мовою програмування Javascript і має типізацію.

JavaScript – прототипна і скриптова мова програмування, яка немає строгої типізації.

Bootstrap – це набір інструментів з відкритим програмним кодом для створення дизайну сторінок веб-додатків, який містить свої шаблони HTML і CSS. Ці шаблони використовуються для створення різниз компонентів на сторінці: кнопки, форми, навігація та інші.

Angular material – це набір інструментів з відкритим програмним кодом для створення дизайну сторінок веб-додатків, який встановлюється разом з фреймворком Angular.

AI (англ. Artetificial intelligence) – штучний інтелект.

ПЗ – програмне забезпечення.

DOM (англ. Document Object Model)– специфікація прикладного програмного інтерфейсу для роботи зі структурованими документами.

API (англ. Application Programming Interface) – набір визначень підпрограм, протоколів взаємодії та засобів для створення програмного забезпечення.

ВСТУП

Сьогодні, з кожнем днем росте кількість веб-додатків і проблема їх моніторингу стає нагальною. Виникає досить значне навантаження на адміністраторів веб-додатків, які не встигають контролювати їх роботу. Через це виникають збої в роботі додатків, що може призвести до їх недоступності. Однією з головних проблем, яка виникає при роботі веб-додатків є їх неправильне адміністрування і не своєчасне прийняття рішень, щодо управління додатком [6]. Неправильне адміністрування додатку може призвести до перебоїв в його роботі, що може зменшити зацікавленість і кількість користувачів. Щоб запобігти цим проблемам необхідно використовувати моніторингову систему, яка буде інформувати адміністраторів при винекненні проблем з веб-додатком. Для кращого аналізу система повина забезпечувати обрахунок статистичних показників [7], таких як:

- мінімальне/максимальне значення для метрики;
- середнього значення;
- середнє квадратичне відхилення.

Ефективна моніторингова система є важливим елементом для хорошого управління додатком. Вона підтримує поінформованість та своєчасність прийняття рішень системними адміністраторами, керівниками проєктів та іншими зацікавленими сторонами [8]. Така система є ключовою частиною процесу управління проєктом.

Існуючі аналоги моніторингових систем не є досконалими і не мають системи сповіщень для популярних месенджерів і підрахунку статистичних показників по користувацьким метрикам.

Основна мета даного дипломного проєкту – це створення моніторингової системи, яка буде відображати і обраховувати статистику по користувацьким метрикам в реальному часі і відправляти сповіщення при винекненні збоїв в користувацьких додатках.

1. АНАЛІЗ ІСНУЮЧИХ РІШЕНЬ ТА ОБГРУНТУВАННЯ ТЕМИ ДИПЛОМНОГО ПРОЄКТУ

1.1. Загальні положення та аналіз предметної області

З розвитком технологій і збільшенням кількості веб-додатків виникла проблема їх моніторингу для подальшої підтримки і розвитку. Моніторинг широко використовується у сфері веб-технологій для оцінки продуктивності і збору інформації про використання веб-додатків, що дозволяє слідкувати за діяльністю та визначати ефективність додатку протягом всього циклу його життя.

Моніторингова система надає можливість збирати і обробляти різні метрики (системні і користувацькі) для їх аналізу, що дозволить прийняти рішення стосовно подальшого управління додатком. Споживачами програмної моніторингової системи є адміністратори, які відповідають за підтримку і правильне функціонування ресурсу. На основі зібраних данних вони формують вимоги до додатку над яким ведеться спостереження, а також встановлюють причини, які перешкоджають нормальному функціонуванню веб-ресурсу.

Найпоширеніші причини проблемності роботи веб-додатків для вирішення, яких застосовують моніторингові системи:

- неефективно написаний код. Неефектно написаний написаний код може призвести до безлічі проблем із веб-додатками, включаючи неефективні алгоритми, витоки пам’яті та недоліки програми. Старі версії програмного забезпечення або застарілі інтегровані системи також можуть знижувати продуктивність [9];
- неоптимізовані бази даних. Оптимізована база даних забезпечує найвищий рівень безпеки та продуктивності. Відсутність індексів сповільнює продуктивність запитів до бази даних, що може сповільнити роботу веб-додатку загалом [10];

- обробка значної кількості даних. Веб-додатки мають тенденцію до збільшення кількості даних, які потрібно обробляти та надавати користувачам. Розробка плану управління та моніторингу такими даними (з урахуванням їх збільшення) необхідна для коректної роботи будь-якого веб-додатку [11];
- різке збільшення кількості користувачів. Збільшення користувачів, як правило, вважається показником популярності веб-додатку. Однак потрібно правильно розраховувати ресурси серверної частини, щоб вона могла витримати необхідне навантаження. Планування ресурсів заздалегідь є ключовим моментом для роботи веб-додатку. Таким чином, можна запобігти проблемі, перш ніж кількість користувачів додатку стане критичною [12];
- неправильний розподіл навантаження на серверах. Неправильний розподіл навантаження може зменшити час відповіді на запити до веб-додатку, неправильно розподіливши використання серверів для нових користувачів веб-додатку. Для запобігання цієї проблеми потрібно налаштувати балансир, який буде розподіляти використання серверів на яких знаходиться веб-додаток [13];
- мережеве підключення та доступність веб-додатку. Відсутність мережевого підключення на сервері не дозволить відвідувачам отримувати доступ до додатку та призведе до різних помилок, таких як 404 (ресурс не знайдено). Крім того, підключення до мережі та ефективність брандмауера мають вирішальне значення для доступу та продуктивності. Потрібно завжди перевіряти доступність додатку і в разі якихось помилок сигналізувати про них [14].

В таблиці 1 наведено аналіз існуючих проблем роботи веб-додатків.

Таблиця 1

Аналіз існуючих проблем роботи веб-додатків

Критерій / Проблема	Рішення проблеми	Критичність (1-5)	Наслідки
Неефективний написаний код	Аналіз програмного коду на наявність помилок та оптимізація основних алгоритмів.	3	Додаток працює повільно, можливі збої
Неоптимізована база даних	Додавання індексів, які підвищать швидкість запитів	2	На обробку запитів до бази даних витрачається багато часу
Обробка великої кількості даних	Збільшення системних ресурсів і оптимізація алгоритмів обробки даних	3	Сповільнення роботи веб-додатку, використовується багато системних ресурсів
Різке збільшення кількості користувачів	Збільшення системних ресурсів сервера і кількості серверів	4	Сповільнення роботи веб-додатку, використовується багато системних ресурсів
Неправильний розподіл навантаження на серверах	Правильно налаштувати балансир, який буде розподіляти навантаження на сервери	3	Один чи декілька з серверів будуть перенавантаженні, в той час як ресурси інших не будуть використовуватись

Продовження табл. 1

Мережеве підключення та доступність веб-додатку	Запустити додаток на резервних серверах, для того щоб коли пропаде підключення на основному додаток продовжив працювати за рахунок інших серверів	5	Недоступність веб-додатку

Щоб відсліткувати і діагностувати дані проблеми моніторингова система повина відслітковувати такі основні метрики:

- кількість активних користувачів системи;
- використання системних ресурсів (CPU, RAM, ROM);
- час відклику веб-додатку.

1.2. Аналіз існуючих програмних рішень

Моніторингова система SmartBear

SmartBear – моніторингова система, яка допомагає відслідковувати основні показника вашого веб-додатку, такі як:

- час завантаження DOM (час який потрібний для розбору документа сторінки);
- час на завантаження самої сторінки (завантаження всіх скриптів, стилів, картинок);
- швидкість виконання запитів до API (зберігання, редагування чи видалення даних);
- валідація даних, які повертає API (перевірка даних);

- перевірка на доступність веб-додатку.

На рис. 1 зображено вигляд користувацького інтерфейсу моніторингової системи SmartBear.

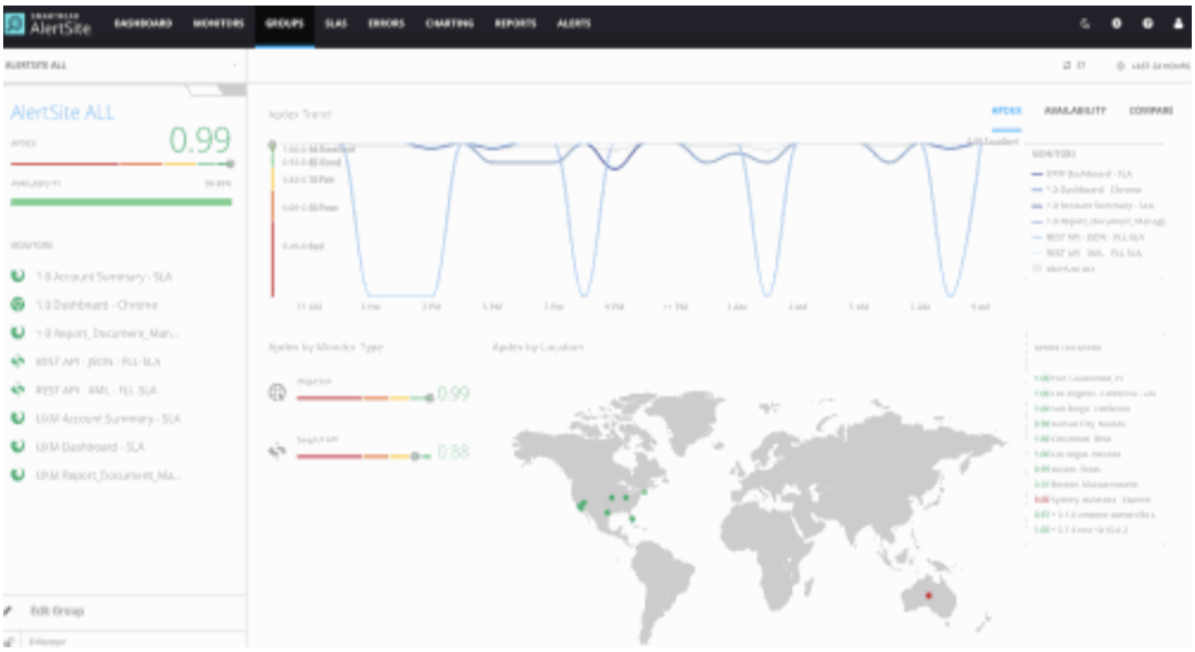


Рис. 1. Приклад інтерфейсу моніторингової системи SmartBear

Переваги ПЗ:

- зручний інтерфейс користувача;
- наявність функціоналу для аналізу даних;
- проста в налаштуванні.

Недоліки ПЗ:

- невеликий список даних, які можна відслідковувати;
- немає повної інтеграції з веб-додатком;
- відсутня система сповіщень;
- не можливо відслідкувати системні метрики;
- відсутні статистичні показники.

Моніторингова система Dotcom-monitor

Dotcom-monitor – моніторингова система, яка надає можливість відслідковувати роботу веб-додатку. Основною задачею цієї системи є перевірка доступності додатку. На рис. 2 зображено вигляд користувацького інтерфейсу моніторингової системи Dotcom-monitor.

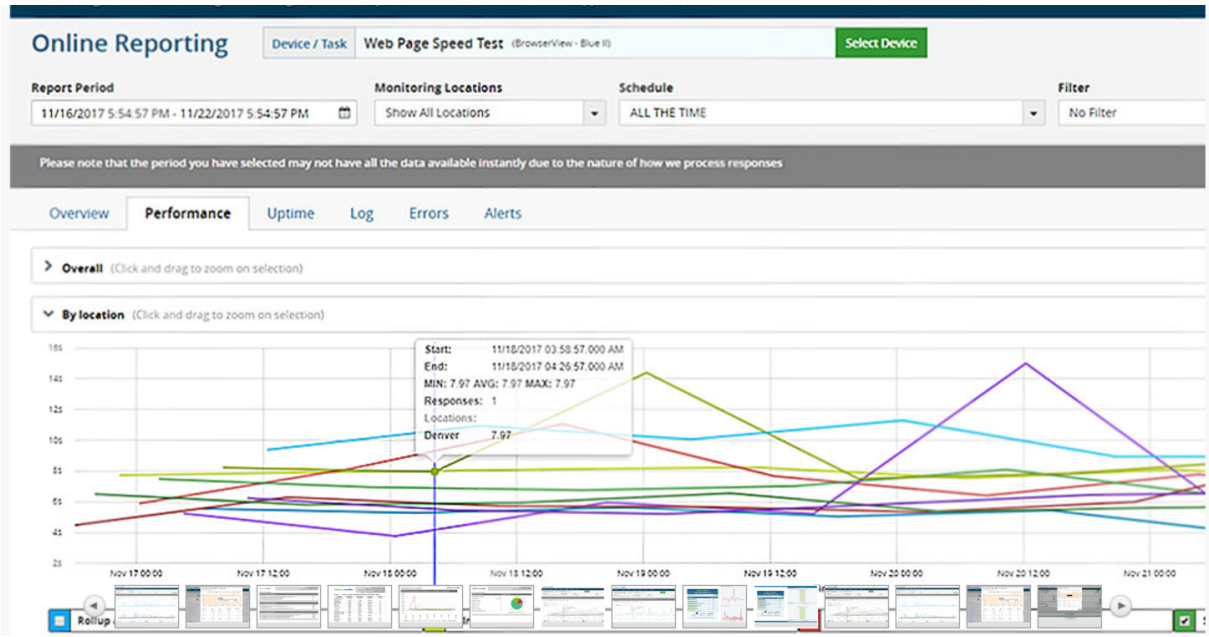


Рис. 2. Приклад інтерфейсу моніторингової системи Dotcom-monitor

Переваги ПЗ**:**

- перевіряє доступність всіх ресурсів веб-сайту;
- існує система сповіщень, яка сигналізує про помилки;
- дані відображаються на графіках;
- має можливість інтеграції через API;
- зберігається відео при перевірці доступності ресурсу.

Недоліки ПЗ:

- незручний інтерфейс користувача;
- відсутні підрахунок статистичних показників;
- велика плата за використання системи.

Моніторигова система vRealize Hyperic

Моніторингова система VRealize Hyperic контролює інфраструктуру, операційну систему та різні ПО з єдиною зручною панеллю моніторингу. Дана система сумісна з більш ніж 70 типами програм, вона може відстежувати до 50 000 метрик, в той же час, має систему споавіщень. Адміністратори можуть настроювати та персоналізувати параметри оповіщень, гарантуючи, що всі вони будуть відправлені. На рис. 3 зображено вигляд користувацького інтерфейсу моніторингової системи VRealize Hyperic.

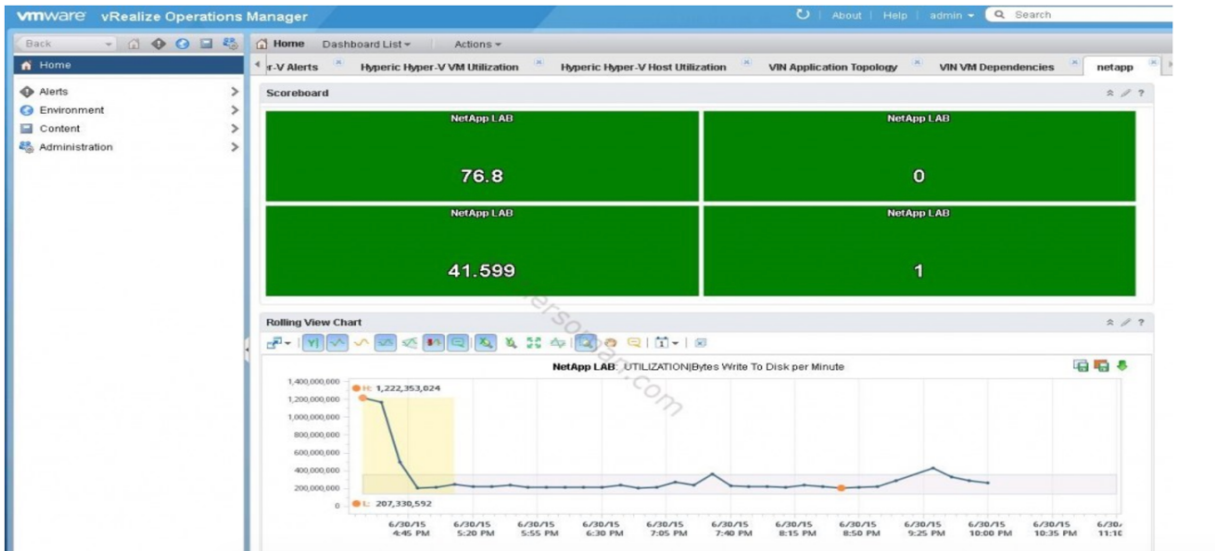


Рис. 3. Приклад інтерфейсу моніторингової системи VRealize Hyperic

Дана моніторингова система підходить для адмінастраторів і розробників веб-додатків, оскільки її API дозволяє користувачам створювати і підключати модулі до своїх веб-додатків.

Переваги ПЗ:

- перевіряє доступність всіх ресурсів веб-сайту;
- існує система сповіщень, яка сигналізує про помилки;
- має можливість інтеграції через API;
- можна відслідковувати велику кількість метрик.

Недоліки ПЗ:

- незручний інтерфейс користувача;
- не має можливості персоналізувати відображення графіків;
- відсутні підрахунок статистичних показників.

Аналіз характеристика існуючих рішень зведено до порівняльної таблиці 2.

Таблиця 2

Порівняльна характеристика існуючих рішень

Основні функції / Система	Підрахунок статистичних даних	Перевірка доступності веб-додатку	Сповіщення користувачів	Візуалізація даних	Аналіз та обробка моніторингових даних
SmartBear	–	+	–	+	+
Dotcom-monitor	–	+	+	+	+
vRealize Hyperic	–	+	+	+	–

В результаті аналізу існуючих рішень було виявлено те, що вони не надають весь необхідні функціональні можливості, які необхідно користувачеві для детального аналізу веб-додатків. Зокрема кожна з них має один чи більше наступних недоліків:

- відсутній підрахунок статистичних даних;
- збір системних метрик;
- система сповіщень.

1.3. Актуальність розробки моніторингової системи

Розроблювана моніторингова система надає користувачам весь необхідний функціонал для відслідковування процесу роботи веб-додатку. Окрім того користувачам не потрібно буде запускати моніторингову систему в себе на комп'ютері чи шукати сервер на якому її запустити, для кожного користувача буде створюватись окремий екземпляр програми до якого буде мати доступ тільки він. Таке рішення спростить користування системою і надасть безперебійний доступ до неї. Також розроблювальна моніторингова система має систему сповіщень, яка буде сигналізувати про системні збої.

Розроблюване програмне забезпечення повине відповідати таким вимогам:

- забезпечення обрахунку та відображення необхідних статистичних даних:
    - мінімальне/максимальне значення для метрики;
    - середнього значення;
    - середнє квадратичне відхилення.
- збереження, фільтрування і сортування даних за часом;
- сповіщення про некоректну роботу:
    - обробка і аналіз метрик, для визначення системних збоїв і перевищення заданих значень;
    - відправка сповіщень про збої в мессенджери.
- система має відображати показники використання системних ресурсів (CPU, RAM і ROM) фізичної машини на якій запущено веб-додаток;
- система має надавати можливість переглядати графіки за певний проміжок часу;
- захист від збоїв:
    - основні статистичні дані зберігаються у файл через певний проміжок часу;

- відновлення системи на випадок збою.

- система має надавати змогу користувачам налаштовувати графіки і сповіщення;
- можливість вибирати тип графіку і вид і кількість графіків на сторінці;
- можливість налаштовувати метрики для, яких потрібно відправляти сповіщення;
- система повинна перевіряти доступність веб-додатку;
- система повинна надавати захист персональних даних користувача і використовувати алгоритми хешування для надійного захисту системи.

Нефункціональні вимоги:

- швидкий доступ до системи;
- зручний і інтуїтивний користувацький інтерфейс;
- легкість в налаштуванні.

1.4. Вимоги до безпеки системи

Безпека користувацьких даних є дуже важливим питанням. Щоб забезпечити їх цілісність і конфіденційність потрібно проаналізувати всі ризики і прийняти заходи для їх запобігання.

Таблиця 3

Загрози безпеці

№ з/с	Уразливість	Загроза	Вимоги до безпеки
1	Поганий захист бази даних	Несанкціонований доступ до бази даних	Надійні паролі для бази даних

Продовження табл. 3

2	Відсутність шифрування	Викрадення даних	Використання актуальних алгоритмів шифрування
3	Невірно введені дані	Збої в роботі системи, загроза несанкціонованого доступу в систему	Перевірка і валідація введених користувацьких даних

Дані загрози є критичними для коректної роботи системи, тому потрібно вжити заходів для запобігання цих загроз, таких як:

- коректна валідація даних;
- використання актуальних алгоритмів шифрування;
- надійні паролі, для бази даних.

1.5. Висновки до розділу

В наш час, коли розробляється безліч веб-застосунків існує проблема моніторингу їх роботи, для відслідковування основних метрик і статистичних показників, таких як: кількість користувачів веб-додатку, кількість переходів, працездатність, безпека та інших.

Розроблена моніторингова система призначена для вирішення даних проблем і надає зручний користувацький інтерфейс і має систему сповіщень.

В цьому розділі були проаналізовані функціональні і нефункціональні вимоги до даного програмного забезпечення. Окрім цього було проведено аналіз ризиків до безпеки даної системи, було створено таблицю ймовірних загроз і дій для їх запобігання. Також було проводено порівняння з аналогами, яке показало, що наявні аналоги моніторингових систем не є досконалими і не мають системи сповіщень для популярних месенджерів чи можливості підрахунку статистичних показників по користувацьким метрикам.

2. ОБҐРУНТУВАННЯ ВИБОРУ ЗАСОБІВ РЕАЛІЗАЦІЇ

2.1. Обґрунтування вибору мови програмування

Моніторингова система реалізується у вигляді веб-додатку, тому потрібно проаналізувати найпопулярніші мови програмування.

Для аналізу взяті найпопулярніші мови програмування, які відповідають таким вимогам:

- мати постійну підтримку і оновлення;
- велику кількість готових бібліотек;
- мати зручний фреймворк для написання веб-додатків.

Мова програмування Java

Java – С-подібна об’єктно-орієнтована мова програмування, яка підтримується компанією Oracle. Синтаксис мови схожий на С, С++, але управлінням пам’ятю займається віртуальна машина, що спрощує роботу програміста. Основною перевагою мови програмування Java є те, що її код може виконуватись на різних платформах. Весь написаний код інтерпретується в байт-код, який може виконуватись на будь-якій платформі за допомогою віртуальної машини. Свою об’єктно-орієнтованість Java запозичила в мови програмування С++, але її було модифіковано і покращенно, для того, щоб спростити роботу розробників. Мова є строго типізованою, це дозволяє зменшити кількість помилок і збільшити надійність програми [15].

Переваги мови програмування Java:

- програмний код може виконуватись на різних платформах і не потребує модифікацій;
- використовується для написання складних систем, через свою надіжність і підтримку;
- С – подібний синтаксис, легка у вивчені і розуміні.

Недоліки:

- низька продуктивність, використання віртуальної машини знижує продуктивність написаного коду, а також Java сама керує використанням пам'яті, що призводить до використання системних ресурсів.

Мова програмування Python

Python – високопродуктивна мова програмування, яка має зручний мінімалістичний синтаксис, а також є інтерпретованою, об’єктно-орієнтованою і має строгу типізацію. Основною перевагою мови програмування Python є низький рівень входження, також вона надає великий набір стандартних функцій. Мова програмування Python не має строгої типізації, що підвищує швидкість написання програм, але знижує безпеку і надійність розробленого програмного забезпечення. Основною перевагою цієї мови програмування є її модульність, розробник може розбивати програму на різні модулі, що є дуже ефективним. Також Python містить багато стандартних модулів, які дозволяють працювати з файлами, створювати графічні інтерфейси і управляти системними викликами.

Переваги:

- простий і зрозумілий синтаксис, та велика кількість стандартних модулів дозволяє швидко розроблювати програмне забезпечення;
- має автоматичне керування пам'яттю;
- має безліч фреймворків і модулів, які використовуються для написання складних систем з використанням машиного навчання.

Недоліки:

- через автоматичне використання пам'яті і нестрогої типізації використання оперативної пам'яті не є мінімальними.

Мова програмування Javascript

JavaScript – прототипна і скриптова мова програмування, яка немає строгої типізації. В основному застосовується на клієнтській частині веб-додатку для керування вмістом сторінки у браузері, але також може застосовуватись для програмування на стороні сервера за допомогою фреймворка NodeJs. На даний момент є однією з найпопулярніших мов програмування, через свою гнучкість і використання у браузерах. Має стандартизацію – EcmaScript, кожного року виходять нові версії, які містять нові функції і покращують використання мови [16].

Javascript має такі властивості:

- немає строгої типізації;
- керування пам'яттю здійснюється автоматично;
- підтримує прототипну і функціональну парадигми програмування;
- має вбудовану підтримку асинхроності.

Переваги:

- підтримується усіма браузерами і є основною мовою для написання сценаріїв для веб-сторінок;
- універсальність – можна використовувати, як для створення клієнтської так і серверної частин.

Недоліки:

- немає строгої типізації.

Мова програмування C#

C# – C-подібна об’єктно-орієнтована мова програмування, яку створила компанія Microsoft для платформи .Net. C# є дуже схожою до мови програмування Java своїми принципами і синтаксисом, так як їніми похідними є такі мови програмування, як:

- C ++;
- Object Pascal;

- C;
- SmallTalk.

Мова програмування C# є строго типізованою і використовується в основному для створення веб-додатків, в основному для передачі і збереження даних використовує мову розмітки XML [17].

Переваги:

- C# використовує об’єктно-орієнтований підхід для програмування. Це означає, що потрібно описувати всі сутності предметної області, для якої ведеться розробка програмного забезпечення і це знижує кількість помилок;
- має досить потужний засіб для розробки – Microsoft Visual Studio. Використовується для розробки як консольних програм так і більш складних програм з графічним інтерфейсом, а також веб-застосунків;
- має багато бібліотек, які допоможуть в розробці.

Недоліки:

- дуже сильно прив'язана до платформи .Net і операційної системи Windows, тому вона є не зовсім кросплатформеною і розробка на інших операційних системах є досить проблематичною.

Проаналізувавши різні мови програмування, для розробки було вибрано мову програмування JavaScript оскільки її можна використовувати одночасно для написання клієнтської і серверної частин. Для розроблення серверної частини використовується подієво-орієнтована платформа NodeJs, який має велику кількість готових бібліотек і фреймворків для розробки веб-застосунків. Дана мова має добру підтримку і входить у список найпопулярніших мов програмування. А також в неї є доступна документація з багатьма прикладами.

2.2. Вибір фреймворку для написання серверної частини

Для розробки серверної частини використовується подієво-орієнтована платформа NodeJs, яка є дуже ефективною для написання веб-застосунків, тому що вона використовує асинхрону модель обробки подій, тобто запити не блокують основний потік виконання програми, а відкладаються в окремий стек і виконуються там. Для розробки веб-завстосунків інує безліч фреймворків розглянемо і проаналізуємо їх [18].

Фреймворк Express.Js

Express.Js – є одним з найдавніших і найпопулярніших фреймворків для платформи Node.Js. Завдяки його стабільності і популярності існує багато прикладів і інструкцій з програмним кодом, які створили інші розробники і протестували на практичних задачах. Основною особливістю є те, що Express.Js володіє необхіднимм мінімальнимм функціональними можливостями для створення веб-додатку, через це являється надійним і добре протестованим. Для створення веб-серверу розробнику потрібно написати всього декілька рядків коду, це означає що розробник отримує легкий і надійний інструмент для вирішення різних задач. Для розширення функціональних можливостей цього фреймворку потрібно використовувати модулі які можна завантажити через пакетний менеджер - npm. Npm містить багато готових пакетів, які допоможуть в розробці.

Переваги:

- простота у використані і налаштуванні;
- гнучкість;
- має добру докуменацію з прикладами програмного коду;
- може поєднуватись з іншими модулями;
- легко масштабується.

Недоліки:

- необхідно шукати необхідні модулі для роботи;
- використовує застарілий підхід callback функцій.

Фреймворк Koa.Js

Koa – фреймворк, який був створений тією ж командою розробників, яка створила Express.Js, як удосконалений його аналог. Він розроблявся, щоб збільшити продуктивність веб-додатку. Він підтримує і використовує стандарт EcmaScript 6, що дозволяє писати більш зрозумілий код. Замість застарілих callback функцій він використовує генератори, які дозволяють підвищити продуктивність роботи. Але даний фреймворк є достатньо новим і не так широко використовується, як Express.js.

Переваги:

- продуктивність і швидкість роботи;
- гнучкість;
- використання генераторів замість callback функцій.

Недоліки:

- невелика популярність зі сторони розробників;
- менше прикладів програмного коду ніж у Express.Js.

Фреймворк Sails.js

Sails.js – це фреймворк, який в себе включає достатні функціональні можливості для створення веб-додатків без використання зовнішніх модулів. Завдяки цьому розробнику потрібно менше часу витрачати для пошуку необхідних модулів. Але це і є недоліком, тому що якщо він не містить необхідного модуля, то знайти потрібний і встановити є проблемою. Даний фреймворк має вбудовану ORM для управління різними базами даних – Waterline ORM. Але вона є недосконалою і не підтримує багато нових функцій, таких як: транзакції і нові функції виправлення помилок.

Переваги:

- широкі базові функціональні можливості;
- підтримка Socket.Io.

Недоліки:

- повільний в роботі;

- обмеження вбудованої ORM;
- документація не містить потрібних прикладів коду.

Таблиця 4

Порівняння фреймворків для платформи Node.Js

Фреймворк для платформи Node.Js	Зручність створення веб додатків	Гнучкість (використання зовнішніх модулів)	Швидкість роботи(1- 5)	Наявність прикладів програмного коду
Express.Js	Є досить зручним для створення веб-додатків, містить всі необхідні функціональні можливості	Легко працює і підтримує зовнішні модулі	4	Документація містить всі необхідні приклади програмного коду
Koa.Js	Є досить зручним для створення веб-додатків, містить весь необхідний мінімальний функціонал	Легко працює і підтримує зовнішні модулі	5	Недостатня кількість прикладів, пов’язана з тим, що фреймворк досить новий
Sails.Js	Має широкий функціонал, що дозволяє зручно створювати веб-додатки без зовнішніх модулів	Важко знайти необхідні модулі і підключити їх	2	Мало прикладів програмного коду, пов’язано з не велекою популярністю фреймворку. Документація не дуже хороша

Проаналізувавши існуючі фреймворки для свторення веб-додатків для платформи Node.Js для створення моніторингової системи було обрано фреймворк Express.Js. Він має більшу надійність, ніж інші фреймоврки, а також має хорошу документацію з великою кількістю програмного коду.

2.3. Вибір бази даних

Розроблюване програмне забезпечення повинно зберігати велику кількість користувацьких даних з складною структурою. Розглянемо і проаналізуєму існуючі системи екрування базами даних.

База даних Postgre SQL

PostgreSQL – широко розповсюдженна база данних, яка має досить потужний функціонал. Належить до групи реляційних баз даних, її вихідний код є відкритим, що дозволяє розробникам і різним компаніям постійно вдосконалювати його. Реляційну базу даних PostgreSQL використовують для розробки складних програмних систем, для яких важливим є питання безпеки їхніх даних. Має вбудовану мову прграмування PL/pgSQL, що дозволяє писати складні сценарії і алгоритми.

Переваги:

- велика підтримка зі сторони великих компаній і велика кількість обновлень;
- може масштабуватись, як і вертикально так і горизонтально;
- є можливість оброблювати і зберігати велику кількість даних;
- має інтуїтивно зрозумілий інтерфейс користувача PgAdmin.

Недоліки:

- складність встановлення і налаштування, а також подальшого адміністрування.

База даних MySQL

MySQL – одна з найбільш використовуваних реляційних баз даних. Має відкритий код системи і весь час оновлюється завдяки своїй популярності. Дана система керування базою даних має підтримку майже у всіх мовах програмування і легка в налаштуванні. Має зручний консольний інтерфейс для управління базами данних а також велику кількість додатків з графічним інтерфейсом для управління базами данних.

Переваги:

- легка в налаштуванні і адмініструванні;
- підтримує велику кількість типів даних і таблиць;
- існує багато програмних прикладів використання;
- легко масштабується і має безліч вбудованих функцій безпеки.

Недоліки:

- потребує більшого часу на виконання запитів до бази данних і виконання дій з даними (створення, оновлення, видалення);
- недуже надійна в процесах роботи з даними (транзакції і аудит).

База даних MongoDb

MongoDb – нереляційна, документо-орієнтована база даних. Для зберігання даних використовує текстовий формат даних – JSON. Використовується для зберігання даних, об’єкти яких можуть не мати чіткої структури, що є її перевагою. Має досить ефективну масштабованість за рахунок шардингу. Для використання різними мовами програмування необхідно встановити драйвер, який надасть користувачу доступ до бази данних за допомогою зручної функціональної мови запитів.

Переваги:

- відсутність схем для даних;
- легко масштабується;
- дані зберігаються у форматі Json;
- швидкий доступ до даних.

Недоліки:

- не можна використовувати мову запитів SQL.

Таблиця 5

Порівняння існуючих баз даних

Назва	Масштабованість	Мова запитів	Гнучкість зберігання даних	Складність в налаштуванні(1-5)	Швидкість обробки запитів
PostgreQsl	Підтримує горизонтальне масштабування, але через додаткові засоби	Sql	Дані зберігаються в таблицях із заданою структурою	5	Висока швидкість обробки вхідних запитів
MySql	Важко масштабується	Sql	Дані зберігаються в таблицях із заданою структурою	3	Обробляє запити повільніше, ніж в інших базах даних
MongoDb	Легко масштабується через вбудований шардинг і реплікацію	Функціональна	Дані не мають чіткої структури	1	Швидко обробляє вхідні запити

Проаналізувавши існуючі системи контролю базами даних для розроблюваної монітирингової системи було обрано СКБД MongoDb. Вона має функціональну мову запитів, що є зручним для розробки так, як мова JavaScript є функціональною. Також дана база даних легко масштабується, завдяки вбудованому шардингу і реплікації, на відміну від інших баз даних,

які потрібно налаштовувати за допомогою додаткового ПО і це є досить важким процесом. Її налаштування не потребує складних зусиль і для використання потрібно встановити спеціальний драйвер. Для зручного опису схеми даних існує окрема бібліотека Mongoose, що дозволяє описувати схему даних і покращує операції з даними підвищуючи безпеку.

2.4. Вибір засобу для відображення графіків

ChartJs

ChartJs – графічна бібліотека для створення графіків і діаграм, яка на основі вхідних даних будує графіки і повертає сторінку у форматі HTML. Має багато налаштувань для графіків і діаграм(розміри, колір, кількість точок). Дана бібліотека є простою у використанні, але сторінка з графіками генерується на стороні сервера, що не є дуже ефективним рішенням.

Переваги:

- простота у використанні;
- великий набір графіків;
- генерується Html сторінка з графіками.

Недоліки:

- графіки статичні;
- генерація відбувається на стороні сервера.

Grafana

Grafana – платформа з відкритим кодом, яка використовується для відображення користувацьких метрик на графіках в реальному часі. Дана платформа дозволяє користувачам створювати дашборди на яких можна розмістити різні види графіків. Garafana надає користувачеві такий функціонал:

- зміна масштабу графіків;
- сортування значень в таблицях;
- багато видів графіків;

- виділення окремим коліром графіку;
- можливість розміщувати більше одного графіку на дашборді .

Переваги:

- велика кількість налаштувань;
- відображення метрик у реальному часі;
- працює окремо від серверної частини.

Недоліки:

- складність налаштування.

Проаналізувавши дані засоби для візуалізації користувацьких метрик було вибрано платфому Grafana, так як вона надає більш широкі функціональні можливості і надає змогу відображати зміну метрик в реальному часі.

2.5. Висновки до розділу

В даному розділі було проаналізовано технології та програмні засоби, які будуть використовуватись в розроблюваному програмному забезпеченні, такі як: мови програмування, бази даних, фреймворки для платформи Node.Js і технології для відображення графіків. Серед мов програмування було проаналізовано такі мови: Java, Python, Javascript, C#; серед фреймворків для платформи Node.Js: Express.js, Koa.js, Sails.js; серед баз даних: PostgreSQL, MongoDb, MySQL; серед технологій для відображення графіків: Chart.js, Grafana. Було описано їхні основні недоліки і переваги і на їх основі вибрано ті, які будуть використовуватись в розроблюваному програмному забезпеченні.

Серед мов програмування було обрано мову програмування Javacsript, через її універсальність, наявність платформи Node.Js для написання серверної частини системи, а також наявність великої кількості зовнішніх модулів.

Серед фреймворків для платформи Node.Js було обрано Express.Js через його надійність, а також хорошу документацію з великою кількістю програмного коду.

Серед баз даних було обрано MongoDb. Вона легко масштабується, завдяки вбудованому шардингу і реплікації, а також має зручну функціональну мову запитів.

Серед засобів для відображення графіків було обрано Grafana, так як вона надає більш широкі функціональні можливості і надає змогу відображати зміну метрик в реальному часі.

3. РОЗРОБЛЕННЯ МОНІТОРИНГОВОЇ СИСТЕМИ

3.1. Загальний опис архітектури системи

Розроблюввльна моніторингова система представлена у вигляді веб-застосунку, який має клієнт-серверну архітектуру. Дана архітектура є домінуючою серед всіх видів розроблення додатків. На рис. 4 зображено архітектуру системи.

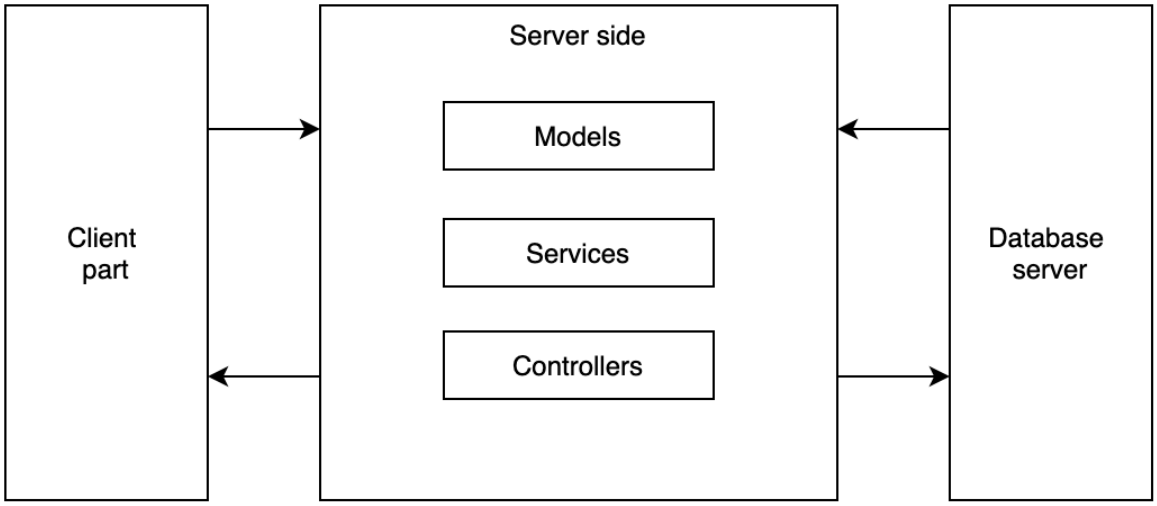


Рис. 4. Архітектура систем

Вона має високу гнучкість і дозволяє відділити клієнтську логіку, яка може виконуватись, як в браузері так і в додатку на мобільному пристрої від серверної частини

Клієнт-серверна архітектура складається з таких компонентів:

- набору серверів, які відповідають за обробку данних;
- набору клієнтів, які користуються функціями серверів;
- мережі, яка забезпечує зв’язок між клієнтами і серверами.

Всі частини в клієнт-серверній архітектурі є окремеми одиницями і функціонують незалежно один від одного. Взаємодія компонентів визначається пеаним розподілом їхніх обов’язків. Можна відокремити такі рівні операцій:

- рівень представлення даних – інтерфейс користувача, який відповідає за відображення даних користувачам;
- прикладний рівень – відповідає за логіку обробки інформації;
- рівень управління даними – надає доступ і управління даними.

Оскільки для створення серверної частини використовувався платформа NodeJs, було обрано фреймворк Express. Даний фреймворк використовує архітектурний шаблон MVC(Model-View-Controller):

- Model – представлення даних, зазвичай описується структура бази даних, відповідає за зміну даних при певній команді з контролера;
- View – відображає дані моделі користувачу;
- Controller – реагує на дії користувача і надає команду моделі про зміну даних.

На рис. 5 показано схему архітектурного шаблону MVC.

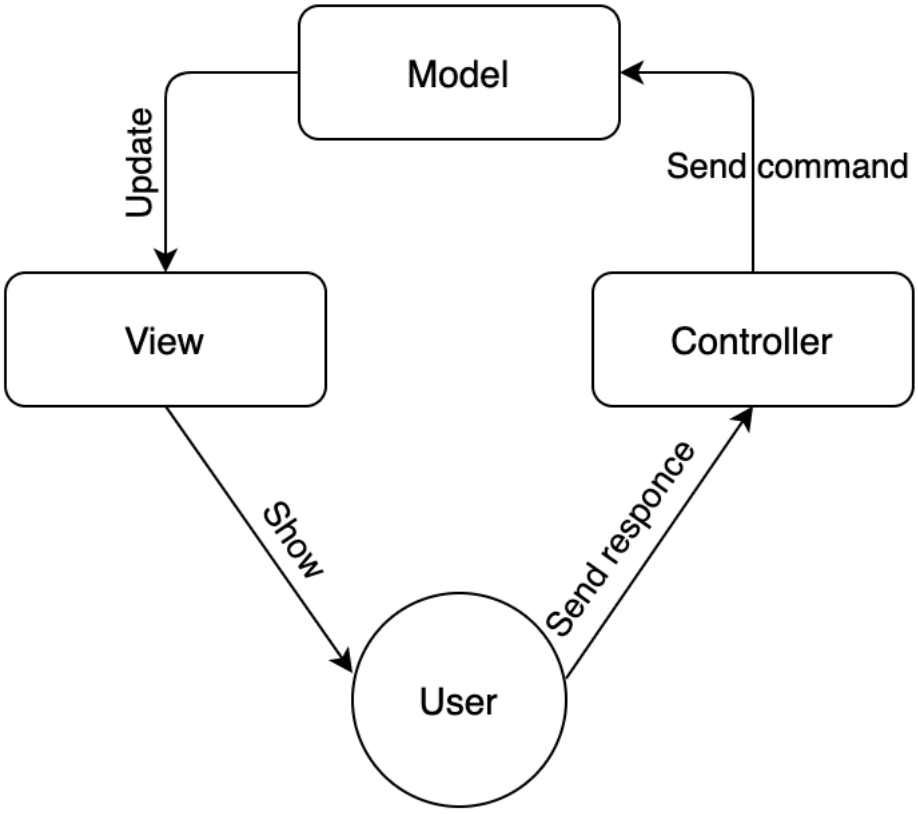


Рис. 5. Схема MVC

Серверна частина розроблювальної моніторингової системи виконує такі функції:

- обробка вхідних запитів;
- обрахунок та відображення необхідних статистичних даних:
  - мінімальне/максимальне значення для метрики;
  - середнього значення;
  - середнє квадратичне відхилення.
- збереження, фільтрування і сортування даних за часом;
- сповіщення про некоректну роботу:
  - обробка і аналіз метрик, для визначення системних збоїв і перевищення заданих значень;
  - відправка сповіщень про збої в мессенджери.
- захист від збоїв: основні статистичні дані зберігаються у файл через певний проміжок часу; відновлення системи на випадок збою;
- система надає змогу користувачам налаштовувати графіки і сповіщення;

  система має захист персональних даних користувача і використовувати алгоритми хешування для надійного захисту системи.

Клієнтська частина надає користувачам інтерфейс для перегляду їх додатків і надає доступ до сторінки перегляду статистики по конкретному додатку. Також клієнтська частина представленна у вигляді модуля, який потрібно підключити до свого додатку і налаштувати, які дані необхідно відображувати на графіках і які статистичні дані необхідно обраховувати.

Клієнтська частина розроблювальної моніторингової системи виконує такі функції:

- надає користувацький інтерфейс для управління сисьемою і відображення користувацьких даних:
  - реєстрація користувача;

- перегляд додатків, над якими ведеться моніторингу;
  доступ до дашбордів зі статистичними даними;
- внесення змін до інформації про користувача;
- управління додатками користувача.

- надає модуль для підключення до клієнтського додатку, який має свої налаштування:
  - вид метрик;
  - кількість і вид графіків на сторінці Grafana сервісу;
    збір системних метрик;
  - перевірка доступності веб-ресурсів;
  - відправка сповіщень.

На рис. 6 показано схему використання системи на якій показано взаємодію користувача з клієнтською і серверною частинами моніторингової системи.

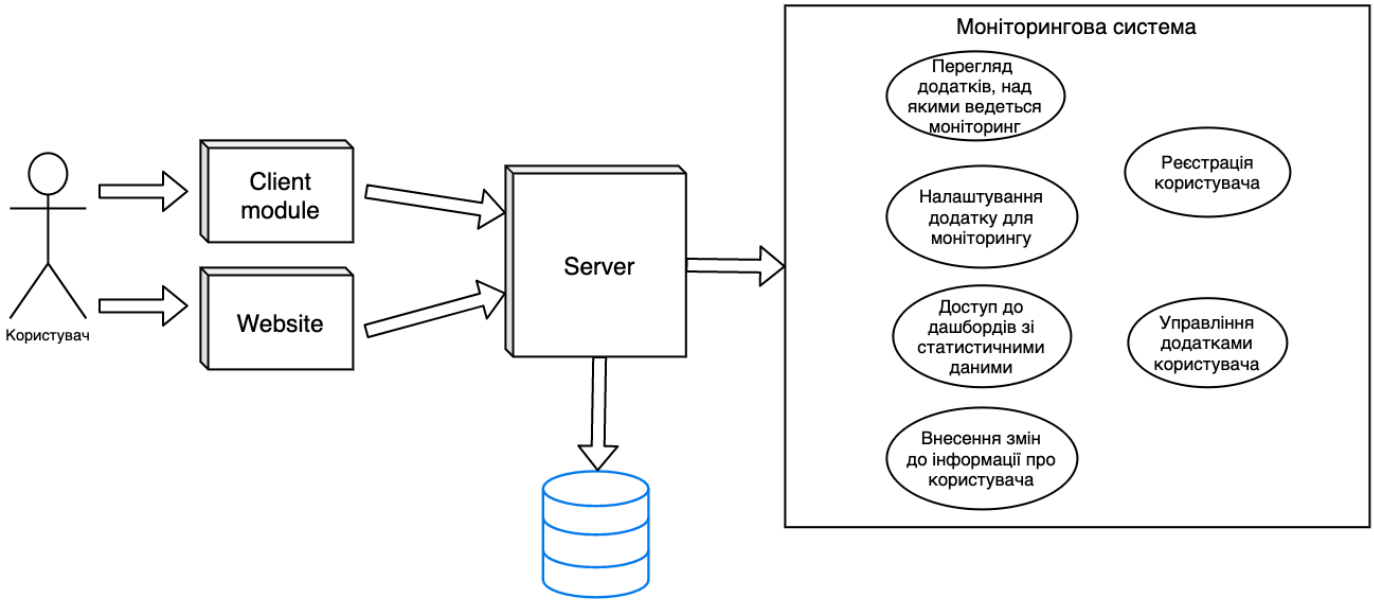


Рис. 6. Схема використання системи

3.2. Опис серверної частини

Серверна частина складається з таких сервісів:

- Profile API – мікросервіс, який відповідає за авторизацію і аутентифікацію користувачів. Для авторизації користувачів сервіс використовує JWT токени. Токен створюється на основі даних користувача і містить в собі 3 основні частини:
    - Header – містить алгоритм хешування;
    - Payload – містить основні дані про користувача;
    - Sigature – підпис згенерований на основі двох попередніх частин.

  Токен має певний час дії, після якого його потрібно оновити. Це дозволяє збільшити надійність і безпеку, тому що якщо токен буде викрадено, то через деякий час срок його дії закнічиться і він не буде валідним;
- Statsmeter API – мікросеріс, який відповідає за обробку всіх клієнтських метрик, підраховує статистичні показники по користувацьким метрикам, такі як:
    - Мінімальне значення;
    - Максимальне значеня;
    - Середнє значення;
    - Середнє квадратичне відхилення [19];
    - Кванилі [20].

  Даний мікросервіс отримує дані від користувача за допомогою вебсокетів. Дані передаються у зашифрованому вигляді, сервіс декодує, обробляє і зберігає їх для подальшого відображення на графіках. Statsmeter API містить такі сервіси:
    - AlertsService – сервіс, який відповідає за відправку повідомлень користувачам, коли якась метрика перевищила задане значення;

- EventBusService – сервіс, в який записуються нові користувацькі метрики, які потім обробляються і видаляються.
- MetricsStorageService – сервіс, який відповідає за збереження користувацьких метрик;
- ListenerService – сервіс, який відповідає за з'єднання із клієнською частиною і відповідає за відправку даних за допомогою вебсокетів.

- Cluster API – мікросервіс, який відповідає за створення клієнтських контейнерів для сервісів Grafana, Graphite і Statsmeter, які взаємодіють один з одним. Даний сервіс створює конфігурації для інших сервісів і запускає контейнери з ними за допомогою системи Kubernetes. Cluster Api містить такі сервіси:
  - ClusterService – сервіс, який відвідає за створення кластерів, надає базовий CRUD (create, read, update delete операції), а кінцеві точки для розгортання і згортання кластерів додатку;
  - ClusterHealthCheckJob – сервіс, який перевіряє доступність кластерів на платформі Kubernetes, а також звільняє ресурси, які не використовуються;
  - ConfigComposerService – сервіс, який створює файли конфігурацій для сервісів Grafana, Graphite і Statsmeter;
  - ConfigValidator – сервіс, який відповідає за валідацію конфігурацій сервісів;
  - DeploymentService – сервіс, який відповідає за розгортання і запуск користувацьких кластерів на платформі Kubernetes, а також за їх видалення;

    - DiskResizeService – сервіс, який перевіряє кількість зайнятого диску на кластерах користувача і якщо місця недостатньо, то кластеру виділяється більше місця.
- Grafana service – сервіс, який відповідає за відображення користувацьких метрик на графіках в реальному часі [21]. Дана платформа дозволяє користувачам створювати дашборди на яких можна розмістити різні види графіків. Garafana надає користувачеві такий функціонал:
    - Зміна масштабу графіків;
    - Сортування значень в таблицях;
    - Багато видів графіків;
    - Виділення окремим коліром графіку;
    - Можливість розміщувати більше одного графіку на дашборді.
- Graphite service – сервіс, який відповідає за збереження моніторингових даних від різних клієнтів у базу даних і їх відображення [22];
- Slack-bot Api – сервіс, який відповідає за відправкусповіщень користувачам в месенджер Slack.

На рис. 7 зображено сервіси з яких складається серверна частина.

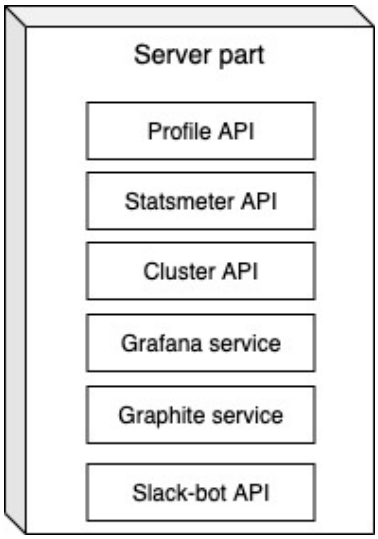


Рис. 7. Структура сервісів серверної частина

3.3. Опис клієнтської частини

Клієнтська частина моніторигової системи складається з двох частин:

- Веб інтерфейс користувача;
- Модуль (бібліотека) клієнта, який підключається до веб-додатку за яким потрібно вести моніторинг.

Веб-інтерфейс надає можливість переглядати додатки за якими ведеться моніторинг і зупиняти цей процес при необхідності.

Клієнтський модуль надає змогу пористувачам встановити спеціальний пакет до свого веб-додатку і налаштувати відправку метрик і їх тип за допомогою спеціальних налаштувань.

Таблиця 6

Налаштування системи

token	Токен авторизації
passwordProtect	Налаштування захису паролем
application	Назва додатку
flushIntervalInSeconds	Інтервал через який потрібно обновлювати значення на графіку
Retention.frequency	Частота оновлення
Retention.keep	Час зберігання метрик
admins	Масив з адміністраторами веб-додатку
plugins	Масив з плагінами. Є два плагіни system-monitor – відповідає за збір системних метрик і health-check – перевіряє доступність ресурсів

Відправка даних здійснюється за допомогою веб сокетів. Це спеціальна технологія, яка дозволяє створювати з’єднання між клієнтом і сервером для обміну повідомленнями в реальному часі. Ця технологія

використовує двоспрямований потік даних, щодозволяє не дублювати повідомлення і надає бистрий зв'язок для відправки даних в реальному часі. На рис. 8 зображено приклад відображення користувацьких метрик.

Рис. 8. Приклад відображення даних на графіках

3.4. Опис структур даних для зберігання і підрахунку статистики

Для зберігання і підрахунку статистичних даних система використовує структуру даних основану на алгоритмі FIFO (first in first out) [23]. Даний алгоритм працює по принципу черги – елементи добавляються в дану чергу і видаляються через певні проміжки часу. Дана структура даних має обмежену кількість елементів в черзі, які видаляються через певний проміжок часу.

Структура даних, яка зображена на рис. 9 має фіксований розмір і час протягом якого вона зберігає дані, час розділяється на кожну комірку і протягом певного проміжку часу статистичні дані накопичуються в певній комірці даної структури даних. Статистичні дані переобраховуються зразу при додаванні нової метрики. Існують такі стистичні показники:

- Мінімальне значення;
- Максимальне значення;

- Середнє значення;
- Кількість елементів;
- Середнє квадратичне відхилення;

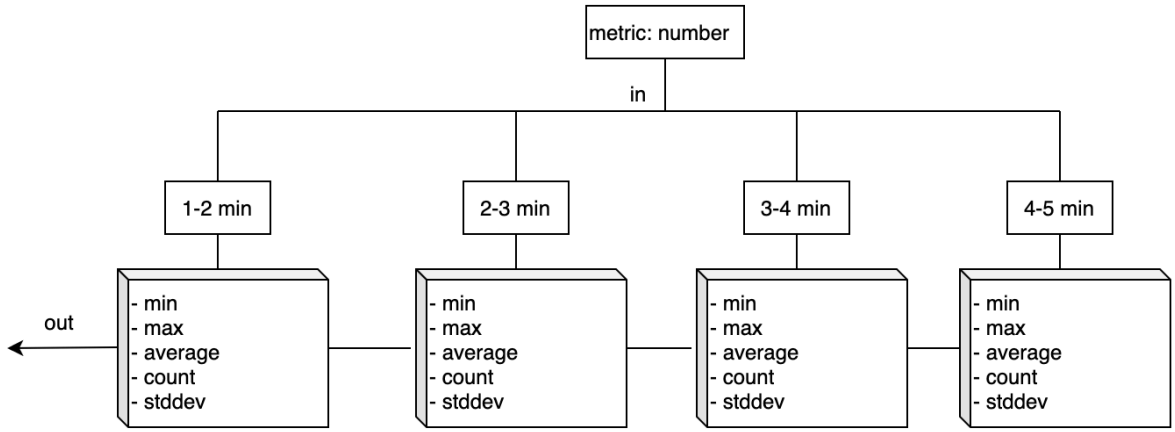


Рис. 9. Опис структури даних

Лістинг 1. Приклад коду для обчислення статистичних показників

```
_updateStatisticsOnAdd(el) {
  if (el !== undefined && el !== null) {
    this.statistics.count += 1;
    this.statistics.sum += el;
    this._updateStatisticsMinAndMaxOnAdd(el);
    this.statistics.sumOfSquares += Math.pow(el, 2);
    if (this.statistics.count > 0) {
      this.statistics.average = this.statistics.sum /
this.statistics.count;
      let difOfSums =
this._calculateDifferenceOfSums(this.statistics.sumOfSquares,
      this.statistics.sum, this.statistics.count);
      if (this.statistics.count > 1) {
        this.statistics.msdev = parseFloat(Math.sqrt(difOfSums /
this.statistics.count));
        this.statistics.stddev = parseFloat(Math.sqrt(difOfSums /
(this.statistics.count - 1)));
      } else {
        this.statistics.msdev = undefined;
        this.statistics.stddev = undefined;
      }
    }
  }
_updateStatisticsMinAndMaxOnAdd(el) {
  if (this.statistics.max < el || this.statistics.max === undefined ||
this.statistics.max === null) {
```

Продовження лістингу 1

```
    this.statistics.max = el;
  }
  if (this.statistics.min > el || this.statistics.min === undefined ||
this.statistics.min === null) {
    this.statistics.min = el;
  }
}
```

Середнє квадратичне відхилення обраховується за формулою

$$\sigma = \sqrt{\overline{x}^2 - (\overline{x})^2} = \sqrt{\frac{\sum x^2}{N} - (\frac{\sum x}{N})^2} \quad (1)$$

Дана формула дозволяє переобчислюватисереднє квадратичне відхилення на основі попередніх значень, це означає, що потрібно переобчислювати тільки суму квадратів і загальну суму.

Середнє квадратичне відхилення це статистичний показник, який показує на скільки в середньому відхиляються значення від середнього їх значення.

3.5. Опис структури бази даних

Для реалізації моніторингової системи було використано базу даних MongoDb, які використовуються системою. Також для збереження даних використовується жорсткий диск на якому зберігаються статистичні дані. Система має 5 сутностей:

- Profile
- Cluster
- Task
- Ports
- Tags

На рис. 10 зображено схему бази даних системи.

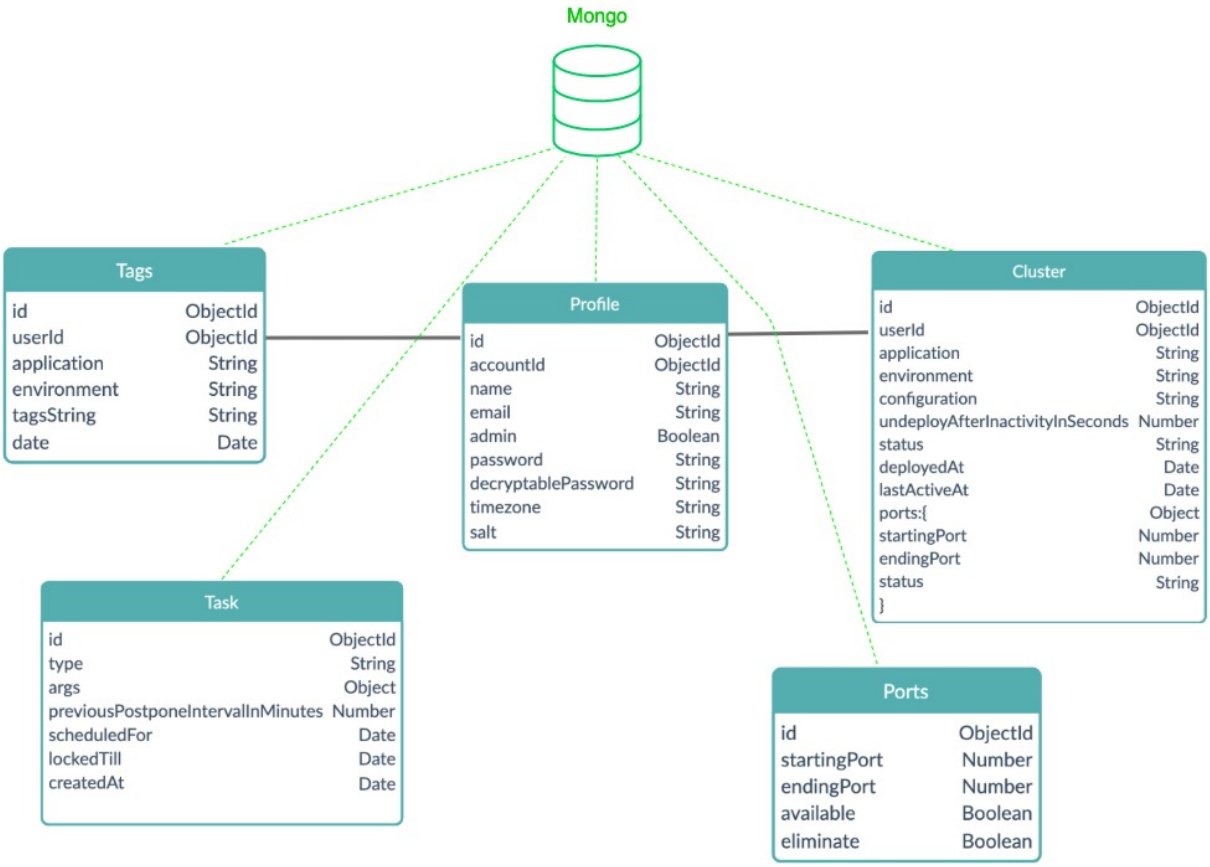


Рис. 10. Схема бази даних

Profile

Таблиця Profile зберігає інформацію про всіх користувачів системи. В таблиці 7 показано назви полів і їх типи даних.

Таблиця 7

Поля сутності Profile

Назва поля	Тип даних
id	ObjectId
accountId	ObjectId
name	String
email	String
admin	Boolean

Продовження табл. 7

password	String
timezone	String
salt	String

Cluster

Таблиця Cluster зберігає інформацію про запущенні для моніторингу користувацькі додатки. В таблиці 8 показано назви полів і їх типи даних.

Таблиця 8

Поля сутності Cluster

Назва поля		Тип даних
id		ObjectId
userId		ObjectId
application		String
environment		String
configuration		String
undeployAfterInactivityInSeconds		Number
status		String
deployedAt		Date
lastActiveAt		Date
ports		Object
startingPort	Number	
endingPort	Number	

Task

Таблиця Task зберігає інформацію про завдання, які використовуються планувальником для розгортання кластерів з користувацькими сервісами (Statsmeter, Grafana, Graphite). В таблиці 9 показано назви полів і їх типи даних.

Таблиця 9

Поля сутності Task

Назва поля	Тип даних
id	ObjectId
type	String
args	Object
previousPostponeIntervalInMinutes	Number
scheduledFor	Date
lockedTill	Date
createdAt	Date

Ports

Таблиця Ports зберігає інформацію про використанні порти користувацькими додаткам, для виділення вільних портів. В таблиці 10 показано назви полів і їх типи даних.

Таблиця 10

Поля сутності Ports

Назва поля	Тип даних
id	ObjectId
startingPort	Number

Продовження табл. 10

endingPort	Number
available	Boolean
eliminate	Boolean

Tags

Таблиця Tags зберігає інформацію про використанні теги користувацьких додатків. В таблиці 11 показано назви полів і їх типи даних.

Таблиця 11

Поля сутності Tags

Назва поля	Тип даних
id	ObjectId
userId	ObjectId
application	String
environment	String
tagsString	String
date	Date

3.6. Висновки розділу

В цьому розділі було розглянуто архітектуру розроблюваної моніторингової системи і створено схему використання і структурну схему системи. При описі архітектури системи було розглянуто архітектурну модель MVC, яка використовується в системі.

Перевагами даної моделі є:

- єдина концепція системи;
- незалежність елементів один від одного;

- легко протестувати окремі компоненти.

Також було показано структуру даних, яка використовується в системі для обрахунку статистичних показників і базується на принципі FIFO (first in first out). Перевагою даної структури даних є те, що вона використовує формули для обрахунку статистичних показників на основі попередніх обрахунків. Було показано схему бази даних на якій зображено всі сутності, які наявні в ситемі і описано тип даних для кожного поля сутності.

4. АНАЛІЗ РОЗРОБЛЕНОЇ МОНІТОРИНГОВОЇ СИСТЕМИ

4.1. Аналіз реалізованої моніторингової системи

Розроблена моніторингова система призначена для відслідковування користувацьких і системих метрик і надання зручного інтерфейсу системним адміністраторам для їх відслідковування. Вона надає такий функціонал своїм користувачам:

- можливість зареєструватись в системі. Реєстрація нового користувача відбувається на серверній частині:
  - Перевіряється, чи такий користувач ще не зареєстрований в системі;
  - Використовується алгоритм SHA512 для хешування паролю;
  - Всі дані користувача із захешованим паролем зберігаються в базу даних;
  - За допомогою модуля jsonwebtoken створюється JWT токен, який відправляється користувачу і він його використовує в запитах до системи.
- налаштування типу графіків, які будуть відображатись користувачеві. Налаштування кількості і типу графіків в системі реалізовано за допомогою створення файлів конфігурацій в яких описано, які саме графіки і метрики повині бути на сторінці і їх кількість.

  Лістинг 2. Приклад файлу конфігурації

```
{
  "dashboards": [
    {
      "id": "system-metrics",
      "name": "System metrics",
      "tags": [["region"], ["region",
      "rows": [
        {
          "charts": [
          "name": "Sign in attempts",
```

Продовження лістингу 2.

```
              "metrics": [
                "auth.sign-in-attempts"
              ]
            },
            {
              "name": "# of active users",
              "metrics": [
                "app.active-users"
              ]
            }
          ]
        }
      ]
    }
  ]
}
```

- налаштування метрик, які будуть відслідковуватись. Існує 5 видів метрик:
    - Gauge – вид метрики, який показує поточне значення метрики;
    - Histogram – вид метрики, який показує скільки значень потрапляє у конкретні інтервали значень протягом конкретного часу;
    - Counter – вид метрики, який показує кількість певних метрик;
    - Statistic – вид метрики, який показує на графіку зміну статистичних показники таких, як:
        - Min – мінмальне значення;
        - Max – максимальне значення;
        - Median – статистична медіана, характеристика розподілу випадкових велечин;
        - Stddev – стандартне відхилення;
        - Avg – середнє значення;
        - P90, p70 – квантилі;

Лістинг 3. Налаштування метрики

```
{
  "metrics": {
    "app.active-users": {
      "event": "app.active-users",
      "type": "gauge",
      "flushIntervalInSeconds": 5,
      "retention":
        {
          "frequency": "10s",
          "keep": "2d"
        }
      ]
    }
  }
}
```

- налаштування сповіщень. Відбувається при створенні метрики.

Лістинг 4. Налаштування сповіщень

```
"auth.sign-in.attemps.overflow": {
  "metric": {
    "metric": "auth.sign-in-attempts",
    "interval": "10m”
  },
  "tags": [[], ["region", "*"]],
  "threshold": 100,
  "condition": "gt",
  "unhealthyDelay": "2m",
  "healthyDelay": "1m",
  "admins": ["developer"],
  "channels": ["slack"],
  "message": "Too many sign in attempts",
  "repeatInterval": "1m"
}
```

Таблиця 12

Пояснення полів метрики

Назва	Пояснення
metric	Назва метрики
tags	Масив тегів для яких налаштовувати сповіщення
threshold	Значення метрики при якому потрібно відправляти сповіщення про її перевищення
condition	“gt” чи “lt” – значення має вижче чи нижче зазначеної метрики

Продовження табл. 12

unhealthyDelay	Час протягом якого значення метрики повино бути в зоні перевищення
healthyDelay	Час протягом якого значення метрики повино бути в допустимих межах для видалення сповіщення
admins	Адміністратори, яким потрібно надсилати сповіщення
channels	Канали в які потрібно відправляти сповіщення
message	Текст повідомлення
repeatInterval	Інтервал протягом якого потрібно повторно надіслати сповіщення

- налаштування плагінів для збору системних метрик і перевірки доступності ресурсів.

Лістинг 5. Налаштування плагінів

```
"plugins": [
   {
     "name": "system-monitor",
     "settings": {
     "enableAlerts": true,
       "admins": ["developer"],
       "channels": ["slack"]
     }
   }, {
     "name": "healthcheck",
     "settings": {
       "healthChecks": [
         {
           "name": "google",
           "url": "https://www.google.com/"
         },
         {
           "name": "google1",
           "url": "https://www.google1.com/"
         }
       ]
     }
  }]
```

Система складається з 2 частин клієнтської і серверної, які є взаємозв'язаними. На рис. 11 зображено структурну схему системи.

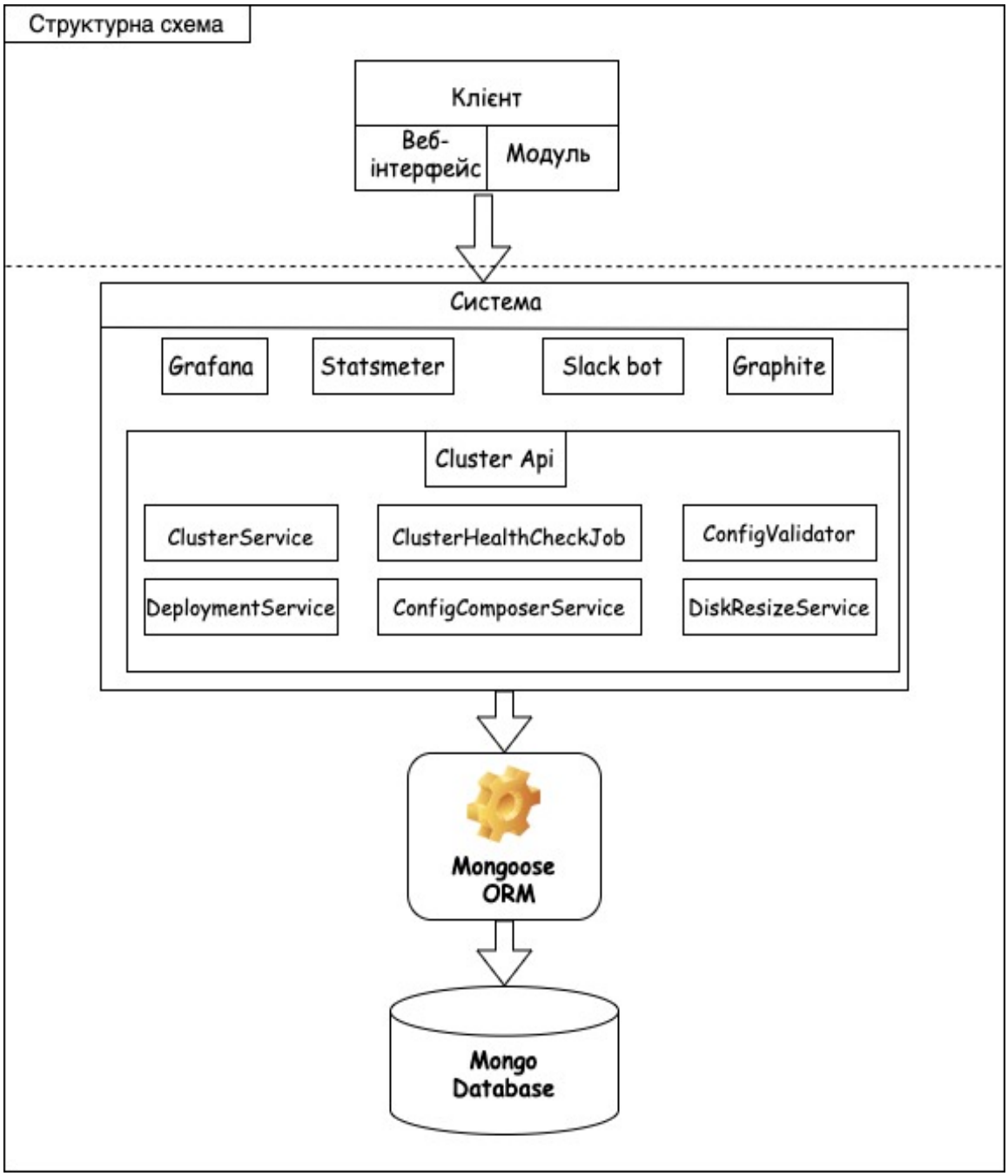


Рис. 11. Структурна схема

4.2. Дизайн та вміст веб-сторінок

Щоб створити дизайн і вміст сторінок системи було використано такі технології:

- HTML;
- CSS;
- Javascript.

Для створення клієнтської части було застосовано фреймворк Angular, який надає весь необхідний функціонал для створення веб-сайту. Також для створення адапнивного дизайну сайту було використано Angular Material і Bootstrap 4, що дозволяє адаптувати веб-сторінки до різних пристроїв.

Головною сторінкою в системі є сторінка перегляду веб-додатків над якими ведеться моніторинг. На цій сторінці користувач може бачити список своїх додатків і перейти на сторінку з графіками на яких відображається зміна користувацьких метрик. Також в користувача є можливість скопіювати токен доступу, який потрібно розмістити в файлі конфігурацій додатку над яким ведеться моніторинг, для подальшої автентифікації користувача. На рис. 12 зображено головну сторінку системи.

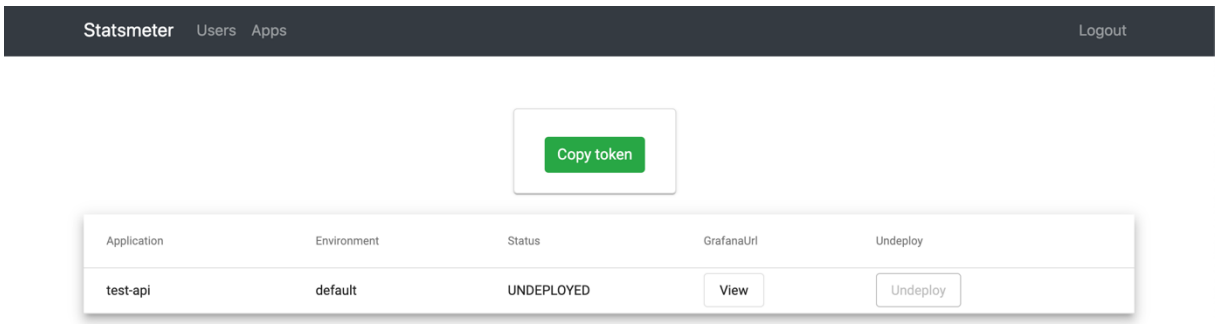


Рис. 12. Вигляд сторінки з користувацькими додатками

На рис. 13 зображено форму для входу в систему. Користувач повинен ввести свій логін і пароль для входу в систему. Якщо дані не вірні на сторінці з’явиться сповіщення з цією інформацією. Якщо введені дані вірні,

то користувача перенаправить на сторінку з додатками над якими ведеться моніторинг.

Рис. 13. Форма входу в систему

Коли користувач увійшов в систему у нього є можливість переглянути свої додатки над якими ведеться моніторинг, перейти на сторінку з графіками, на якій будуть показанні користувацькі метрики на графіках, зупинити моніторинг і внести зміни до своїх даних на персональній сторінці.

При переході на сторінку з графіками відкриється сторінка в сервісі Grafana, яка містить всі налаштовані графіки користувача. На графіках відображаються всі метрики в реальному часі, на графіку можна виділити проміжок часу на якому потрібно більш детально розглянути зміну метрик чи змінити його розмір в налаштуваннях. Також в користувачів системи є можливість добавити потрібні графіки на свій сайт чи на сторінку адміністратора для зручнішого моніторингу і доступу до моніторингових даних. Для цього потрібно скопіювати необхідний елемент в налаштуваннях певного графіку.

На рис. 14 і рис. 15 зображенно вигляд сторінок із відображенням графіків із системними і користувацькими метриками.

Рис. 14. Сторінка відображення системних метрик

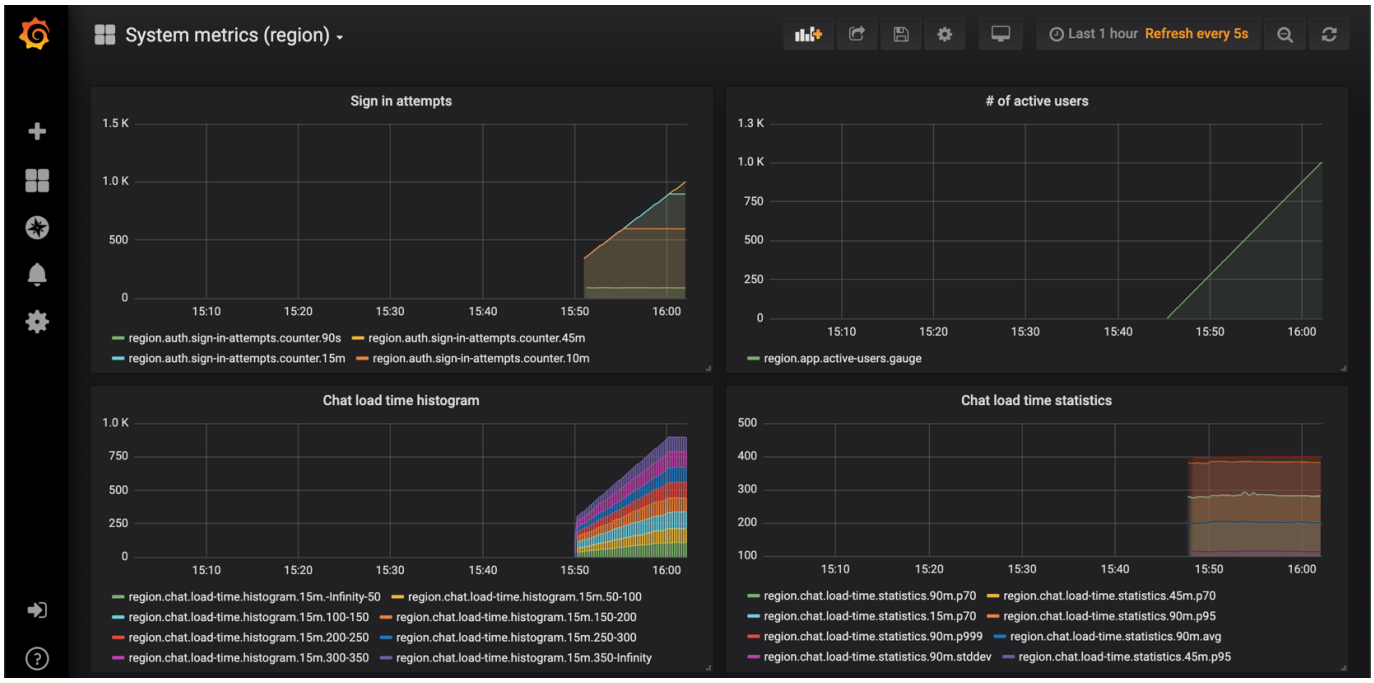


Рис. 15. Сторінка відображення користувацьких метрик

4.3. Тестування системи

Одним з факторів ефективної моніторингової системи є показник кількості оброблених користувацьких метрик. Протестуємо систему і створену структуру даних тестом на продуктивність (performance testing),

щоб дізнатись кількість метрик, яку може оброблювати система. Данний вид тестування дозволяє протистувати продуктивність системи і визначити кількість даних, які може оброблюватись системою.

Для даного тесту було створено файл із 1000000 числових записів. Дані з цього файлу були прочитанні і поступово добавлялись в створену структуру даних, яка обраховує користувацькі метрики і виводились проміжні логи при оброкі кожних 100000 метрик. На рис. 16 показано результати тесту на продуктивність обробки даних.

Рис. 16. Результати тесту на продуктивність обробки даних

З результатів тестування на продуктивність структури даних бачимо, що для обробки 1000000 записів було витрачено 1367 мілісекунд, тобто в секунду структура даних може обробити 731528 записів. Даний показник показує, що система може справитись з значними навантаженнями.

Щоб дізнатись скільки метрик може обробити система, якщо дані метрикик надсилає користувач. Було створено інтеграційний тест в якому створюється клієнт, який надсилає користувацькі метрики за допомогою веб-сокетів. На рис. 17 показано результати тесту на продуктивність системи.

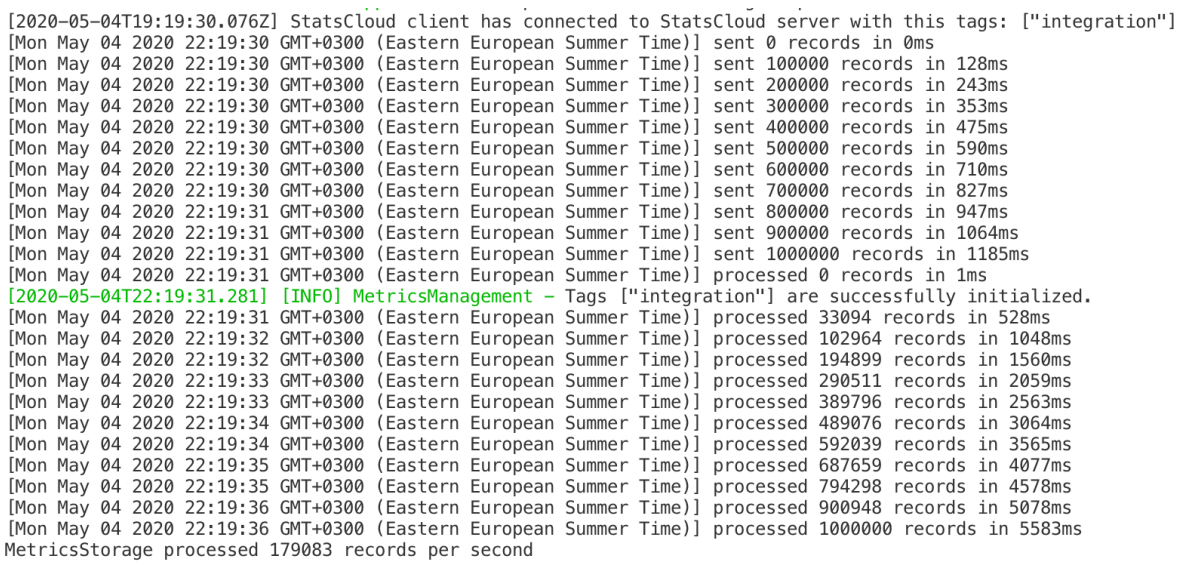


Рис. 17. Результати теста на продуктивність системи

З результатів тестування на продуктивність структури даних бачимо, що для відпраки і обробки 1000000 записів було витрачено 5583 мілісекунд, тобто в секунду система може обробити 179083 записів. Даний показник показує, що система може справитись з значними навантаженнями при отримані і відправці метрик.

Також для основних модулів системи написані юніт тести. Юніт тестування дозволяє перевірити передбачену поведінку окремого модуля і впевнетись, що всі його функції вірно працюють з різними наборами даних. На рис. 18 показано результати юніт тестів модуля, який оброблює користувацькі метрики.

```
  MetricsStorage
[2020-05-05T13:22:03.498] [INFO] Application - Express server listening on port 3030
[2020-05-05T13:22:03.502] [INFO] MetricsManagement - Tags ["region","server"] are successfully initialized.
    ✓ should return stats by metric name
    ✓ should return stats by metric name and tags
    Counters
      ✓ should return initial zero values
      ✓ should update reservoir and return correct value
      ✓ should update reservoir after interval pass when getting stats
      ✓ should update reservoir after interval pass when event fired
      ✓ should evict old metric from reservoir after interval passed
    Gauges
      ✓ should update gauge value
      ✓ should return metric stats
    Histogram
      ✓ should return initial zero values
      ✓ should update reservoir for left interval on getting stats if time interval passed
      ✓ should update reservoir for middle interval on getting stats if time interval passed
      ✓ should update reservoir for right interval on getting stats if time interval passed
      ✓ should update reservoir on event fire if interval passed
      ✓ should update reservoir if time passed
      ✓ should evict old metric from reservoir after interval passed
    Statistics
      ✓ should calculate statistics (41ms)
      Percentile
        ✓ should not return percentiles data before any data arrived
        ✓ should calculate percentiles just after new data arrived
        ✓ should calculate percentile of single signed of integer part
        ✓ should calculate percentile of percentile value with trailing zeros
        ✓ should calculate percentiles of two-sined fractional part percentile
        ✓ should calculate percentiles of three-sined fractional part percentile
        ✓ should calculate percentiles correctly
        ✓ should return percentiles for actual records only
      Median
        ✓ should return no median value before any data arrived
        ✓ should calculate medians just after new data arrived
        ✓ should return median for actual records only
        ✓ should return center reservoir element value as median if reservoir length is odd
        ✓ should return average of two central reservoir elements if reservoir length is even
      Min
        ✓ should return no min value before any data arrived
        ✓ should calculate min just after new data arrived
        ✓ should return min for actual records only
        ✓ should calculate min of negative elements
        ✓ should return min value for reservoir of one element that eqls to that element
      Max
        ✓ should return no max value before any data arrived
        ✓ should calculate max just after new data arrived
        ✓ should return max for actual records only
        ✓ should calculate max element
        ✓ should calculate max element of negative digits correctly
        ✓ should return max value for reservoir of one element that eqls to that element
      Standard deviation
        ✓ should return no max value before any data arrived
        ✓ should return no standard deviation result for reservoir of 1 element
        ✓ should return standard deviation for actual records only
        ✓ should calculate standard deviation just after new data arrived
      Average
        ✓ should return no average value before any data arrived
        ✓ should calculate average just after new data arrived
        ✓ should return average for actual records only
        ✓ should calculate average value
    Store reservoir
[2020-05-05T13:22:27.655] [WARN] MetricsStorage - Reservoirs directory does not exist. Creating...
[2020-05-05T13:22:27.655] [DEBUG] MetricsStorage - Recorded reservoirs for 104 metrics to persistent storage in 0 ms
      ✓ should save gauge metric (105ms)
[2020-05-05T13:22:03.763] [WARN] MetricsStorage - Reservoirs directory does not exist. Creating...
[2020-05-05T13:22:03.763] [DEBUG] MetricsStorage - Recorded reservoirs for 104 metrics to persistent storage in 0 ms
      ✓ should create reservoirs directory if it doesn't exist
[2020-05-05T13:23:15.791] [WARN] MetricsStorage - Reservoirs directory does not exist. Creating...
[2020-05-05T13:23:15.791] [DEBUG] MetricsStorage - Recorded reservoirs for 104 metrics to persistent storage in 0 ms
      ✓ should save counter metrics (138ms)
[2020-05-05T13:23:03.925] [WARN] MetricsStorage - Reservoirs directory does not exist. Creating...
[2020-05-05T13:23:03.925] [DEBUG] MetricsStorage - Recorded reservoirs for 104 metrics to persistent storage in 0 ms
      ✓ should save histogram metrics (132ms)
[2020-05-05T13:22:28.060] [WARN] MetricsStorage - Reservoirs directory does not exist. Creating...
[2020-05-05T13:22:28.060] [DEBUG] MetricsStorage - Recorded reservoirs for 104 metrics to persistent storage in 0 ms
      ✓ should save statistic metrics (65ms)
    Restore reservoir
[2020-05-05T13:22:04.129] [INFO] MetricsManagement - Metrics are restored from persistent storage
      ✓ should restore gauge metric
[2020-05-05T13:22:04.132] [INFO] MetricsManagement - Metrics are restored from persistent storage
      ✓ should restore counter metrics
[2020-05-05T13:22:04.137] [INFO] MetricsManagement - Metrics are restored from persistent storage
      ✓ should not restore counter metric which is not present in config
[2020-05-05T13:22:04.140] [INFO] MetricsManagement - Metrics are restored from persistent storage
      ✓ should restore histogram metrics reservoir
[2020-05-05T13:22:04.152] [INFO] MetricsManagement - Metrics are restored from persistent storage
      ✓ should not restore histogram metric which is not present in config
[2020-05-05T13:22:04.154] [INFO] MetricsManagement - Metrics are restored from persistent storage
      ✓ should restore statistic metrics reservoir
[2020-05-05T13:22:04.158] [INFO] MetricsManagement - Metrics are restored from persistent storage
      ✓ should not restore statistic metric which is not present in config
```

Рис. 18. Приклад результатів модульних тестів

4.4. Рекомендації щодо вдосконалення

Щодо подальшого вдосконалення моніторингової системи можна виділити наступні пункти:

- створити модуль які буде оброблювати статистичні дані за допомогою AI (Artetificial intelligence [24]) і робити передбачення на основі статистичних даних;
- добавити можливість інтегрувати всі графіки автоматично на сайт користувача системи;
- створити cron job [25], який буде періодично перевіряти чи запущена моніторингова система і якщо виникла помилка, то буде її перезапускати;
- добавити comandline interface для зручного налаштування системи і створення всіх необхідних файлів конфігурації.

4.5. Висновки до розділу

В цьому розділі було проаналізовано розроблену моніторингову систему. Було проаналізовано весь функціонал створеної моніторингової системи і показано налаштування за допомогою яких користувач може налаштувати систему для свого користування. Також було описано, які функціональні можливості потрібно буде добавити до системи.

Важливим пунктом в даному розділі є пункт “Тестування системи”. В даному пункті були описані юніт тести і тести на продуктивність системи. За результатами даних тестів видно, що система добре протестована і може витримати значні навантаження. Велика кількість користувацьких метрик, які потрібно обробити не буде проблемою для створеної моніторингової системи.

ВИСНОВКИ

Основною метою даного дипломного проєкту було створення моніторингової системи, яка буде відображати і обчислювала статистику по користувацьким метрикам в реальному часі і відправляти сповіщення при винекнення збоїв в додатках.

Було проведено аналіз проблем в існуючих системах моніторингу веб-додатків і було створенно функціональні та нефункціональні вимоги до розроблюваної моніторингової системи. На основі вимог до даної системи та аналізу існуючих програмних рішень для розробки було обрано основні засоби реалізації, а саме:

- платформа Node.js для створення серверної частини системи;
- база даних MongoDb;
- Grafana – сервіс для візуалізації графіків.

Користувачі розробленої моніторингової системи мають можливість переглядати всю необхідну статистичну інформацію для тестування та визначення найпопулярніших ресурсів, а також переглядати навантаження на сервер, на якому запускається їх додаток. За допомогою системних показників можна визначати і прогнозувати, які системні ресурси потрібні для коректної роботи веб-додатку.

Розроблена моніторингова система надає такі можливості:

- зберігання та відображення користувацьких метрик;
- обрахунок та відображення необхідних статистичних даних (мінімальних/максимальних значень, середнього значення і середнього квадратичного відхилення);
- відправка сповіщень в месенджер, якщо якась метрика перевищила задане значення;
- відображення системних метрик, використання (CPU, RAM і ROM) на фізичній машині на якій запущено веб-додаток;
- перегляд графіків за певний проміжок часу.

Моніторингова система була протестована на продуктивність. За результатами даних тестів видно, що система може витримати значні навантаження. Велика кількість користувацьких метрик, які потрібно обробити не буде проблемою для створеної моніторингової системи.

Щоб вдосконалити дану моніторингову систему, необхідно створити модуль які буде оброблювати статистичні дані за допомогою AI і робити передбачення на основі статистичних даних;

Моніторингова система була створена відповідно до сформованих вимог і є універсальним рішенням для підтримки і адміністрування веб-додатків.

СПИСОК ВИКОРИСТАНИХ ЛІТЕРАТУРНИХ ДЖЕРЕЛ

1. Проєктування системи моніторингу [Електронний ресурс]. — Режим доступу до ресурсу: http://lib.iitta.gov.ua/4864/1/%D0%9Fi%D0%B4%D1%85i%D0%B4_%D0%BF%D1%80%D0%BE%D0%B5%D0%BA%D1%82%D1%83%D0%B2%D0%B0%D0%BD%D0%BD%D1%8F_%D0%B4%D0%BE_%D1%81%D0%B8%D1%81%D1%82%D0%B5%D0%BC%D0%B8_%D0%BC%D0%BE%D0%BDi%D1%82%D0%BE%D1%80%D0%B8%D0%BD%D0%B3%D1%83.pdf
2. Оптимізація баз даних [Електронний ресурс]. — Режим доступу до ресурсу: https://habr.com/ru/post/349910/
3. CPU [Електронний ресурс]. — Режим доступу до ресурсу: https://sohabr.net/habr/post/331004/
4. RAM [Електронний ресурс]. — Режим доступу до ресурсу: https://uk.wikipedia.org/wiki/%D0%9F%D0%B0%D0%BC%27%D1%8F%D1%82%D1%8C_%D0%B7_%D0%B4%D0%BE%D0%B2%D1%96%D0%BB%D1%8C%D0%BD%D0%B8%D0%BC_%D0%B4%D0%BE%D1%81%D1%82%D1%83%D0%BF%D0%BE%D0%BC
5. ROM [Електронний ресурс]. — Режим доступу до ресурсу: https://uk.wikipedia.org/wiki/%D0%9F%D0%BE%D1%81%D1%82%D1%96%D0%B9%D0%BD%D0%B8%D0%B9_%D0%B7%D0%B0%D0%BF%D0%B0%D0%BC%27%D1%8F%D1%82%D0%BE%D0%B2%D1%83%D0%B2%D0%B0%D1%87
6. Scott G. Chaplowe & J. Bradley Cousins. (2016).Monitoring and Evaluation Training: A Systematic Approach.Thousand Oaks, CA: Sage. 464
7. Ивченко Григорий Иванович, Медведев Юрий Иванович Математическая статистика: Учебник. — М.: Книжный дом «ЛИБРОКОМ», 2014. — 352 с.
8. Germano, B.P., S.A. Cesar and G. Ricci. 2007. Enhancing Management Effectiveness of Marine Protected Areas: A Guidebook for Monitoring and

Evaluation. Marine Laboratory, Institute of Tropical Ecology, Leyte State University, Visca, Baybay, Leyte 6521-A, Philippines.

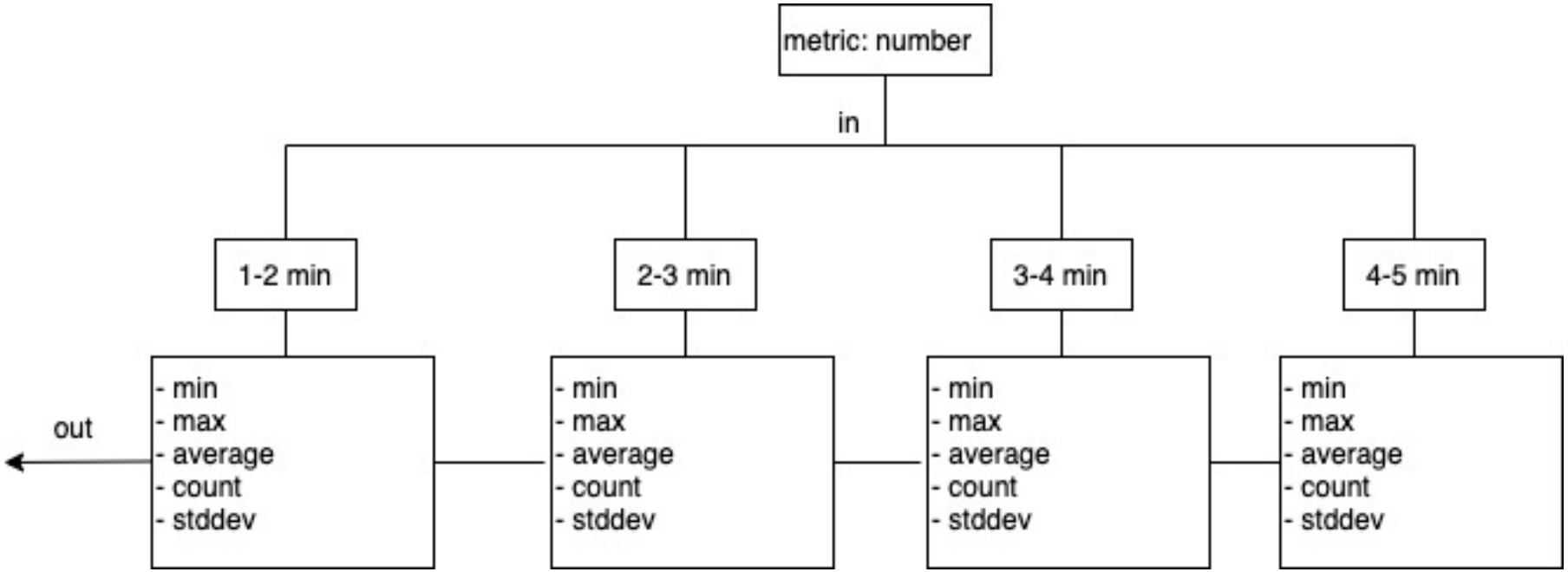
9. Проблема швидкості завантаження веб-ресурсу [Електронний ресурс]. — Режим доступу до ресурсу: - https://netpeak.net/ru/blog/sayt-zagruzhayetsya-medlenno-kogda-ne-nuzhno-panikovat/
10. Оптимізація баз даних [Електронний ресурс]. — Режим доступу до ресурсу: https://habr.com/ru/post/349910/
11. Проблема обробки великої кількості даних [Електронний ресурс]. — Режим доступу до ресурсу: https://www.jetinfo.ru/bolshie-dannye-bolshaya-problema/
12. Проблема швидкості загрузки веб-ресурсу [Електронний ресурс]. — Режим доступу до ресурсу: - https://netpeak.net/ru/blog/sayt-zagruzhayetsya-medlenno-kogda-ne-nuzhno-panikovat/
13. Балансування навантаження на сервері [Електронний ресурс]. — Режим доступу до ресурсу: https://ru.wikipedia.org/wiki/%D0%91%D0%B0%D0%BB%D0%B0%D0%BD%D1%81%D0%B8%D1%80%D0%BE%D0%B2%D0%BA%D0%B0_%D0%BD%D0%B0%D0%B3%D1%80%D1%83%D0%B7%D0%BA%D0%B8
14. Проблеми налаштування серверу [Електронний ресурс]. — Режим доступу до ресурсу: https://www.faq.in.ua/articles/26-problemy-z-serverom-shcho-robyty.html
15. Мова програмування Java - [Електронний ресурс]. — Режим доступу до ресурсу: https://uk.wikipedia.org/wiki/Java
16. Мова програмування Javascript - [Електронний ресурс]. — Режим доступу до ресурсу: https://uk.wikipedia.org/wiki/JavaScript
17. Мова програмування C# - [Електронний ресурс]. — Режим доступу до ресурсу: https://uk.wikipedia.org/wiki/C_Sharp
18. Порівняння фреймворків для платформи Node.JS - [Електронний ресурс]. — Режим доступу до ресурсу: https://umbrellait.com/ru/blog/choosing-the-best-nodejs-framework/

19. Бобик О. І., Берегова Г. І., Копитко Б. І. Теорія ймовірностей і математична статистика: Навч. підручник. 2006. – 440 с
20. Чернова Н. И. Математическая статистика: Учеб. пособие / Новосиб. гос. ун-т. Новосибирск, 2007. 148 с
21. Grafana - Електронний ресурс]. — Режим доступу до ресурсу: https://en.wikipedia.org/wiki/Grafana
22. Graphite - Електронний ресурс]. — Режим доступу до ресурсу: https://en.wikipedia.org/wiki/Graphite_(software)
23. John Bullinaria, Data Structures and Algorithms, UK Version of 27 March 2019, CA: 656 pages
24. AI - [Електронний ресурс]. — Режим доступу до ресурсу: https://en.wikipedia.org/wiki/Artificial_intelligence
25. Cron - [Електронний ресурс]. — Режим доступу до ресурсу: https://en.wikipedia.org/wiki/Cron

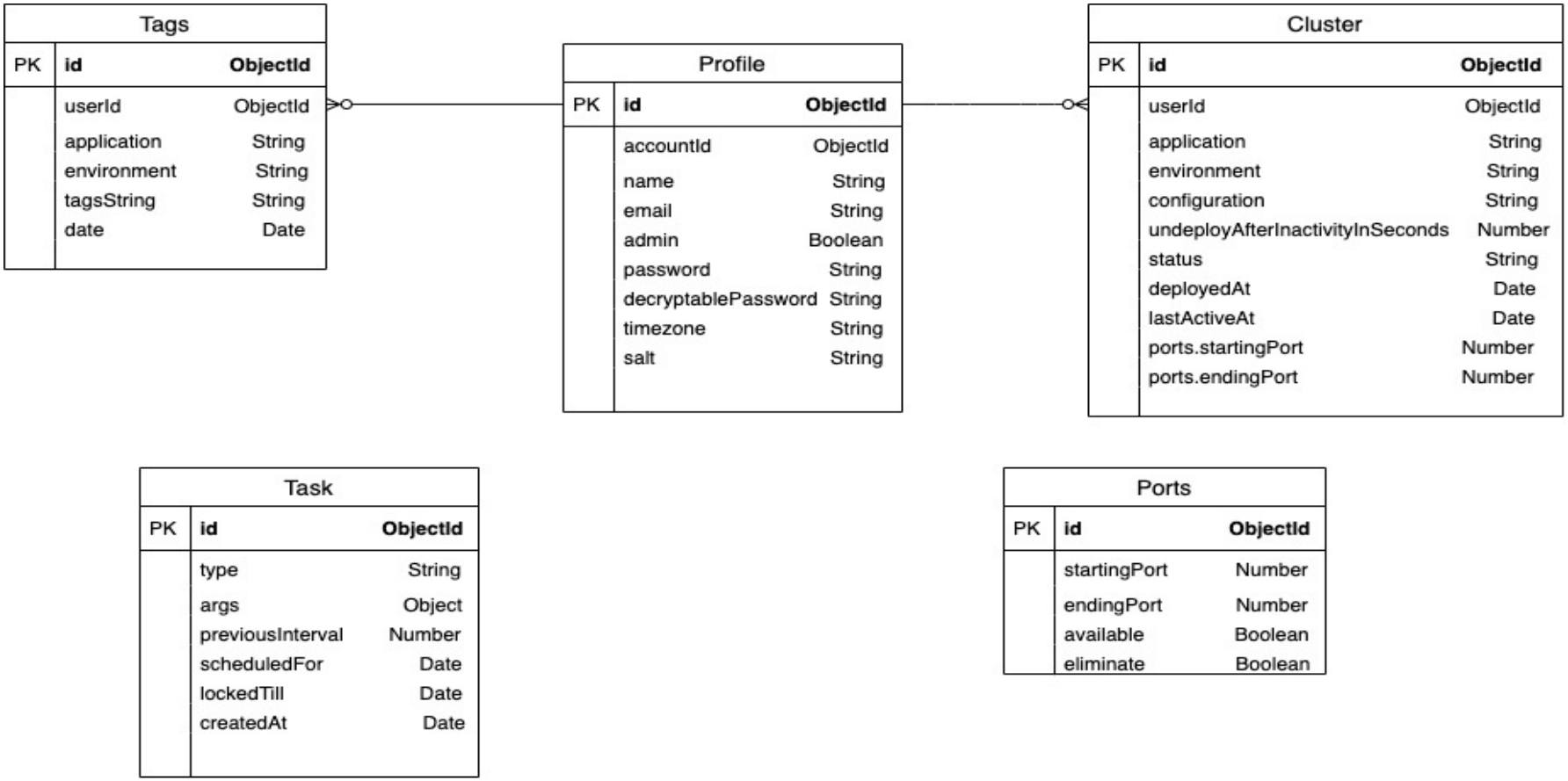
ДОДАТКИ

Додаток 1

Копії графічних матеріалів

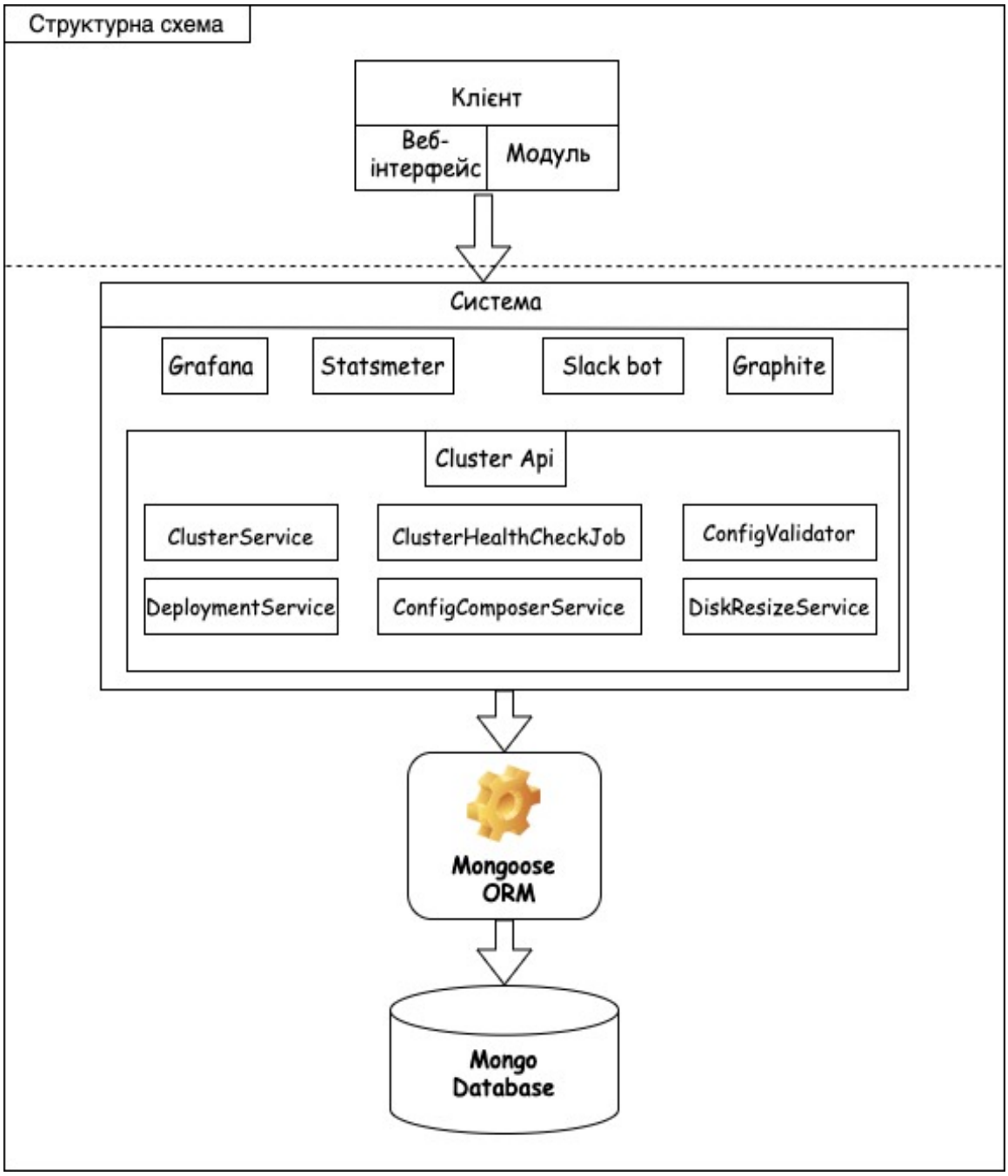


ДП.045440-06-99. Програмна система моніторингу веб-додатків в реальному часі. Додавання нової метрики. Схема додавання метрики



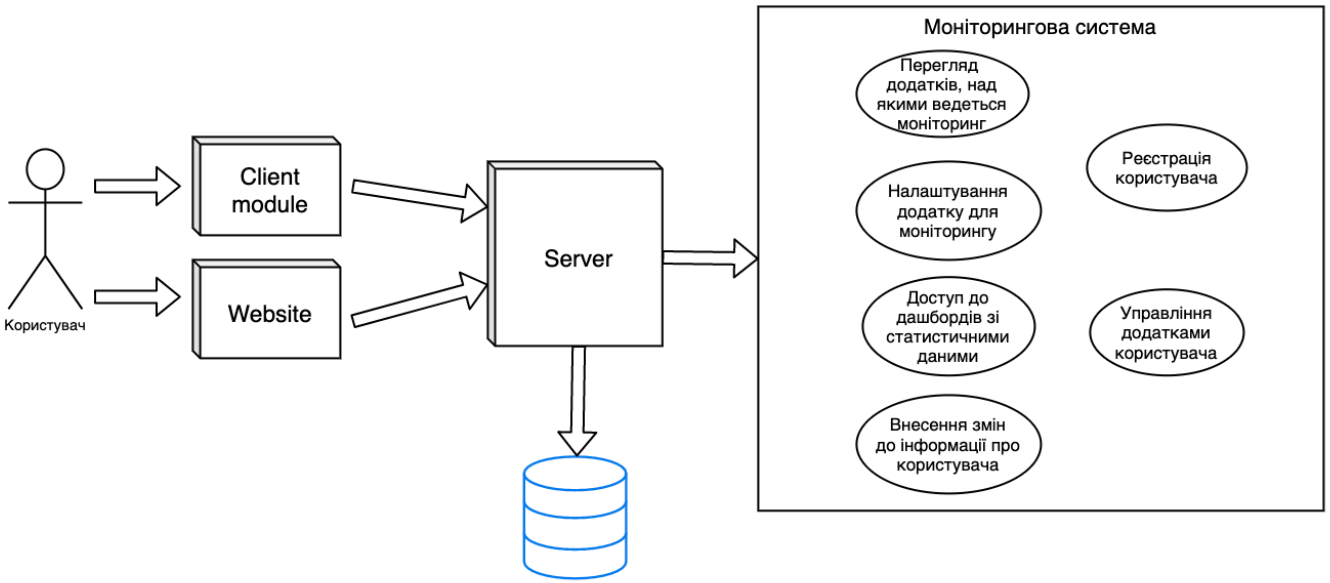
ДП.045440-07-99. Програмна система моніторингу веб-додатків в реальному часі. Структура бази даних. ERD-діаграма

СТРУКТУРНА СХЕМА ПРОГРАМНОЇ СИСТЕМИ МОНІТОРИНГУ ВЕБ-ДОДАТКІВ В РЕАЛЬНОМУ ЧАСІ



Щербина Вадим Олегович,
гр. КП-61

СХЕМА ВИКОРИСТАННЯ ПРОГРАМНОЇ СИСТЕМИ МОНІТОРИНГУ ВЕБ-ДОДАТКІВ В РЕАЛЬНОМУ ЧАСІ



Щербина Вадим Олегович,
гр. КП-61

Додаток 2

Копія презентації

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
“КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ ІМЕНІ ІГОРЯ СІКОРСЬКОГО”

ФАКУЛЬТЕТ ПРИКЛАДНОЇ МАТЕМАТИКИ

КАФЕДРА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ КОМП’ЮТЕРНИХ СИСТЕМ

Програмна система моніторингу веб-додатків в реальному часі

Виконав: Щербина Вадима Олегович

Керівник: Ст. викладач кафедри ПЗКС, к.т.н. Люшенко Леся Анатоліївна

Київ – 2020

Постановка задачі

Мета проєкту: розробити програмну систему моніторингу веб-додатків в реальному часі

Завдання:

1. Проаналізувати існуючі аналоги моніторингових систем
2. Обрати засоби для реалізації ПЗ
3. Розробити моніторингову систему
4. Протестувати систему
5. Сформулювати рекомендації, щодо подальшого вдосконалення

Актуальність

- Велика кількість веб-додатків
- Значне навантаження на адміністраторів
- Перебої в роботі веб-додатків

Вимоги до системи

- Можливість відправляти сповіщення в месенджери (Slack та RocketChat)
- Підрахунок статистичних показників по користувацьким метрикам
- Відображення системних метрик (System load, CPU, Memory, Disk, Network, Battery)

Порівняння з аналогами

	SmartBear	Dotcom-monitor	vRealize Hyperic
Підрахунок статистичних даних	–	–	–
Перевірка доступності веб-додатку	+	+	+
Сповіщення користувачів в месенджарах	–	+	–
Аналіз та обробка моніторингових даних	+	+	+

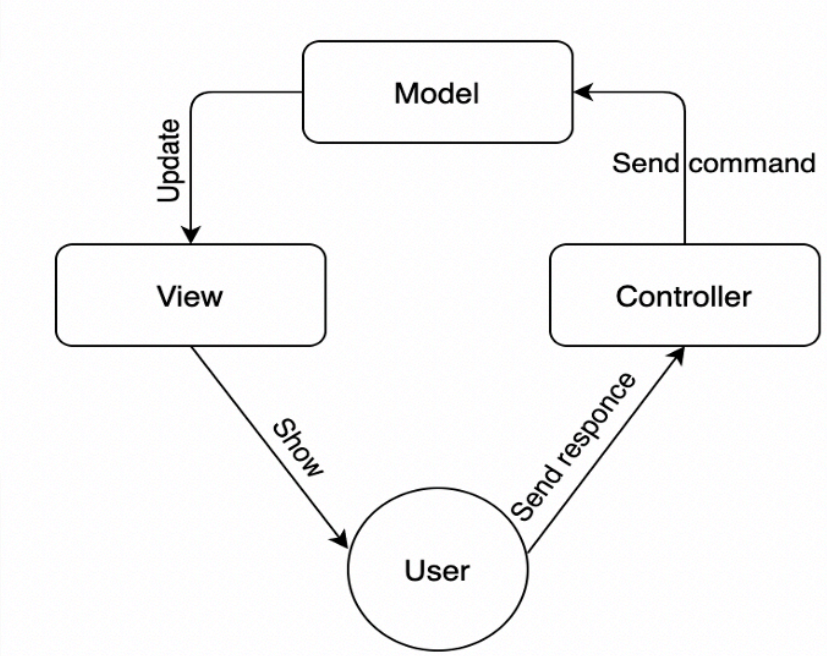
Засоби реалізації

- MEAN stack – Mongo, Express, Angular, Node.js

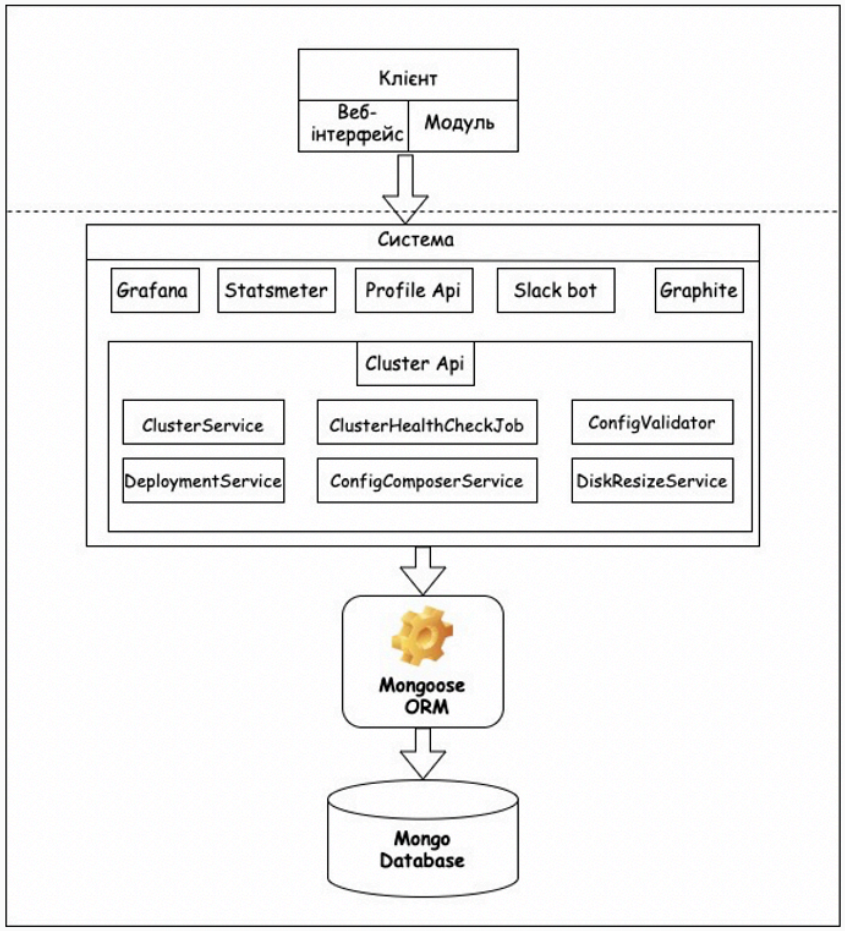
Схема взаємодії компонентів

Архітектурний шаблон MVC:

- Model
- View
- Controller



Опис системи



Статистичні показники системи

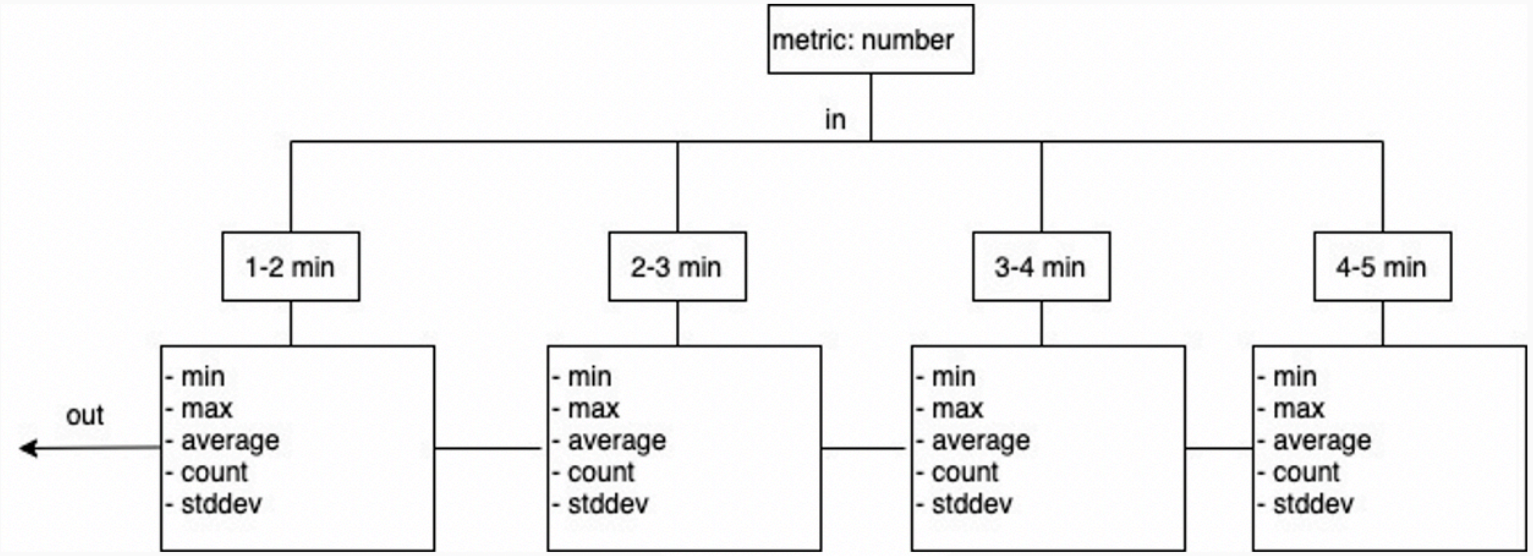
Система обраховує такі статистичні показники:

- Мінімальне значення;
- Максимальне значення;
- Середнє значення;
- Кількість елементів;
- Середнє квадратичне відхилення;

Формула для підрахунку середнього квадратичного відхилення

$$\sigma = \sqrt{\overline{x}^2 - (\overline{x})^2} = \sqrt{\frac{\sum x^2}{N} - (\frac{\sum x}{N})^2}$$

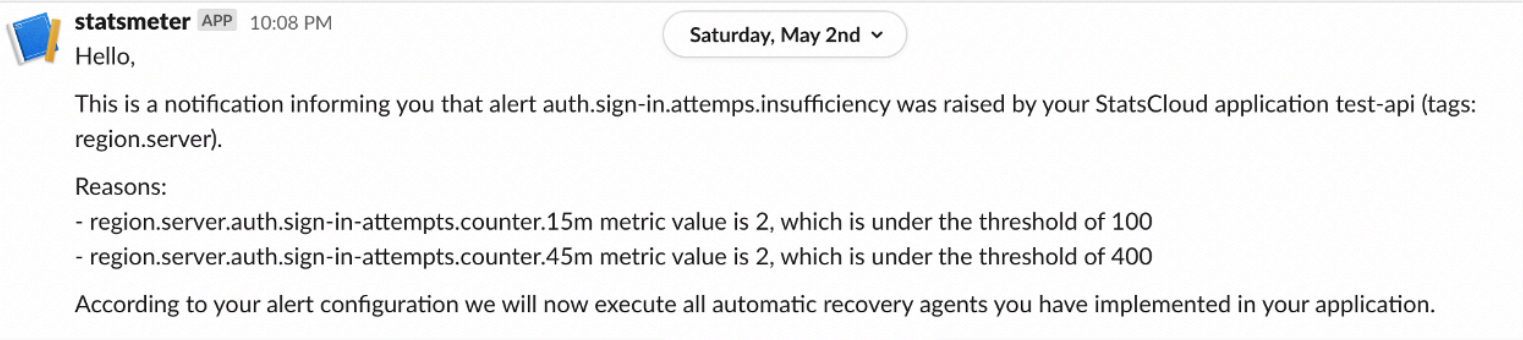
Додавання метрики



Система сповіщень

Система відправляє сповіщення в 2 месенджери:

- Slack
- RocketChat



Аналіз розробленої системи

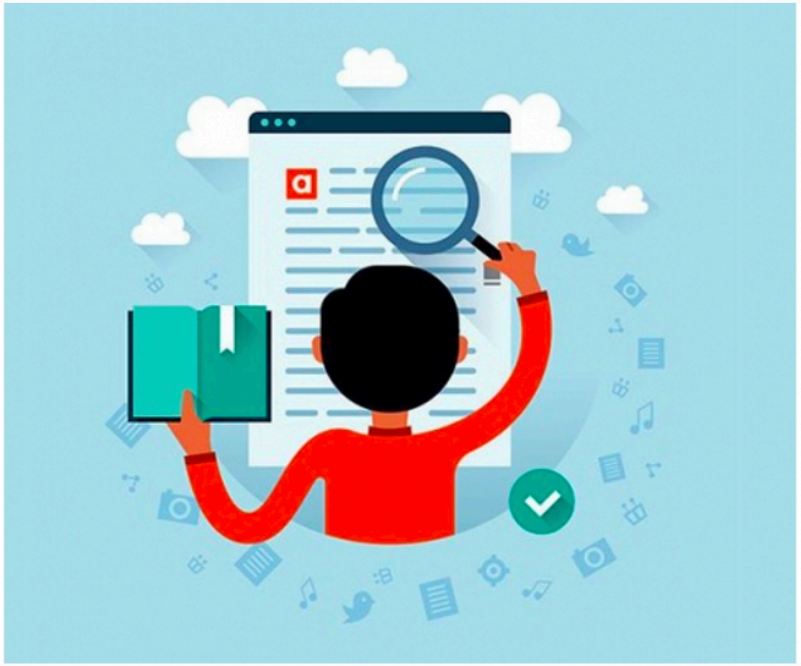
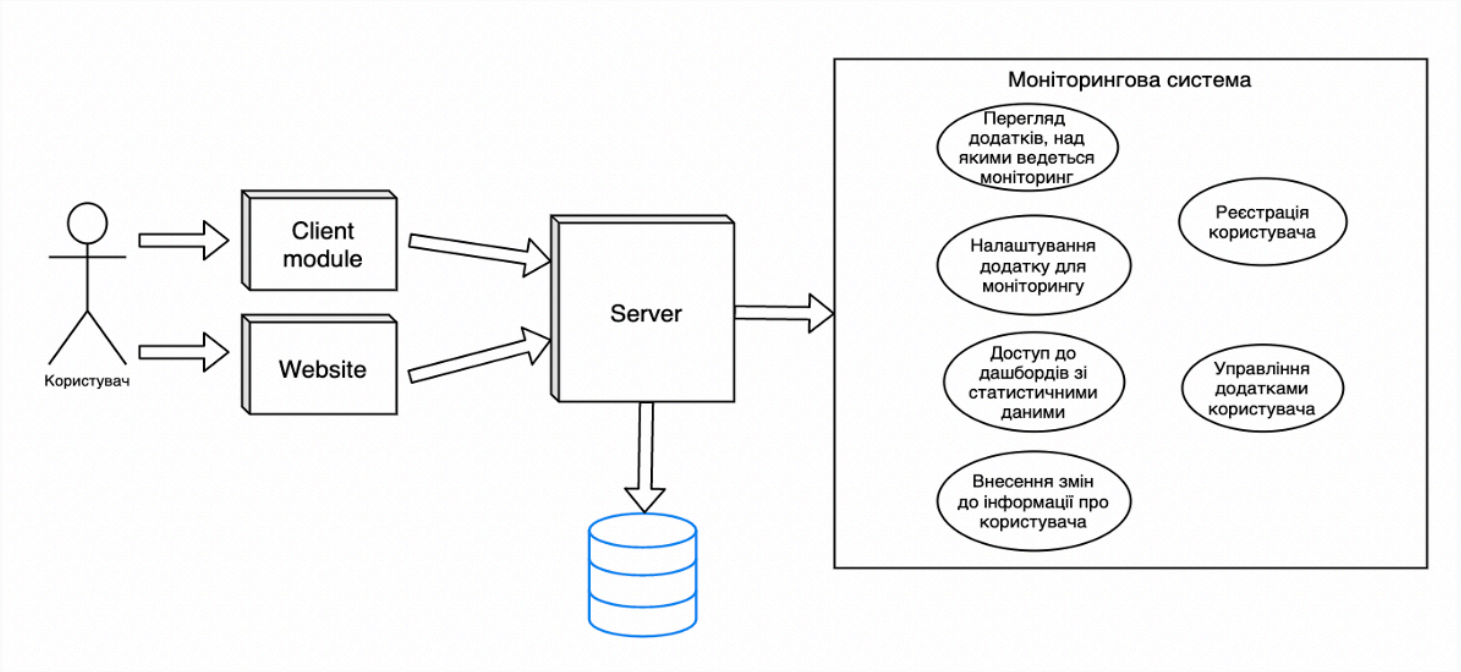


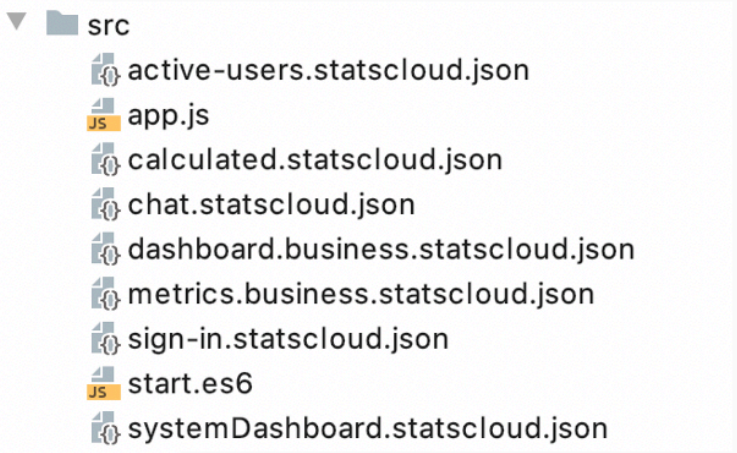
Схема використання системи



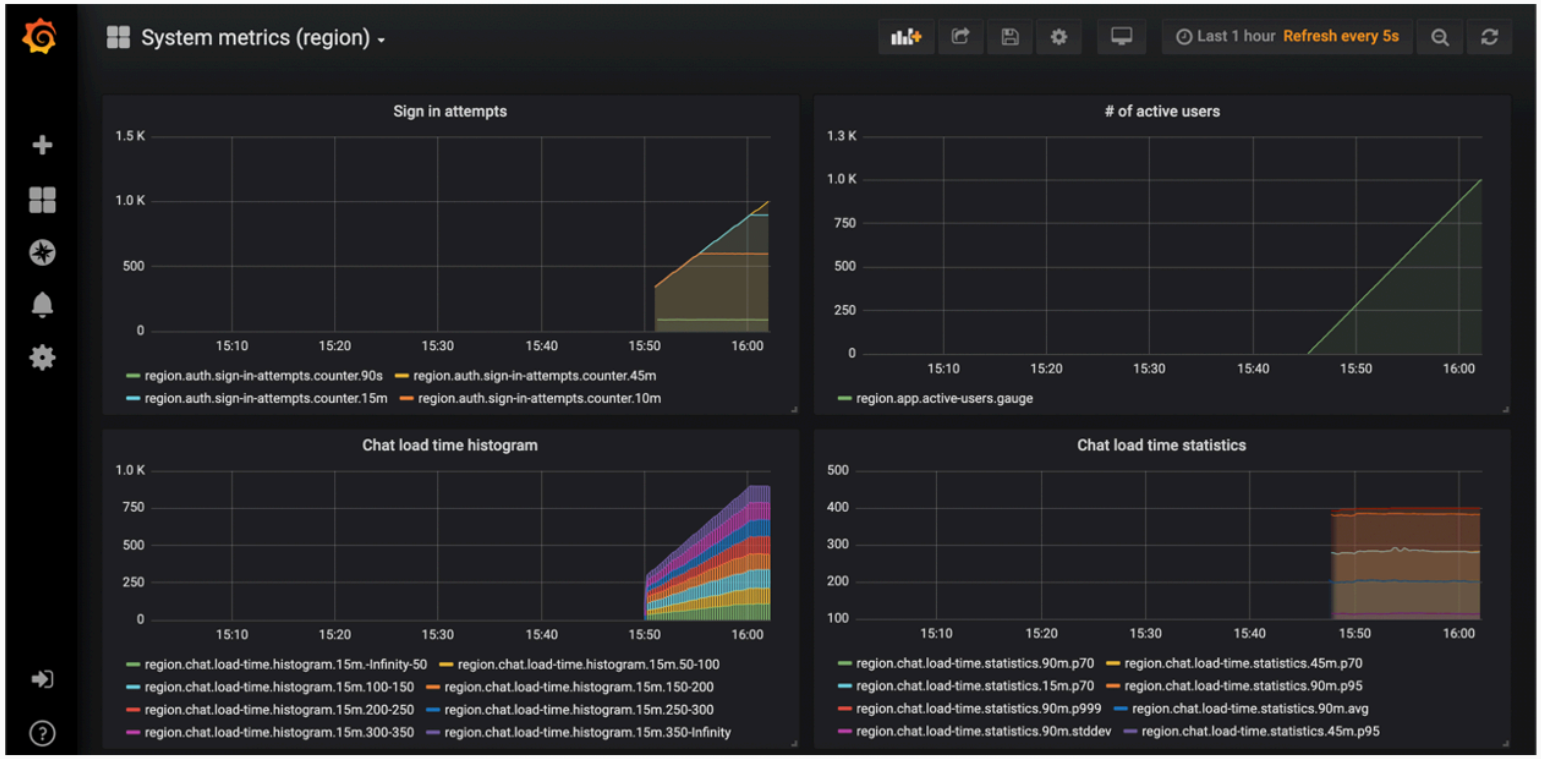
Налаштування

Налаштування системи здійснюється за допомогою файлів з розширенням .statscloud.
Користувач описує:

- Метрики:
  - Gauge
  - Histogram
  - Counter
  - Statistic
- Сторінки з дашбордами
- Здійснює відправку метрик

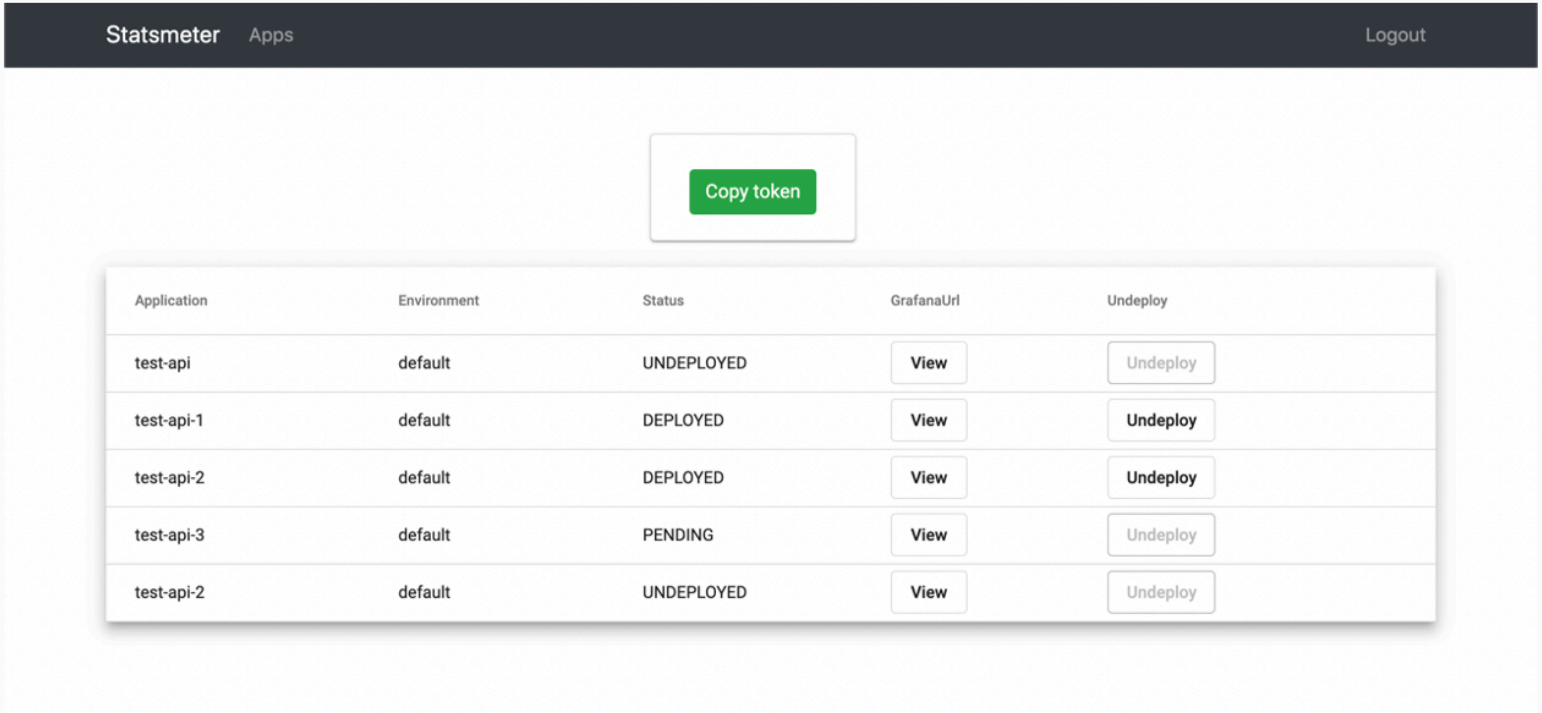


Інтерфейс системи (користувацькі метрики)



Інтерфейс системи (системні метрики)

Інтерфейс системи (сторінка з додатками)

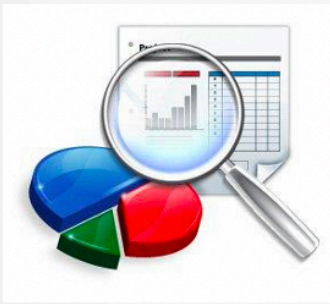


Тестування системи

- Було проведено мануальне тестування коду і його вдосконалення
- Здійснено тестування на продуктивність. Система може обробити 179083 записи за секунду
- Було проведено тестування реальними користувачами та виправлено недоліки

```
[Mon May 04 2020 22:19:31 GMT+0300 (Eastern European Summer Time)] processed 33094 records in 528ms
[Mon May 04 2020 22:19:32 GMT+0300 (Eastern European Summer Time)] processed 102964 records in 1048ms
[Mon May 04 2020 22:19:32 GMT+0300 (Eastern European Summer Time)] processed 194899 records in 1560ms
[Mon May 04 2020 22:19:33 GMT+0300 (Eastern European Summer Time)] processed 290511 records in 2059ms
[Mon May 04 2020 22:19:33 GMT+0300 (Eastern European Summer Time)] processed 389796 records in 2563ms
[Mon May 04 2020 22:19:34 GMT+0300 (Eastern European Summer Time)] processed 489076 records in 3064ms
[Mon May 04 2020 22:19:34 GMT+0300 (Eastern European Summer Time)] processed 592039 records in 3565ms
[Mon May 04 2020 22:19:35 GMT+0300 (Eastern European Summer Time)] processed 687659 records in 4077ms
[Mon May 04 2020 22:19:35 GMT+0300 (Eastern European Summer Time)] processed 794298 records in 4578ms
[Mon May 04 2020 22:19:36 GMT+0300 (Eastern European Summer Time)] processed 900948 records in 5078ms
[Mon May 04 2020 22:19:36 GMT+0300 (Eastern European Summer Time)] processed 1000000 records in 5583ms
MetricsStorage processed 179083 records per second
```

Рекомендації для подальшого вдосконалення



- Створити модуль, який буде оброблювати статистичні дані за допомогою AI
- Інтеграція всіх графіків автоматично на адмінську панель сайту користувача за допомогою html тегів
- Відправка сповіщень в інші месенджери
- Comandline interface для зручного налаштування системи

Перевірка на плагіат

Власник документу:
Люшенко Леся Анатоліївна gmail

ID перевірки:
1003866014

Дата перевірки:
08.06.2020 12:29:51 EEST

Тип перевірки:
Doc vs Internet + Library

Дата звіту:
08.06.2020 19:35:03 EEST

ID користувача:
91603

Назва документу: **Щербина_КП-61**

ID файлу: **1003867311** **Кількість сторінок:** **57** **Кількість слів:** **7145** **Кількість символів:** **55849** **Розмір файлу:** **3.05 MB**

1.57% Схожість

Найбільша схожість: **0.57%** **з джерело** **https://pastebin.com/hU9E9UPp**

1.11% Схожість з Інтернет джерелами	10	Page 59
0.91% Текстові збіги по Бібліотеці акаунту	95	Page 59

ВИСНОВКИ

1. Було проаналізовано існуючі аналоги моніторингових систем та відповідно сформовано вимоги до ПЗ У вигляді функціональних та нефункціональних вимог
2. Після аналізу технологій для реалізації було обрано мову програмування JavaScript, платформу Node.js, фреймворк Express.js і базу даних MongoDb
3. Було створено моніторингову систему для веб-додатків у вигляді веб-застосунку і клієнтської бібліотеки
4. Систему було протестовано і усунено недоліки
5. Розробка програмного забезпечення викона в повному обсязі і відповідає поставленим вимогам

Дякую за увагу!

Додаток 3

Лістинг програми

```
'use strict';

/**
 * FIFO-like reservoir of a fixed size capable
 * calculating running sums, min/max, average, msdev and stddev
 * msdev and stddev calculation by Naive algorithm
 * (Mean square deviation) msdev = sqrt((∑{i = from 1 to n}(Xi)^2 -(∑{i =
from 1 to n}Xi)^2 / N) / N)
 * (Standard deviation) stddev = sqrt((∑{i = from 1 to n}(Xi)^2 -(∑{i =
from 1 to n}Xi)^2 / N) / N - 1)
 * link: https://goo.gl/MAEGP2
 */
export default class Reservoir {

  /**
   * Constructs Reservoir
   * @param {number} size Reservoir size
   * @param {number} observationIntervalInMS Reservoir observation
Interval In ms
   */
  constructor(size, observationIntervalInMS, object) {
    if (!object) {
      this.array = new Array();
      this.size = size;
      this._interval = (observationIntervalInMS / size);
      this._queueEndTime = Date.now();
      this._fisrtQueueIndex = 0;
      this._intermediaryRecord = undefined;
      this.statistics = {
        count: 0,
        sum: 0,
        max: undefined,
        min: undefined,
        average: 0,
        sumOfSquares: 0,
        msdev: 0,
        stddev: 0
      };
    } else {
      this.array = object.array;
      this.size = object.size;
      this._interval = object._interval;
      this._queueEndTime = object._queueEndTime;
      this._fisrtQueueIndex = object._fisrtQueueIndex;
      this._intermediaryRecord = object._intermediaryRecord;
      this.statistics = this.checkStatisticsOnRestore(object.statistics);
    }
  }

  checkStatisticsOnRestore(statistics) {
    if (statistics.count === 0) {
      statistics = {
        count: 0,
        sum: 0,
        max: undefined,
        min: undefined,
        average: undefined,
        sumOfSquares: 0,
        msdev: undefined,
        stddev: undefined
      };
    } else if (statistics.count < 2) {
      statistics.msdev = undefined;
      statistics.stddev = undefined;
```

```
  }
  return statistics;
}

/**
 * Add element to Reservoir
 * @param {Number} data to add
 */
pushMeasurement(data) {
  if (isFinite(data)) {
    this._updateQueue();
    this._updateIntermediaryRecord(data);
    this._updateStatisticsOnAdd(data);
  }
}

/**
 * return Reservoir statistics
 * @return {Object} Reservoir statistics
 */
getStatistics() {
  this._updateQueue();
  return this.statistics;
}

toPlainObject() {
  this._updateQueue(true);
  return {
    array: this.array,
    size: this.size,
    _interval: this._interval,
    _queueEndTime: this._queueEndTime,
    _fisrtQueueIndex: this._fisrtQueueIndex,
    _intermediaryRecord: this._intermediaryRecord,
    statistics: this.statistics
  };
}

_updateQueue() {
  let intervalsCount = this._takeTimeIntervalsCount();
  let emptyElementsCount = this._takeEmptyElementsAddCount();
  if (emptyElementsCount > 0) {
    this._addRecord(emptyElementsCount);
    this._queueEndTime += intervalsCount * this._interval;
  }
}

_takeEmptyElementsAddCount() {
  let emptyElementsCount = this._takeTimeIntervalsCount();
  if (emptyElementsCount > this.size) {
    emptyElementsCount = this.size;
  }
  return emptyElementsCount;
}

_takeTimeIntervalsCount() {
  let timeNow = Date.now();
  let timeDiff = timeNow - this._queueEndTime;
  let timeIntervalsCount = Math.floor(timeDiff / this._interval);
  return timeIntervalsCount;
}

_updateRunningStatisticsOnRemove(removeCount) {
```

```
    let removeElementIndex = this._fisrtQueueIndex + 1;
    for (let i = 0; i < removeCount; i++) {
      if (removeElementIndex >= this.size) {
        removeElementIndex = 0;
      }

      this._updateStatisticsOnRemove(this.array[removeElementIndex],
removeElementIndex);
      this.array[removeElementIndex] = {
        count: 0,
        sum: 0,
        max: undefined,
        min: undefined,
        average: 0,
        sumOfSquares: 0
      };
      removeElementIndex++;
    }
    removeElementIndex--;
    if (removeElementIndex < 0) {
      removeElementIndex = this.size - 1;
    }
    return removeElementIndex;
  }

  _updateStatisticsOnRemove(removeElement, removeElementIndex) {
    if (removeElement !== undefined && removeElement !== null) {
      this.statistics.count -= removeElement.count;
      this.statistics.sumOfSquares -= removeElement.sumOfSquares;
      this.statistics.sum -= removeElement.sum;
      this._updateStatisticsMinAndMaxOnRemove(removeElement,
removeElementIndex);
      if (this.statistics.count > 0) {
        this.statistics.average = this.statistics.sum /
this.statistics.count;
        if (this.statistics.count > 1) {
          let difOfSums =
this._calculateDifferenceOfSums(this.statistics.sumOfSquares,
            this.statistics.sum, this.statistics.count);
          this.statistics.msdev = parseFloat(Math.sqrt(difOfSums /
this.statistics.count));
          this.statistics.stddev = parseFloat(Math.sqrt(difOfSums /
(this.statistics.count - 1)));
        } else {
          this.statistics.stddev = undefined;
          this.statistics.msdev = undefined;
        }
      } else {
        this.statistics.average = undefined;
        this.statistics.stddev = undefined;
        this.statistics.msdev = undefined;
      }
    }
  }

  _updateStatisticsMinAndMaxOnRemove(removeElement, removeElementIndex) {
    if (removeElement.max !== undefined && removeElement.max ===
this.statistics.max) {
      this.statistics.max = this._findMax(removeElementIndex);
    }

    if (removeElement.min !== undefined && removeElement.min ===
this.statistics.min) {
```

```
      this.statistics.min = this._findMin(removeElementIndex);
    }
  }

  _updateStatisticsOnAdd(el) {
    if (el !== undefined && el !== null) {
      this.statistics.count += 1;
      this.statistics.sum += el;
      this._updateStatisticsMinAndMaxOnAdd(el);
      this.statistics.sumOfSquares += Math.pow(el, 2);
      if (this.statistics.count > 0) {
        this.statistics.average = this.statistics.sum /
this.statistics.count;
        let difOfSums =
this._calculateDifferenceOfSums(this.statistics.sumOfSquares,
          this.statistics.sum, this.statistics.count);
        if (this.statistics.count > 1) {
          this.statistics.msdev = parseFloat(Math.sqrt(difOfSums /
this.statistics.count));
          this.statistics.stddev = parseFloat(Math.sqrt(difOfSums /
(this.statistics.count - 1)));
        } else {
          this.statistics.msdev = undefined;
          this.statistics.stddev = undefined;
        }
      }
    }
  }

  _updateStatisticsMinAndMaxOnAdd(el) {
    if (this.statistics.max < el || this.statistics.max === undefined ||
this.statistics.max === null) {
      this.statistics.max = el;
    }
    if (this.statistics.min > el || this.statistics.min === undefined ||
this.statistics.min === null) {
      this.statistics.min = el;
    }
  }

  _addRecord(emptyElementsCount) {
    if (this._intermediaryRecord !== undefined) {
      this.array[this._fisrtQueueIndex] = this._intermediaryRecord;
      this._intermediaryRecord = undefined;
    }
    let curIndexInArray =
this._updateRunningStatisticsOnRemove(emptyElementsCount);
    this._fisrtQueueIndex = curIndexInArray;
  }

  _calculateDifferenceOfSums(sum1, sum2, count) {
    let dif = sum1 - Math.pow(sum2, 2) / count;
    return dif;
  }

  _updateIntermediaryRecord(el) {
    if (this._intermediaryRecord === undefined) {
      this._intermediaryRecord = {
        count: 1,
        sum: el,
        max: el,
        min: el,
        average: el,
```

```
        sumOfSquares: Math.pow(el, 2)
      };
    } else {
      if (this._intermediaryRecord.max < el) {
        this._intermediaryRecord.max = el;
      }
      if (this._intermediaryRecord.min > el) {
        this._intermediaryRecord.min = el;
      }
      this._intermediaryRecord.count += 1;
      this._intermediaryRecord.sum += el;
      this._intermediaryRecord.sumOfSquares += Math.pow(el, 2);
    }
  }

  _findMin(index) {
    let min = Infinity;
    this.array.forEach((el, i) => {
      if (el !== null && el !== undefined && el.min !== undefined &&
el.min < min && i !== index) {
        min = el.min;
      }
    });
    if (min === Infinity) {
      return (this._intermediaryRecord !== undefined) ?
this._intermediaryRecord.min : undefined;
    }
    return min;
  }

  _findMax(index) {
    let max = -Infinity;

    this.array.forEach((el, i) => {
      if (el !== null && el !== undefined && el.max !== undefined &&
el.max > max && i !== index) {
        max = el.max;
      }
    });
    if (max === -Infinity) {
      return (this._intermediaryRecord !== undefined) ?
this._intermediaryRecord.max : undefined;
    }
    return max;
  }
}

---

export class MetricsStorage {

  /**
   * Metrics service class constructor
   * @param {ConfigComposer} configComposer config composer service
   * @param {MetricsManagement} metricsManagement metrics management
service
   * @param {CalculatedMetricsSupport} calculatedMetricsSupport addon to
support calculated metrics
   * @param {BillingService} billingService billing service
   */
  constructor(configComposer, metricsManagement,
calculatedMetricsSupport, billingService) {
    this._logger = log4js.getLogger('MetricsStorage');
```

```
    this._config = configComposer.config;
    this._metricsManagement = metricsManagement;
    this._calculated = calculatedMetricsSupport;
    this._billingService = billingService;
  }

  init() {
    this._calculated.initDependencies(this, this._metricsManagement);
    this._metricsManagement.init();
  }

  /**
   * Saves metrics data to file
   * @returns {Promise<>} returns promise which will be resolved when
metrics data will be saved
   */
  async saveReservoirs() {
    if (!this.canStoreReservoirs) {
      return;
    }

    const startStoringReservoirs = new Date();
    const dir = this._config.persistence.reservoirsDir;
    try {
      await fsPromised.access(dir, fs.F_OK);
      // directory exists
    } catch (error) {
      // directory does not exists
      if (error.code === 'ENOENT') {
        this._logger.warn('Reservoirs directory does not exist.
Creating...');
        await fsPromised.mkdir(dir);
      } else {
        throw error;
      }
    }

    const data = await this._metricsManagement.toBson();

    if (data.length > 5) {
      return
fsPromised.writeFile(path.join(this._config.persistence.reservoirsDir,
        this._config.persistence.fileName), data)
        .then(() => {
          this._logger.debug('Recorded reservoirs for ' +
this._metricsManagement.countMetrics() +
            ' metrics to persistent storage in ' + (new Date() -
startStoringReservoirs) + ' ms');
        });
    }
  }

  /**
   * Restores metrics data
   * @returns {Promise<>} returns promise which will be resolved when
metrics will be restored
   */
  async restoreReservoirs() {
    try {
      const data = await fsPromised.readFile(
        path.join(this._config.persistence.reservoirsDir,
this._config.persistence.fileName)
      );
```

```
      this._metricsManagement.fromBson(data);
    } catch (error) {
      if (error.code !== 'ENOENT') {
        this._logger.error('An error occurred while reading reservoirs
file', error);
      }
      this.init();
    } finally {
      this.canStoreReservoirs = true;
    }
  }

  /**
   * Returns listener function for specific event
   * @param {String} eventName event name
   * @returns {function} returns listener function
   */
  getEventListener(eventName) {
    const dontRecordEventFor = ['AlertsHealth', 'StatsmeterHealth'];
    return (event, tags=[]) => {
      if (!dontRecordEventFor.includes(eventName)) {
        this._billingService.recordEvent();
      }
      for (const metric of
this._metricsManagement.metricsByEvents(eventName)) {
        const type = metric.metricConfig.type;
        if (!this._metricsManagement.doesTagsContainTags(tags,
metric.tags)) {continue;}
        switch (type) {
        case 'counter':
          //for each interval
          this._counterListener(metric);
          break;
        case 'statistic':
          //for each interval
          this._statisticListener(metric, event);
          break;
        case 'gauge':
          this._gaugeListener(metric, event);
          break;
        case 'histogram':
          this._histogramListener(metric, event);
          break;
        }
      }
    };
  }

  _counterListener(metric) {
    this._updateReservoir(metric);
    metric.value++;
  }

  _statisticListener(metric, event) {
    this._logger.trace('Received statistic event for metric ' +
metric.metricName, event);
    metric.reservoir.pushMeasurement(event);
    metric.value = event;
    metric.statisticsReservoir.pushMeasurement(event);
    if (this._logger.level === 'TRACE') {
      this._logger.trace('Pushed measurement for ' + metric.metricName +
        ' to reservoir, reservoir length is \n' +
```

```
        metric.statisticsReservoir.length + ', reservoir is ' +
this._printStatisticsReservoir(metric.statisticsReservoir.toArray()));
    }
  }

  _gaugeListener(metric, event) {
    metric.value = event;
  }

  _histogramListener(metric, event) {
    if (this._checkIfValueFallsInRange(event, metric.range)) {
      this._updateReservoir(metric);
      metric.value++;
    }
  }

  _updateReservoir(metric) {
    const now = moment().valueOf();
    let timeElapsed;
    switch (metric.metricConfig.type) {
    case 'counter':
    case 'histogram':
      metric.reservoir.pushMeasurement(metric.value);
      metric.value = 0;
      break;
    }
    if (metric.metricConfig.type === 'statistic') {
      if (this._logger.level === 'TRACE') {
        this._logger.trace('Updated ' + metric.metricName + ' reservoirs,
reservoir is \n' +
this._printStatisticsReservoir(metric.statisticsReservoir.toArray()) +
'\n' +
          'running statistics is' + '\n' +
util.inspect(metric.reservoir.getStatistics())
        );
      }
    }
  }

  _printStatisticsReservoir(reservoir) {
    if (reservoir.length) {
      if (reservoir.length < 10) {
        return util.inspect(reservoir.map(item => {
          return {time: new Date(item.time), data: item.data};
        }));
      } else {
        return util.inspect(reservoir.filter((item, index) => index < 5
|| index > reservoir.length - 6).map(item => {
          return {time: new Date(item.time), data: item.data};
        }));
      }
    } else {
      return '';
    }
  }

  /**
   * Returns metric data for all tags or specific one
   * @param {String} metricName metric name
   * @param {Array} tags tags
   * @returns {Object} metrics stats
```

```
   */
  getStats(metricName, tags) {
    const metrics =
this._metricsManagement.metricsByMetricNames(metricName);
    if (!metrics || !metrics[0]) {return [];}

    if (metrics[0].metricConfig.type === 'calculated') {
      return this._getStatsForCalculated(metricName, metrics, tags);
    } else {
      return this._getStatsForPrimitiveMetrics(metricName, metrics,
tags);
    }
  }

  _getStatsForCalculated(metricName, metrics, mainTags) {
    const metricsStats = [];
    const suffixedMetricName = addSuffix(metricName,
this._config.metrics[metricName].type);

    const groupsOfTags = mainTags ? [mainTags] :
this._metricsManagement.getAllInitializedTags();

    for (const tags of groupsOfTags) {
      const wholeMetricName = this._metricsManagement.addTagsPrefix(tags,
suffixedMetricName);
      metricsStats.push({
        metricName,
        wholeMetricName,
        tags,
        stat: this._getMetricsStats(metrics, tags)
      });
    }

    return metricsStats;
  }

  _getStatsForPrimitiveMetrics(metricName, metrics, mainTags) {
    const metricsStats = [];

    const suffixedMetricName = addSuffix(metricName,
this._config.metrics[metricName].type);
    const sortedMetricsByTags =
this._metricsManagement.sortMetricsByTags(metrics);

    const groupOfTagsString = mainTags ?
      [this._metricsManagement.tagsToString(mainTags)] :
      Object.keys(sortedMetricsByTags);

    for (const tagsString of groupOfTagsString) {
      const metricsByTags = sortedMetricsByTags[tagsString] || [];
      const tags = this._metricsManagement.tagsFromString(tagsString);

      const wholeMetricName =
this._metricsManagement.addTagsPrefix(tagsString, suffixedMetricName);

      metricsStats.push({
        metricName,
        wholeMetricName,
        tags,
        stat: this._getMetricsStats(metricsByTags)
      });
    }
```

```
    return metricsStats;
  }

  _getMetricsStats(metrics, tags) {
    let stats;
    switch (metrics[0].metricConfig.type) {
    case 'counter':
      stats = this._getCounterStats(metrics);
      break;
    case 'histogram':
      stats = this._getHistogramStats(metrics);
      break;
    case 'statistic':
      stats = this._getStatisticStats(metrics);
      break;
    case 'gauge':
      stats = this._getGaugeStats(metrics);
      break;
    case 'calculated':
      stats = this._calculated.getStats(metrics, tags);
      break;
    }

    if (stats !== undefined) {
      return [stats];
    } else {
      return [];
    }
  }

  _getCounterStats(metrics) {
    return this._calculateIntervalsStats(metrics,
metrics[0].metricConfig.intervals, (m) => {
      let rate = 0;
      if (m.reservoir._interval !== undefined) {
        rate = m.reservoir.getStatistics().count;
      }
      return [rate];
    });
  }

  _getHistogramStats(metrics) {
    return this._calculateIntervalsStats(metrics,
metrics[0].metricConfig.intervals, (m) => {
      let rate = 0;
      if (m.reservoir._interval !== undefined) {
        rate = m.reservoir.getStatistics().count;
      }
      return [{[m.range]: rate}];
    });
  }

  _getStatisticStats(metrics) {
    return this._calculateIntervalsStats(metrics,
metrics[0].metricConfig.intervals, (m) => {
      return this._getStatisticAllStatistics(m);
    });
  }

  _getGaugeStats(metrics) {
    const gaugeMetric = metrics[0];
    return gaugeMetric.value;
  }
```

```
  _calculateIntervalsStats(metrics, intervals, getIntervalValues) {
    const allIntervalsValues = {};
    for (const metric of metrics) {
      for (const intervalValue of getIntervalValues(metric)) {
        // interval value can be object or any primitive value
        if (typeof intervalValue === 'object') {
          for (const key of Object.keys(intervalValue)) {
            if (!allIntervalsValues[metric.interval])
{allIntervalsValues[metric.interval] = {};}
            allIntervalsValues[metric.interval][key] =
intervalValue[key];
          }
        } else {
          allIntervalsValues[metric.interval] = intervalValue;
        }
      }
    }

    if (Object.keys(allIntervalsValues).length !== 0) {
      return allIntervalsValues;
    }
  }
}
```

Факультет прикладної математики

Кафедра програмного забезпечення комп'ютерних систем

«ЗАТВЕРДЖЕНО»
Науковий керівник кафедри
_____ Іван ДИЧКА
«___»_____2019 р.

ПРОГРАМНА СИСТЕМА МОНІТОРИНГУ ВЕБ-ДОДАТКІВ В РЕАЛЬНОМУ ЧАСІ

Програма та методика тестування

ДП.045440-04-51

"ПОГОДЖЕНО"

Керівник проєкту:

_____ Леся ЛЮШЕНКО

Нормоконтроль:

_____ Микола ОНАЙ

Виконавець:

_____ Вадим ЩЕРБИНА

2019

ЗМІСТ

1. ОБ’ЄКТ ВИПРОБУВАНЬ

Програмна система моніторингу веб-додатків в реальному часі, яка реалізована у вигляді веб-додатку і клієнтської бібліотеки, яку необхідно підключити до свого додатку. Користувачі даної системи мають можливість відслідковувати необхідні метрики і переглядати їх на зручних графіках.

2. МЕТА ТЕСТУВАННЯ

У процесі тестування має бути перевірено наступне:

- коректність налаштувань системи;
- коректність модулю обробки користувацьких метрик;
- коректність модулю підрахунку статистичних показників;
- продуктивність системи і відповідність кількості метрик, які може обробити система функціональним вимогам;
- коректність та стабільність роботи системи.
- відповідність інтерфейсу нефункціональним вимогам (час відгуку, зрозумілість, зручність використання тощо).
- зручність інтерфейсу.

3. МЕТОДИ ТЕСТУВАННЯ

Виконується модульне, інтеграційне і тестування на продуктивність.

- мета модульного тестування: перевірка коректності роботи окремих модулів, що забезпечують відповідно:
    - відправка користувацьких метрик ;
    - обробка користувацьких метрик і підрахунок статистичних показників;

    - робота модулю по збору системних метрик(CPU, RAM, ROM);
    - робота модулю по запуску контейнерів з користувацькими сервісами;
    - інтерфейс користувача.
- мета інтеграційного тестування: перевірка взаємодії всіх молулів системи на відповідність функціональним вимогам;
- мета тестування на продуктивність: перевірка системи під навантаженням, щоб дізнатись яку кількість метрик вона може обробити.

Використовуються наступні методи:

- ручне тестування інтерфейсу;
- функціональне тестування;
- тестування взаємодії.

4. ЗАСОБИ ТА ПОРЯДОК ТЕСТУВАННЯ

Працездатність програмного продукту перевіряється шляхом:

- динамічного ручного тестування – введенням граничних та недопустимих значень в поля, які доступні для використання у рамках даного програмного проєкту;
- динамічного ручного тестування на відповідність функціональним вимогам;
- статичного тестування коду;
- тестування стабільності роботі при різних зовнішніх умовах;
- тестування зручності використання;
- тестування інтерфейсу.

Факультет прикладної математики

Кафедра програмного забезпечення комп'ютерних систем

«ЗАТВЕРДЖЕНО»
Науковий керівник кафедри
_____ Іван ДИЧКА
«___»_____2020 р.

ПРОГРАМНА СИСТЕМА МОНІТОРИНГУ ВЕБ-ДОДАТКІВ В РЕАЛЬНОМУ ЧАСІ

Керівництво користувача
ДП.045440-05-34

«ПОГОДЖЕНО»

Керівник проєкту:

_____ Леся ЛЮШЕНКО

Нормоконтроль:

_____ Микола ОНАЙ

Виконавець:

_____ Вадим ЩЕРБИНА

2020

ЗМІСТ

1. ОПИС ПРОГРАМНОГО ЗАСТОСУНКУ

Програмна система моніторингу веб-додатків в реальному часі, реалізована у вигляді веб-додатку і клієнтської бібліотеки, яку необхідно підключити до свого додатку. Користувачі даної системи мають можливість відслідковувати необхідні метрики і переглядати всю необхідну статистичну інформацію для тестування та визначення найпопулярніших ресурсів, а також переглядати навантаження на сервер, на якому запускається їх додаток. За допомогою системних показників можна визначати і прогнозувати, які системні ресурси потрібні для коректної роботи веб-додатку.

2. ОПИС ПРОЦЕДУРИ НАЛАШТУВАННЯ СИСТЕМИ

Для роботи системи потрібно встановити клієнтську бібліотеку і налаштувати відправку користувацьких метрик.

Щоб налаштувати метрики їх потрібно описати в файлах конфігурацій з розширенням .statscloud.json, а також добавити у файли конфігурацій дашбордів.

Лістинг 1. Приклад налаштування метрики

```
{
  "metrics": {
    "app.active-users": {
      "event": "app.active-users",
      "type": "gauge",
      "flushIntervalInSeconds": 5,
      "retention":
        {
          "frequency": "10s",
          "keep": "2d"
        }
      ]
    }
  }
}
```

Налаштування сповіщень. Відбувається при створенні метрики.

Лістинг 2. Приклад налаштування сповіщень

```
"auth.sign-in.attemps.overflow": {
  "metric": {
    "metric": "auth.sign-in-attempts",
    "interval": "10m”
  },
  "tags": [[], ["region", "*"]],
  "threshold": 100,
  "condition": "gt",
  "unhealthyDelay": "2m",
  "healthyDelay": "1m",
  "admins": ["developer"],
  "channels": ["slack"],
  "message": "Too many sign in attempts",
  "repeatInterval": "1m"
}
```

Користувацька бібліотека надає можливість для підключення плагінів, для збору системних метрик і для перевірки доступності веб-ресурсів. Щоб Налаштувати плагіни для збору системних метрик і перевірки доступності ресурсів неоюхідно додати .

Лістинг 3. Налаштування плагінів

```
"plugins": [
    {
      "name": "system-monitor",
      "settings": {
        "enableAlerts": true,
        "admins": ["developer"],
        "channels": ["slack"]
      }
    },
    {
      "name": "healthcheck",
      "settings": {
        "healthChecks": [
          {
             "name" : "google",
             "url": "https://www.google.com/"
          },
          {
            "name" : "google1",
            "url": "https://www.google1.com/"
          }
        ]
      }
    }
  ]
```

Лістинг 5. Приклад програмного коду для відправки користувацких мерик

```
import statscloud from 'statscloud.io-client';
statscloud.withTags('region', 'server').start()
  .then(() =>
     setInterval(() => statscloud.recordEvent('auth.sign-in-
attempt'), 1000);
     let gauge = 0;
     setInterval(() => statscloud.recordEvent('app.active-users',
gauge++), 1000);
     setInterval(() => statscloud.recordEvent('chat.page-loaded',
Math.random() * 400), 1000);
});
const exitHandler = () => {
       statscloud.stop()
      .then(() => businessClient.stop())
      .then(() => console.log('Successfully cleaned up'))
      .then(() => process.exit(), () => process.exit());
};
process.on('exit', exitHandler);
process.on('SIGINT', exitHandler);
```

Лістинг 6. Приклад налаштування дашборду

```
{
  "dashboards": [
    {
      "id": "business",
      "name": "Business dashboard",
      "rows": [
        {
          "charts": [
            {
              "name": "Counter",
              "metrics": [
                "business.counter"
              ]
            }
          ]
        }
      ]
    }
  ]
}
```

3. ОПИС ВЕБ-СТОРІНОК

Моніторингова система легко інтегрується до користувацьких веб-додатків, тому для початку використання системи необхідно зареєструватись і скопіювати токен доступу, який потім необхідно розмітити у файлі конфігурації системи. На рис. 1 зображено форму реєстації в системі.

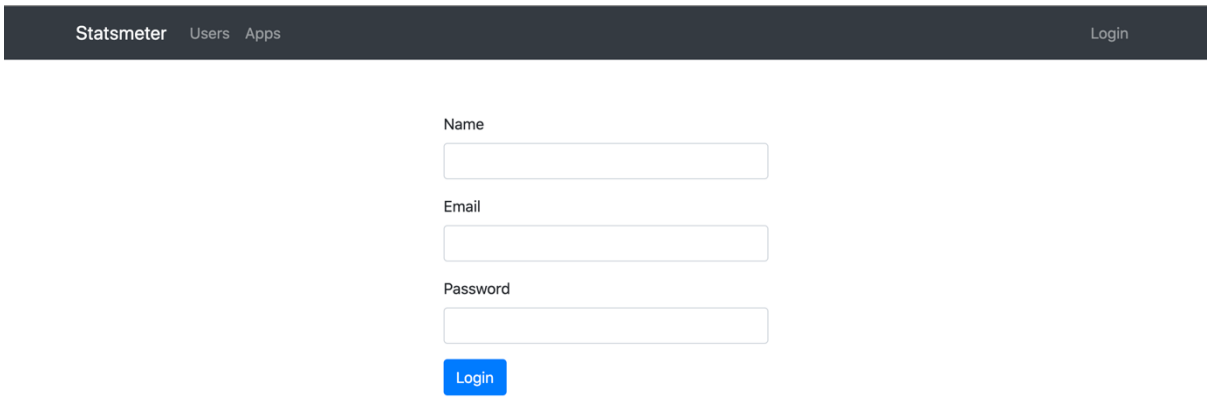


Рис. 1. Форма реєстрації в системі

Для входу входу в систему користувач повинен ввести свій логін і пароль. На рис. 2 зображено форму для входу в систему.

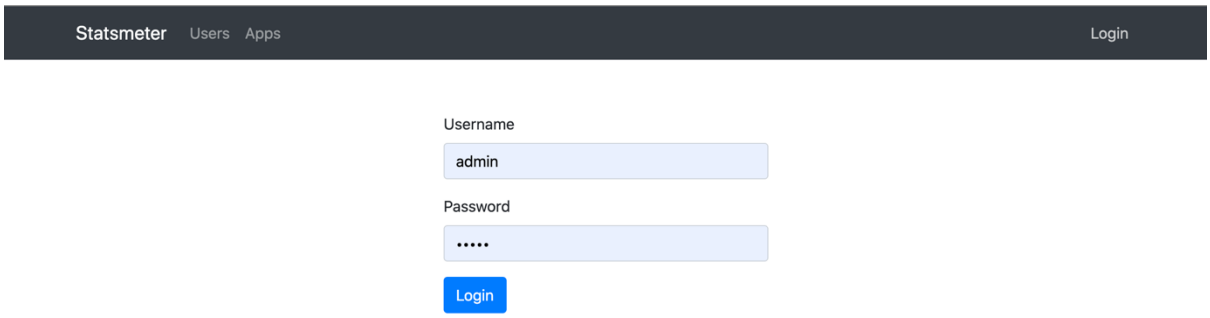


Рис. 2. Форма входу в систему

Після того, як користувач був авторизований в системі він попадає на сторінку з усіма своїми додатками над якими ведеться моніторинг. Він має можливість перейти на сторінку з графіками і переглянути поточний стан метрик, або зупинити моніторинг даного додатку (рис. 3).

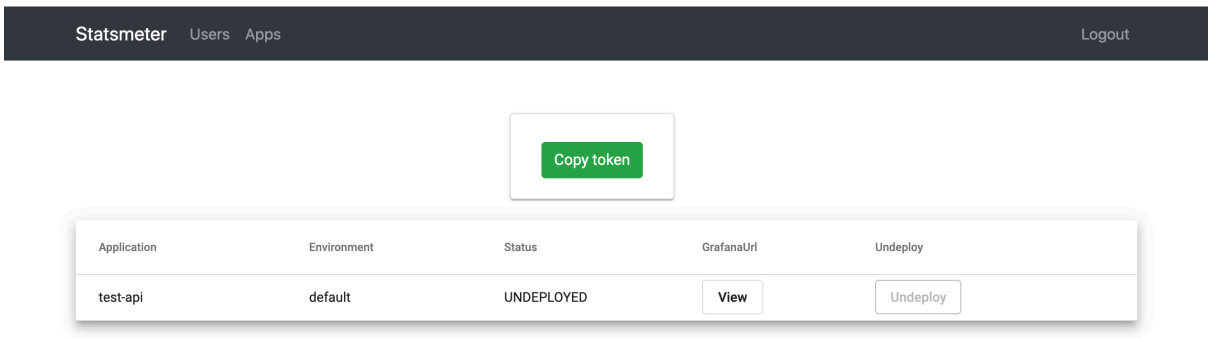


Рис. 3 Сторінка з користувацькими додатками

Після того, як користувач натиснув на кнопку View, він попадає на сторінку сервісу Grafana, на якій можна відслідкувати свої метрики. На рис. 4 зображено графіки з користувацькими метриками.

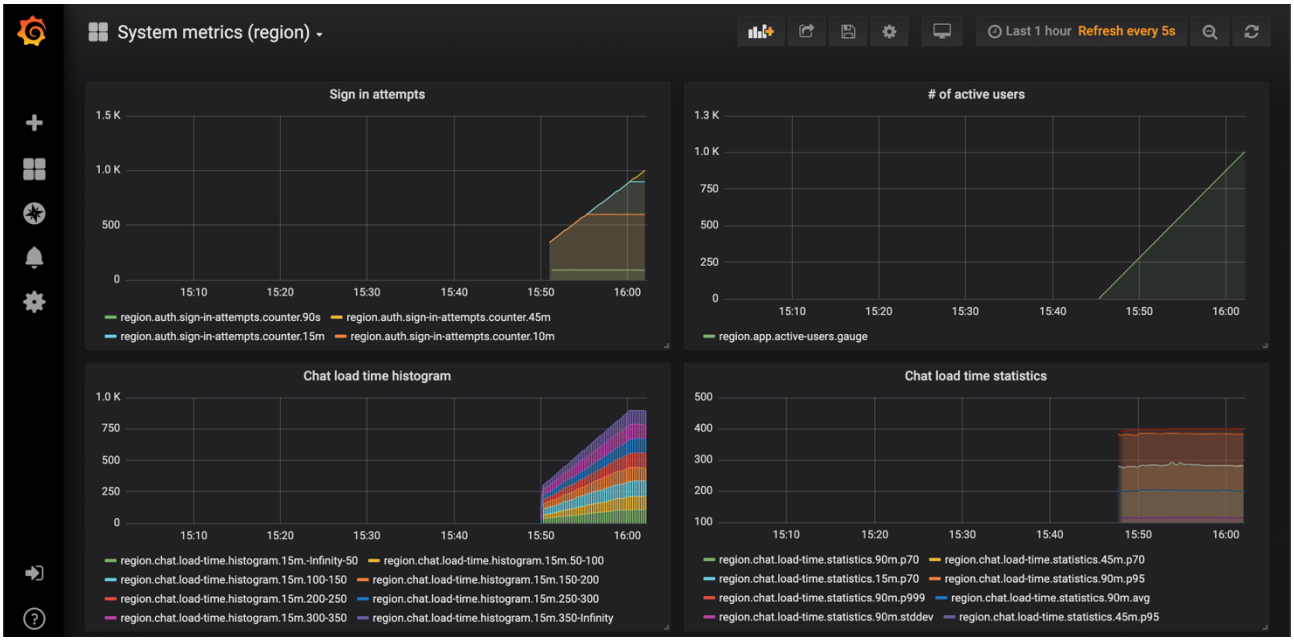


Рис. 4 Сторінка з користувацькими метриками