НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ імені ІГОРЯ СІКОРСЬКОГО»
Факультет інформатики та обчислювальної техніки
(повна назва інституту/факультету)

Кафедра інформатики та програмної інженерії
(повна назва кафедри)

«До захисту допущено»
Завідувач кафедри
_____ Едуард ЖАРІКОВ
(підпис) (ім'я прізвище)
"___"_____2023 р.

Дипломний проєкт

на здобуття ступеня бакалавра

за освітньо-професійною програмою «Інженерія програмного забезпечення інформаційних систем»

спеціальності «121 Інженерія програмного забезпечення»

на тему: Програмне забезпечення для збору програмних метрик застосунків

Виконав студент IV курсу, групи ІП-91
(шифр групи)
Шаховська Дар'я Олександрівна _____
(прізвище, ім'я, по батькові) (підпис)

Керівник ст. викл., Халус О. А. _____
(посада, науковий ступінь, вчене звання, прізвище та ініціали) (підпис)

Консультант з графічної документації доцент, к.т.н., доц. Ліщук К. І. _____
(посада, науковий ступінь, вчене звання, прізвище та ініціали) (підпис)

Рецензент професор кафедри ІСТ, д.т.н., проф., Теленик С. Ф. _____
(посада, науковий ступінь, вчене звання, прізвище та ініціали) (підпис)

Засвідчую, що у цій дипломній роботі немає запозичень з праць інших авторів без відповідних посилань.
Студент _____
(підпис)

Київ –2023

Національний технічний університет України
«Київський політехнічний інститут імені Ігоря Сікорського»

Факультет інформатики та обчислювальної техніки

Кафедра інформатики та програмної інженерії

Рівень вищої освіти – перший (бакалаврський)

Спеціальність – 121 Інженерія програмного забезпечення

Освітньо-професійна програма – Інженерія програмного забезпечення інформаційних систем

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____ Едуард ЖАРІКОВ
(підпис) (ім'я прізвище)
"___"_____2023 р.

ЗАВДАННЯ
на дипломний проєкт студенту

Шаховській Дар'ї Олександрівні
(прізвище, ім'я, по батькові)

1. Тема проєкту Програмне забезпечення для збору програмних метрик застосунків

керівник проєкту Халус Олена Андріївна, ст. викл.
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

затверджені наказом по університету від «31» травня 2023 р. № 2101-с

2. Термін подання студентом проєкту « 17 » червня 2023 року

3. Вихідні дані до проєкту: технічне завдання

4. Зміст пояснювальної записки
1) Аналіз вимог до програмного забезпечення: загальні положення, змістовний опис і аналіз предметної області, аналіз існуючих технологій та технічних рішень, аналіз вимог до програмного забезпечення, постановка задачі. .
2) Моделювання та конструювання програмного забезпечення: моделювання та аналіз програмного забезпечення, архітектура програмного забезпечення, конструювання програмного забезпечення, аналіз безпеки даних.
3) Аналіз та тестування програмного забезпечення: аналіз якості ПЗ, опис процесів тестування, опис контрольного прикладу.
4) Впровадження та супровід програмного забезпечення: розгортання програмного забезпечення, підтримка програмного забезпечення.

5. Перелік графічного матеріалу

1) Схема структурна варіантів використання

2) Схема структурна компонентів ПЗ

3) Схема структурна класів програмного забезпечення

6. Консультанти розділів проєкту

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв

7. Дата видачі завдання «10» березня 2023 року

Календарний план

№ з/п	Назва етапів виконання дипломного проєкту	Термін виконання етапів проєкту	Примітка
1	Вивчення рекомендованої літератури	10.03.2023	
2	Аналіз існуючих методів розв’язання задачі	20.04.2023	
3	Постановка та формалізація задачі	23.04.2023	
4	Проектування структури програмного забезпечення, проектування компонентів	25.04.2023	
5	Обґрунтування вибору використаних технічних засобів	04.05.2023	
6	Розробка програмного забезпечення	15.05.2023	
7	Налагодження програми	20.05.2023	
8	Виконання графічних документів	23.05.2023	
9	Оформлення пояснювальної записки	01.06.2023	
10	Подання ДП на попередній захист	02.06.2023	
11	Подання ДП рецензенту	12.06.2023	
12	Подання ДП на основний захист	17.06.2023	

Студент _____ Дар’я ШАХОВСЬКА
(підпис) (ініціали, прізвище)

Керівник _____ Олена ХАЛУС
(підпис) (ініціали, прізвище)

АНОТАЦІЯ

Пояснювальна записка дипломного проєкту складається з чотирьох розділів, містить 34 таблиці, 32 рисунка та 15 джерел – загалом 61 сторінка.

Дипломний проєкт присвячений розробці програмного забезпечення для збору програмних метрик застосунків.

Мета створення даного дипломного проєкту: полегшення збору та візуалізації метрик застосунків для їх подальшого аналізу за рахунок розробки SDK, яке легко підключається та інтегрується у програмний код.

Об'єкт дослідження: програмне забезпечення для збору програмних метрик застосунків.

Предмет дослідження: процес розроблення програмного забезпечення для збору програмних метрик застосунків.

У розділі «Аналіз вимог до програмного забезпечення» було викладено загальні положення по предметній області та її аналіз, проведено аналіз існуючих технологій та успішних продуктів, наведено діаграму варіантів використання і сформульовано технічні вимоги до програмного забезпечення.

Розділ «Моделювання та конструювання програмного забезпечення» присвячений розробці архітектури програмного забезпечення. У ньому також наведено діаграми бізнес-процесів, діаграму компонентів та діаграму класів, описано структуру бази даних, а також проаналізовано безпеку даних.

У розділі «Аналіз якості та тестування програмного забезпечення» було описано процес тестування компонентів ПЗ та основного контрольного прикладу.

Розділ «Впровадження та супровід програмного забезпечення» присвячений опису алгоритму розгортання програмного забезпечення у Docker.

КЛЮЧОВІ СЛОВА: ПРОГРАМНЕ ЗАБЕЗПЕЧЕННЯ, МЕТРИКА, МОНОЛІТ, .NET, БАЗА ДАНИХ, POSTGRESQL, DOCKER.

ABSTRACT

The explanatory note of the diploma project consists of four sections, contains 34 tables, 32 figures and 15 sources – in total 61 pages.

The diploma project is devoted to the development of software for collecting software application metrics.

The purpose of creating this diploma project: facilitating the collection and visualization of application metrics for their further analysis by developing an SDK that is easily connected and integrated into the program code.

Research object: software for collecting application software metrics.

The subject of the study: the process of developing software for collecting application software metrics.

In the "Analysis of software requirements" section, the general provisions of the subject area and its analysis were outlined, an analysis of existing technologies and successful products was carried out, a diagram of use cases was given, and technical requirements for the software were formulated.

The "Software Modeling and Design" section is devoted to the development of software architecture. It also provides a business process diagrams, component diagram, and class diagram, describes the database structure, and analyzes data security.

In the section "Software quality analysis and testing" the process of testing software components and the main test case was described.

The "Software Deployment and Maintenance" section describes the software deployment algorithm in Docker.

KEYWORDS: SOFTWARE, METRICS, MONOLITH, .NET, DATABASE, POSTGRESQL, DOCKER.

№ з/п	Формат	Позначення	Найменування	Кількість листів	Примітка
1	А4		Завдання на дипломний проєкт	2	
2	А4	КПІ.ІС-9328.045200.01.91	Технічне завдання	11	
3	А4	КПІ.ІС-9328.045200.02.81	Пояснювальна записка	61	
4	А4	КПІ.ІС-9328.045200.03.12	Текст програми	10	
5	А4	КПІ.ІС-9328.045200.04.51	Програма та методика тестування	6	
6	А4	КПІ.ІС-9328.045200.05.34	Керівництво користувача	9	
7	А3	КПІ.ІС-9328.045200.06.99	Схема структурна варіантів	1	
			використань		
8	А3	КПІ.ІС-9328.045200.06.99	Схема структурна компонентів	1	
			системи		
10	А3	КПІ.ІС-9328.045200.06.99	Схема структурна класів	1	
			програмного забезпечення		

					КПІ.ІС-9328.045200.00.90		
Змін.	Арк.	№ докум.	Підп.	Дата			
Розроб.		Шаховська Д.О.			Відомість дипломного проєкту	Літ.	Аркуш 1 / Аркушів
Перевір.		Халус О.А.					
						КПІ ім. Ігоря Сікорського ФІОТ каф. ІПІ гр. ІП-91	
Н.контр.		Ліщук К.І.					
Затв.		Жаріков Е.В.					

Факультет інформатики та обчислювальної техніки
Кафедра інформатики та програмної інженерії

“ЗАТВЕРДЖЕНО”
Завідувач кафедри
_____ Едуард ЖАРІКОВ
“___” _____ 2023 р.

ПРОГРАМНЕ ЗАБЕЗПЕЧЕННЯ ДЛЯ ЗБОРУ ПРОГАМНИХ МЕТРИК ЗАСТОСУНКІВ

Технічне завдання

КПІ.ІС-9328.045200.01.91

“ПОГОДЖЕНО”
Керівник проєкту:
_____Олена ХАЛУС

Нормоконтроль:
_____ Катерина ЛІЩУК

Виконавець:
_____ Дар’я ШАХОВСЬКА

Київ – 2023

ЗМІСТ

1 НАЙМЕНУВАННЯ ТА ГАЛУЗЬ ЗАСТОСУВАННЯ

Назва розробки: Програмне забезпечення для збору програмних метрик застосунків

Галузь застосування:

Наведене технічне завдання поширюється на розробку програмного забезпечення для збору програмних метрик застосунків, котра використовується для полегшення збору програмних метрик та призначена для керівників та розробників програмних продуктів, які бажають впровадити у свій продукт збір програмних метрик для детального розуміння внутрішніх процесів та їх подальшого аналізу

2 ПІДСТАВА ДЛЯ РОЗРОБКИ

Підставою для розробки програмного забезпечення для збору програмних метрик застосунків є завдання на дипломне проєктування, затверджене кафедрою інформатики та програмної інженерії Національного технічного університету України «Київський політехнічний інститут імені Ігоря Сікорського».

3 ПРИЗНАЧЕННЯ РОЗРОБКИ

Розробка призначена для керівників та розробників програмних продуктів, які бажають впровадити у свій продукт збір програмних метрик для детального розуміння внутрішніх процесів та їх подальшого аналізу

Метою розробки є полегшення збору та візуалізації метрик застосунків для їх подальшого аналізу за рахунок розробки SDK, яке легко підключається та інтегрується у програмний код.

4 ВИМОГИ ДО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

4.1 Вимоги до функціональних характеристик

Програмне забезпечення повинно забезпечувати виконання наступних основних функції:

4.1.1 Для користувача:

– перегляд правил для відправки сповіщень;
– візуалізація зібраних метрик;
– перегляд інформації стосовно поточного рантайму;
– перегляд конфігурацій сервера;
– перегляд статусу ендпоінтів.

4.1.2 Додаткові вимоги:

– простий і зрозумілий інтерфейс;
– підтримання різними веб-браузерами.

4.2 Вимоги до надійності

Передбачити захист від некоректних дій користувача. Забезпечити цілісність інформації в базі даних.

4.3 Умови експлуатації

Умови експлуатації згідно СанПін 2.2.2.542 – 96.

4.3.1 Вид обслуговування

Вимоги до виду обслуговування не висуваються.

4.3.2 Обслуговуючий персонал

Вимоги до обслуговуючого персоналу не висуваються.

4.4 Вимоги до складу і параметрів технічних засобів

Для коректної роботи програмного забезпечення рекомендується наступна конфігурація технічних засобів:

– тип процесору: Intel Core i7;
– об‘єм ОЗП: 8-16 Гб;
– підключення до мережі Інтернет зі швидкістю від 100 мегабіт;
– не менше 300 Мб вільної пам’яті.

4.5 Вимоги до інформаційної та програмної сумісності

Програмне забезпечення повинно працювати під управлінням операційних систем сімейства WIN64 або Linux та підтримувати браузери Google Chrome, Microsoft Edge, Mozila Firefox.

4.5.1 Вимоги до вхідних даних

Вхідні дані повинні надходити в результаті використання методів SDK.

4.5.2 Вимоги до вихідних даних

Результати повинні бути представлені в наступному форматі: сповіщення у Telegram-BOT.

4.5.3 Вимоги до мови розробки

Розробку виконати на мові програмування C#.

4.5.4 Вимоги до середовища розробки

Розробку виконати за допомогою середовища Visual Studio 2022.

4.5.5 Вимоги до представленню вихідних кодів

Вихідний код програми має бути представлений у вигляді SDK та Docker-image-ів.

4.6 Вимоги до маркування та пакування

Вимоги до маркування та пакування не висуваються.

4.7 Вимоги до транспортування та зберігання

Вимоги до транспортування та зберігання не висуваються.

4.8 Спеціальні вимоги

Згенерувати Docker-image програмного забезпечення.

5 ВИМОГИ ДО ПРОГРАМНОЇ ДОКУМЕНТАЦІЇ

5.1 Попередній склад програмної документації

У склад супроводжувальної документації повинні входити наступні документи на аркушах формату А4:

– пояснювальна записка;

– технічне завдання;

– текст програми;

– програма та методика тестування;

– керівництво користувача.

Графічна частина повинна бути виконана на аркушах формату А3 та містити наступні документи:

– схема структурна варіантів використання;

– схема структурна компонентів програмного забезпечення;

– схема структурна класів програмного забезпечення.

5.2 Спеціальні вимоги до програмної документації

Програмні модулі, котрі розробляються, повинні бути задокументовані, тобто тексти програм повинні містити всі необхідні коментарі.

6 СТАДІЇ І ЕТАПИ РОЗРОБКИ

№	Назва етапу	Строк	Звітність
1.	Вивчення рекомендованої літератури	10.03.2023	
2.	Аналіз існуючих методів розв’язання задачі	20.04.2023	Пояснювальна записка
3.	Постановка та формалізація задачі	23.04.2023	Технічне завдання
4.	Проектування структури програмного забезпечення, проектування компонентів	25.04.2023	Схема структурна компонентів програмного забезпечення та діаграма класів
5.	Обґрунтування вибору використаних технічних засобів	04.05.2023	Пояснювальна записка
6.	Розробка програмного забезпечення	15.05.2023	Тексти програмного забезпечення
7.	Налагодження програми	20.05.2023	Тексти програмного забезпечення
8.	Виконання графічних документів	23.05.2023	Графічний матеріал проекту
9.	Оформлення пояснювальної записки	01.06.2023	Пояснювальна записка
10.	Подання ДП на попередній захист	02.06.2023	Пояснювальна записка
11.	Подання ДП рецензенту	12.06.2023	Пояснювальна записка
12.	Подання ДП на основний захист	17.06.2023	Пояснювальна записка, презентація

7 ПОРЯДОК КОНТРОЛЮ ТА ПРИЙМАННЯ

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

Пояснювальна записка
до дипломного проєкту

на тему: Програмне забезпечення для збору програмних метрик застосунків

КПІ.ІС-9328.045200.02.81

Київ – 2023

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

IDE	-	Integrated Development Environment – інтегроване середовище розробки
SDK	-	Software development kit
IT	-	Інформаційні технології
MVP	-	Minimum Viable Product
ОС	-	Операційна система
БД	-	База даних
ПЗ	-	Програмне забезпечення
СУБД	-	Система управління базами даних

ВСТУП

В наш час IT-індустрія розвивається досить швидкими темпами. З’являються сотні нових продуктів і відкриваються десятки бізнесів, внаслідок чого стрімко зростає конкуренція, кількість користувачів різних застосунків та веб-сайтів, а також і сам функціонал продуктів, що спричиняє більше навантаження та задіяння більшої кількості ресурсів.

Для того, щоб встигати за розвитком сфери, протистояти конкуренції і зробити продукт зручним у користуванні, керівники компаній вдаються до багатьох різних рішень, таких як: аналіз ринку та конкурентів, запуск реклами та просування у соціальних мережах, внесення новизни, формування конкурентоспроможної ціни та багато всього іншого. Але найголовнішим є повне та детальне розуміння свого продукту, особливо коли стаються якісь збої або несправності. Це потрібно для зручності користування продуктом користувачами та якісного розвитку продукту, як з боку розробки так і зі сторони бізнесу.

Для вирішення цієї проблеми в дипломному проєкті пропонується програмне забезпечення для збору та візуалізації програмних метрик, а саме метрик представлених в формі часових рядів. А також для надання можливості легко та зручно збирати дані, які необхідні для дата аналізу та аналізу ефективності роботи застосунку.

Мета створення даного дипломного проєкту: полегшення збору та візуалізації метрик застосунків для їх подальшого аналізу за рахунок розробки SDK, яке легко підключається та інтегрується у програмний код.

Практичне значення: результатом роботи є програмне забезпечення для збору програмних метрик застосунків задля детального розуміння внутрішніх процесів, які відбуваються у програмі.

1 АНАЛІЗ ВИМОГ ДО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

1.1 Загальні положення

Даний дипломний проєкт присвячений розробці програмного забезпечення, яке повинно полегшити збір і візуалізацію програмних метрик застосунків для подальшого розвитку та вдосконалення продукту.

Перш ніж розглянути всі переваги використання систем даного типу, зануримось у загальні визначення, які стосуються предметної області.

Метрика – це величина, яка дозволяє кількісно виміряти певні міри програмного забезпечення. Метрики бувають декількох типів: базові або їх ще називають абсолютні, тобто ті які не потребують додаткового обрахунку, та обчислені, які на відміну від базових піддаються розрахункам для їх подальшого аналізу.

За типом даних, що збираються, класифікують такі типи метрик:

– метрики кількісних даних: ці метрики вимірюють числові значення та кількісні атрибути;

– метрики категоріальних даних: дані метрики застосовуються до категоріальних даних, які представляють різні класи чи категорії;

– метрики бінарних даних: метрики застосовуються до даних, які приймають тільки два значення так/ні;

– метрики текстових даних: ці метрики використовуються ля аналізу текстових даних, наприклад наборів слів, речень, текстових документів;

– метрики часових рядів (time-series): даний тип метрик використовується для аналізу та оцінки даних впорядкованих у часі. Time-series метрики найбільше підходять для вирішення задач поставлених до програмного забезпечення дипломного проєкту. Тому що завдяки цьому типу даних можна відслідковувати зміну даних у часі, відслідковувати взаємодію різних показників та знаходити певні закономірності, що значно полегшує процес знаходження помилок в коді.

Цілі використання метрик можуть варіюватися в залежності від типу продукту, однак все одно можна виділити основні:

– вимірювання якості: зібрані величини допомагають оцінити на скільки продукт якісний, виявити наявні або потенційні проблеми, виявити на скільки продукт задовольняє очікування користувачів;

– управління процесом розробки: метрики дозволяють відслідковувати прогрес розробки, приймати рішення щодо використання певної кількості ресурсів;

– прийняття рішень: метрики надають фактичну інформацію про властивості програмного забезпечення, на основі яких можуть прийматись обґрунтовані рішення. Метрики можуть бути задіяні у оцінці АБ-тестів, в результаті яких обирається краща версія продукту;

– полегшення та прискорення процесу пошуку несправностей: у випадку появи критичних помилок використання метрик дозволяє ефективно знайти проблему та усунути її в коротші терміни [1].

Враховуючи вище описані цілі використання метрик, можемо дійти висновку, що метрики потрібні та важливі на всіх етапах розробки та підтримки продукту, як для великих корпорацій, так і для нових стартап-проєктів або навіть домашніх пет-проєктів.

1.2 Змістовний опис і аналіз предметної області

В сучасних умовах розвитку ІТ-продуктів на ринку, коли з великою швидкістю збільшується функціонал і кількість користувачів, надзвичайно популярним і важливим є збір програмних метрик, їх візуалізація, а також подальший аналіз, так як це надає важливу інформацію розробнику або тестувальнику про те, на скільки швидко та якісно функціонує їх застосунок, на скільки він задовольняє такі потреби користувачів як: швидкість, безперебійність та доступність. Для замовників або керівників продуктів це також має певні плюси, одним з них є можливість побачити та правильно розрахувати навантаження на БД або сервер з економним використанням

ресурсу і коштів. Також вкрай важливою перевагою збору програмних величин є можливість легко та швидко виявити проблеми у продукті, що значно зменшує час їх виправлення і надання доступу користувачу до правильно працюючого програмного забезпечення [2].

Існує чимало реалізацій систем даного типу, але вони мають декілька недоліків. Серед них є такі як: відсутність власного інструменту візуалізації зібраних даних або інтеграції з існуючими сервісами, які пропонують даний набір інструментів, відсутність відправки сповіщень, коли якась з метрик переходить визначений максимум або мінімум.

У дипломному проєкті пропонується покращений варіант програмного забезпечення якраз за рахунок легкості налаштування та підключення SDK у програму, можливості інтеграції з відомими сервісами для візуалізації та зручного аналізу, а також з відправкою оповіщень.

1.3 Аналіз існуючих технологій та успішних ІТ-проєктів

Проаналізуємо відоме на сьогодні алгоритмічне забезпечення у даній області та технічні рішення, що допоможуть у реалізації програмного забезпечення для збору програмних метрик застосунків. Далі будуть розглянуті допоміжні програмні засоби, засоби розробки та готові програмні рішення.

1.3.1 Аналіз відомих алгоритмічних та технічних рішень

Як правило, в розробці систем збору та моніторингу даних використовуються загальні алгоритми, тому немає потреби робити аналіз алгоритмічних рішень. Більш детально розглянемо відомі архітектури застосунків, так як реалізація дипломного проєкту будується на архітектурних рішеннях.

До найбільш відомих типів архітектури для система данного типу відносять: монолітну та мікросервісну архітектури. Розглянемо детальніше кожен з цих шаблонів, порівняємо їх переваги та недоліки в реалізації

програмного забезпечення представленого в дипломному проєкті та оберемо найбільш відповідну під вимоги.

Монолітна архітектура програмного забезпечення являє собою єдиний компонент, в якому знаходиться весь функціонал застосунку. Перевагами даного типу архітектури є:

– легкість реалізації та відлагоджування (debugging), так як всі компоненти системи знаходяться в єдиній кодовій базі;

– легкість розгортання: витрачається менше налаштувань та конфігурацій для розгортання ПЗ; Повноцінний застосунок розгортається за допомогою єдиного контейнеру розгортання (deployment pipeline);

– продуктивність: весь функціонал знаходиться в одному сервісі або застосунку, за рахунок чого всі компоненти системи при взаємодії мають низький час відгуку через відсутність викликів через мережу до інших компонентів.

Враховуючи вищесказане, можна зробити висновок, що такий вид архітектури підходить для MVP (Minimum Viable Product) продуктів або невеликих застосунків. Але попри всі ці плюси монолітна архітектура має і значні мінуси, які проявляються при збільшенні розміру та складності застосунку, такі як:

– масштабування: є проблематичним через те, що замість масштабування одного компонента доводиться масштабувати весь застосунок;

– розгортання: відсутність можливості розгортання окремого компонента;

– надійність: якщо виникає проблема у конкретному компоненті, це може вплинути на роботу цілого застосунку;

– час розробки: чим більшою стає кодова база, тим важче її підтримувати. Це спричинено високою зв’язаністю компонентів і великим об’ємом застосунку;

– відсутність гнучкості при виборі технологій: застосунок обмежений тими технологіями, які вже використовує. Це унеможливлює зміну та покращення технології на рівні конкретного модуля, так як це потрібно робити на рівні цілого застосунку.

Наступним типом архітектури є мікросервісна архітектура. Вона представлена у вигляді невеликих незалежних компонентів-сервісів, кожен з яких виконує певну частину функціоналу продукту. Цей тип архітектури має суттєві плюси у використанні, а саме:

– розподіл на окремі сервіси: кожен компонент має свій окремий функціонал, який розробляється та розгортається незалежно від інших компонентів застосунку;

– масштабування: як було зазначено вище, кожен мікросервіс – це окремий компонент, саме це дозволяє масштабувати один з них, не вносячи змін до інших;

– гнучкість: кожен з мікросервісів може бути реалізований з використанням різних технологій та мов програмування, що надає можливість обрати найбільш відповідні інструменти для реалізації функціоналу кожного з них;

– легкість підтримки кодової бази застосунку.

До мінусів можна віднести такі характеристики, як:

– більший час відгуку: комунікація між мікросервісами відбувається через мережеві виклики, що може спричинити більший час відгуку;

– складність управління та тестування: чим більше стає сервісів, тим ними складніше управляти;

– більші затрати на ресурси: кожен мікросервіс потребує виділення під себе власних ресурсів та інфраструктури, що може призвести до збільшення витрат на обслуговування.

Беручи до уваги наведені переваги та недоліки мікросервісної архітектури, можна сказати що вона добре підходить для хмарних рішень,

але не on-premise. Одна з таких причин це розгортання, яка ускладнюється наявністю багатьох компонентів. У випадку on-premise застосунків, розгортання відбувається користувачем вручну, і дана архітектура робить цю задачу дуже складною. Але, враховуючи функціонал даного програмного забезпечення та плюси обох патернів, в даному випадку краще застосувати гібридну версію двох даних архітектур.

1.3.2 Аналіз допоміжних програмних засобів та засобів розробки

Для реалізації серверної застосунків використовують досить багато різних високорівневих мов програмування, таких як: Go [4], Java [5], Python [6], C#(ASP.NET) [7]. Перелічені мови програмування мають багато плюсів, але мінуси також наявні. Тому в залежності від поставлених задач застосовується та чи інша мова.

Почнемо з першої у списку мови – Go, також відома як Golang. Це відносно молода мова програмування, звідси і походять її основний недолік, такий як відсутність підтримки деякого функціоналу наявного в інших мовах, які наведені у списку вищі, а саме: обмежені можливості абстракції та повторного використання коду, відсутність наслідування класів та перегрузки операторів, досить невелика та нерізноманітна база бібліотек і таке інше. Серед плюсів мови можна визначити такі характеристики як: простий та зрозумілий синтаксис, що робить її легкою для використання, підтримка кросплатформеності, що дозволяє запускати застосунки на різних операційних системах, висока продуктивність програм, так як Go компілюється в машинний код.

Розглядаючи переваги мови програмування Python можна виділити те, що вона підтримає багато бібліотек, які можна застосовувати до найрізноманітніших задач, що значно полегшує розробку. Але в цьому є й мінус – Python занадто залежить від сторонніх бібліотек. Також одним з недоліків є те, що мова не строго типізована, а це значить, що для знаходження помилок у коді доведеться виділити багато часу.

Звертаючи увагу на Java можна виокремити те, що мова досить гарно підходить для написання масштабованих сервісів, у ній наявна підвищена безпека, так як віртуальна машина Java перевіряє байт-код Java, щоб запобігти поширенню вірусів. Але, напевно, найбільшими її недоліками є: Java працює повільно через використання віртуальної машини для компіляції, проблеми з очищенням пам’яті, блокування потоків та погані налаштування кешування.

Якщо розглядати C#(ASP.NET), як мову програмування для серверної частини застосунку, то можна виділити такі переваги: об’єктно-орієнтована кросплатформена мова, яка підходить для вирішення великого спектру задач, має зворотну сумісність та цілісність. Серед мінусів можна виокремити те, що деякі технології .NET Framework недоступні в поточній версії .NET Core, але для написання даного дипломного проєкту наявне все необхідне. Тому серверна частина програмного забезпечення буде написана саме на мові програмування C#.

Головним середовищем розробки програмного забезпечення буде обрано Visual Studio 2022, так як це інтегроване середовище розробки, що розробляється та підтримується Microsoft, як і сама мова програмування C#. Visual Studio 2022 надає широкий набір інструментів, що робить процес написання програм ефективним та зручним: автодоповнення коду, швидкий пошук, керування проектами та багато чого іншого.

Для розгортання програмного забезпечення буде обрано Docker [8], так як він має декілька значних переваг у використанні: Docker-контейнери легко переносяться між різними операційними системами і хостами, що дозволяє не перекомпільовувати або не змінювати код, Docker-контейнери забезпечують ізоляцію застосунків від інших процесів, що попереджає конфлікти та забезпечує безпечність роботи програми. Також Docker-контейнери досить швидко розгортаються та потребують менше ресурсів на відміну від віртуальних машин [9].

1.3.3 Аналіз відомих програмних продуктів

Як було зазначено вище, на просторах інтернету можна знайти багато аналогів систем для збору та моніторингу даних, тож оберемо найвідоміші і розглянемо їх більш детально.

Graphite – це пасивна time-series база даних з власною мовою запитів та набором інструментів для візуалізації. Головними функціональними можливостями є :

– зберігання зібраних даних локально на жорсткому диску;

– кожна серія даних зберігається у новому файлі і нові дані перезаписують старі дані через деякий проміжок часу;

– наявність власної мови запитів;

– наявність власних засобів візуалізації зібраних даних [10].

InfluxDB – це time-series база даних з відкритим кодом з комерційною можливістю для масштабування та кластеризації. До головного функціоналу продукту входять:

– модель даних має пари ключ-значення як мітки, які називаються тегами, також існує другий тип міток, який називається полями;

– підтримує часові мітки з точністю до наносекунд та типи даних: int64, float64, bool та string;

– зберігання зібраних даних локально на жорсткому даних;

– розподілене сховище даних, хостинг та підтримка доступні лише за комерційною підпискою [11].

Nagios – це система моніторингу, яка була створена у 90-х роках і називалась NetSaint. Головний фунціонал включає в себе:

– не має сховища даних окрім поточного стану, сховище даних можливе лише через зовнішні розширення;

– сервера є самостійними, всі конфігурації відбуваються через файл;

– має можливість відправки сповіщень [12].

Для порівняння дипломної роботи з аналогом можна скористатись таблицею 1.1.

Таблиця 1.1 – Порівняння з аналогом

Функціонал	Дипломний проєкт	Graphite	InfluxDB	Nagios	Пояснення
Безкоштовне ПЗ	+	+	-	-	Весь функціонал ПЗ безкоштовний
Система з відкритим кодом	+	+	+	+	Система має код у відкритому для всіх доступі
Легкість налаштування SDK	+	+	-	+	Система легка у налаштуванні
Збір time-series метрик	+	+	+	-	Система збирає time-series метрики
Засоби візуалізації	+	+	-	-	Система надає можливість візуалізувати зібрані дані
Відправлення сповіщень	+	-	-	+	Система відправляє оповіщення, коли одна з метрик переходить визначений максимум/мінімум

1.4 Аналіз вимог до програмного забезпечення

Головною функцією програмного забезпечення є збір програмних метрик застосунків, більше функцій можна побачити на рисунку 1.1.

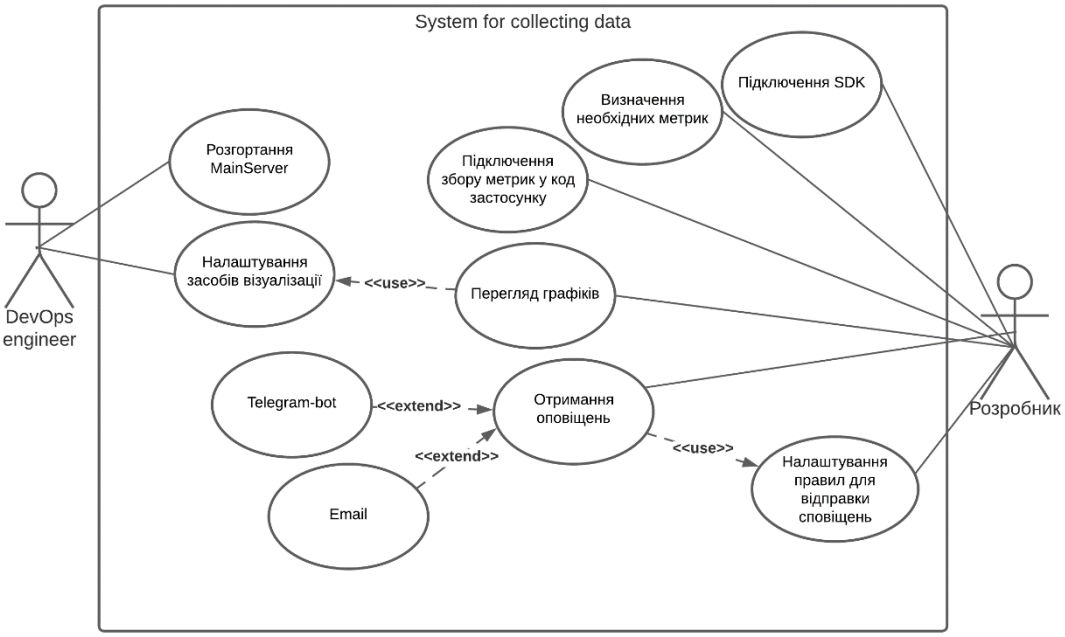


Рисунок 1.1 – Діаграма варіантів використання

В таблицях 1.2 - 1.11 наведені варіанти використання програмного забезпечення.

Таблиця 1.2 - Варіант використання UC-1

Use case name	Підключення SDK
Use case ID	UC-01
Goals	Підключити бібліотеки
Actors	Розробник
Trigger	Розробник отримав завдання підключити SDK в програму
Pre-conditions	-
Flow of Events	Розробник відкриває NuGet Package Manager, завантажує відповідний NuGet і проводить налаштування конфігурацій.
Extension	У випадку налаштування неправильних конфігурацій не вдасться підключити SDK
Post-Condition	SDK підключений

Таблиця 1.3 - Варіант використання UC-2

Use case name	Визначення необхідних метрик
Use case ID	UC-02
Goals	Визначити метрики для збору даних
Actors	Розробник
Trigger	Розробник підключив SDK
Pre-conditions	-
Flow of Events	Розробник аналізує програмне забезпечення та обирає метрики, які треба збирати
Extension	У випадку неправильного вибору метрик, отримані результати не будуть нести корисної інформації
Post-Condition	Обрані метрики для підключення

Таблиця 1.4 - Варіант використання UC-3

Use case name	Підключення збору метрик у код застосунку
Use case ID	UC-03
Goals	Підключити збір метрик у програмний код
Actors	Розробник
Trigger	Розробник обрав метрики для збору даних
Pre-conditions	-
Flow of Events	Розробник використовує методи з SDK у програмному коді застосунку
Extension	У випадку неправильного підключення збору метрик, метрики не будуть збиратися або будуть збиратися не вірно
Post-Condition	Метрики успішно записуються в базу даних

Таблиця 1.5 - Варіант використання UC-4

Use case name	Налаштування правил для відправки сповіщень
Use case ID	UC-04
Goals	Налаштувати правила для відправки сповіщень
Actors	Розробник
Trigger	Розробник отримав завдання налаштувати відправку сповіщень

Продовження таблиці 1.5

Pre-conditions	-
Flow of Events	Розробник створює правила відправлення оповіщень, використовуючи кінцеву точку HTTP Alert Manager-а
Extension	У випадку неправильного налаштування правил оповіщення не будуть відправлятись
Post-Condition	Правила для отримання оповіщень налаштовані

Таблиця 1.6 - Варіант використання UC-5

Use case name	Отримання оповіщень
Use case ID	UC-05
Goals	Отримувати повідомлення
Actors	Розробник
Trigger	Одна з зібраних метрик підпала під правила прописані програмістом
Pre-conditions	-
Flow of Events	Розробник отримує оповіщення про те, що одна з метрик підпадає під прописані правила(перевищує або занадто занижена відносно норми), переходить до повідомлення і бачить інформацію про метрику та її величину
Extension	У випадку не підключення до інтернету оповіщення не буде отримано
Post-Condition	Отримано оповіщення

Таблиця 1.7 - Варіант використання UC-6

Use case name	Telegram-bot
Use case ID	UC-06
Goals	Отримувати повідомлення в Telegram-bot
Actors	Розробник
Trigger	Одна з зібраних метрик підпала під правила прописані програмістом
Pre-conditions	-

Продовження таблиці 1.7

Flow of Events	Розробник отримує оповіщення про те, що одна з метрик підпадає під прописані правила(перевищує або занадто занижена відносно норми), переходить у Telegram-bot і бачить інформацію про метрику та її величину
Extension	У випадку не підключення до інтернету оповіщення не буде отримано
Post-Condition	Отримано оповіщення у Telegram-bot

Таблиця 1.8 - Варіант використання UC-7

Use case name	Email
Use case ID	UC-07
Goals	Отримувати повідомлення в Email
Actors	Розробник
Trigger	Одна з зібраних метрик підпала під правила прописані програмістом
Pre-conditions	-
Flow of Events	Розробник отримує оповіщення про те, що одна з метрик підпадає під прописані правила(перевищує або занадто занижена відносно норми), заходить в електронну пошту і бачить інформацію про метрику та її величину
Extension	У випадку не підключення до інтернету оповіщення не буде отримано
Post-Condition	Отримано оповіщення на email

Таблиця 1.9 - Варіант використання UC-9

Use case name	Розгортання MainServer
Use case ID	UC-08
Goals	Розгорнути MainServer
Actors	DevOps Engineer
Trigger	DevOps Engineer отримав завдання розгорнути MainServer
Pre-conditions	-
Flow of Events	DevOps Engineer скачує docker-image та розгортає його у Docker

Продовження таблиці 1.9

Extension	У випадку неправильного розгортання серверу DevOps Engineer отримає відповідну помилку
Post-Condition	MainServer розгорнутий у Docker

Таблиця 1.10 - Варіант використання UC-9

Use case name	Налаштування засобів візуалізації
Use case ID	UC-09
Goals	Налаштувати засоби візуалізації
Actors	DevOps Engineer
Trigger	DevOps Engineer отримав завдання налаштувати засоби візуалізації
Pre-conditions	-
Flow of Events	DevOps Engineer підключає та налаштовує засоби візуалізації
Extension	У випадку неправильного розгортання серверу DevOps Engineer отримає відповідну помилку
Post-Condition	Засоби візуалізації налаштовані

Таблиця 1.11 - Варіант використання UC-10

Use case name	Перегляд графіків
Use case ID	UC-10
Goals	Переглянути графіки
Actors	Розробник
Trigger	Пройшов деякий час збору даних
Pre-conditions	-
Flow of Events	Розробник заходить у інструменти візуалізації, переходить у вкладку Dashboard, обирає необхідну метрику та обирає за який час хоче переглянути дані
Extension	У випадку не підключення до Інтернету графіки не будуть відображатись
Post-Condition	Графіки переглянуті

1.4.1 Розроблення функціональних вимог

Програмне забезпечення розділене на модулі. Кожен модуль має свій певний набір функцій. В таблицях 1.12 – 1.15 наведений опис функціональних вимог до програмного забезпечення. Матрицю трасування вимог можна побачити на рисунку 1.2.

Таблиця 1.12 – Функціональна вимога FR-1

Назва	Збір програмних метрик
Опис	Система повинна надавати можливість збирати програмні метрики застосунку шляхом підключення та налаштування SDK

Таблиця 1.13 – Функціональна вимога FR-2

Назва	Запит даних через HTTP-API
Опис	Система повинна надавати можливість виконувати запити через HTTP-API.

Таблиця 1.14 – Функціональна вимога FR-3

Назва	Відправка оповіщень
Опис	Система повинна надавати можливість відправляти оповіщення про те, коли одна з метрик попадає під прописані правила.

Таблиця 1.15 – Функціональна вимога FR-4

Назва	Візуалізація даних
Опис	Система повинна надавати можливість візуалізувати зібрані дані

	FR-1	FR-2	FR-3	FR-4
UC-1	+			
UC-2	+			
UC-3	+			
UC-4			+	
UC-5			+	
UC-6			+	
UC-7			+	
UC-8	+			
UC-9		+		+

Рисунок 1.2 – Матриця трасування вимог

1.4.2 Розроблення нефункціональних вимог

Програмне забезпечення повинно бути:

– легким у використанні;

– зрозумілим для користувача;

– підтримуватись на різних браузерах.

1.5 Постановка задачі

Розробка призначена для керівників та розробників програмних продуктів, які бажають впровадити у свій продукт збір програмних метрик для детального розуміння внутрішніх процесів та їх подальшого аналізу.

Розробка ставить перед собою досягнення наступних цілей:

– надати можливості легко підключити та розгорнути програмне забезпечення;

– забезпечити збір програмних метрик;

– надати можливість отримувати сповіщення на електронну пошту або у телеграм-бот у випадку, коли одна з метрик підпала під правила;

– забезпечити можливість візуалізації зібраних даних.

Мета створення програмного забезпечення: полегшення збору та візуалізації метрик застосунків для їх подальшого аналізу за рахунок розробки SDK, яке легко підключається та інтегрується у програмний код.

Програма призначена для збору та візуалізації програмних метрик застосунків.

Висновки до розділу

В першому розділі дипломного проєкту було розглянуто загальні відомості по предметній області, наведено загальні теоретичні визначення, визначено основні цілі застосування метрик, їх типи. Також був наведений змістовний аналіз предметної області, виділено проблематику та актуальність теми дипломного проєкту.

Було проаналізовано відомі архітектурні рішення та обрано гібридну версію архітектури моноліту та мікросервісів для розробки програмного забезпечення для збору програмних метрик додатків. Крім того було зроблено аналіз найбільш вживаних мов програмування для реалізації серверної частини застосунків та обрано мову програмування C# для написання програмного забезпечення.

Також було розглянуто схожі відомі програмні продукти, надано короткий опис, виділені переваги та недоліки по кожному з них, наведено порівняльну таблицю.

У розділі була наведена Use Case діаграма з детальним описом кожного з варіантів використання. До того ж було виділено функціональні вимоги до програмного забезпечення, на основі яких була побудована матриця трасування вимог. Були зазначені нефункціональні вимоги та виділено постановку задачі.

2 МОДЕЛЮВАННЯ ТА КОНСТРУЮВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

2.1 Моделювання та аналіз програмного забезпечення

Для опису головного бізнес-процесу програмного забезпечення використовується BPMN модель представлена на рисунку 2.1.

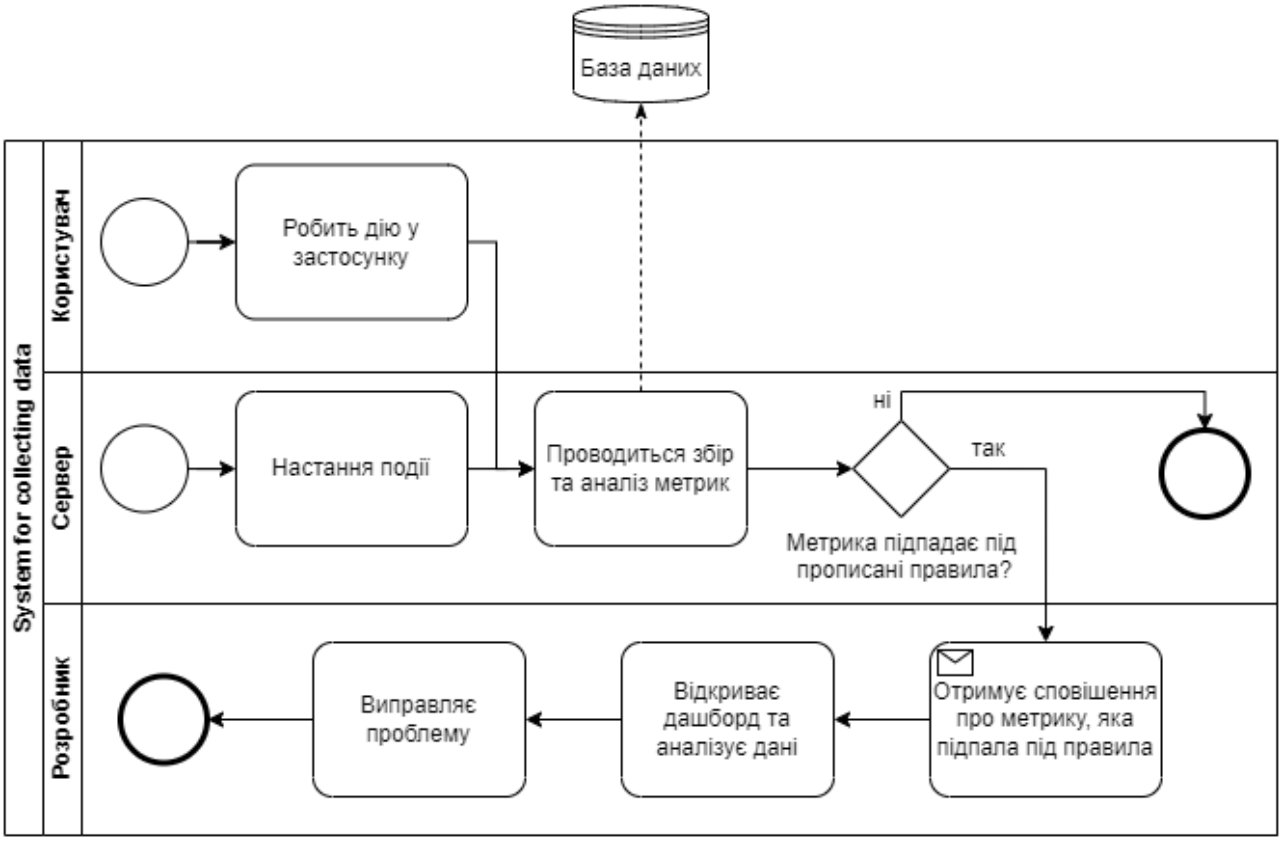


Рисунок 2.1 – Схема головного бізнес-процесу

Опис головного бізнес-процесу, а саме послідовності збору програмних метрик:

– користувач робить якусь дію у застосунку або на сервері стається якась подія, яка є тригерною для збору метрик;

– на сервері проводиться збір метрик, тобто запис у базу, та аналіз;

– якщо метрика підпадає під визначені правила, то розробник отримує оповіщення про це;

– розробник відкриває дашборд та аналізує зібрані дані, шукаючи причину того, чому метрика перейшла визначений мінімум або максимум;

– розробник, знайшовши проблему, виправляє її і проводить відповідні тестування.

Для опису бізнес-процесу отримання статусу роботи застосунку використовується BPMN модель представлена на рисунку 2.2.

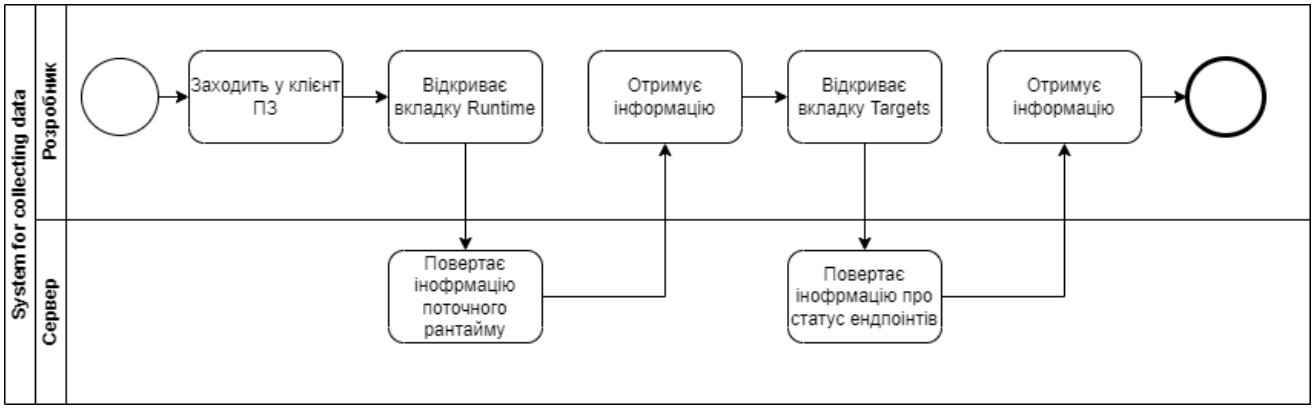


Рисунок 2.2 – Схема бізнес-процесу отримання статусу роботи застосунку

Опис процесу отримання статусу роботи застосунку:

– розробник заходить у клієнт програмного забезпечення;

– зі спадного списку Status обирає вкладку Runtime;

– переглядає інформацію поточного рантайму, а саме: коли запустився сервер, статус останнього перезавантаження, коли відбувалось останнє перезавантаження та кількість днів протягом яких дані зберігаються у базі;

– з спадного списку Status обирає вкладку Targets;

– переглядає інформацію по підключеним таргетам, тобто по статусу ендпоінтів, коли відбувалося останнє стягування данних, як довго воно тривало та наявні помилки на даний момент.

Для опису бізнес-процесу налаштування правил відправки сповіщень використовується BPMN модель представлена на рисунку 2.3.

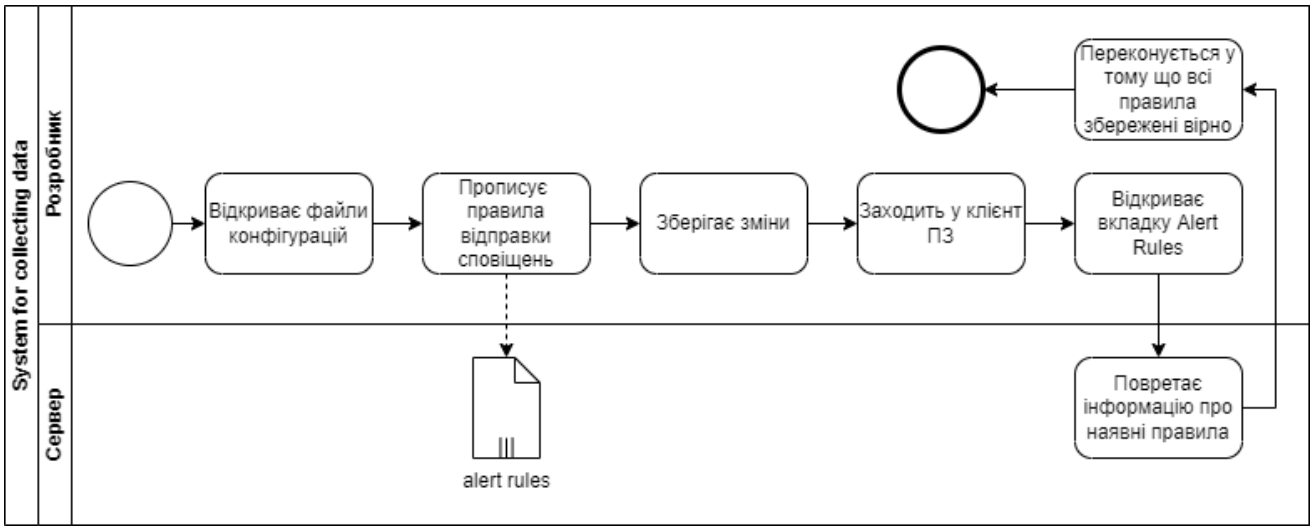


Рисунок 2.3 – Схема бізнес-процесу налаштування правил відправки повідомлень

Опис процесу налаштування правил відправки повідомлень:

– розробник відкриває файл конфігурацій, у якому міститься правила відправки сповіщень;

– для обраних метрик прописує правила, заповнюючи такі поля, як: ім’я сповіщення, умова відправки сповіщення, час, протягом якого буде відбуватись обробка правила, текст самого повідомлення;

– зберігає внесені зміни, у системі оновлюється файл з конфігураціями;

– розробник заходить у клієнт програмного забезпечення;

– з спадного списку Status обирає вкладку Alert Rules;

– сервер повертає запитувану інформацію

– переконується у тому, що всі правила, які були додані, наявні у системі.

Розглянемо діаграму послідовності зображену на рисунку 2.4.

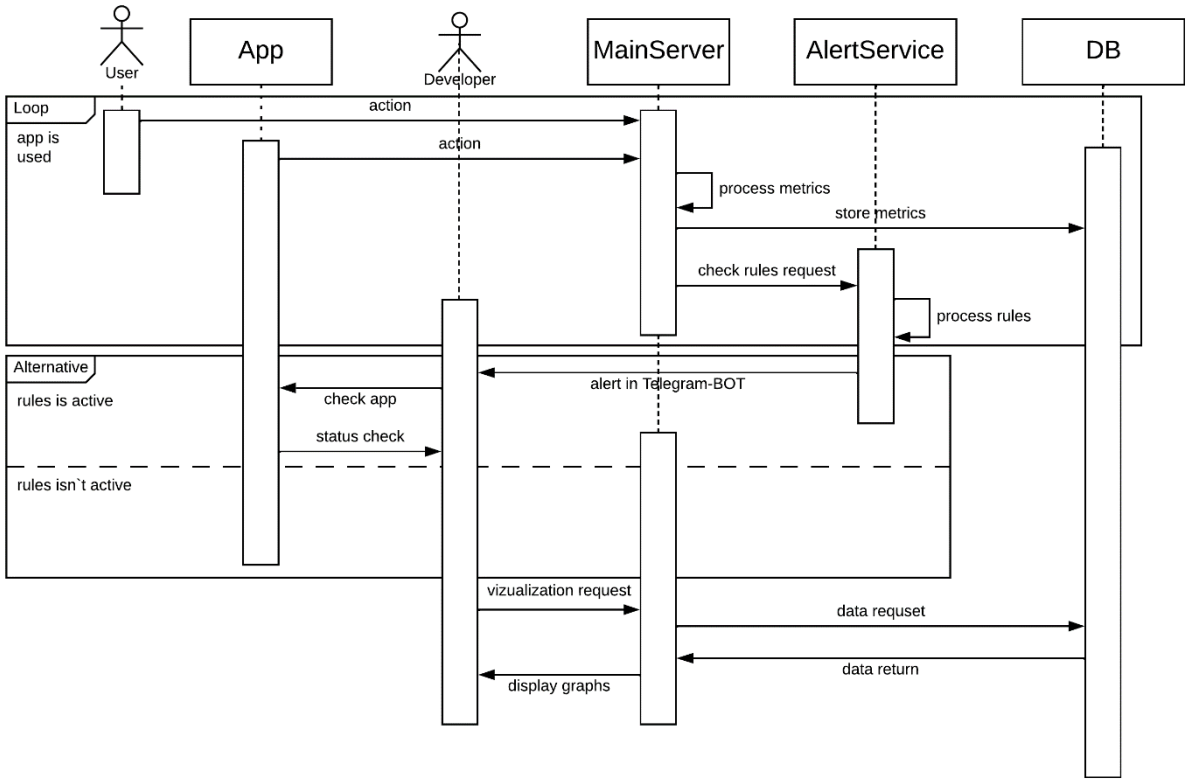


Рисунок 2.4 – Діаграма послідовності

Опишемо діаграму послідовності. Користувач заходить у застосунок, у який підключено збір програмних метрик та робить певну дію, на яку навішено тригер збору метрик, наприклад завантаження певної сторінки або безпосередньо у самому цьому застосунку виконується якась дія на яку теж відбувається збір метрик, наприклад запит до бази даних. Проводиться збір метрик і відправляється запит на головний сервер. На сервері відбувається обробка метрик відправляється запит отриманих даних у базу даних та відправляється запит на сервіс обробки сповіщень. AlertService опрацьовує даний запит і перевіряє чи метрика підпадає під прописані правила. Даний процес відбувається постійно, так як застосунок знаходиться у використанні, тому ці дії відбуваються циклічно.

Якщо правила виявились активними, то сервіс відправляє alert розробнику, у якому міститься інформація про те яка метрика підпала під правила та відповідний текст повідомлення. Отримання даного сповіщення свідчить про те, що у застосунку відбуваються певні процеси, які не є

коректними. Тому розробник аналізує текст повідомлення і визначає те місце у коді, яке могло спровокувати такий результат. Після проведення необхідних тестів або знаходження і виправлення помилок, система відправляє статус, про те що недоліки виправлені. У випадку, коли метрика не підпадає під правила, то сповіщення не відправляються, а це значить, що застосунок знаходиться у задовільному стані.

Розробнику необхідно переглянути візуалізацію певної метрики, тому він відкриває вкладку Dashboard, обирає метрику та час, за який він хоче переглянути зміни. Відповідно обрані фільтри відправляються запитом на головний сервер, який в свою чергу відправляє запит на витягування необхідних даних з бази даних. База даних повертає запитувані дані і відповідно сервер повертає візуалізацію розробнику. В залежності від побаченого розробник приймає рішення про свої подальші дії.

2.2 Архітектура програмного забезпечення

Програмне забезпечення реалізоване використовуючи гібрид монолітної і мікросервісної архітектури (рисунок 2.5).

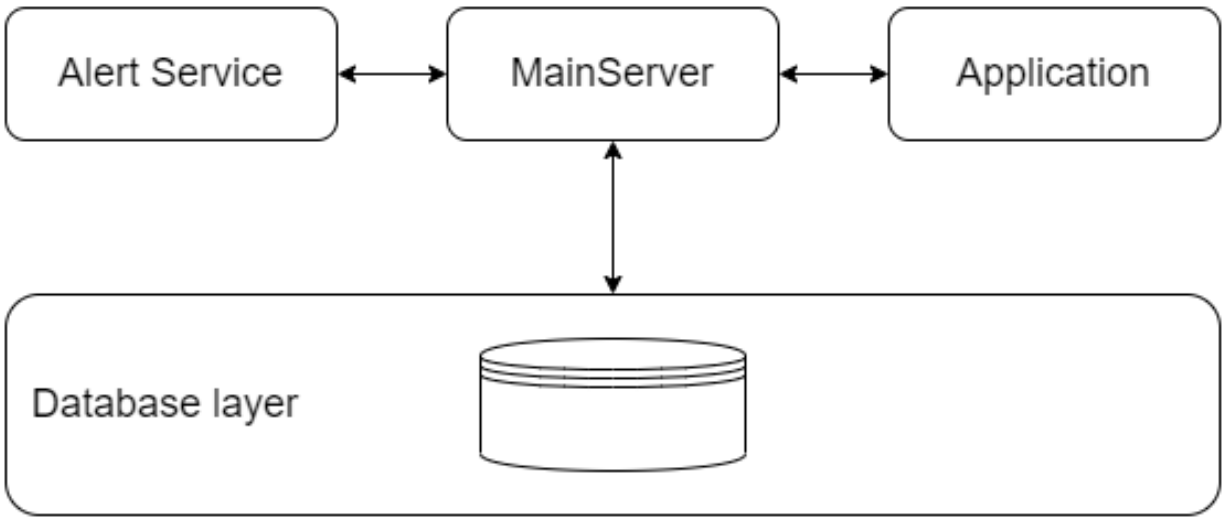


Рисунок 2.5 – Архітектура програмного забезпечення

Дане представлення архітектури включає в себе переваги обох патернів.

Від моноліту були взяті такі плюси, як: легкість реалізації , debugging та продуктивність за рахунок того, що всі компоненти знаходяться в одному місці.

Переваги наявні у даному програмному забезпеченні і характерні для мікросервісної архітектури це: гнучкість і масштабованість, так як різний тип функціоналу знаходиться у різних компонентах, незалежне розгортання, що дозволяє досить швидко впровадити зміни у програмний код.

Розгортання даного забезпечення є також досить легким, так як використання сервісу для відправки сповіщень не є обов’язковим, користувачі самі приймають рішення про його використання.

Також для полегшення розробки і підтримки ПЗ та забезпечення слабкої зв’язаності коду, функціонал був розбитий на модулі. Структурна схема компонентів системи відображена на рисунку 2.6.

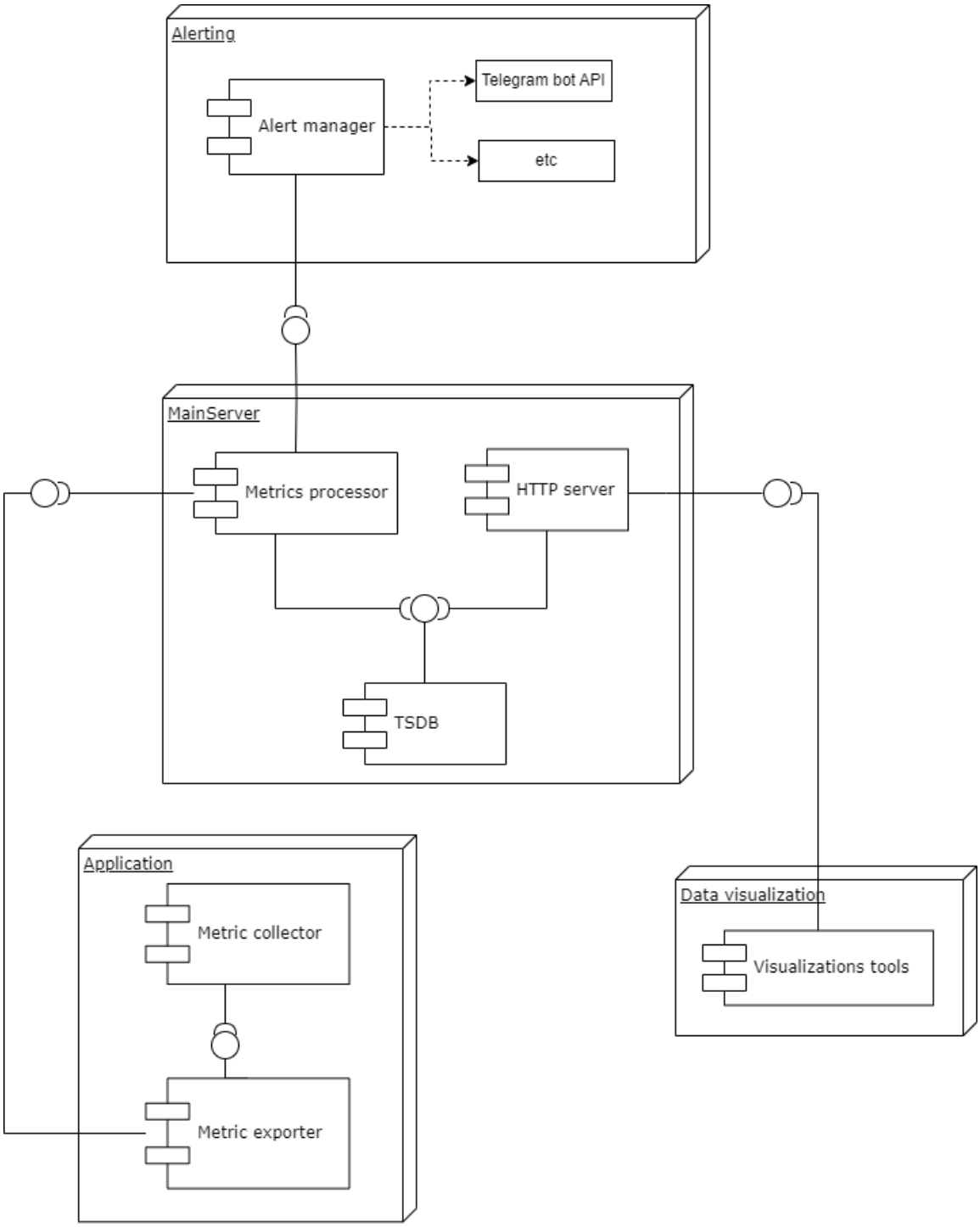


Рисунок 2.6 – Діаграма компонентів ПЗ

Як можна бачити на діаграмі наведеній вище у даному програмному забезпеченні присутні 4 компоненти, такі як: MainServer, Application, Alerting, Data Vizualization. Опис основних компонентів системи наведено в таблиці 2.1.

Таблиця 2.1 – Опис основних компонентів

Назва	Опис
MainServer	Модуль, що відповідає за отримання, обробку, та зберігання метрик, а також надає API для керування сервером і отримання оброблених даних з метою подальшої візуалізації
Data visualization	Модуль, що відповідає за візуалізацію оброблених даних
Alerting	Модуль, що відповідає за відправку оповіщень через email, або інші сторонні сервіси, такі як Telegram Bot
Application	Модуль, який знаходиться на стороні розробника і відповідає за першочерговий збір та зберігання метрик

У системі наявний головний модуль MainServer, який в свою чергу взаємодіє з усіма іншим компонентами, такими як: Data visualization, Alerting, Application, що в свою чергу не взаємодіють між собою.

У таблиці 2.2 наведено більш детальний опис компонентів.

Таблиця 2.2 – Детальний опис компонентів системи

Назва	Опис
Metrics processor	Модуль, що відповідає за отримання метрик від застосунку розробника, і їх подальшу обробку
HTTP server	API для керування сервером і отримання оброблених даних з метою подальшої візуалізації
TSDB	База даних для зберігання даних у time-series форматі
Metrics collector	Модуль на стороні застосунку розробника, що відповідає за збір та зберігання метрик

Продовження таблиці 2.2

Metrics exporter	Модуль на стороні застосунку розробника, що відповідає за відправлення зібраних метрик на основний сервер
Alert manager	Модуль, що відповідає за відправку оповіщень через email, або інші сторонні сервіси, такі як Telegram Bot
Visualization tools	Модуль, що відповідає за візуалізацію зібраних даних

Опишемо взаємодію підкомпонентів системи. Vizualiation tools відправляє HTTP запит на HTTP server, який в свою чергу іде в базу даних – TSDB, щоб повернути відповідь по запитуваним даним. У Metric Processor відбувається обробка метрик, відправляється запит на Alert Manager, коли одна з них підпадає під правила то користувачу відправляється повідомлення у Telegram. Application являє собою застосунок користувача, у якому наявні два модулі Metrics collector та Metrics exporter. Metrics collector збирає та зберігає дані локально до тих пір, поки Metrics exporter не прийде за тим, щоб забрати ці дані та відправити їх на компонент Metrics processor головного серверу, який у свою чергу проведе запис отриманих даних у TSDB.

2.3 Конструювання програмного забезпечення

Для написання серверної частини було обрано мову C#(.NET), так як дана мова програмування має декілька досить вагомих переваг, а саме: кросплатформеність, об’єкто-орієнтованість, досить широкий набір бібліотек та фреймворків, вона забезпечує високу продуктивність виконання коду завдяки використанню компіляції в машинний код та оптимізаціям JIT (Just-In-Time) та і в цілому має досить легкий і зрозумілий синтаксис, що значно полегшує процес розробки.

Також при розробці використовувався деякий набір бібліотек, зокрема Grpc.Core.Api та Grpc.Net.Client.Web для забезпечення комунікації основного сервера з алерт-сервісом. Newtonsoft.Json викростовувалася для серіалізації даних. Щоб забезпечити доступ до бази даних була використана бібліотека Npgsql. Бібліотека System.Linq.Async та DeepCloner використовувалася для асинхронного linq(асинхронні методи розширення для колекцій) та клонування об’єктів відповідно. Для ретраїв була використаний пакет Polly. Зокрема для тестування програмного забезпечення був використаний фреймворк MSTest.TestFramework та бібліотека FluentValidation.

Для реалізації клієнтської частини, а саме засобів візуалізації було обрано мову програмування TypeScript. Дана мова програмування має статичну типізацію та додаткові переваги, на відміну від JavaScript. Також серед плюсів TypeScript можна виділити те, що дана мова програмування надає розширені функції розробки, такі як: автодоповнення коду, рефакторінг, статичний аналіз коду, а також підказки про типи даних, що значно полегшує та пришвидшує розробку програмного забезпечення.

Для полегшення написання клієнтської частини використовувалася бібліотека React. Для побудови графіків була застосована бібліотека D3. А бібліотеки: Redux, Formik, react-if, lodash, react-select, immutable мають досить багато синтаксичного цукору, що значно полегшує реалізацію.

На рисунку 2.7 зображена діаграма класів програмного забезпечення.

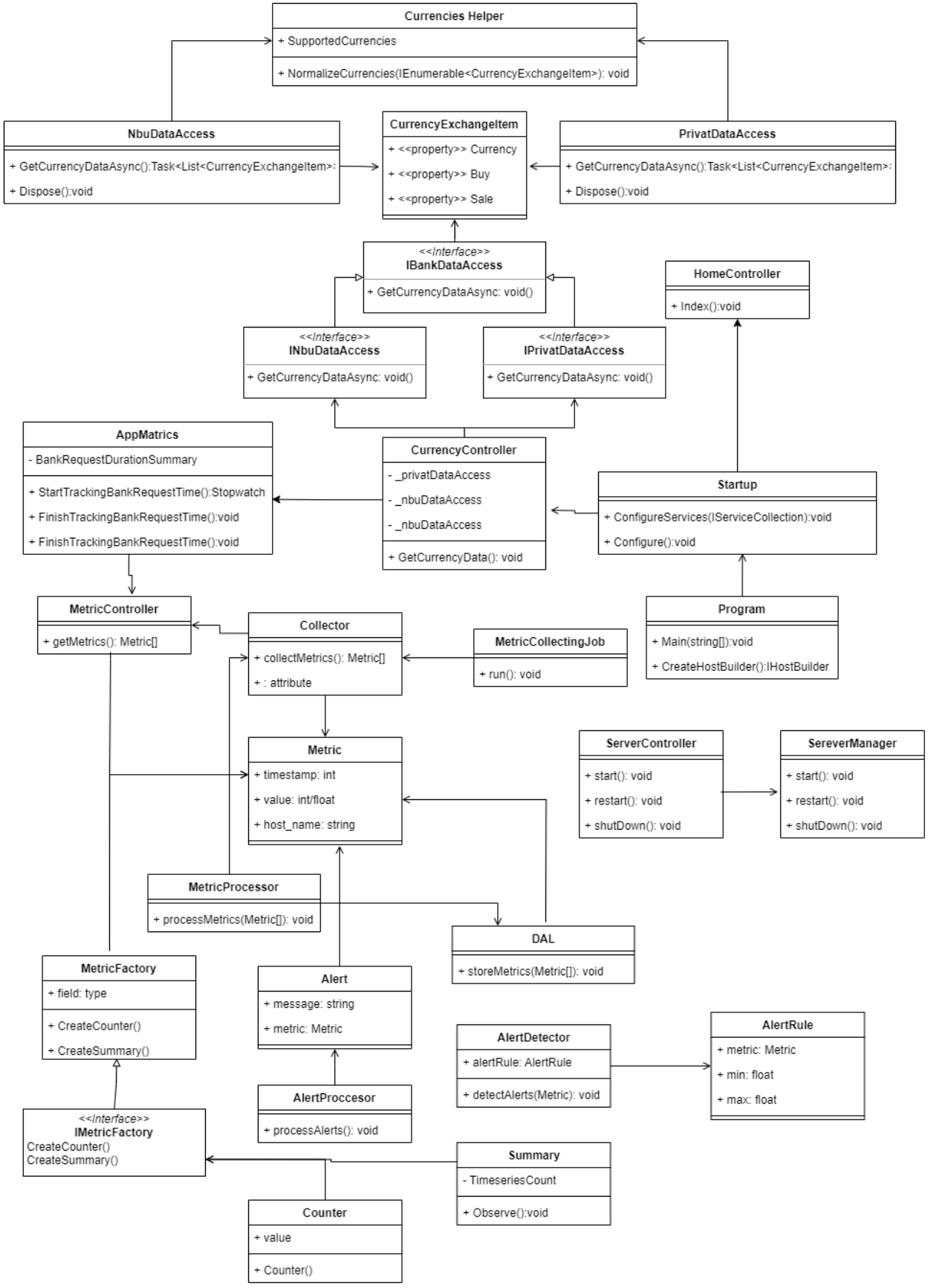


Рисунок 2.7 – Схема структурна класів

У таблиці 2.3 наведений опис основних класів програмного забезпечення.

Таблиця 2.3 – Опис класів програмного забезпечення

Назва	Опис
Metric	Даний клас є сутністю Metric
Collector	Даний клас відповідає за збір програмних метрик
MetricController	Даний клас відповідає за доступ до метрик
MetricCollectingJob	Даний клас відповідає за запуск job-ів, які збирають дані
MetricProccessor	Даний клас відповідає за обробку метрик
DAL	Даний клас відповідає за доступ до даних і їх зберігання
Alert	Даний клас є сутністю Alert
AlertDetector	Даний клас відповідає за розпізнавання підпадання метрики під задані правила
AlertProccesor	Даний клас відповідає за обробку сповіщень
AlertRule	Даний клас є сутністю правил відправки сповіщень
ServerController	Даний клас відповідає за надання API для керування сервером
ServerManager	Даний клас містить бізнес-логіку керування сервером

Програмне забезпечення має набір API, опишемо їх у таблиці 2.4.

Таблиця 2.4 – Опис API

Точка входу	HTTP метод	Опис
…/healthy	GET	Повертає статус сервера на даний момент часу

Продовження таблиці 2.4

…/ready	GET	Повертає статус готовності відповіді на запити, які надходять на сервер
…/reload	PUT	Запускає перезавантаження файлів конфігурацій головного сервера
…/quit	PUT	Запускає завершення роботи головного сервера

В якості СУБД у дипломному проєкті використовується PostgreSQL, адже це реляційна база даних, яка має досить потужну систему контролю цілісності даних, забезпечує збереження даних, обробку помилок та високий рівень безпеки даних.

База даних серверу призначена для зберігання зібраних метрик, за замовчуванням дані у базі зберігаються не більше 15 днів, але якщо змінити відповідні налаштування у потрібному файлі конфігурацій, то можна підвищити цей термін.

На кожну метрику, що збирається, створюється окрема таблиця з тим самим ім’ям, що і у самої метрика. Таблиці ніяк не пов’язані між собою та кожна з них має стандартну структуру для кожної метрики. Опис таблиці бази даних наведено у таблиці 2.5.

Таблиця 2.5 – Опис таблиці metric_name

Таблиця	Назва поля	Тип даних	Опис
metric_name	timestamp	unix	часова мітка певної події
	value	int/float	кількісне значення метрики
	host_name	string	ім’я хоста, з якого відправляються дані

Дана база даних буде мати дві сутності: Metric і Host. ER-діаграма представлена на рисунку 2.8.

Рисунок 2.8 – ER-діаграма

У таблиці 2.6 наведено опис виділених сутностей.

Таблиця 2.6 – Опис виділених сутностей бази даних

Сутність	Опис
Metric	Інформує про зібрану метрику
Host	Інформує про хоста, який проводить збір метрик

Ці сутності будуть мати певні відношення, а саме один/багато до багатьох, так як різні хости можуть проводити збір однієї метрики або ж різні хости можуть проводити збір різних метрик.

Опис утиліт, бібліотек та іншого стороннього програмного забезпечення, що використовується у розробці наведено в таблиці 2.7.

Таблиця 2.7 – Опис утиліт

№ п/п	Назва утиліти	Опис застосування
1	Visual Studio 2022	Головне середовище розробки програмного забезпечення серверної частини дипломного проєкту.
2	Visual Studio Code	Головне середовище розробки програмного забезпечення клієнтської частини дипломного проєкту.
3	Postman	Програмне забезпечення необхідне для тестування rest запитів. Використовувалось для тестування API інтерфейсів, та клієнтських запитів.

Продовження таблиці 2.7

4	.NET 7	Мова розробки серверної частини програмного забезпечення.
5	Docker	Середовище, у якому розгортається MainServer
6	PostgreSQL	Релятивна СУБД для збору даних.
7	Telegram BOT API	Telegram-API для відправки сповіщень у Telegram-bot.
8	TypeScript	Мова розробки клієнтської частини програмного забезпечення
9	D3	Бібліотека, за допомогою якою будуються графіки для візуалізації зібраних метрик
10	React	Бібліотека для створення користувацьких інтерфейсів
11	Webpack	Інструмент для збору модулів клієнтської частини
12	Babel	Інструмент для транспіляції коду JS
13	Figma	Сервіс для розробки вигляду екранних форм

2.4 Аналіз безпеки даних

Дана система — це ПЗ з багатьма компонентами та багатьма інтеграціями з іншими системами. Воно може бути розгорнуте в різних надійних і ненадійних середовищах.

MainServer – безпека основного сервера цілком залежить від середовища в якому він розгортається. Передбачається, що користувачі, які знаходяться у одній мережі з сервером, мають доступ до кінцевої точки HTTP та усіх зібраних даних. Тому безпека повинна забезпечуватись системними адміністраторами або DevOps-ами, через налаштування відповідних політик доступу до мережі.

Alerting - будь-який користувач, який має доступ до кінцевої точки HTTP AlertManager, має доступ до його даних. Вони можуть створювати, модифікувати та видаляти сповіщення.

SDK – передбачається, що SDK є вбудована у застосунок розробника. Тому розробник повинен забезпечити відповідні мережеві налаштування для доступу до кінцевої точки HTTP, щоб вона не була використана зловмисниками для спричинення надмірного навантаження на застосунок, або крадіжки даних.

Конфіденційні дані – ПЗ не повертає жодних конфіденційних даних через кінцеві точки HTTP, так як метрики, логи, та метадані не вважаються конфіденційними даними.

Висновки до розділу

Уданому розділі дипломного проєкту було наведено та описано схему головного бізнес-процесу програмного забезпечення, а саме збору програмних метрик застосунків.

Крім того, було відображено структурну схему компонентів програмного забезпечення, наведено таблиці, що містять більш загальний та детальний опис основних компонентів системи, описано їх взаємодію.

Було наведено діаграму класів програмного забезпечення, а також у таблиці зібрано та описано кожен з них. У таблицю винесено наявні точки входу та описано їх

Також було описано структуру бази даних та таблиць, що будуть у ній міститися.

У окремій таблиці представлено короткий опис використаних утиліт та іншого стороннього програмного забезпечення, що використовується у реалізації програмного забезпечення. Також було розглянуто безпеку даних застосунку.

3 АНАЛІЗ ЯКОСТІ ТА ТЕСТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

3.1 Аналіз якості ПЗ

Аналіз якості програмного забезпечення є важливим етапом циклу розробки програмного забезпечення, так як даний аналіз дозволяє виявити сильні та слабкі сторони продукту, місця, які потребують покращення або оптимізації.

Процес аналізу якості відбувається за допомогою аналізу метрик, які можуть свідчити про те наскільки код відповідає певним стандартам, таким як: зрозумілість, складність, форматування, дублювання шматків коду та ще багато інших.

Проведемо статичний аналіз якості кодової бази програмного забезпечення за допомогою розширення PVS-Studio [13] для середовища розробки Visual Studio 2022.

Даний інструмент перевіряє код на наявність у ньому помилок, недоліків, неправильного форматування та потенційно-небезпечних місць у яких можуть виникати баги. Результат аналізу коду можна побачити на рисунку 3.1.

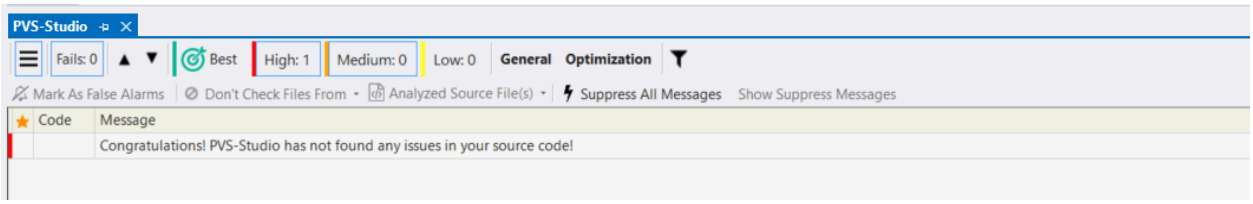


Рисунок 3.1 – Результат перевірки коду

Як можемо бачити по результатам, дане програмне забезпечення успішно пройшло перевірку якості коду.

3.2 Опис процесів тестування

Один з дуже важливих етапів у розробці програмного забезпечення є саме тестування. Бувають досить різні види тестування, такі як: unit-тести, компонент-тести, E2E(End-to-end) тести, а також мануальні тести. Для того,

щоб виконати тестування, підключимо розроблене SDK у код готового застосунку, у якому наявний клієнт, та протестуємо функціонал мануально.

Таблиця 3.1 – Тест 1

Тест	Завантаження SDK з NuGet Package Manager
Номер тесту	1
Початковий стан системи	Користувач знаходиться у Visual Studio 2022
Вхідні данні	-
Опис проведення тесту	Відкривається NuGet Package Manager, у поле пошуку вводиться відповідна назва пакету, серед запропонованих підказок обирається потрібний NuGet, натискається кнопка «Install».
Очікуваний результат	Після кінця завантаження даний NuGet відображається у Packages в солюшені проекту.
Фактичний результат	Після кінця завантаження даний NuGet відображається у Packages в солюшені проекту.

Таблиця 3.2 – Тест 2

Тест	Коректний запис метрики у базу
Номер тесту	2
Початковий стан системи	Користувач знаходиться на сторінці застосунку, у який підключено SDK
Вхідні данні	-
Опис проведення тесту	Відкриваємо застосунок, у який підключили SDK, робимо дію, на яку попередньо налаштували збір метрик і запам’ятовуємо час. Відкриваємо таблицю у базі з назвою відповідної метрики.
Очікуваний результат	У базі у потрібній таблиці є запис з відповідним часом дії і числове значення метрики.

Продовження таблиці 3.2

Фактичний результат	У базі у потрібній таблиці є запис з відповідним часом дії і числове значення метрики.

Таблиця 3.3 – Тест 3

Тест	Перевірка правильності відображення інформації по поточному рантайму
Номер тесту	3
Початковий стан системи	Користувач знаходиться на сторінці власних засобів візуалізації
Вхідні данні	-
Опис проведення тесту	Користувач проводить розгортання серверів та запам’ятовує час і налаштування у файлах конфігурацій, переходить у інструмент для засобів візуалізації з спадного списку Status обирає Runtime та звіряє відображену інформацію
Очікуваний результат	Відображена інформація збіглася
Фактичний результат	Відображена інформація збіглася

Таблиця 3.4 – Тест 4

Тест	Відправка сповіщень у Telegram-BOT
Номер тесту	4
Початковий стан системи	Користувач знаходиться на сторінці застосунку, у який підключено SDK
Вхідні данні	-
Опис проведення тесту	У застосунку робимо дію, на яку підключили збір метрик та налаштували правила, тобто визначили допустимий мінімум та максимум. Правила налаштували так аби сповіщення точно було відправлено.

Продовження таблиці 3.4

Очікуваний результат	У Telegram-BOT приходить сповіщення про те, що дана метрика підпала під правила відправки оповіщень
Фактичний результат	У Telegram-BOT приходить сповіщення про те, що дана метрика підпала під правила відправки оповіщень

Таблиця 3.5 – Тест 5

Тест	Сповіщення не відправляються, коли метрика не підпадає під правила
Номер тесту	5
Початковий стан системи	Користувач знаходиться на сторінці застосунку, у який підключено SDK
Вхідні данні	-
Опис проведення тесту	У застосунку робимо дію, на яку підключили збір метрик та налаштували правила, тобто визначили допустимий мінімум та максимум. Правила налаштували так аби сповіщення точно не було відправлено.
Очікуваний результат	Не отримали сповіщення про те, що дана метрика підпала під правила відправки оповіщень
Фактичний результат	Не отримали сповіщення про те, що дана метрика підпала під правила відправки оповіщень

Таблиця 3.6 – Тест 6

Тест	Розгортання Docker-image
Номер тесту	6
Початковий стан системи	Користувач знаходиться у Docker Hub
Вхідні данні	-
Опис проведення тесту	У поле пошуку в Docker Hub пишемо відповідну назву Docker-image, знаходимо потрібний і завантажуємо, налаштовуємо потрібні конфігурації

Продовження таблиці 3.6

Очікуваний результат	MainServer розгорнутий у Docker
Фактичний результат	MainServer розгорнутий у Docker

Таблиця 3.7 – Тест 7

Тест	Можливість вибрати різні метрики для візуалізації
Номер тесту	7
Початковий стан системи	Користувач знаходиться на сторінці власних засобів візуалізації
Вхідні данні	-
Опис проведення тесту	Користувач знаходить поле вибору метрик для візуалізації, розгортає список доступних
Очікуваний результат	Можливо вибрати різні метрики
Фактичний результат	Можливо вибрати різні метрики

Таблиця 3.8 – Тест 8

Тест	Зміна графіку відповідну до дії
Номер тесту	8
Початковий стан системи	Користувач знаходиться на сторінці застосунку, у який підключено SDK
Вхідні данні	-
Опис проведення тесту	Користувач виконує дію, на яку підключено збір метрик. Переходить у вікно засобів візуалізації, обирає метрику, на яку робив дію.
Очікуваний результат	Бачить зміну графіку відповідно до виконаної дії

Продовження таблиці 3.8

Фактичний результат	Бачить зміну графіку відповідно до виконаної дії

Таблиця 3.9 – Тест 9

Тест	Піднятий стан сервера застосунку
Номер тесту	9
Початковий стан системи	Користувач знаходиться на сторінці власних засобів візуалізації
Вхідні данні	-
Опис проведення тесту	Користувач переконується у тому, що сервер застосунку піднятий, заходить у вкладку, де відображається стан серверу на даний момент
Очікуваний результат	Бачить «Up» у стані сервера
Фактичний результат	Бачить «Up» у стані сервера

Таблиця 3.10 – Тест 10

Тест	Стан сервера застосунку – «Down»
Номер тесту	10
Початковий стан системи	Користувач знаходиться на сторінці власних засобів візуалізації
Вхідні данні	-
Опис проведення тесту	Користувач переконується у тому, що сервер застосунку не піднятий, заходить у вкладку, де відображається стан серверу на даний момент
Очікуваний результат	Бачить «Down» у стані сервера
Фактичний результат	Бачить «Down» у стані сервера

Таблиця 3.11 – Тест 11

Тест	Перевірка коректності роботи засобів візуалізації у Microsoft Edge
Номер тесту	11
Початковий стан системи	Користувач відкриває браузер Microsoft Edge
Вхідні данні	-
Опис проведення тесту	Користувач вводить посилання на засоби візуалізації та тестує коректну роботу застосунку у даному браузері
Очікуваний результат	Робота коректна
Фактичний результат	Робота коректна

Таблиця 3.12 – Тест 12

Тест	Перевірка коректності відображення правил для відправки сповіщень
Номер тесту	12
Початковий стан системи	Користувач знаходиться на сторінці власних засобів візуалізації
Вхідні данні	-
Опис проведення тесту	Користувач дивиться наявні правила у файлі конфігурацій та порівнює їх з наявними правилами, які відображені у вкладці Alert Rules
Очікуваний результат	Всі наявні правила коректно відображаються
Фактичний результат	Всі наявні правила коректно відображаються

3.3 Опис контрольного прикладу

Відтворимо процес контрольного прикладу опираючись на розроблені діаграми бізнес-процесів.

Відкриваємо Visual Studio 2022 з готовим додатком, у який хочемо підключити збір метрик, заходимо в NuGet Package Manager. У поле пошуку вводимо shakhovska_metrcis та shakhovska_metrcis.AspNetCore, завантажуємо ці два пакети. Як має виглядати NuGet Package Manager продемонстровано на рисунку 3.2.

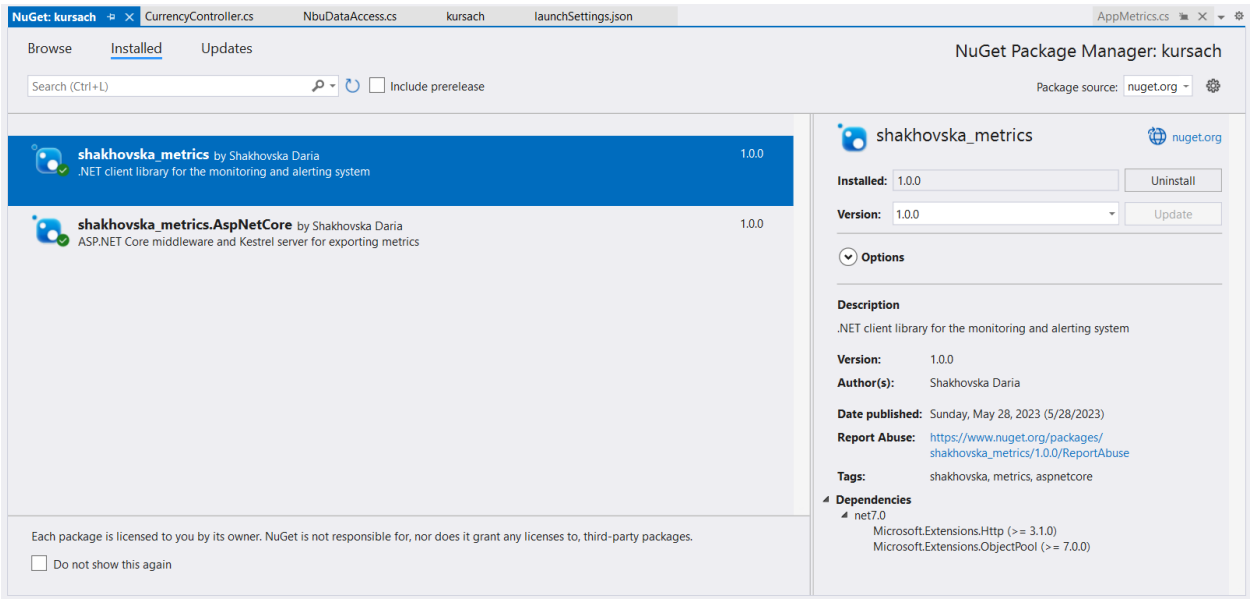


Рисунок 3.2 – Додавання NuGet пакетів у додаток

В налаштуваннях проекту перевіряємо успішне завантаження, як на рисунку 3.3.

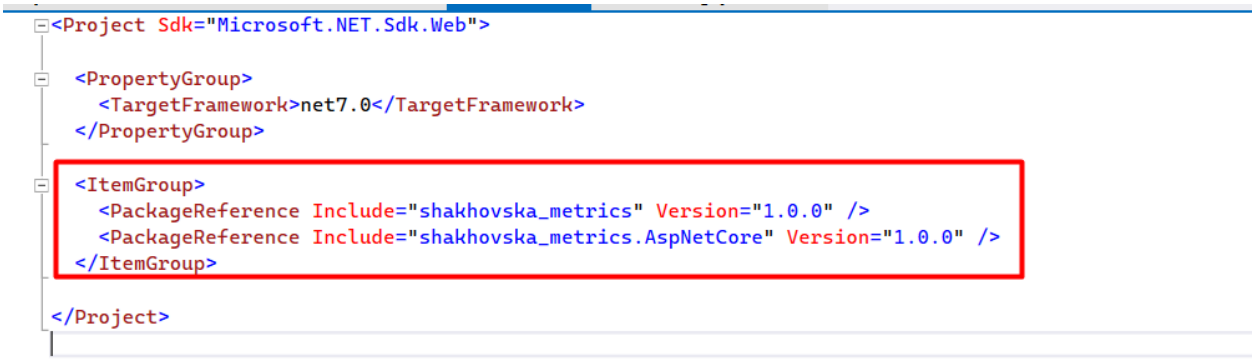


Рисунок 3.3 – Перевірка успішного завантаження пакетів

Додаємо простір імен ShakhovskaSdk у потрібні файли, де будуть використовуватись методи для збору метрик. Використовуємо методи SDK

для збору даних у коді застосунку. Приклад використання методів показаний на рисунку 3.4.

```
[HttpGet]
[ActionName("Data")]
0 references
public async Task<JsonResult> GetCurrencyData()
{
    var sw = AppMetrics.StartTrackingBankRequestTime();
    var privatData = await _privatDataAccess.GetCurrencyDataAsync();
    AppMetrics.FinishTrackingBankRequestTime(sw);
```

Рисунок 3.4 – Приклад збору метрик застосунку

Відкриваємо застосунок, у якому налаштували збір метрик та робимо дії, які призводять до збору часових міток. Відкриваємо інструмент для візуалізації зібраних даних, переходимо у вкладку Dashboard, обираємо потрібну нам метрику, наприклад bank_request, та виставляємо час, за який хочемо переглянути дані. Вигляд даної вкладки наведений на рисунку 3.5.

Рисунок 3.5 – Візуалізація зібраних метрик

Переходимо у вкладку Alerts та можемо переглянути список наявних сповіщень, приклад вкладки продемонстрований на рисунку 3.6.

Рисунок 3.6 – Перегляд правил відправки сповіщень

Бачимо, що дане правило перевіряє, чи піднятий сервер. Якщо ні і протягом хвилини ситуація не змінюється, то в Телеграм-БОТ відправляється відповідне сповіщення. Скріншот сповіщення, яке отримали, наведений на рисунку 3.7.

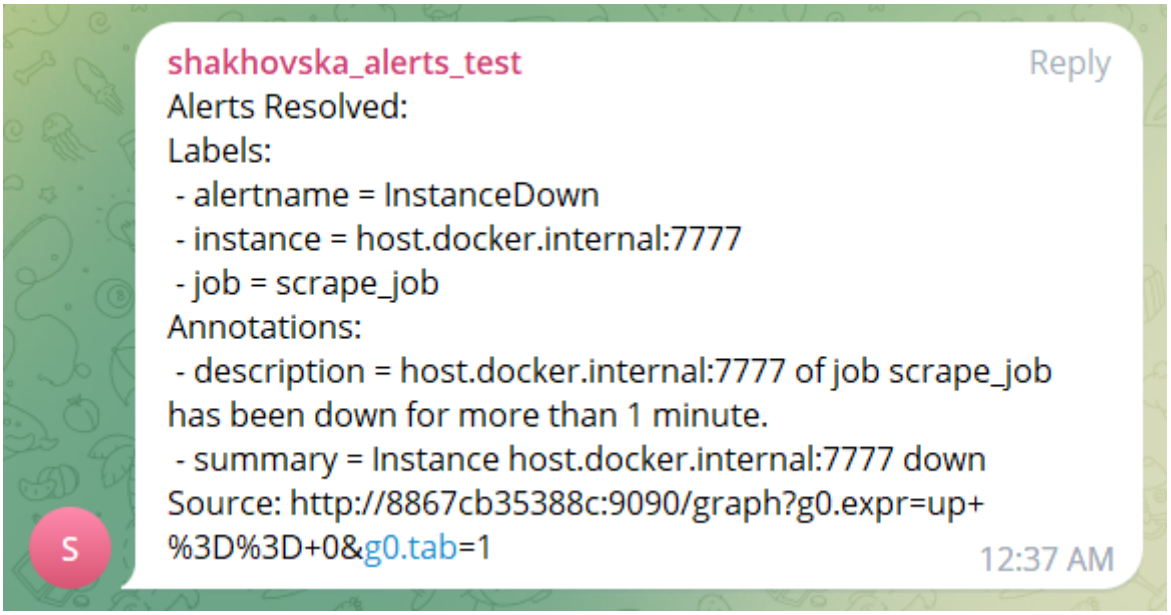


Рисунок 3.7 – Відправлене сповіщення у Телеграм-БОТ

З випадаючого списку Status обираємо Configuration та можемо подивитись наявні конфігурації серверу, такі як: частота скрапінгу даних, кінцеві ендпоінти і таке інше. Вміст даної вкладки наведений на рисунку 3.8.

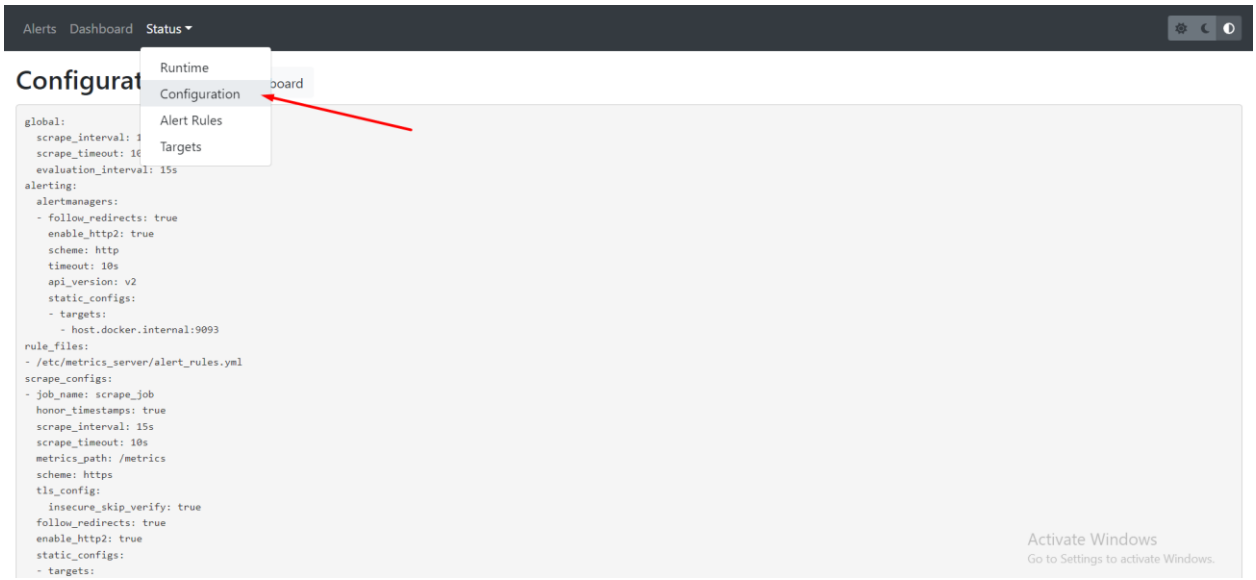


Рисунок 3.8 – Вміст вкладки Configuration

З випадаючого списку Status обираємо Targets та можемо побачити список наявних таргетів, їх статус, коли відбувався останній скрапінг даних, тривалість цього скрапінгу та наявні помилки. Вміст даної вкладки наведений на рисунку 3.9.

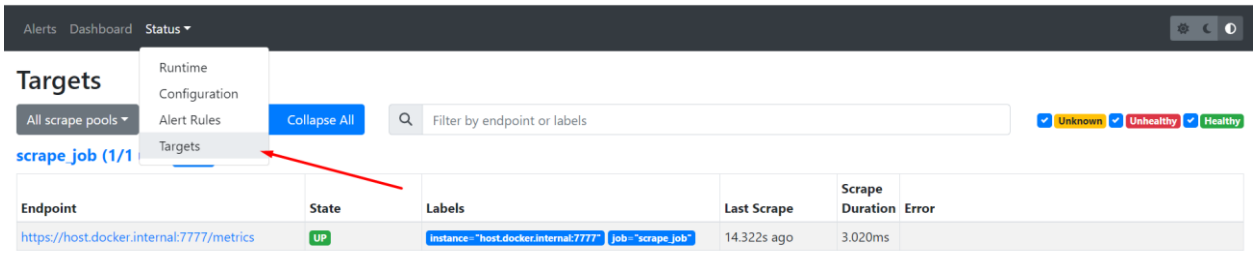


Рисунок 3.9 – Вміст вкладки Targets

З випадаючого списку Status обираємо Alert Rules та можемо переглянути список наявних правил для відправки оповіщень. Вміст даної вкладки наведений на рисунку 3.10.

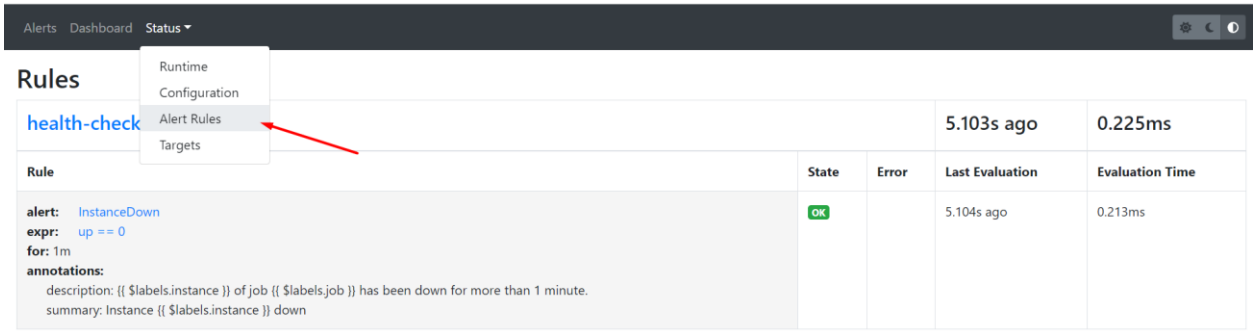


Рисунок 3.10 – Вміст вкладки Alert Rules

З випадаючого списку Status обираємо Runtime та можемо переглянути інформацію стосовно поточного рантайму. Вміст даної вкладки наведений на рисунку 3.11.

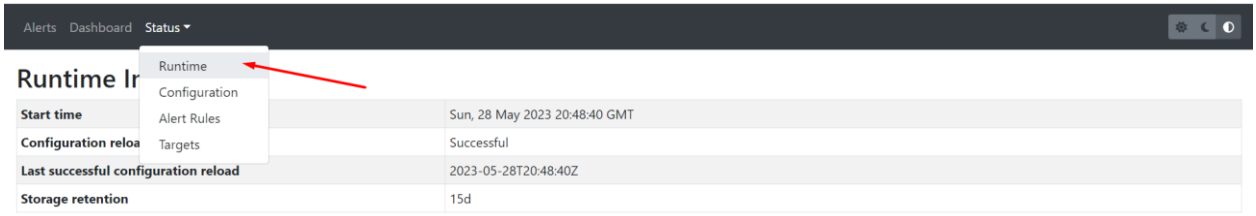


Рисунок 3.11 – Вміст вкладки Runtime

У випадку, коли є потреба змінити метрику для візуалізації, переходимо у вкладку Dashboard. У полу пошуку вводимо назву метрики (рисунок 3.12).

Рисунок 3.12 – Зміна обраної метрики

Якщо хочемо побачити більш детальну картину зміни метрики, зменшуємо час, за який відображаються дані, на панельні як показано на рисунку 3.13.

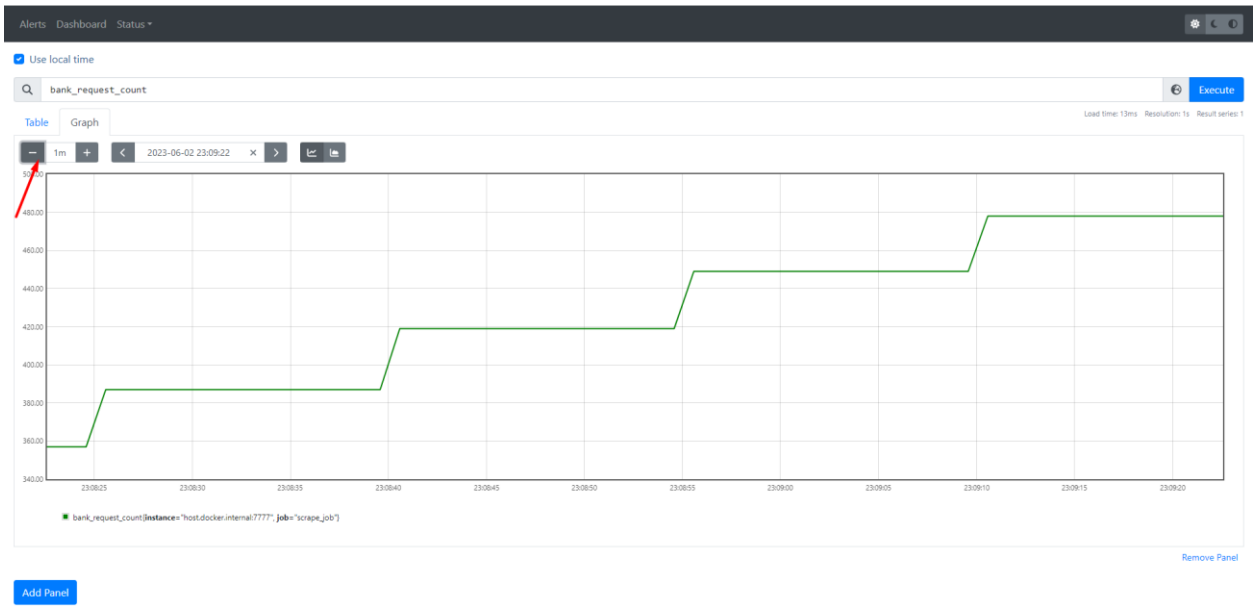


Рисунок 3.12 – Зміна часу за який відображається візуалізація

Даний інтерфейс надає можливість одночасно показувати декілька графіків, для цього треба натиснути кнопку Add Panel та провести ті самі дії, як описано вище (рисунок 3.13-3.14).

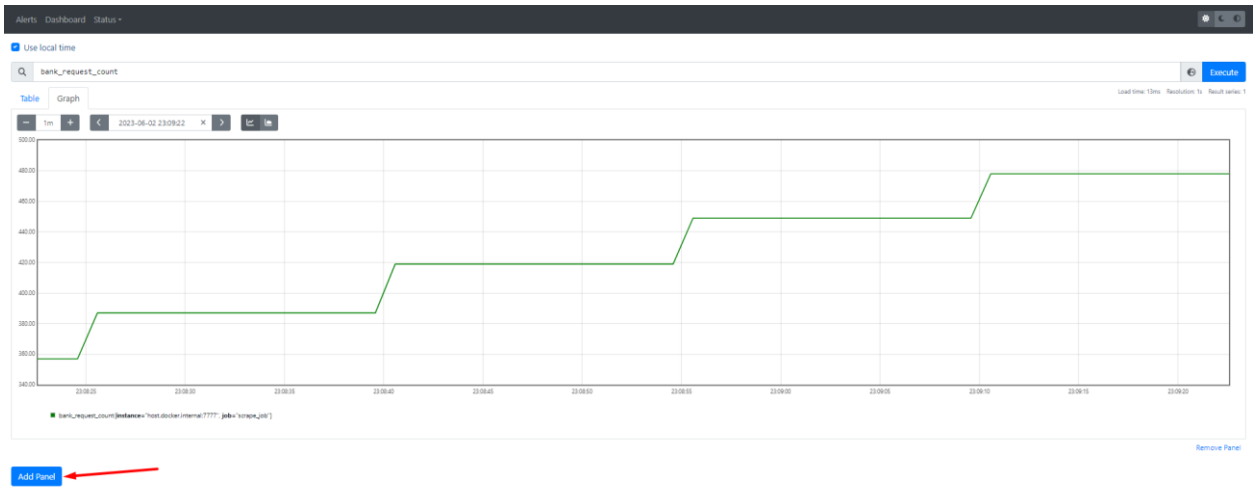


Рисунок 3.13 – Додавання ще одної панелі візуалізації

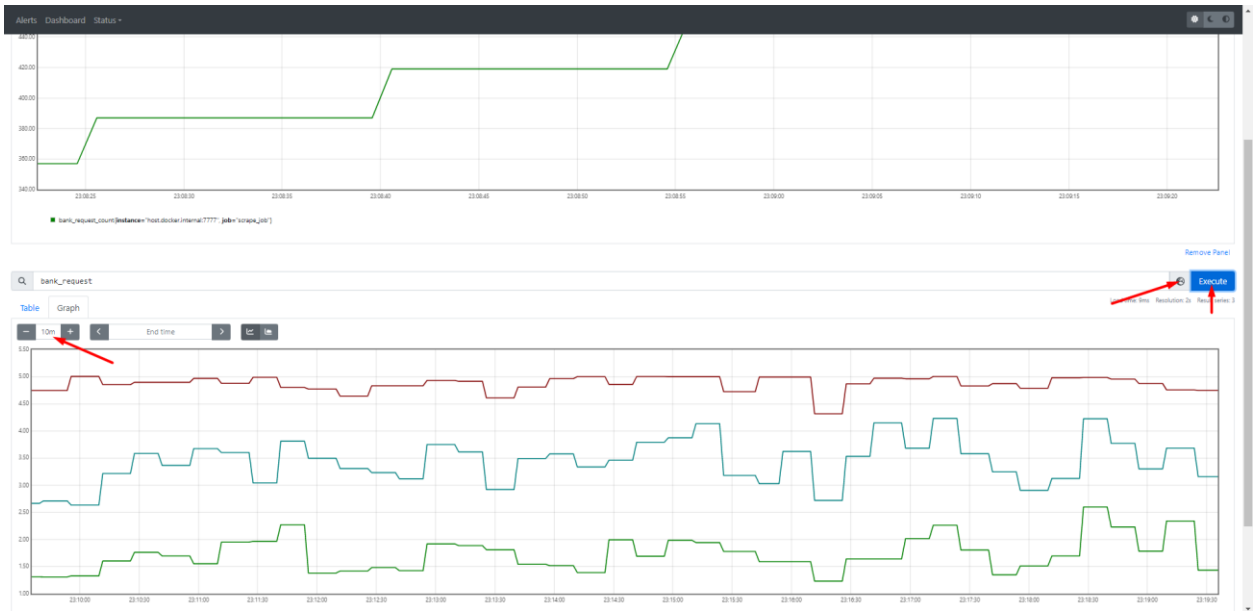


Рисунок 3.14 – Налаштування другої панелі візуалізації

Висновки до розділу

У даному розділі дипломного проєкту було проведено аналіз якості програмного забезпечення за визначеними метриками. В результаті отримали достатньо позитивні оцінки даного аналізу, з чого можемо зробити висновок, що дане програмне забезпечення відповідає якісним характеристикам.

Також було проведено мануальне тестування застосунку та детально описано 12 тест-кейсів.

Крім того був розглянутий контрольний приклад головного бізнес процесу з усіма можливими кейсами. Контрольний приклад підкріплений наведенням скріншотів до кожного кроку.

4 ВПРОВАДЖЕННЯ ТА СУПРОВІД ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

4.1 Розгортання програмного забезпечення

Дане програмне забезпечення є on-premise, а тому розгортається безпосередньо на пристроях користувача. Програмне забезпечення складається з трьох компонентів: MainServer, AlertService та SDK. Остання версія SDK знаходиться у NuGet Gallery-і [14] (рисунок 4.1).

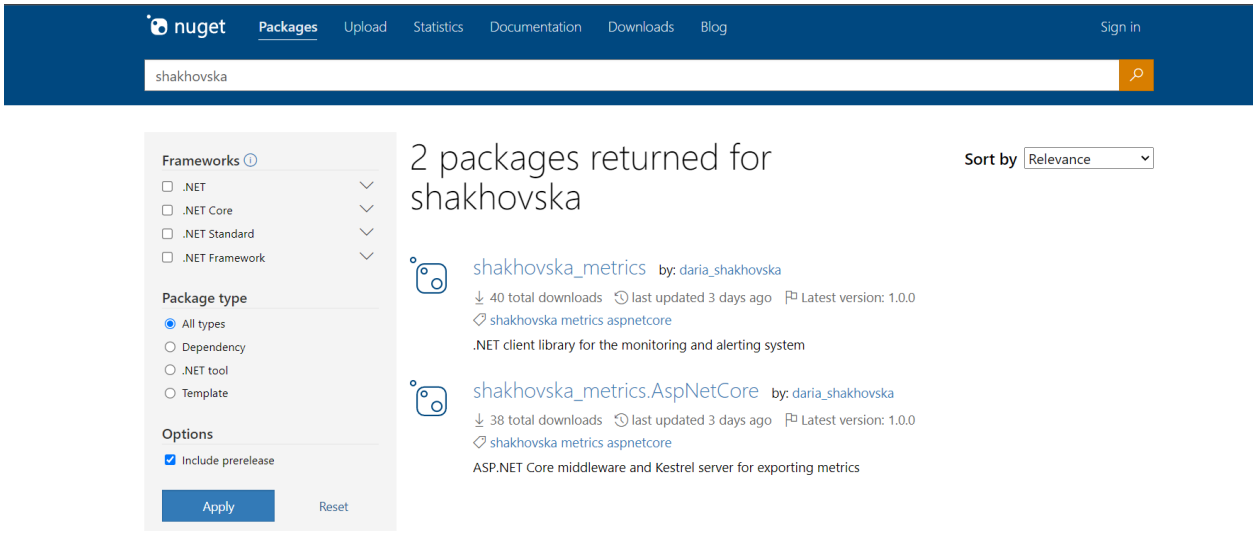


Рисунок 4.1 – Наявність SDK у NuGet Gallery

MainServer та AlertService поставляються користувачам у вигляді готових Docker-image-ів, у яких є все необхідне середовище для запуску. Користувач може завантажити відповідні Docker-image-і з Docker Hub [15]. Вигляд завантажених Docker-image-ів можна переглянути на рисунку 4.2.

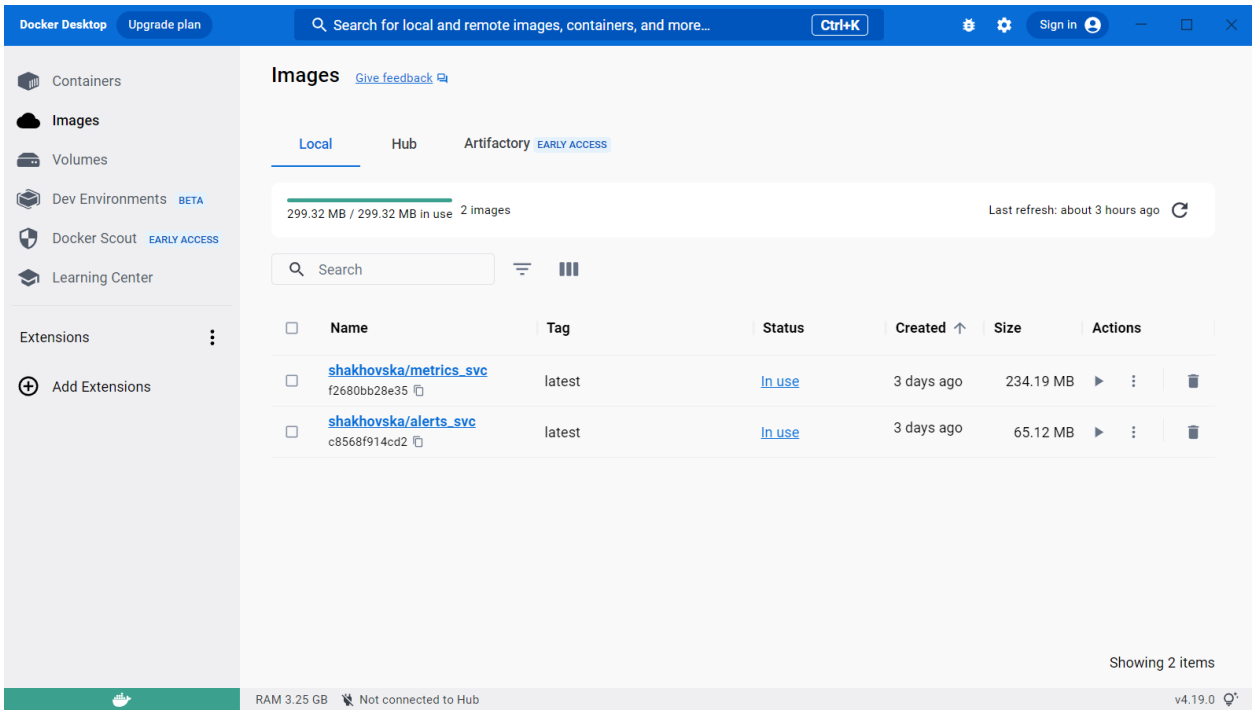


Рисунок 4.2 – Завантажені Docker-image-і

Вводячи команду «docker run --name metrics_svc -d -p 9090:9090 -v C:\Users\Daria\Desktop\configs:/etc/metrics_server shakhovska/metrics_svc» у термінал відбувається розгортання Docker-image-а MainServer (рисунок 4.3).

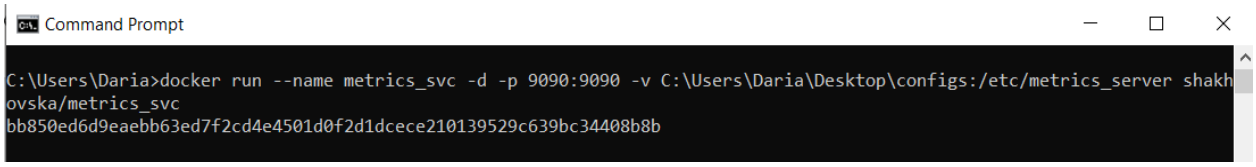


Рисунок 4.3 – Розгортання MainServer

Вводячи команду «docker run --name alerts_svc -d -p 9093:9093 -v C:\Users\Daria\Desktop\configs:/etc/alertmanager shakhovska/alerts_svc» у темінал відбувається розгортання Docker-image-а AlertService (рисунок 4.4).

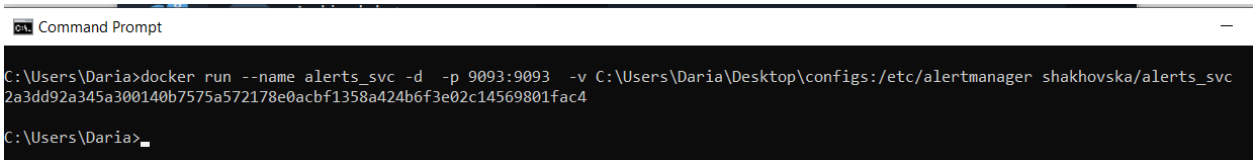


Рисунок 4.4 – Розгортання AlertService

При розгортанні Docker-image-у потрібно вказати файли конфігурацій, які містять всі потрібні налаштування для MainServer та AlertService відповідно. Існують три файли конфігурацій: alert_rules.yml, alertmanager.yml та server_config.yml, які відповідають за конфігурації правил відправки

сповіщень, конфігурації сервіса відправки сповіщень, конфігурації основного сервера відповідно (рисунок 4.5-4.7).

alert_rules.yml | server_config.yml | alertmanager.yml

C: > Users > Daria > Desktop > configs > alert_rules.yml

```yaml
groups:
- name: health-check
  rules:
  # Alert for any instance that is unreachable for >1 minute.
  - alert: InstanceDown
    expr: up == 0
    for: 1m
    annotations:
      summary: "Instance {{ $labels.instance }} down"
      description: "{{ $labels.instance }} of job {{ $labels.job }} has been down for more than 1 minute."
```

Рисунок 4.5 – Приклад файлу alert_rules.yml

alert_rules.yml | server_config.yml | alertmanager.yml

C: > Users > Daria > Desktop > configs > alertmanager.yml

```yaml
# The root route on which each incoming alert enters.
route:
  group_by: ['alertname']

  # A default receiver
  receiver: telegram

receivers:
  - name: telegram
    telegram_configs:
      - bot_token: 6228806646:AAHWalU8qskDX1LEUcG4OWPF4zDzXbVAWDY
        chat_id: -846515129
        api_url: 'https://api.telegram.org'
        parse_mode: ''
```

Рисунок 4.6 – Приклад файлу alertmanager.yml

alert_rules.yml | server_config.yml | alertmanager.yml

C: > Users > Daria > Desktop > configs > server_config.yml

```yaml
global:
  scrape_interval: 15s
  evaluation_interval: 15s

scrape_configs:
  - job_name: scrape_job
    scheme: https
    tls_config:
      insecure_skip_verify: true
    static_configs:
      - targets: ["host.docker.internal:7777"]

rule_files:
  - "alert_rules.yml"

alerting:
  alertmanagers:
  - static_configs:
      - targets:
        - host.docker.internal:9093
```

Рисунок 4.7 – Приклад файлу server_config.yml

4.2 Підтримка програмного забезпечення

В подальшому планується випускати нові версії SDK з більш розширеним функціоналом, які будуть доступні для скачування у NuGet Package Manager-і з тими назвами, що були наведені вище, тому користувачі завжди зможуть мати доступ до найновішої версії бібліотеки.

Для того, щоб оновити версію Docker-image-а необхідно спочатку переглянути поточну версію імеджа через команду «docker images», поле tag відповідає за версію. Після цього необхідно переглянути новіших версій у реєстрі імеджів Docker Hub. При наявності нової версії потрібно її скачати за допомогою команди «docker pull» і зупинити старі запущені контейнери та розгорнути нові, як описано вище у пункті 4.1.

В майбутньому планується забезпечення інтеграції з сторонніми сервісами візуалізації, такими як Grafana, для більшої гнучкості і розширення функціоналу даної системи. Також один з варіантів розвитку продукту полягає у розширенні варіантів отримання сповіщень, не тільки на електронну пошту і в Телеграм-бот, а й в інші месенджери або застосунки. Це надасть можливість більш ширшого та зручнішого способу отримання оповіщень.

Висновки до розділу

В даному розділі дипломного проєкту було наведено детальний опис процесу розгортання серверу у Docker.

Також було описано підтримку програмного забезпечення, а саме як користувачі зможуть отримати доступ до нової версії SDK.

ВИСНОВКИ

У першому розділі пояснювальної записки дипломного проєкту було проаналізовано предметну область, детально наведені загальні положення. Також було визначено проблематику та актуальність даної роботи, яка полягає у полегшенні збору та візуалізації метрик програмістам, що в майбутньому надасть внутрішню інформацію про процеси, які відбуваються у коді, також полегшить пошук наявних проблем та пришвидшить процес їх виправлення. Крім цього, було проаналізовано найбільш відомі технічні рішення. В результаті даного аналізу було обрано:

– C# як мову серверної частини програмного забезпечення;

– TypeScript як мову клієнтської частини програмного забезпечення;

– PostgreSQL як СУБД для зберігання зібраних даних;

– Docker як середовище, у якому буде розгорнуте програмне забезпечення;

– Visual Studio 2022 як середовище розробки серверної частини застосунку;

– Visual Studio Code як середовище розробки клієнтської частини застосунку.

Також був здійснений пошук аналогів даного програмного забезпечення, проведений аналіз їх функціоналу. Далі була наведена діаграма варіантів використання з детальним описом кожного варіанта. Були визначені функціональні та нефункціональні вимоги та побудована матриця трасування вимог.

У другому розділі була наведена та детально описана діаграма головного бізнес-процесу застосунку. Крім того, було обрано монолітну архітектуру для реалізації програмного забезпечення. Наведено діаграму компонентів та діаграму класів системи. Було описано структури бази даних

та у окрему таблицю винесено опис використаних утиліт та стороннього програмного забезпечення. Було також проаналізовано безпеку даних.

У наступному розділі пояснювальної записки було оцінено якість застосунку, а також проведено мануальне тестування і наведені 12 тест-кейсів з детальним описом проведення. Далі був наведений контрольний приклад на основі головного бізнес-процесу застосунку, який був підкріплений наведенням скріншотів на кожному етапі.

Четвертий розділ пояснювальною записки описує детальний процес розгортання застосунку у Docker та подальше підтримання програмного забезпечення.

Всі поставлення задачі реалізації даного дипломного проєкту були виконані, а саме: надання можливості легко підключити та розгорнути програмне забезпечення, забезпечити збір програмних метрик, надати можливість отримувати сповіщення у Телеграм-бот, забезпечити можливість візуалізації зібраних даних, а мета досягнена.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1) Software Metrics and Software Metrology / Alain Abram, 2010 – 328 с.

2) Software Metrics: A Rigorous and Practical Approach, second edition / Norman E. Fenton, Shari Lawrence Pfleeger, 1997 – 651 с.

3) Fundamentals of Software Architecture: An Engineering Approach / Mark Richards, Neal Ford, 2020 – 396 с.

4) Go official documentation [Електронний ресурс]. – 2023. – Режим доступу: https://go.dev/doc/

5) Java official documentation [Електронний ресурс]. – 2023. – Режим доступу: https://www.java.com/en/

6) Python official documentation [Електронний ресурс]. – 2023. – Режим доступу: https://www.python.org/

7) C# (ASP.NET) official documentation [Електронний ресурс]. – 2023. – Режим доступу: https://learn.microsoft.com/en-us/aspnet/core/?view=aspnetcore-7.0

8) Docker [Електронний ресурс]. – 2023. – Режим доступу: https://www.docker.com/

9) Docker in Action, second edition / Jeff Nickoloff, Stephen Kuenzli, 2019 – 336 с.

10) Graphite official documentation [Електронний ресурс]. – 2023. – Режим доступу: https://graphite.readthedocs.io/en/latest/#

11) InfluxDB [Електронний ресурс]. – 2023. – Режим доступу: https://www.influxdata.com/

12) Nagios [Електронний ресурс]. – 2023. – Режим доступу: https://www.nagios.org/

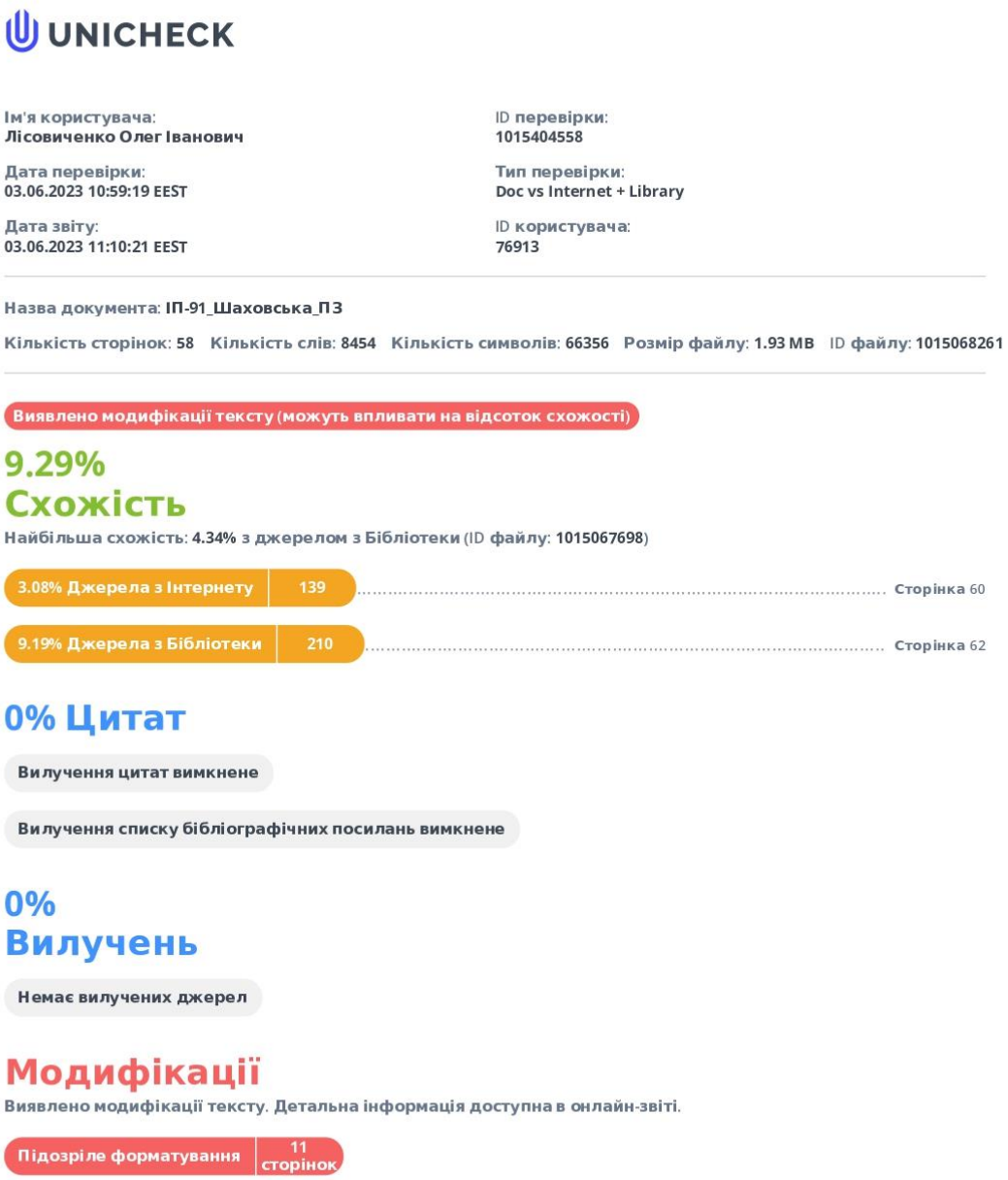
13) PVS-Studio [Електронний ресурс]. – 2023 – Режим доступу: https://pvs-studio.com/en/

14) NuGet Gallery [Електронний ресурс]. – 2023 – Режим доступу: https://www.nuget.org/

15) Docker Hub [Електронний ресурс]. – 2023 – Режим доступу: https://hub.docker.com/

ДОДАТКИ

ДОДАТОК А Звіт подібності



UNICHECK

Ім'я користувача:
Лісовиченко Олег Іванович

ID перевірки:
1015404558

Дата перевірки:
03.06.2023 10:59:19 EEST

Тип перевірки:
Doc vs Internet + Library

Дата звіту:
03.06.2023 11:10:21 EEST

ID користувача:
76913

Назва документа: ІП-91_Шаховська_ПЗ

Кількість сторінок: 58 Кількість слів: 8454 Кількість символів: 66356 Розмір файлу: 1.93 MB ID файлу: 1015068261

Виявлено модифікації тексту (можуть впливати на відсоток схожості)

9.29%
Схожість

Найбільша схожість: 4.34% з джерелом з Бібліотеки (ID файлу: 1015067698)

3.08% Джерела з Інтернету 139 ..... Сторінка 60

9.19% Джерела з Бібліотеки 210 ..... Сторінка 62

0% Цитат

Вилучення цитат вимкнене

Вилучення списку бібліографічних посилань вимкнене

0%
Вилучень

Немає вилучених джерел

Модифікації

Виявлено модифікації тексту. Детальна інформація доступна в онлайн-звіті.

Підозріле форматування 11 сторінок

Рисунок А.1 – Звіт подібності

Факультет інформатики та обчислювальної техніки
Кафедра інформатики та програмної інженерії

“ЗАТВЕРДЖЕНО”
Завідувач кафедри
_____ Едуард ЖАРІКОВ
“___” _____ 2023 р.

ПРОГРАМНЕ ЗАБЕЗПЕЧЕННЯ ДЛЯ ЗБОРУ ПРОГРАМНИХ МЕТРИК ЗАСТОСУНКІВ

Текст програми

КПІ.ІС-9328.045200.03.12

“ПОГОДЖЕНО”
Керівник проєкту:
_____ Олена ХАЛУС

Нормоконтроль:
_____ Катерина ЛІЩУК

Виконавець:
_____ Дар’я ШАХОВСЬКА

Київ – 2023

Файл MyBackgroundService.cs

```
using System;
using System.Threading;
using System.Threading.Tasks;
using diploma.Diagnostics;
using Microsoft.Extensions.Hosting;

namespace diploma.BackgroundProcesses
{
   public class MyBackgroundService : IHostedService, IDisposable
   {
      public Task StartAsync(CancellationToken stoppingToken)
      {
         _timer = new Timer(DoWork, null, TimeSpan.Zero, TimeSpan.FromMilliseconds(500));

         return Task.CompletedTask;
      }

      private async void DoWork(object state)
      {
         var sw = AppMetrics.StartTrackingBankRequestTime();
         await Task.Delay(new Random().Next(500, 5000));
         AppMetrics.FinishTrackingBankRequestTime(sw);
      }

      public Task StopAsync(CancellationToken stoppingToken)
      {
         _timer?.Change(Timeout.Infinite, 0);

         return Task.CompletedTask;
      }

      public void Dispose()
      {
         _timer?.Dispose();
      }

      private Timer _timer = null;
   }
}
```

Файл AppMetrics.cs

```
using ShakhovskaSdk;
using System.Diagnostics;
using System;

namespace diploma.Diagnostics
{
   public static class AppMetrics
   {
      public static Stopwatch StartTrackingBankRequestTime() => Stopwatch.StartNew();

      public static void FinishTrackingBankRequestTime(Stopwatch sw)
      {
         BankRequestDurationSummary
            .Observe(sw.Elapsed.TotalSeconds);
      }

      private static readonly Summary BankRequestDurationSummary = Metrics
         .CreateSummary("bank_request", $"Summary of bank request duration (in seconds).",// over last
{SummaryMaxAgeMinutes} minutes.",
            new SummaryConfiguration
            {
```

```
                MaxAge = TimeSpan.FromSeconds(20),
                Objectives = new[]
                {
                    new QuantileEpsilonPair(0.3, 0.0001),
                    new QuantileEpsilonPair(0.7, 0.0001),
                    new QuantileEpsilonPair(1, 0.0001),
                }
            });
        //private const int SummaryMaxAgeMinutes = 1;
    }
}
```

Файл Program.cs

```
using Microsoft.AspNetCore.Hosting;
using Microsoft.Extensions.Configuration;
using Microsoft.Extensions.Hosting;
using Microsoft.Extensions.Logging;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;

namespace diploma
{
    public class Program
    {
        public static void Main(string[] args)
        {
            CreateHostBuilder(args).Build().Run();
        }

        public static IHostBuilder CreateHostBuilder(string[] args) =>
            Host.CreateDefaultBuilder(args)
                .ConfigureWebHostDefaults(webBuilder =>
                {
                    webBuilder.UseStartup<Startup>();
                });
    }
}
```

Файл Startup.cs

```
using diploma.BackgroundProcesses;
using diploma.Controllers;
using diploma.DataAccess;
using diploma.DataAccess.Contracts;
using Microsoft.AspNetCore.Builder;
using Microsoft.AspNetCore.Hosting;
using Microsoft.Extensions.Configuration;
using Microsoft.Extensions.DependencyInjection;
using Microsoft.Extensions.Hosting;
using ShakhovskaSdk;

namespace diploma
{
    public class Startup
    {
        public Startup(IConfiguration configuration)
        {
            Configuration = configuration;
        }

        public IConfiguration Configuration { get; }

        // This method gets called by the runtime. Use this method to add services to the container.
```

```
        public void ConfigureServices(IServiceCollection services)
        {
            services.AddControllersWithViews();

            services.AddSingleton<IPrivatDataAccess, PrivatDataAccess>();
            services.AddSingleton<INbuDataAccess, NbuDataAccess>();

            services.AddSingleton<HomeController>();

            services.AddHostedService<MyBackgroundService>();
        }

        // This method gets called by the runtime. Use this method to configure the HTTP request pipeline.
        public void Configure(IApplicationBuilder app, IWebHostEnvironment env)
        {
            if (env.IsDevelopment())
            {
                app.UseDeveloperExceptionPage();
            }
            else
            {
                app.UseExceptionHandler("/Home/Error");
                // The default HSTS value is 30 days. You may want to change this for production scenarios, see
https://aka.ms/aspnetcore-hsts.
                app.UseHsts();
            }
            app.UseShakhovskaSdkServer();

            app.UseHttpsRedirection();
            app.UseStaticFiles();

            app.UseRouting();

            app.UseAuthorization();

            app.UseEndpoints(endpoints =>
            {
                endpoints.MapControllerRoute(
                    name: "default",
                    pattern: "{controller=Home}/{action=Index}/{id?}");
            });
        }
    }

    public static class ApplicationBuilderExtensions
    {
        public static IApplicationBuilder UseShakhovskaSdkServer(this IApplicationBuilder app)
        {
            Metrics.SuppressDefaultMetrics();

            app.Map("/metrics", metricsApp =>
            {
                metricsApp.UseMetricServer(port: 7777, url: string.Empty);
            });

            return app;
        }
    }
}
```

Файл HomeController.cs

```
using Microsoft.AspNetCore.Mvc;
```

```
namespace diploma.Controllers
{
    public class HomeController : Controller
    {
        public IActionResult Index()
        {
            return View();
        }
    }
}
```

Файл Controller.cs

```
using diploma.DataAccess;
using diploma.DataAccess.Contracts;
using diploma.Diagnostics;
using diploma.ViewModels;
using Microsoft.AspNetCore.Mvc;
using System.Threading.Tasks;


namespace diploma.Controllers
{
    public class CurrencyController : Controller
    {
        public CurrencyController(IPrivatDataAccess privatDataAccess,
            INbuDataAccess nbuDataAccess)
        {
            _privatDataAccess = privatDataAccess;
            _nbuDataAccess = nbuDataAccess;
        }

        [HttpGet]
        [ActionName("Data")]
        public async Task<JsonResult> GetCurrencyData()
        {
            var sw = AppMetrics.StartTrackingBankRequestTime();
            var privatData = await _privatDataAccess.GetCurrencyDataAsync();
            AppMetrics.FinishTrackingBankRequestTime(sw);

            var nbuData = await _nbuDataAccess.GetCurrencyDataAsync();
            privatData.NormalizeCurrencies();
            nbuData.NormalizeCurrencies();

            var vm = new CurrencyDataViewModel(new[] {
                privatData.ToViewModel(BankNames.Privat),
                nbuData.ToViewModel(BankNames.Nbu)
            });

            return new JsonResult(vm);
        }

        private readonly IPrivatDataAccess _privatDataAccess;
        private readonly INbuDataAccess _nbuDataAccess;
    }
}
```

Файл PrivatDataAccess.cs

```
using diploma.DataAccess.Contracts;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Net.Http;
using System.Text.Json;
```

```
using System.Threading.Tasks;

namespace diploma.DataAccess
{
    public class PrivatDataAccess : IPrivatDataAccess, IDisposable
    {
        public async Task<List<CurrencyExchangeItem>> GetCurrencyDataAsync()
        {
            await Task.Delay(new Random().Next(500, 2500));

            const string Url = "https://api.privatbank.ua/p24api/pubinfo?json&exchange&coursid=11";
            var resp = await _httpClient.GetAsync(Url);
            var json = await resp.Content.ReadAsStringAsync();
            var allData = JsonSerializer.Deserialize<List<PrivatCurrencyItem>>(json);
            var fileteredData = allData
                .Where(x => CurrenciesHelper.SupportedCurrencies.Contains(x.ccy))
                .Select(x => x.ToContract())
                .ToList();

            return fileteredData;
        }

        private readonly HttpClient _httpClient = new HttpClient();

        public void Dispose()
        {
            _httpClient?.Dispose();
        }

        private class PrivatCurrencyItem
        {
            public string ccy { get; set; }
            public string base_ccy { get; set; }
            public string buy { get; set; }
            public string sale { get; set; }

            public CurrencyExchangeItem ToContract()
            {
                return new CurrencyExchangeItem()
                {
                    Currency = ccy,
                    Buy = Convert.ToDouble(buy),
                    Sale = Convert.ToDouble(sale)
                };
            }
        }
    }
}
```

Файл DataAccess.cs

```
using diploma.DataAccess.Contracts;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Net.Http;
using System.Text.Json;
using System.Threading.Tasks;

namespace diploma.DataAccess
{
    public class NbuDataAccess : INbuDataAccess, IDisposable
    {
        public async Task<List<CurrencyExchangeItem>> GetCurrencyDataAsync()
```

```
        {
            const string Url = "https://bank.gov.ua/NBUStatService/v1/statdirectory/exchange?json";
            var resp = await _httpClient.GetAsync(Url);
            var json = await resp.Content.ReadAsStringAsync();
            var allData = JsonSerializer.Deserialize<List<NbuCurrencyItem>>(json);
            var fileteredCurrencies = allData
                .Where(x => CurrenciesHelper.SupportedCurrencies.Contains(x.cc))
                .Select(x => x.ToContract())
                .ToList();

            return fileteredCurrencies;
        }

        public void Dispose()
        {
            _httpClient?.Dispose();
        }

        private readonly HttpClient _httpClient = new HttpClient();

        private class NbuCurrencyItem
        {
            public string cc { get; set; }
            public double rate { get; set; }

            public CurrencyExchangeItem ToContract()
            {
                return new CurrencyExchangeItem()
                {
                    Currency = cc,
                    Buy = rate
                };
            }
        }
    }
}
```

Файл index.cshtml

```
@{
    Layout = "_Layout";
    ViewData["Title"] = "diploma";
}


<span>| </span>
<!-- ko foreach: $data.currencyData() -->
<li data-bind="click: $parent.toggleActive" style="display:inline">
    <a data-bind="attr: {href: '#' + $data.bankName}, text: bankName, css: $parent.selectedBank() == $data ? 'active':
""></a>
</li>
<span> | </span>
<!-- /ko -->

<div></div>

<table style="display: inline-table">
    <thead>
        <tr>
            <th>Currency</th>
            <th>Buy</th>
            <th>Sale</th>
        </tr>
    </thead>
```

```
    <tbody data-bind="foreach: $data.selectedBank().data">
      <tr>
        <td data-bind="text: currency"></td>
        <td data-bind="text: buy"></td>
        <td data-bind="text: sale"></td>
      </tr>
    </tbody>
</table>

<div style="margin: 30px 0 20px 0; background-color: #303030; width: 360px">
    <select data-bind="options: calcConvertOptions, value: calcConvertSelectedOption"></select>
    <div style="padding: 20px">
      <!-- ko if: calcConvertSelectedOption() === 'Buy' -->
      <span class="text-white" style="margin: 0 10px 0 10px">UAH</span>
      <!-- /ko -->
      <!-- ko if: calcConvertSelectedOption() === 'Sale' -->
      <select data-bind="options: exchangeCurrOptions, value: exchangeCurr"></select>
      <!-- /ko -->
      <input data-bind="textInput: exchangeVal1" style="margin-left: 30px" />
    </div>
    <img src="~/images/arrow-down.png" style="margin-left: 60px" />
    <div style="padding: 20px">
      <!-- ko if: calcConvertSelectedOption() === 'Sale' -->
      <span class="text-white" style="margin: 0 10px 0 10px">UAH</span>
      <!-- /ko -->
      <!-- ko if: calcConvertSelectedOption() === 'Buy' -->
      <select data-bind="options: exchangeCurrOptions, value: exchangeCurr"></select>
      <!-- /ko -->
      <input data-bind="textInput: exchangeVal2" style="margin-left: 30px" />
    </div>
</div>
```

Файл _Layout.cshtml

```
@addTagHelper *, Microsoft.AspNetCore.Mvc.TagHelpers

<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="utf-8" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
    <title>@ViewData["Title"] - diploma</title>
    <link rel="stylesheet" href="~/lib/bootstrap/dist/css/bootstrap.min.css" />
    <link rel="stylesheet" href="~/css/site.css" />
    <script type='text/javascript' src="~/lib/knockout-3.5.1.js"></script>
    <script type='text/javascript' src="~/lib/knockout.mapping-latest.js"></script>
</head>
<body>
    <header class="header">
        <nav class="navbar navbar-expand-sm navbar-toggleable-sm navbar-light border-bottom box-shadow mb-3">
            <div class="container">
                <h3>currency aggregator</h3>
                <button class="navbar-toggler" type="button" data-toggle="collapse" data-target=".navbar-collapse" aria-
controls="navbarSupportedContent"
                        aria-expanded="false" aria-label="Toggle navigation">
                    <span class="navbar-toggler-icon"></span>
                </button>
                <div class="navbar-collapse collapse d-sm-inline-flex justify-content-between" style="margin-left:
20px;">
                    <ul class="navbar-nav flex-grow-1">
                        <li class="nav-item">
                            <a class="nav-link text-white" asp-area="" asp-controller="Home" asp-
action="Index">Exchange</a>
                        </li>
```

```
                </ul>
            </div>
        </div>
    </nav>
</header>
<div class="container">
    <main role="main" class="pb-3">
        @RenderBody()
    </main>
</div>

<footer class="border-top footer">
    <div class="container">
        &copy; 2021 - currency aggregator
    </div>
</footer>
<script src="~/lib/jquery/dist/jquery.min.js"></script>
<script src="~/lib/bootstrap/dist/js/bootstrap.bundle.min.js"></script>
<script src="~/js/site.js" asp-append-version="true"></script>
@await RenderSectionAsync("Scripts", required: false)
</body>
</html>
```

Файл CurrencyDataItemViewModel.cs

```
using diploma.DataAccess.Contracts;
using System.Collections.Generic;
using System.Linq;

namespace diploma.ViewModels
{
    public class CurrencyDataItemViewModel
    {
        public string bankName { get; set; }
        public List<CurrencyViewModel> data { get; set; }
    }

    public static class CurrencyDataItemViewModelExtensions
    {
        public static CurrencyDataItemViewModel ToViewModel(this IEnumerable<CurrencyExchangeItem> items,
string bankName)
        {
            return new CurrencyDataItemViewModel()
            {
                bankName = bankName,
                data = items.Select(x => x.ToViewModel()).ToList()
            };
        }
    }
}
```

Файл CurrencyDataViewModel.cs

```
using System.Collections.Generic;

namespace diploma.ViewModels
{
    public class CurrencyDataViewModel : List<CurrencyDataItemViewModel>
    {
        public CurrencyDataViewModel()
        {
        }

        public CurrencyDataViewModel(IEnumerable<CurrencyDataItemViewModel> collection)
            : base(collection)
        {
```

```
        }
    }
}
```

Файл appsettings.json

```
{
  "Logging": {
    "LogLevel": {
      "Default": "Information",
      "Microsoft": "Warning",
      "Microsoft.Hosting.Lifetime": "Information"
    }
  },
  "AllowedHosts": "*"
}
```

Факультет інформатики та обчислювальної техніки
Кафедра інформатики та програмної інженерії

“ЗАТВЕРДЖЕНО”
Завідувач кафедри
_____ Едуард ЖАРІКОВ
“___” _____ 2023 р.

ПРОГРАМНЕ ЗАБЕЗПЕЧЕННЯ ДЛЯ ЗБОРУ ПРОГРАМНИХ МЕТРИК ЗАСТОСУНКІВ

Програма та методика тестування

КПІ.ІС-9328.045200.04.51

“ПОГОДЖЕНО”
Керівник проєкту:
_____ Олена ХАЛУС

Нормоконтроль:
_____ Катерина ЛІЩУК

Виконавець:
_____ Дар’я Шаховська

Київ – 2023

ЗМІСТ

1 ОБ’ЄКТ ВИПРОБУВАНЬ

Об’єктом випробування є програмне забезпечення для збору програмних метрик застосунків.

2 МЕТА ТЕСТУВАННЯ

Метою тестування є наступне:

- перевірка правильності роботи програмного забезпечення відповідно до функціональних вимог;
- перевірка правильності збереження даних;
- перевірка сумісності застосунку з різними операційними системами (Windows, Linux, …);
- знаходження проблем, помилок і недоліків з метою їх усунення.

3 МЕТОДИ ТЕСТУВАННЯ

Для тестування програмного забезпечення використовуються такі методи:

- статичне тестування – перевіряється програма разом з усією документацією, яка аналізується на предмет дотримання стандартів програмування;
- функціональне тестування – полягає у перевірці відповідності реальної поведінки програмного забезпечення очікуваній;
- системне тестування – перевіряється усе програмне забезпечення в цілому;
- мануальне тестування – тестування без використання автоматизації, тест-кейси пише особа, що тестує програмне забезпечення.

4 ЗАСОБИ ТА ПОРЯДОК ТЕСТУВАННЯ

Тестування виконується мануально, з метою знаходження помилок та недоліків як у функціональній частині програмного забезпечення так і в зручності користування. Для того, щоб перевірити працездатність та відмовостійкість застосунку, необхідно провести наступні тестування:

– динамічне тестування на відповідність функціональним вимогам;

– тестування на виведення повідомлень про помилку, коли це необхідно;

– тестування клієнтської частини на сучасних браузерах (Google Chrome, Mozila Firefox, Microsoft Edge);

– тестування працездатності програми у випадку відсутності з’єднання до мережі;

– тестування інтерфейсу користувача;

– тестування зручності використання.

Факультет інформатики та обчислювальної техніки
Кафедра інформатики та програмної інженерії

“ЗАТВЕРДЖЕНО”
Завідувач кафедри
_____ Едуард ЖАРІКОВ
“___” _____ 2023 р.

ПРОГРАМНЕ ЗАБЕЗПЕЧЕННЯ ДЛЯ ЗБОРУ ПРОГРАМНИХ МЕТРИК ЗАСТОСУНКІВ

Керівництво користувача

КПІ.ІС-9328.045200.05.34

“ПОГОДЖЕНО”
Керівник проєкту:
_____ Олена ХАЛУС

Нормоконтроль:
_____ Катерина ЛІЩУК

Виконавець:
_____ Дар’я ШАХОВСЬКА

Київ – 2023

ЗМІСТ

1 ПРИЗНАЧЕННЯ ПРОГРАМИ

«Metric Collecting» - це програмне забезпечення для збору та візуалізації програмних метрик застосунків. Програма надає можливість збирати та візуалізувати зібрані дані, а також відправляти сповіщення за правилами налаштованими користувачем.

2 ПІДГОТОВКА ДО РОБОТИ З ПРОГРАМНИМ ЗАБЕЗПЕЧЕННЯМ

2.1 Системні вимоги для коректної роботи

Для успішної роботи даного програмного забезпечення необхідне виконання наступних вимог:

– наявність пристрою з сучасним браузером: Google Chrome, Mozila Firefox, Google Edge;

– наявність доступу до Інтернету;

– для розгортання серверу необхідно не менше 300 МБ вільної пам’яті.

2.2 Завантаження застосунку

Для завантаження SDK з NuGet Package Manager-а необхідно у поле пошуку ввести shakhovska_matrics та shakhovska_matrics.AspNetCore та завантажити ці два пакети.

Для того, щоб розгорнути сервери у Docker необхідно з Docker Hub завантажити 2 image-а: shakhovska/metrics_svc та shakhovska/alerts_svc.

2.3 Перевірка коректної роботи

Після завантаження SDK у налаштуваннях проекту, в який ми додаємо SDK, потрібно перевірити список залежностей у файлі проекту (рисунок 2.1).

```
<Project Sdk="Microsoft.NET.Sdk.Web">

  <PropertyGroup>
    <TargetFramework>net7.0</TargetFramework>
  </PropertyGroup>

  <ItemGroup>
    <PackageReference Include="shakhovska_metrics" Version="1.0.0" />
    <PackageReference Include="shakhovska_metrics.AspNetCore" Version="1.0.0" />
  </ItemGroup>

</Project>
```

Рисунок 2.1 – Перевірка успішного завантаження пакетів

Після завантаження імеджів з Docker Hub необхідно перевірити список наявних імеджів командою: docker image list (рисунок 2.2).

```
C:\Users\Daria>docker image list
REPOSITORY                TAG       IMAGE ID
shakhovska/metrics_svc    latest    f2680bb28e35
shakhovska/alerts_svc     latest    c8568f914cd2
```

Рисунок 2.2 – Перевірка наявності імеджів

3 ВИКОНАННЯ ПРОГРАМИ

Після успішного завантаження SDK, додаємо простір імен ShakhovskaSdk у потрібні файли, де будуть використовуватись методи для збору метрик. Використовуємо методи SDK для збору даних у коді застосунку. Приклад використання методів показаний на рисунку 3.1.

```
[HttpGet]
[ActionName("Data")]
0 references
public async Task<JsonResult> GetCurrencyData()
{
    var sw = AppMetrics.StartTrackingBankRequestTime();
    var privatData = await _privatDataAccess.GetCurrencyDataAsync();
    AppMetrics.FinishTrackingBankRequestTime(sw);
```

Рисунок 3.1 – Приклад збору метрик застосунку

Для налаштування відправки сповіщень у файлі з конфігураціями alert_rules.yml вказуємо правила за якими буде здійснюватися відправка (рисунок 3.2).

alert_rules.yml

C: > Users > Daria > Desktop > configs > alert_rules.yml

```
groups:
- name: health-check
  rules:
  # Alert for any instance that is unreachable for >1 minute.
  - alert: InstanceDown
    expr: up == 0
    for: 1m
    annotations:
      summary: "Instance {{ $labels.instance }} down"
      description: "{{ $labels.instance }} of job {{ $labels.job }} has been down for more than 1 minute."
```

Рисунок 3.2 – Приклад налаштування правил для відправки сповіщень

Відкриваємо застосунок, у якому налаштували збір метрик та робимо дії, які призводять до збору часових міток. Відкриваємо інструмент для візуалізації зібраних даних, переходимо у вкладку Dashboard, обираємо потрібну нам метрику, наприклад bank_request, та виставляємо час, за який хочемо переглянути дані. Вигляд даної вкладки наведений на рисунку 3.3.

Рисунок 3.3 – Візуалізація зібраних метрик

Переходимо у вкладку Alerts та можемо переглянути список наявних сповіщень, приклад вкладки продемонстрований на рисунку 3.4.

Рисунок 3.4 – Перегляд правил відправки сповіщень

Бачимо, що дане правило перевіряє, чи піднятий сервер. Якщо ні і протягом хвилини ситуація не змінюється, то в Телеграм-БОТ відправляється відповідне сповіщення. Скріншот сповіщення, яке отримали, наведений на рисунку 3.5.

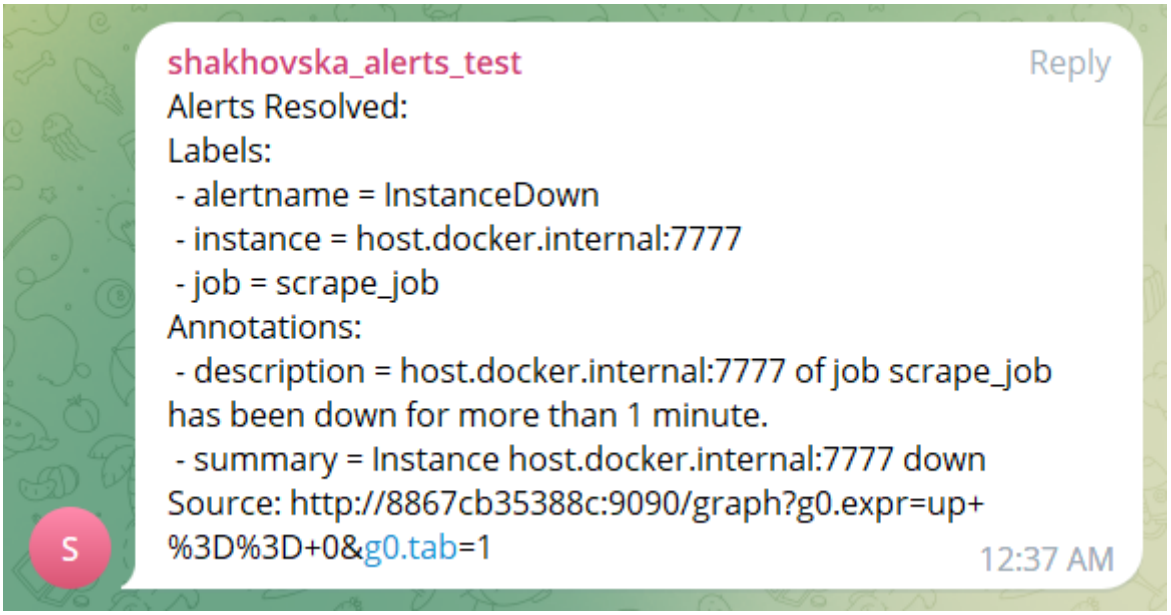


Рисунок 3.5 – Відправлене сповіщення у Телеграм-БОТ

З спадного списку Status обираємо Configuration та можемо подивитись наявні конфігурації серверу, такі як: частота скрапінгу даних, кінцеві ендпоінти і таке інше. Вміст даної вкладки наведений на рисунку 3.6.

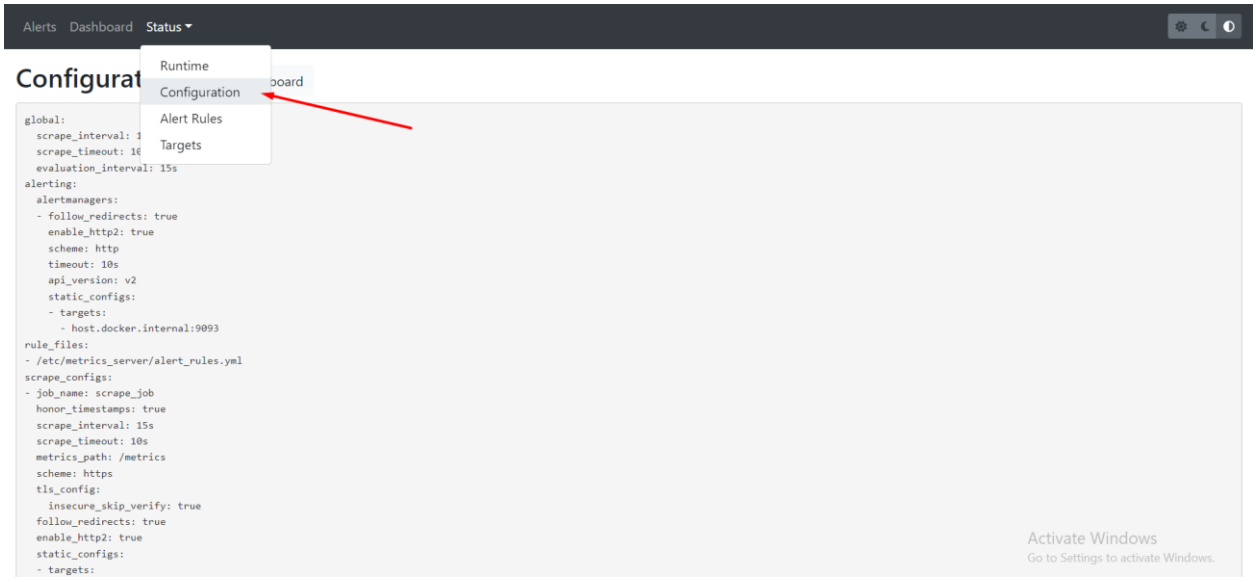


Рисунок 3.6 – Вміст вкладки Configuration

З спадного списку Status обираємо Targets та можемо побачити список наявних таргетів, їх статус, коли відбувався останній скрапінг даних, тривалість цього скрапінгу та наявні помилки. Вміст даної вкладки наведений на рисунку 3.7.

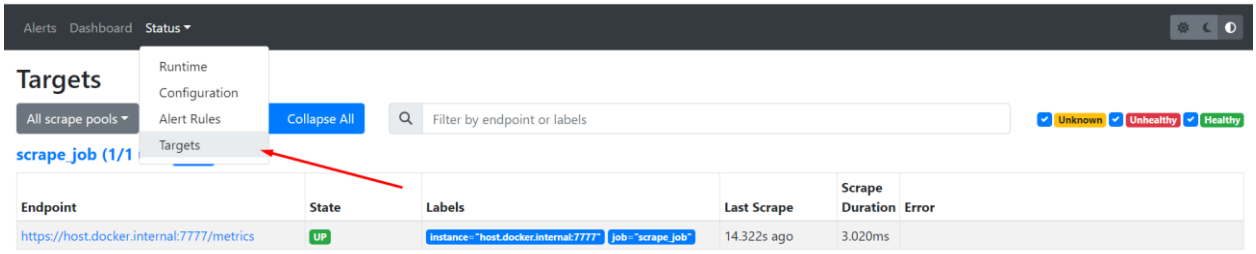


Рисунок 3.7 – Вміст вкладки Targets

З спадного списку Status обираємо Alert Rules та можемо переглянути список наявних правил для відправки оповіщень. Вміст даної вкладки наведений на рисунку 3.8.

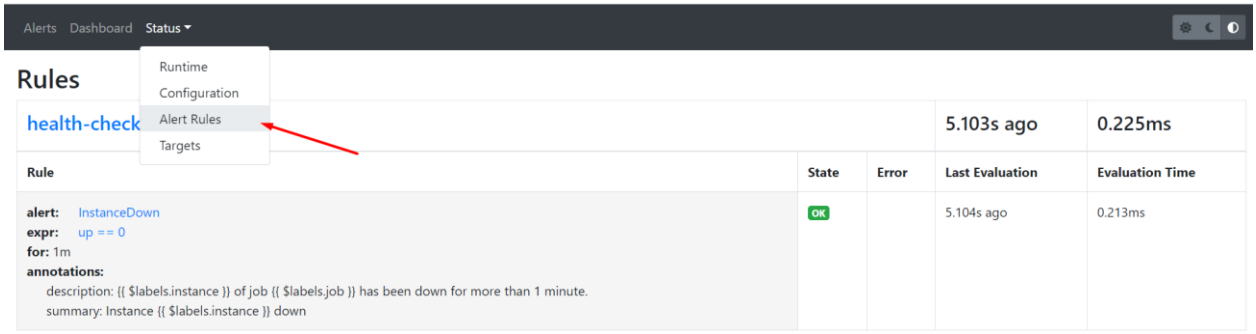


Рисунок 3.8 – Вміст вкладки Alert Rules

З спадного списку Status обираємо Runtime та можемо переглянути інформацію стосовно поточного рантайму. Вміст даної вкладки наведений на рисунку 3.9.

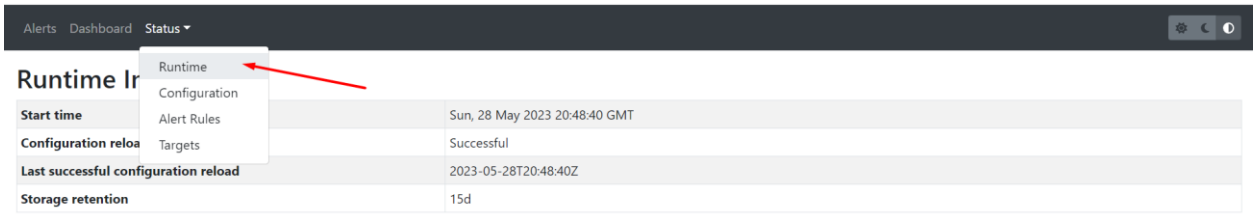


Рисунок 3.9 – Вміст вкладки Runtime

Факультет інформатики та обчислювальної техніки
Кафедра інформатики та програмної інженерії

“ЗАТВЕРДЖЕНО”
Завідувач кафедри
_____ Едуард ЖАРІКОВ
“___” _____ 2023 р.

ПРОГРАМНЕ ЗАБЕЗПЕЧЕННЯ ДЛЯ ЗБОРУ ПРОГРАМНИХ МЕТРИК ЗАСТОСУНКІВ

Графічний матеріал

КПІ.ІС-9328.045200.06.99

“ПОГОДЖЕНО”
Керівник проєкту:
_____ Олена ХАЛУС

Нормоконтроль:
_____ Катерина Ліщук

Виконавець:
_____ Дар’я Шаховська

Київ – 2023

					КПІ.ІС-9328.045200.06.99.ССВ	
Зм.	Арк.	№ документа	Підпис	Дата	Схема структурна ваірантів використань	Літера / Маса / Масштаб
Розробив		Шаховська Д.О				
Перевірив		Халус О.А.				Аркуш / Аркушів
Т. кон.						
Н. кон.		Ліщук К.І.			Програмне забезпечення для збору програмних метрик застосунків	КПІ ім.Ігоря Сікорського Кафедра ІПІ гр. ІП-91
Затвердив		Жаріков Е.В.				

Зм.	Арк.	№ докум.	Підп.	Дата
Розроб.		Шаховська Д.О.		
Перев.		Харус О.А.		
Т. Кон.				
Н. Кон.		Пшук К.І.		
Затв.		Жарікова Е.В.		

КПІ.ІС-9328.045200.06.99.ССМ

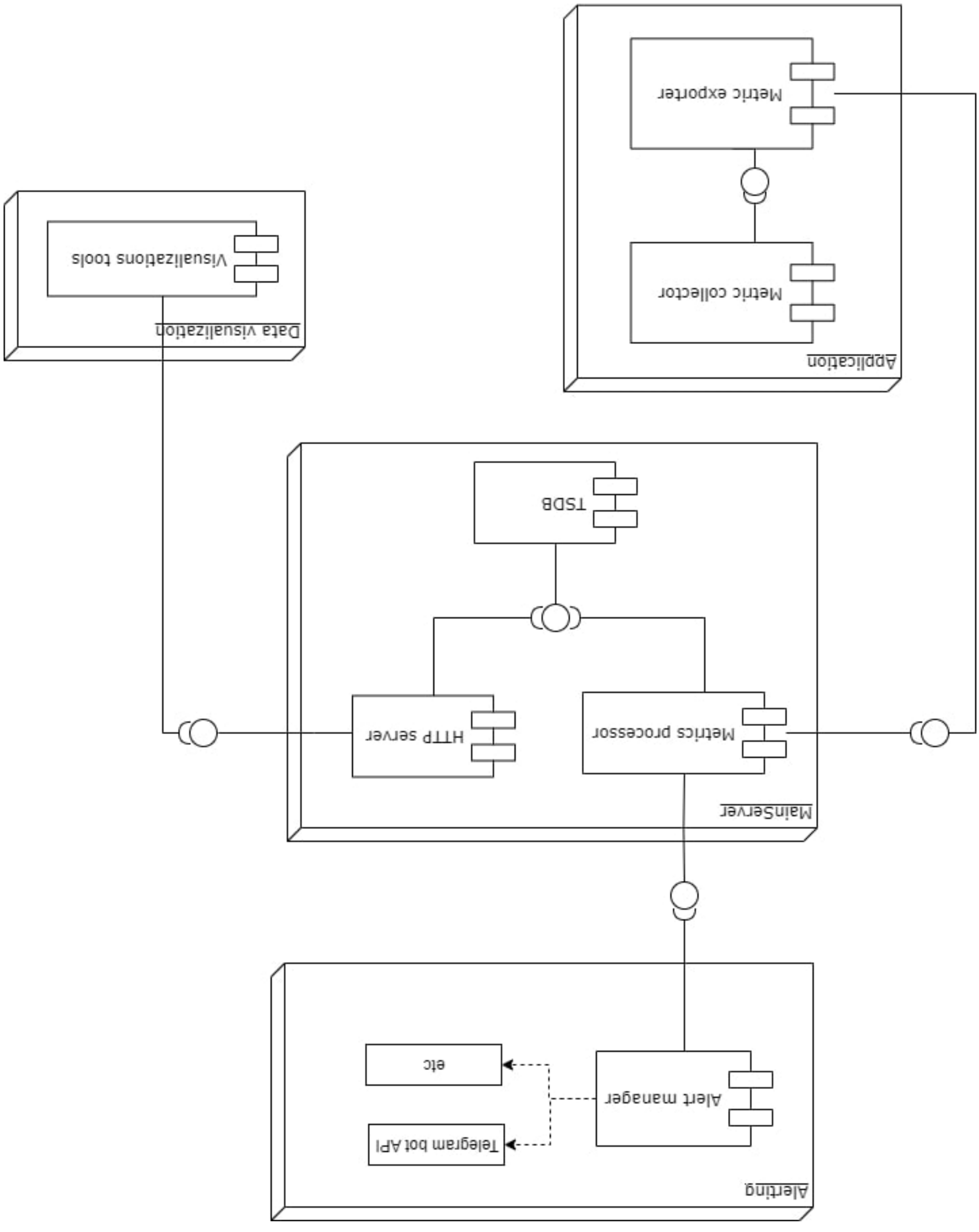
Схема структурна компоненті програмного забезпечення

Літ.	Арк.	Аркушів
		1

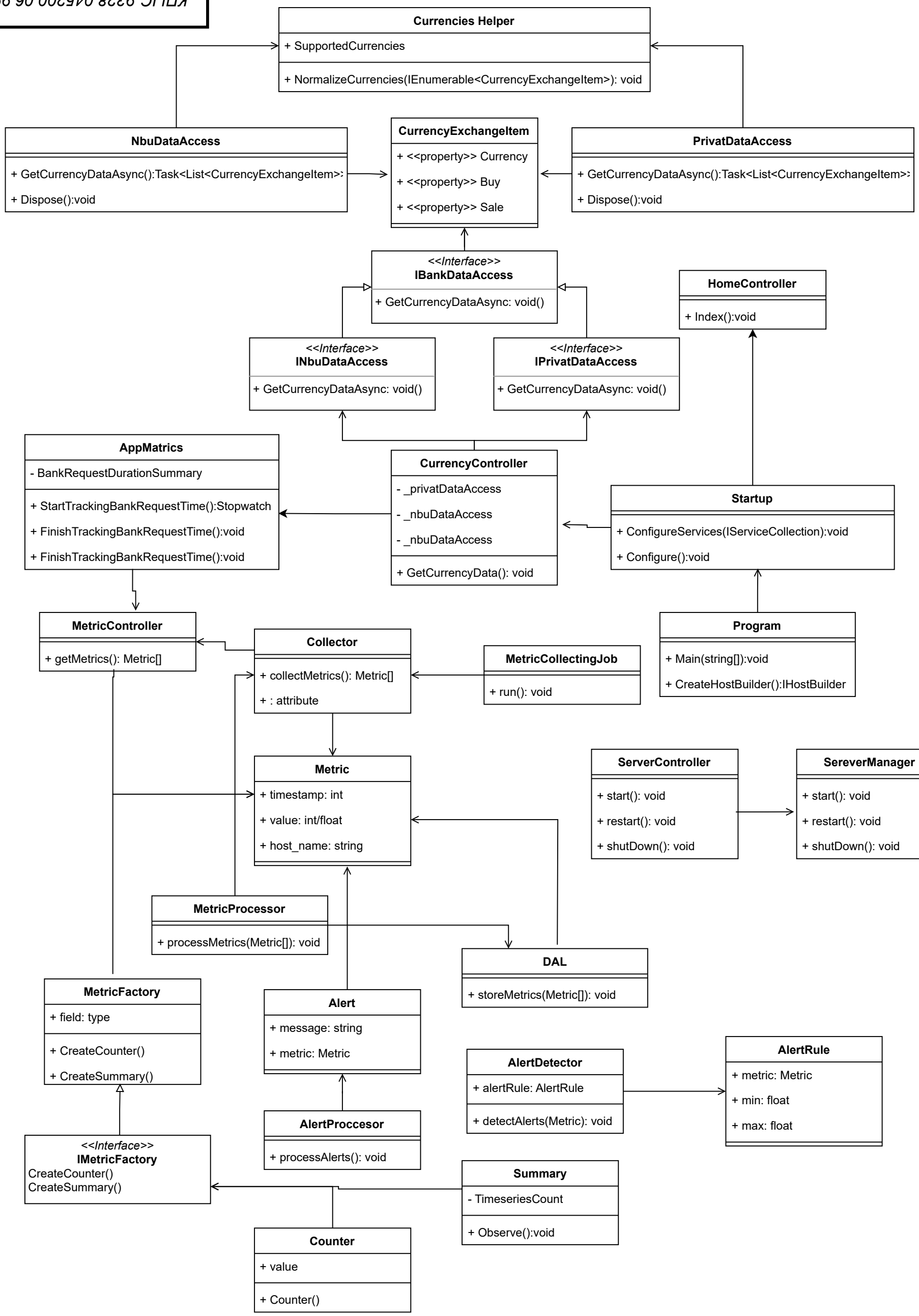
Аркуш | Аркушів

Програмне забезпечення для збору програмних метрик застосунків

КПІ ім.Ігоря Сікорського
Кафедра ІПІ
гр. ІП-91



КПІ.ІС-9328.045200.06.99.ССК



Зм.	Арк.	№ докум.	Підп.	Дата
Розроб.		Шаховська Д.О.		
Перев.		Халус О.А.		
Т. Кон.				
Н. Кон.		Ліщук К.І.		
Затв.		Жаріков Е.В.		

КПІ.ІС-9328.045200.06.99.ССК

Схема структурна класів програмного забезпечення

Лит.	Арк.	Аркушів
		1
Аркуш	Аркушів	

Програмне забезпечення для збору програмних метрик застосунків

КПІ ім.Ігоря Сікорського
Кафедра ІПІ
гр. ІП-91