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

Навчально-науковий інститут атомної та теплової енергетики

Кафедра цифрових технологій в енергетиці

«До захисту допущено»
Завідувач кафедри ЦТЕ
Наталія АУШЕВА
«___» _____2026р.

Дипломна робота
на здобуття ступеня бакалавр

За освітньою програмою «Цифрові технології в енергетиці»
Спеціальність 122 «Комп'ютерні науки»
на тему: «Серверна частина вебсистеми моніторингу доступності програмних застосунків»

Виконав: студент 4 курсу, групи ТР-23
Шаповалов Денис Едуардович _____
(прізвище, ім'я, по батькові) (підпис)

Керівник: асистент Антон ПАСІЧНЮК _____
(посада, науковий ступінь, вчене звання, ім'я, ПРІЗВИЩЕ) (підпис)

Рецензент: _____
(посада, науковий ступінь, вчене звання, ім'я, ПРІЗВИЩЕ) (підпис)

Н.контроль: _____ _____
(посада, ім'я, ПРІЗВИЩЕ) (підпис)

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

Київ – 2026

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

НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ АТОМНОЇ ТА ТЕПЛОВОЇ ЕНЕРГЕТИКИ

Кафедра ЦИФРОВИХ ТЕХНОЛОГІЙ В ЕНЕРГЕТИЦІ

Рівень вищої освіти – перший (бакалаврський)

спеціальність 122 «Комп'ютерні науки»

Освітньо-професійна програма «Цифрові технології в енергетиці»

ЗАТВЕРДЖУЮ
Завідувач кафедри ЦТЕ
Наталія АУШЕВА
«___» _____2026р.

ЗАВДАННЯ
на дипломну роботу студенту

Шаповалову Денису Едуардовичу
(прізвище, ім'я, по батькові)

1. Тема роботи Серверна частина вебсистеми моніторингу доступності програмних застосунків

Науковий керівник Пасічнюк Антон Олексійович, асистент
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

затверджені наказом по університету від «28» травня 2026 року № 1992-с

2. Термін подання роботи студентом _____

3. Вихідні дані до роботи моніторинг доступності вебзастосунків; аналіз простоїв системи; реалізація механізмів перевірки;

4. Зміст роботи аналіз проблем забезпечення бесперервної роботи вебзастосунків та наслідків простоїв системи; огляд існуючих підходів і сервісів моніторингу доступності.

5. Орієнтовний перелік графічного матеріалу архітектура системи; діаграми взаємодії компонентів системи; таблиця порівняння сервісів; діаграма активності; діаграма розгортання застосунку.

6. Дата видачі завдання 13.09.2025

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів виконання дипломної роботи	Термін виконання етапів роботи	Примітка
1.	Вибір теми роботи	14.12.25	Виконано
2.	Аналіз методів та засобів розв'язання задачі	23.03.2026-19.04.2026	Виконано
3.	Розробка архітектури та загальної структури системи	20.04.2026-26.04.2026	Виконано
4.	Розробка окремих підсистем	27.04.2026-03.05.2026	Виконано
5.	Програмна реалізація системи	04.05.2026-17.05.2026	Виконано
6.	Оформлення пояснювальної записки	18.05.2026-30.05.2026	Виконано
7.	Захист програмного забезпечення	13.05.26	Виконано
8.	Передзахист	3.06.26	Виконано
9.	Захист	16.06.25	Виконано

Студент _____ Шаповалов Д.Е.
(підпис) (прізвище та ініціали)

Керівник роботи _____ Пасічнюк А.О.
(підпис) (прізвище та ініціали)

АНОТАЦІЯ

Дипломна робота виконана 79 на сторінках, містить 10 ілюстрацій, додатки та список джерел в переліку посилань.

Метою роботи є розробка серверної частини програмної системи моніторингу доступності вебзастосунків, що забезпечує автоматичну перевірку стану сервісів, виявлення інцидентів та інформування користувачів, з можливістю налаштування параметрів моніторингу та відображення статусу системи.

Методи та засоби: аналіз підходів до моніторингу доступності вебсервісів, використання HTTP перевірок; застосування асинхронної обробки даних; модульне проектування архітектури системи; платформа для розробки програмного забезпечення .NET, використання вебфреймворку ASP.NET Core; реалізація REST API для взаємодії з системою; використання JWT токену для автентифікації та авторизації користувачів; застосування хмарної платформи Microsoft Azure для розгортання, тестування та налагодження системи.

Результати: розроблено програмну систему моніторингу доступності вебзастосунків, яка забезпечує автоматичну перевірку стану сервісів, виявлення інцидентів та їх обробку. Система надає можливість створення та управління моніторами, підтримує асинхронну обробку перевірок за допомогою черг повідомлень, а також реалізує механізми обробки інцидентів. Передбачено функціональність сторінки статусу для відображення стану сервісів у реальному часі. Система протестована у хмарному середовищі Microsoft Azure та є готовим рішенням, рекомендованим для подальшого розвитку та впровадження у реальних умовах.

Ключові слова: МОНІТОРИНГ, БЕЗПЕРЕРВНІСТЬ РОБОТИ СЕРВІСІВ, .NET, ASP.NET CORE, REST API, AZURE, АСИНХРОННА ОБРОБКА

ANNOTATION

The thesis is completed on 79 pages, contains 10 illustrations, appendices and sources in the reference list.

Objective: development of a web application availability monitoring system that provides automatic service status checks, incident detection, and user notification, with the ability to configure monitoring parameters and display the system status.

Methods and tools: analysis of approaches to web service availability monitoring; use of HTTP checks; application of asynchronous data processing and message queues; modular system architecture design; use of the .NET platform and C# programming language with the ASP.NET Core framework; implementation of a REST API for system interaction; use of JWT tokens for user authentication and authorization; deployment, testing, and debugging using the Microsoft Azure cloud platform.

Results: a web application availability monitoring system was developed, providing automatic service status checks, incident detection, and handling. The system enables the creation and management of monitors, supports asynchronous processing of checks using message queues, and implements incident handling mechanisms. A status page functionality is included to display real-time service status. The system was tested in the Microsoft Azure cloud environment and represents a ready solution recommended for further development and deployment in real-world conditions.

Keywords: MONITORING, SERVICE AVAILABILITY, .NET, ASP.NET CORE, REST API, AZURE, MESSAGE QUEUES, ASYNCHRONOUS PROCESSING

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

API – Application Programming Interface (програмний інтерфейс для взаємодії між компонентами системи)

HTTP – HyperText Transfer Protocol (протокол передачі гіпертексту)

HTTPS – HyperText Transfer Protocol Secure (захищена версія протоколу HTTP)

REST – Representational State Transfer (архітектурний стиль побудови вебсервісів)

JWT – JSON Web Token (формат токена для автентифікації та передачі даних)

JSON – JavaScript Object Notation (формат обміну даними у вигляді текстових структур)

ORM – Object-Relational Mapping (технологія взаємодії з базою даних через об’єкти)

CRUD – Create, Read, Update, Delete (базові операції створення, читання, оновлення та видалення даних)

SQL – Structured Query Language (мова структурованих запитів до бази даних)

CPU – Central Processing Unit (центральний процесор)

RAM – Random Access Memory (оперативна пам’ять)

DNS – Domain Name System (система доменних імен)

TCP – Transmission Control Protocol (протокол керування передачею даних)

.NET – програмна платформа для розробки додатків

EF Core – Entity Framework Core (ORM-бібліотека для роботи з базою даних)

Azure – хмарна платформа Microsoft

Docker – платформа контейнеризації застосунків

CI/CD – Continuous Integration / Continuous Deployment (безперервна інтеграція та доставка)

ВСТУП

У сучасних умовах стрімкого розвитку інформаційних технологій вебзастосунки стали критично важливою складовою бізнес-процесів, цифрових сервісів та інфраструктури підприємств. Безперервність їх роботи безпосередньо впливає на якість обслуговування користувачів, фінансові показники компаній та їхню репутацію. Водночас висока залежність від мережевих технологій, складність програмних систем та постійні зміни у навантаженні створюють значні ризики виникнення збоїв і простоїв. Навіть короткочасна недоступність сервісів може призвести до втрати клієнтів, зниження довіри та фінансових збитків.

У сучасних мережевих системах вебзастосунки функціонують як частина глобальної інфраструктури, що базується на протоколах передачі даних, таких як HTTP та TCP/IP. Вебкомунікація здійснюється за принципом «запит-відповідь», що визначає основу взаємодії клієнта та сервера [2]. У зв’язку з цим стабільність роботи сервісів напряму залежить від коректності функціонування мережевих протоколів та серверної інфраструктури.

Сучасні вебзастосунки часто функціонують у розподілених середовищах, використовують мікросервісну або гібридну архітектуру, інтегруються з зовнішніми API та хмарними сервісами. Така складність підвищує ймовірність виникнення збоїв, оскільки відмова навіть одного компонента може вплинути на роботу всієї системи. Крім того, зростання кількості користувачів та обсягу даних створює додаткове навантаження, що потребує постійного контролю за станом ресурсів та швидкістю обробки запитів.

У таких умовах традиційні підходи до контролю стану систем, що базуються на ручному спостереженні або періодичній перевірці, є недостатньо ефективними. По-перше, сучасні вебзастосунки працюють у цілобовому режимі, що унеможливлює постійний контроль з боку людини. По-друге, швидкість змін стану системи вимагає оперативного реагування, яке складно забезпечити без автоматизації. По-третє, людський фактор може призводити до пропуску

критичних інцидентів або несвоєчасного реагування на них. Крім того, складні розподілені системи часто включають велику кількість компонентів, взаємодія між якими ускладнює визначення джерела проблеми без використання спеціалізованих інструментів.

Ще одним важливим аспектом є необхідність не лише фіксації факту відмови, але й оцінки якості роботи системи. Навіть якщо вебзастосунок формально залишається доступним, збільшення часу відгуку або часткові помилки можуть суттєво впливати на досвід користувачів. Саме тому сучасні системи моніторингу повинні забезпечувати збір та аналіз різних показників, що характеризують стан сервісу, а також дозволяти виявляти тенденції до погіршення його роботи.

Актуальність теми даної дипломної роботи зумовлена необхідністю створення ефективної системи моніторингу доступності вебзастосунків, яка забезпечує автоматичну перевірку стану сервісів, своєчасне виявлення інцидентів та інформування користувачів. Використання таких систем дозволяє мінімізувати час простою, підвищити надійність роботи програмного забезпечення та забезпечити оперативне реагування на проблеми. Особливу важливість має також можливість відображення актуального стану системи у зручному вигляді, що сприяє підвищенню прозорості для користувачів і зацікавлених сторін.

Незважаючи на наявність значної кількості готових рішень для моніторингу, вони часто мають обмеження щодо гнучкості налаштування, масштабованості або інтеграції з іншими системами. Комерційні сервіси можуть вимагати регулярних фінансових витрат, що не завжди є доцільним для невеликих проєктів або навчальних цілей. У свою чергу, відкриті рішення потребують додаткових зусиль для розгортання та підтримки, а також не завжди дозволяють реалізувати специфічну логіку роботи без суттєвих змін у їхній структурі. Це обумовлює доцільність розробки власної системи, яка може бути адаптована під конкретні вимоги та забезпечити необхідний рівень контролю й функціональності.

Особливістю сучасних систем моніторингу є використання асинхронних підходів до обробки даних, що дозволяє ефективно працювати з великою кількістю

перевірок. Застосування черг повідомлень та фонових процесів дає змогу розподіляти навантаження, підвищувати продуктивність та забезпечувати безперервність роботи системи. Крім того, інтеграція з хмарними платформами відкриває можливості для масштабування та підвищення доступності розробленого рішення.

Метою даної дипломної роботи є розробка серверної частини системи моніторингу доступності вебзастосунків. Система повинна підтримувати асинхронну обробку перевірок, забезпечувати масштабованість та інтеграцію з сучасними технологіями.

Для досягнення поставленої мети визначено такі завдання:

- проаналізувати існуючі методи моніторингу вебзастосунків;
- здійснити огляд сучасних систем моніторингу;
- сформувати вимоги до розроблюваної системи;
- спроєктувати архітектуру серверної частини;
- обрати засоби розробки;
- реалізувати основні модулі системи;
- забезпечити взаємодію із системою через REST API;
- виконати розгортання системи у хмарному середовищі.

Об’єктом дослідження є процес моніторингу доступності вебзастосунків, а предметом – методи та засоби розробки серверних систем моніторингу з використанням сучасних технологій.

Дипломна робота складається зі вступу, п’яти основних розділів, висновків, списку використаних джерел та додатків. У роботі послідовно розглянуто аналіз предметної області, проєктування архітектури системи, реалізацію програмних модулів та результати її функціонування.

1 ЗАДАЧА МОНІТОРИНГУ ДОСТУПНОСТІ ВЕБЗАСТОСУНКІВ

У сучасних умовах функціонування вебзастосунків, забезпечення їхньої доступності та стабільної роботи є одним із ключових завдань для IT-систем. Більшість сучасних сервісів працюють у цілодобовому режимі, обслуговуючи значну кількість користувачів, тому навіть короткочасні збої можуть призводити до фінансових втрат, зниження якості обслуговування та втрати довіри користувачів. Особливістю таких систем є їхня розподіленість, залежність від мережевої інфраструктури та використання хмарних технологій, що ускладнює контроль їхнього стану.

Сучасні вебзастосунки часто інтегруються з великою кількістю зовнішніх сервісів, баз даних та API, що додатково підвищує складність їх функціонування. Відмова одного з компонентів може призвести до часткової або повної недоступності всієї системи. При цьому проблема не завжди проявляється у вигляді повного відключення сервісу - вона може виражатися у збільшенні часу відповіді, нестабільній роботі або періодичних помилках. Такі ситуації складніше виявити без використання спеціалізованих інструментів моніторингу.

У таких умовах використання автоматизованих систем моніторингу доступності стає необхідністю. Вони дозволяють здійснювати регулярну перевірку стану сервісів, виявляти інциденти та оперативно інформувати про них. Крім того, такі системи забезпечують накопичення історичних даних, що дозволяє аналізувати поведінку сервісів у довгостроковій перспективі та виявляти тенденції до погіршення їхньої роботи.

Важливим аспектом є також необхідність зменшення впливу людського фактору. Ручний контроль стану систем є неефективним через обмеженість ресурсів та високу ймовірність пропуску критичних подій. Автоматизований моніторинг дозволяє забезпечити постійний контроль без перерв, а також швидке

реагування на будь-які відхилення.

Проте існуючі рішення не завжди забезпечують необхідний рівень гнучкості, масштабованості або адаптації під конкретні задачі. Деякі системи орієнтовані на загальні сценарії використання і не дозволяють реалізувати специфічні вимоги, пов’язані з особливостями конкретної інфраструктури.

Деякі існуючі рішення можуть бути складними у налаштуванні або вимагати значних ресурсів для підтримки. Це обумовлює доцільність розробки власної системи, яка буде максимально відповідати поставленим задачам. Така система повинна забезпечувати створення та управління моніторами доступності вебзастосунків, виконувати перевірки сервісів за допомогою HTTP/HTTPS-запитів, вимірювати час відгуку та аналізувати продуктивність. Також важливими функціями є виявлення інцидентів на основі результатів перевірок, зменшення кількості хибних спрацювань за рахунок повторних перевірок або порогових значень, надсилання сповіщень про збої, відображення поточного стану сервісів на сторінці статусу та забезпечення взаємодії із системою через API.Додатково система повинна забезпечувати можливість зберігання історії перевірок, що дозволяє проводити подальший аналіз та оцінювати стабільність роботи сервісів. Наявність статистичних даних дає змогу визначати показники доступності, виявляти частоту виникнення інцидентів та оцінювати якість роботи системи в цілому.

Виходячи з цього, основною задачею даної дипломної роботи є розробка серверної частини системи моніторингу доступності вебзастосунків, яка реалізує зазначений функціонал. Система повинна бути спроєктована з урахуванням модульності, що забезпечить можливість її подальшого розширення, а також масштабованості для обробки великої кількості перевірок. Крім того, важливими вимогами є надійність, ефективність та безпека роботи системи.

Особлива увага приділяється використанню асинхронних підходів до обробки даних. Це дозволяє ефективно розподіляти навантаження між компонентами системи, уникати блокування процесів та забезпечувати стабільну

роботу навіть при значній кількості одночасних перевірок. Використання черг повідомлень та фонових процесів є ключовим елементом для реалізації такого підходу.

Для успішної реалізації поставленої задачі необхідно виконати низку взаємопов’язаних етапів розробки:

- Спроєктувати архітектуру системи: визначити основні компоненти системи, зокрема модулі конфігурації моніторингу, перевірки доступності, виявлення інцидентів, сповіщень і зберігання даних; обрати технологічний стек для реалізації;
- Розробити модуль перевірки доступності: реалізувати механізм виконання HTTP/HTTPS-запитів для перевірки стану вебзастосунків із можливістю налаштування інтервалів виконання перевірок;
- Розробити модуль обробки результатів і виявлення інцидентів: забезпечити аналіз отриманих результатів, визначення стану контрольованих сервісів і виявлення збоїв або погіршення показників їхньої роботи;
- Розробити модуль сповіщень: реалізувати механізм інформування користувачів про виникнення інцидентів і відновлення роботи сервісів із можливістю налаштування способів отримання повідомлень;
- Інтегрувати компоненти системи та організувати обробку даних: забезпечити взаємодію між модулями з використанням асинхронних механізмів і черг повідомлень;
- Провести розгортання системи: розгорнути реалізований застосунок у хмарному середовищі Microsoft Azure.

Таким чином, реалізація поставленої задачі дозволить створити ефективну систему моніторингу доступності вебзастосунків, яка забезпечить автоматизацію контролю стану сервісів, своєчасне виявлення інцидентів та підвищення надійності роботи інформаційних систем. Розроблена система може бути використана як основа для подальшого розвитку, розширення функціоналу та інтеграції з іншими компонентами IT-інфраструктури.

2 АНАЛІЗ МЕТОДІВ МОНІТОРИНГУ ВЕБЗАСТОСУНКІВ

Ефективна розробка системи моніторингу доступності вебзастосунків потребує аналізу існуючих підходів до контролю стану сервісів та інструментів, що застосовуються для цієї задачі. У даному розділі розглядаються основні методи моніторингу, включаючи перевірку доступності, аналіз продуктивності та підходи до виявлення інцидентів.

Огляд сучасних рішень у сфері моніторингу вебзастосунків дозволить визначити переваги та недоліки, а також обґрунтувати доцільність розробки власної системи, адаптованої до поставлених вимог.

2.1 Загальні підходи до моніторингу вебсервісів

Моніторинг вебсервісів є важливою складовою забезпечення їх стабільної та безперервної роботи. Він дозволяє контролювати стан системи, своєчасно виявляти збої та оцінювати якість її функціонування. У сучасних умовах, коли застосунки працюють у розподілених і хмарних середовищах, ефективний моніторинг стає необхідним для підтримки надійності сервісів.

Існує кілька основних підходів до моніторингу вебсервісів, які відрізняються способом отримання даних та рівнем деталізації. Найпоширенішими є активний моніторинг, що передбачає виконання зовнішніх перевірок сервісу, та пасивний моніторинг, який базується на аналізі внутрішніх даних системи, таких як логи та метрики.

Сучасні підходи до моніторингу передбачають використання як активних, так і пасивних методів збору даних. Активний моніторинг базується на генерації запитів до системи з метою перевірки її стану, тоді як пасивний моніторинг аналізує вже наявні дані, такі як журнали подій та метрики. Комбінування цих підходів дозволяє отримати більш повну картину стану системи та підвищити точність

виявлення проблем.

Крім того, моніторинг може здійснюватися на різних рівнях: перевірка доступності сервісу, аналіз його продуктивності та контроль коректності виконання бізнес-логіки. Використання комбінованого підходу дозволяє отримати більш повну інформацію про стан системи та підвищити ефективність виявлення проблем.

Таким чином ефективний моніторинг вебзастосунків не обмежується використанням одного підходу або методу. Найбільш результативним є комбінування різних типів моніторингу, що дозволяє отримати повнішу картину стану системи. Зокрема, поєднання перевірки доступності, аналізу продуктивності та механізмів виявлення інцидентів дає змогу не лише фіксувати факт відмови, але й визначати причини її виникнення.

Важливо також враховувати, що сучасні вебсистеми постійно змінюються, а їхня інфраструктура ускладнюється. У зв’язку з цим система моніторингу повинна бути достатньо гнучкою, щоб адаптуватися до нових умов, а також забезпечувати можливість розширення функціоналу без суттєвих змін у її архітектурі.

У наступних підрозділах детальніше розглянуто основні методи перевірки доступності, моніторинг продуктивності та підходи до виявлення інцидентів.

2.1.1 Методи перевірки доступності

Перевірка доступності вебсервісів найчастіше здійснюється за допомогою HTTP-запитів, що дозволяє оцінити як факт доступності, так і коректність відповіді. Протокол HTTP є безстанним, що означає незалежність кожного запиту від попередніх. Це спрощує реалізацію перевірок, але водночас вимагає додаткових механізмів для аналізу послідовності взаємодій.

Базовий підхід передбачає надсилання запиту до сервісу та аналіз отриманого HTTP-статус коду. Наприклад, код 200 свідчить про успішну обробку запиту, тоді

як коди 4xx або 5xx можуть вказувати на наявність проблем. Однак перевірка лише статус-коду не завжди є достатньою, оскільки сервіс може відповідати формально коректно, але при цьому працювати некоректно. Тому часто застосовується перевірка вмісту відповіді, яка дозволяє переконатися, що сервіс повертає очікувані дані.

На рисунку 2.1 наведено схему виконання перевірки доступності вебсервісу. Як видно зі схеми, процес включає послідовне надсилання HTTP-запиту, отримання відповіді від сервісу, аналіз статусу та часу відгуку, а також формування результату перевірки. Якщо відповідь відповідає очікуваним умовам, сервіс вважається доступним, і результат фіксується як успішний. У разі отримання помилки або перевищення допустимого часу очікування система визначає це як інцидент. Такий підхід дозволяє автоматизувати контроль стану вебзастосунків і забезпечує основу для подальшого аналізу продуктивності та надійності сервісу.

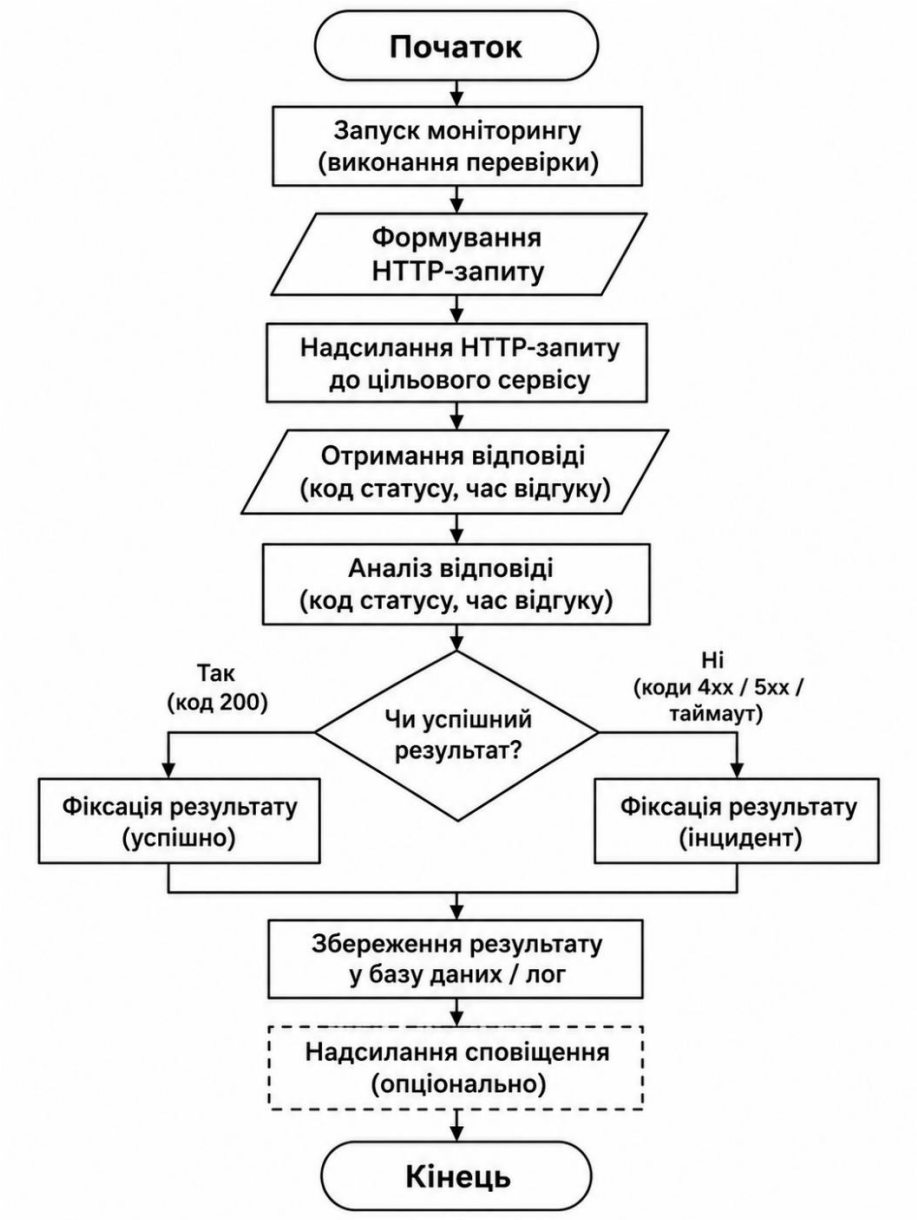


Рисунок 2.1 – Схема виконання перевірки доступності вебсервісу

Важливим аспектом є також тип запиту, який використовується для перевірки. У більшості випадків застосовуються GET-запити, проте для більш точного контролю можуть використовуватися POST або інші типи запитів, що дозволяють перевіряти роботу окремих функціональних можливостей сервісу. Це особливо актуально для систем, у яких основна логіка реалізована через API.

Крім того, важливим параметром є час відгуку сервісу. Навіть якщо сервіс відповідає, але робить це із значною затримкою, це може свідчити про проблеми з

продуктивністю або перевантаженням. У таких випадках встановлюються граничні значення часу відповіді, перевищення яких вважається інцидентом. Таким чином, перевірка доступності поступово переходить у площину аналізу якості роботи сервісу.

Крім HTTP-перевірок, для визначення причин недоступності вебзастосунку можуть застосовуватися перевірки на мережевому рівні, зокрема TCP- та DNS-перевірки. HTTP-запит дозволяє встановити, чи здатний сервіс обробити звернення та повернути коректну відповідь. Проте у випадку помилки він не завжди одразу показує, на якому саме етапі виникла проблема: під час визначення адреси сервера, встановлення мережевого з’єднання або вже безпосередньо під час обробки запиту застосунком.

TCP-перевірка полягає у спробі встановити з’єднання з визначеним сервером через указаний порт, наприклад порт 80 для HTTP або 443 для HTTPS. Якщо з’єднання успішно встановлюється протягом заданого часу, це означає, що вузол доступний через мережу, а відповідний порт відкритий для приймання з’єднань. Якщо ж з’єднання не вдається встановити або виникає таймаут, це може свідчити про недоступність сервера, помилку мережевої конфігурації, блокування порту, несправність балансувальника навантаження чи зупинку відповідного сервісу.

Водночас успішна TCP-перевірка не гарантує коректної роботи самого вебзастосунку. Сервер може приймати мережеві з’єднання, але повертати помилки під час обробки HTTP-запитів або некоректно виконувати бізнес-операції. Тому TCP-перевірка доцільна як базовий рівень діагностики, який дозволяє відокремити мережеву недоступність від помилок прикладного рівня.

DNS-перевірка використовується для контролю коректності перетворення доменного імені вебсервісу на мережеву адресу сервера. Перед надсиланням HTTP-запиту система повинна визначити IP-адресу, що відповідає доменному імені. Якщо DNS-запис відсутній, містить неправильне значення або DNS-сервер не відповідає, користувачі не зможуть звернутися до вебзастосунку навіть тоді, коли сам сервер і застосунок працюють справно.

Під час DNS-перевірки можуть контролюватися успішність отримання IP-адреси, час виконання DNS-запиту та, за потреби, відповідність отриманого результату очікуваному значенню. Це дозволяє виявляти проблеми, пов’язані з неправильним налаштуванням домену, зміною маршрутизації, недоступністю DNS-інфраструктури або помилковим спрямуванням запитів на інший сервер.

Поєднання DNS, TCP та HTTP перевірок забезпечує більш точне визначення причини збою. Якщо не виконується DNS-перевірка, проблема пов’язана з визначенням адреси сервісу. Якщо DNS працює, але TCP-з’єднання не встановлюється, імовірною є мережева проблема або недоступність потрібного порту. Якщо ж DNS- і TCP-перевірки проходять успішно, але HTTP-запит завершується помилкою, проблема, найімовірніше, виникла на рівні самого вебзастосунку або його внутрішньої логіки.

У більш складних сценаріях можуть використовуватися так звані активні перевірки, що імітують поведінку користувача. Вони передбачають виконання послідовності дій, таких як відкриття сторінки, авторизація або взаємодія з елементами інтерфейсу. Такий підхід дозволяє перевіряти не лише доступність сервісу, але й коректність виконання його бізнес-логіки.

Окремо варто відзначити питання частоти виконання перевірок. З одного боку, часті перевірки дозволяють швидше виявляти проблеми, з іншого - створюють додаткове навантаження на систему. Тому важливо обирати оптимальний інтервал перевірок залежно від критичності сервісу та його характеристик. Для високонавантажених або критично важливих систем інтервали можуть бути значно меншими, ніж для другорядних сервісів.

Ще одним важливим аспектом є обробка результатів перевірок. У реальних умовах можливі ситуації, коли короткочасні мережеві збої або нестабільність інфраструктури призводять до хибних спрацювань. Для зменшення таких випадків використовуються механізми повторних перевірок або аналізу декількох послідовних результатів. Наприклад, інцидент може фіксуватися лише у випадку, якщо проблема повторюється кілька разів поспіль.

Крім цього, методи перевірки доступності можуть використовуватися не лише для контролю зовнішніх вебресурсів, але й для моніторингу внутрішніх сервісів, API та окремих компонентів розподілених систем. Це особливо актуально для сучасних мікросервісних архітектур, де робота одного сервісу часто залежить від доступності інших компонентів. У таких умовах навіть короткочасний збій одного з модулів може вплинути на працездатність усієї системи.

Сучасні системи моніторингу також можуть поєднувати результати перевірок із механізмами сповіщень та автоматичного реагування. Наприклад, у випадку виявлення недоступності сервісу система може автоматично надсилати повідомлення адміністраторам або створювати інцидент для подальшого аналізу. Такий підхід дозволяє значно скоротити час реагування на проблеми та підвищити надійність роботи інформаційних систем.

Отже, перевірка доступності є не лише базовим механізмом контролю стану вебзастосунків, але й важливою складовою комплексного моніторингу. Поєднання регулярних перевірок, аналізу результатів та механізмів автоматичного реагування дозволяє забезпечити стабільну та безперервну роботу сучасних вебсервісів.

2.1.2 Моніторинг продуктивності та часу відгуку

Моніторинг продуктивності дозволяє оцінити якість роботи сервісу, зокрема швидкість обробки запитів. Відповідно до J. Turnbull, час відгуку є одним із ключових показників, що характеризує ефективність системи, а його зростання може свідчити про перевантаження або наявність внутрішніх проблем [8].

Час відгуку є одним із ключових параметрів, що характеризує швидкість роботи сервісу. Його збільшення може свідчити про перевантаження серверів, проблеми з базою даних або неефективну реалізацію бізнес-логіки. Регулярний моніторинг цього показника дозволяє виявляти тенденції до погіршення продуктивності та вживати заходів до виникнення критичних ситуацій.

У контексті надійності систем, у підході Site Reliability Engineering підкреслюється важливість аналізу не лише відмов, але й деградації продуктивності, що може передувати критичним збоям [9].

Окрім цього, важливим є контроль кількості помилок, що виникають під час обробки запитів. Зростання кількості помилок може бути ознакою нестабільності системи або наявності програмних дефектів. Аналіз таких даних дозволяє швидше знаходити причини проблем та усувати їх [3].

Моніторинг продуктивності також включає аналіз навантаження на систему, зокрема використання процесора, пам’яті та мережевих ресурсів. Це особливо важливо для систем, що працюють у хмарному середовищі, де ресурси можуть динамічно змінюватися.

На практиці часто використовуються графіки та статистичні показники, які дозволяють відслідковувати зміни у продуктивності протягом часу. Це дає можливість не лише реагувати на поточні проблеми, але й прогнозувати можливі збої.

Таким чином, моніторинг продуктивності дозволяє забезпечити стабільну та ефективну роботу вебсервісів, своєчасно виявляючи проблеми та запобігаючи їхньому розвитку.

2.1.3 Методи виявлення інцидентів та аварійних ситуацій

Виявлення інцидентів є ключовим етапом у системах моніторингу, оскільки саме на його основі приймається рішення про наявність проблеми в роботі сервісу. Інцидентом вважається будь-яке відхилення від нормального стану системи, яке може впливати на її доступність або продуктивність.

У сучасних підходах до експлуатації систем інцидент визначається як подія, що впливає на доступність або якість сервісу. На практиці інциденти класифікуються залежно від їх впливу на користувачів та критичність системи. Це

дозволяє визначати пріоритетність реагування та оптимізувати процеси обробки збоїв.

Найпростішим методом виявлення інцидентів є аналіз результатів перевірок доступності. Якщо сервіс не відповідає або повертає помилку, система фіксує інцидент. Однак такий підхід може бути недостатньо точним, оскільки не враховує короткочасні збої або випадкові помилки.

Більш ефективним є використання порогових значень, які дозволяють визначати інциденти на основі перевищення певних параметрів, наприклад часу відгуку або кількості помилок. Це дає змогу виявляти не лише повну недоступність, але й деградацію роботи сервісу.

Для підвищення точності широко застосовуються механізми повторних перевірок. Інцидент фіксується лише у випадку, якщо проблема підтверджується декількома послідовними перевірками. Це дозволяє значно зменшити кількість хибних спрацювань.

У більш складних системах використовуються методи кореляції подій, які дозволяють аналізувати взаємозв’язок між різними показниками та визначати причину проблеми. Також може застосовуватись аналіз історичних даних для виявлення аномалій.

Важливим аспектом є також своєчасне інформування про інциденти. Система повинна не лише виявляти проблему, але й оперативно повідомляти користувачів або адміністраторів для швидкого реагування.

2.2 Огляд існуючих систем моніторингу

На сучасному етапі розвитку інформаційних технологій існує велика кількість систем моніторингу, які дозволяють контролювати доступність та продуктивність вебзастосунків. Вони відрізняються за функціональністю, підходами до збору даних, масштабованістю та способом розгортання. Умовно такі

рішення можна поділити на комерційні сервіси та системи з відкритим вихідним кодом.

Сучасні системи моніторингу поєднують функціональність збору метрик, логування та візуалізації даних. Ефективний моніторинг повинен забезпечувати не лише фіксацію проблем, але й можливість їх подальшого аналізу та прогнозування. Це дозволяє переходити від реактивного до проактивного підходу управління системами. Крім того, сучасні рішення активно використовують механізми автоматизації, що дозволяє значно скоротити час реагування на інциденти та мінімізувати вплив людського фактора.

Комерційні системи моніторингу, як правило, надаються у вигляді готових хмарних рішень і не потребують складного налаштування. Вони забезпечують широкий набір функцій, включаючи перевірку доступності, моніторинг продуктивності, систему сповіщень та зручні інтерфейси для візуалізації даних. До їх переваг можна віднести простоту використання, швидке розгортання та технічну підтримку. Водночас такі рішення часто мають обмеження у налаштуванні та потребують регулярних фінансових витрат.

Більшість комерційних платформ підтримують інтеграцію із зовнішніми сервісами, такими як електронна пошта, месенджери або системи управління інцидентами. Це дозволяє оперативно повідомляти користувачів про виникнення проблем та забезпечувати швидке реагування на збої. Також важливою особливістю є підтримка централізованого збору даних та побудови звітів, що дозволяє аналізувати стан системи у довгостроковій перспективі.

Системи з відкритим вихідним кодом надають більше можливостей для гнучкого налаштування та інтеграції з іншими компонентами інфраструктури. Вони дозволяють повністю контролювати процес моніторингу, адаптувати функціонал під конкретні потреби та уникнути залежності від сторонніх сервісів. Проте їх використання зазвичай потребує додаткових зусиль для налаштування, підтримки та масштабування.

Особливо актуальними відкриті системи моніторингу є для великих або

специфічних проєктів, де виникає потреба у власних механізмах перевірки та нестандартних сценаріях обробки даних. У таких випадках можливість змінювати вихідний код і розширювати функціонал системи є суттєвою перевагою. Водночас відповідальність за підтримку, оновлення та забезпечення стабільної роботи системи покладається безпосередньо на розробників або адміністраторів.

Серед популярних рішень можна виділити системи, що спеціалізуються на перевірці доступності, а також більш комплексні платформи, які поєднують моніторинг, логування та аналіз метрик. Деякі з них орієнтовані на простоту використання, тоді як інші - на гнучкість і масштабованість у великих системах.

Крім базового моніторингу доступності, сучасні системи підтримують аналіз продуктивності, контроль навантаження на сервери, збір системних метрик та побудову аналітичних панелей. Це дозволяє не лише виявляти факт недоступності сервісу, але й визначати причини погіршення його роботи. Наприклад, система може зафіксувати підвищення часу відповіді, перевантаження процесора або нестачу пам’яті ще до повного виникнення збою.

Також важливою тенденцією є використання хмарних технологій та контейнеризації. Багато сучасних систем моніторингу підтримують роботу у розподілених середовищах та можуть масштабуватися залежно від навантаження. Це особливо актуально для вебзастосунків, які працюють у мікросервісній архітектурі або використовують хмарну інфраструктуру.

Незважаючи на широкий вибір існуючих інструментів, жодне з рішень не є універсальним. Комерційні сервіси можуть бути обмежені у кастомізації, а відкриті рішення - складними у впровадженні. Це створює передумови для розробки власної системи моніторингу, яка дозволить поєднати необхідний функціонал, гнучкість налаштування та ефективність роботи відповідно до конкретних вимог.

2.2.1 Комерційні рішення

Комерційні системи моніторингу зазвичай надаються у вигляді готових хмарних сервісів, що дозволяє швидко розпочати їх використання без необхідності розгортання власної інфраструктури. Вони забезпечують широкий набір функцій, зокрема перевірку доступності вебзастосунків, моніторинг продуктивності, аналіз часу відгуку, а також систему сповіщень у разі виникнення інцидентів.

До популярних комерційних рішень належать такі сервіси, як Pingdom, UptimeRobot та Datadog. Вони пропонують зручний інтерфейс, можливість налаштування різних типів перевірок (HTTP, TCP, DNS), а також інтеграцію з іншими сервісами для сповіщення, такими як електронна пошта, SMS або месенджери. Деякі з них додатково підтримують функціонал синтетичного моніторингу, що дозволяє імітувати дії користувачів, а також збір детальних метрик і логів для глибшого аналізу роботи системи.

Такі рішення також забезпечують централізоване зберігання даних, побудову графіків та звітів, що дозволяє відслідковувати стан системи в динаміці. Завдяки використанню глобально розподіленої інфраструктури перевірки можуть виконуватись з різних географічних точок, що дає більш об’єктивну оцінку доступності сервісу для користувачів.

Основною перевагою комерційних систем є простота використання, швидке налаштування та відсутність необхідності підтримки власних серверів. Крім того, вони зазвичай мають високий рівень надійності, регулярні оновлення та технічну підтримку, що зменшує витрати часу на обслуговування системи.

Водночас такі рішення мають і певні недоліки. Зокрема, вони можуть мати обмеження у гнучкості налаштування, особливо у випадках, коли необхідно реалізувати специфічну логіку перевірок або інтеграцію з внутрішніми системами. Також важливим фактором є вартість використання, яка зростає разом із кількістю моніторів або обсягом зібраних даних. Крім того, використання сторонніх сервісів передбачає залежність від постачальника та обмежений контроль над обробкою

даних.

Проведений аналіз існуючих систем моніторингу було розширено за рахунок порівняння функціональних можливостей різних сервісів, зокрема Postman, Statuspage, Healthchecks, Uptime, AdminLabs. Основною метою такого аналізу було визначення ключових функцій, які використовуються у сучасних рішеннях, а також виявлення можливостей для вдосконалення розроблюваної системи.

Таблиця 2.1 – Порівняння сервісів моніторингу

Назва сервісу	**Безоплатний тариф**	**HTTP/HTTPS-перевірки**	**Heartbeat-перевірки**	**WebSocket перевірки**	**Сценарні перевірки**
Postman	Передбачено з обмеженнями	Передбачено	Не заявлено	Не заявлено	Передбачено
Statuspage	Передбачено з обмеженнями	Не є основним призначенням сервісу	Не заявлено	Не заявлено	Не заявлено
Healthchecks	Передбачено з обмеженнями	Не є основним призначенням сервісу	Передбачено	Не заявлено	Не заявлено
UptimeRobot	Передбачено з обмеженнями	Передбачено	Обмежено тарифом	Не заявлено	Не заявлено
AdminLabs	Ні, лише пробний період	Передбачено	Не заявлено	Не заявлено	Не заявлено

Як видно з таблиці 2.1, більшість сервісів підтримують базові HTTP-перевірки, однак більш розширений функціонал реалізований лише у частині з них.

У процесі аналізу було визначено два напрямки, які є перспективними для розвитку систем моніторингу: реалізація складних активних перевірок та підтримка перевірок на основі протоколу WebSockets.

Складні активні перевірки відрізняються від базових тим, що складаються не

з одного запиту, а з послідовності дій. Наприклад, у сервісі Postman реалізована можливість створення набору API-запитів, які виконуються послідовно, із додатковою перевіркою отриманих результатів. У свою чергу, Pingdom дозволяє моделювати поведінку користувача, виконуючи послідовність дій на вебсторінці, що дає змогу перевіряти не лише доступність, але й коректність роботи інтерфейсу.

З точки зору практичного використання більш доцільним є підхід, орієнтований на поведінку користувача, оскільки він дозволяє оцінювати роботу системи саме з позиції кінцевого користувача, а не лише на рівні окремих запитів.

З огляду на складність реалізації та потреби системи, доцільним є використання базового підходу перевірки встановлення з'єднання, з можливістю подальшого розширення функціональності у майбутньому.

На основі проведеного аналізу було визначено функціональні можливості, які доцільно реалізувати у розроблюваній системі. Насамперед система повинна підтримувати базові перевірки доступності вебзастосунків, дозволяти налаштовувати інтервали перевірок та забезпечувати сповіщення користувачів у разі виникнення збоїв. Також важливою частиною є відображення результатів перевірок на сторінці статусу, щоб користувач міг швидко оцінити поточний стан сервісів. Окрему увагу слід приділити можливості керування інформацією, яка відображається користувачам, а також підтримці пасивного моніторингу для розширення можливостей аналізу.

Крім основного функціоналу, у подальшому система може бути розширена складними активними перевірками та перевірками на основі WebSockets.

2.2.2 Рішення з відкритим вихідним кодом та власна розробка

Рішення з відкритим вихідним кодом є важливою альтернативою комерційним системам моніторингу, оскільки надають повний контроль над функціональністю та можливість гнучкого налаштування під конкретні потреби.

Серед популярних рішень можна виділити Prometheus, Grafana та Zabbix. Вони забезпечують збір і зберігання метрик, візуалізацію даних, а також можливість налаштування сповіщень. Такі інструменти часто використовуються у поєднанні один з одним, наприклад, для збору даних за допомогою Prometheus та їх відображення у Grafana.

Основною перевагою відкритих рішень є їхня гнучкість та відсутність ліцензійних обмежень. Вони дозволяють реалізувати складні сценарії моніторингу, інтегруватися з внутрішніми сервісами та адаптувати систему під специфічні вимоги. Крім того, вони не потребують регулярних фінансових витрат на підписку, що є важливим фактором для довгострокового використання.

Водночас використання таких систем має і певні недоліки. Зокрема, їх впровадження потребує додаткових зусиль для налаштування, підтримки та масштабування.

У зв’язку з цим доцільним є розгляд власної розробки системи моніторингу, яка дозволяє поєднати необхідний функціонал із максимальною гнучкістю. Власна система може бути спроєктована з урахуванням конкретних вимог, інтегрована з іншими компонентами та оптимізована під конкретні задачі, що робить її ефективним рішенням у порівнянні з готовими інструментами.

2.2.3 Переваги розробки власної системи моніторингу

Розробка власної системи моніторингу дозволяє максимально адаптувати функціональність під конкретні вимоги проєкту та особливості інфраструктури. На відміну від готових рішень, така система не має обмежень у реалізації логіки перевірок, способів обробки даних або інтеграції з іншими сервісами, що є важливим для складних або нестандартних систем.

Однією з ключових переваг є гнучкість. Власна система може бути спроєктована з урахуванням конкретних сценаріїв використання, включаючи специфічні типи перевірок, індивідуальні правила виявлення інцидентів та кастомізовані механізми сповіщень. Це дозволяє більш точно відповідати реальним потребам та уникати зайвого функціоналу.

Також важливим фактором є повний контроль над даними та інфраструктурою. У випадку власної реалізації всі дані залишаються в межах системи, що підвищує рівень безпеки та дозволяє уникнути залежності від сторонніх сервісів. Крім того, це дає можливість оптимізувати систему під необхідне навантаження та забезпечити її масштабування відповідно до потреб.

Ще однією перевагою є відсутність постійних фінансових витрат на використання сторонніх сервісів.

Водночас власна система дозволяє поступово розширювати функціональність, додаючи нові можливості без обмежень, що накладаються готовими платформами. Це робить її більш перспективною для розвитку та інтеграції з іншими компонентами системи.

Таким чином, розробка власної системи моніторингу є доцільним підходом у випадках, коли необхідна висока гнучкість, контроль та можливість адаптації під специфічні вимоги.

У довгостроковій перспективі власна система може стати основою для побудови більш складних рішень, що включають аналітику, прогнозування та автоматичне реагування на інциденти.

3 АРХІТЕКТУРА СИСТЕМИ ТА ОПИС ЗАСОБІВ РОЗРОБКИ

Після проведеного аналізу предметної області та існуючих рішень наступним кроком є перехід до проєктування системи. Саме на цьому етапі формується загальне бачення майбутнього програмного продукту, визначається його структура, основні компоненти та логіка їх взаємодії. Від прийнятих архітектурних рішень залежить, наскільки система буде зручною у підтримці, стійкою до навантаження та придатною до подальшого розширення.

Проєктування дозволяє заздалегідь врахувати вимоги до системи та уникнути проблем на етапі реалізації. Особливо це важливо для систем моніторингу, які повинні обробляти велику кількість перевірок, працювати безперервно та швидко реагувати на зміни стану сервісів. Тому при побудові архітектури доцільно використовувати модульний підхід, який дозволяє розділити систему на незалежні частини та спростити їх розвиток.

Сучасні вебзастосунки функціонують як розподілені системи, що складаються з великої кількості взаємодіючих компонентів. Такі системи характеризуються високою складністю та потребують спеціальних підходів до забезпечення надійності [13].

У межах даного розділу буде сформульовано основні вимоги до системи, визначено її архітектуру та описано ключові модулі. Також буде обґрунтовано вибір технологій, які використовуються для реалізації, з урахуванням їхньої ефективності, зручності розробки та можливостей масштабування.

3.1 Вимоги до системи

Формування вимог до системи є важливим етапом проєктування, оскільки саме вони визначають її функціональність, обмеження та очікувану поведінку. Чітко сформульовані вимоги дозволяють побудувати систему, яка відповідатиме поставленим задачам, буде зручною у використанні та придатною до подальшого розвитку.

Для системи моніторингу доступності вебзастосунків вимоги можна умовно поділити на функціональні та нефункціональні. Функціональні вимоги описують, які саме можливості повинна забезпечувати система, тоді як нефункціональні визначають характеристики її роботи, такі як продуктивність, надійність та масштабованість.

Основною задачею системи є автоматична перевірка стану вебсервісів, виявлення інцидентів та інформування користувачів. При цьому система повинна забезпечувати можливість налаштування параметрів моніторингу, обробки результатів перевірок та відображення актуального стану сервісів.

Крім цього, важливим є забезпечення стабільної роботи системи при великій кількості перевірок, ефективна обробка даних та можливість інтеграції з іншими сервісами. Також система повинна бути зручною для розширення, щоб у майбутньому можна було додавати нові функції без значних змін у її архітектурі.

У наступних підрозділах більш детально розглянуто функціональні та нефункціональні вимоги до розроблюваної системи.

3.1.1 Функціональні вимоги

Функціональні вимоги визначають перелік можливостей, які повинна забезпечувати система моніторингу для виконання своїх основних задач. Вони описують поведінку системи та взаємодію з користувачем і іншими компонентами.

На рисунку 3.1 наведено use case діаграму системи моніторингу доступності вебзастосунків, яка відображає основні функції системи та взаємодію з нею користувача і зовнішніх сервісів.

Система повинна забезпечувати створення, редагування та видалення моніторів доступності вебзастосунків. Користувач повинен мати можливість задавати параметри перевірок, зокрема адресу сервісу, інтервал виконання запитів, тип перевірки та умови успішності.

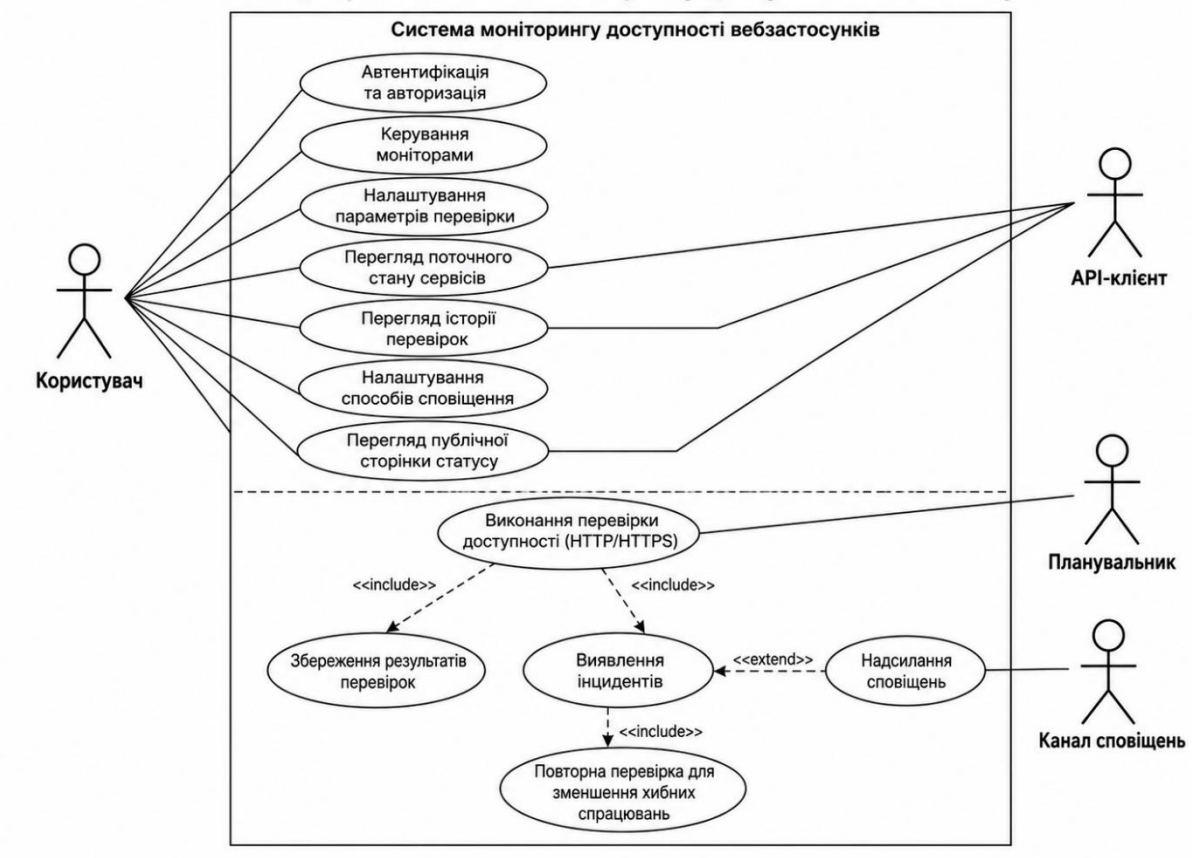


Рисунок 3.1 – Use Case діаграма застосунку

Однією з ключових функцій є виконання перевірок доступності сервісів за допомогою HTTP/HTTPS-запитів. Система повинна зберігати результати перевірок, включаючи статус відповіді та час відгуку, для подальшого аналізу.

Система також повинна визначати інциденти на основі отриманих даних. У разі виявлення проблеми необхідно змінювати стан сервісу та фіксувати відповідні події. При цьому має бути реалізований механізм зменшення хибних спрацювань, наприклад, за рахунок повторних перевірок.

Важливою функцією є надсилання сповіщень користувачам про виникнення інцидентів або відновлення роботи сервісу. Користувач повинен мати можливість налаштовувати способи отримання таких повідомлень.

Система повинна надавати можливість перегляду поточного стану сервісів, а також історії перевірок. Це може бути реалізовано у вигляді API, через який інші компоненти або інтерфейси отримують необхідну інформацію.

Крім цього, система повинна підтримувати механізми автентифікації та авторизації користувачів, забезпечуючи контроль доступу до функціоналу.

Таким чином, функціональні вимоги визначають основні можливості системи та забезпечують виконання її ключових задач у сфері моніторингу доступності вебзастосунків.

3.1.2 Нефункціональні вимоги

Нефункціональні вимоги визначають характеристики якості системи та умови її функціонування. Вони не описують конкретний функціонал, але встановлюють обмеження та критерії, яким повинна відповідати система під час роботи.

Однією з основних вимог є продуктивність. Система повинна обробляти велику кількість перевірок без значних затримок, забезпечуючи своєчасне

отримання результатів та швидке реагування на інциденти. Час обробки запитів і виконання перевірок повинен залишатися стабільним навіть при зростанні навантаження.

Важливою характеристикою є надійність. Система повинна працювати безперервно та забезпечувати коректну обробку даних навіть у випадку помилок або збоїв окремих компонентів. Передбачається використання механізмів обробки помилок та повторних спроб виконання операцій.

Також система повинна бути масштабованою, тобто мати можливість обробляти збільшення кількості моніторів і перевірок без суттєвого зниження ефективності. Це досягається за рахунок використання асинхронної обробки та розподілу навантаження.

Не менш важливою є безпека. Система повинна забезпечувати захист даних користувачів, використовуючи механізми автентифікації та авторизації, а також захист від несанкціонованого доступу.

Крім того, система повинна бути зручною у використанні та підтримці. Архітектура має бути зрозумілою та модульною, що дозволить легко вносити зміни, додавати новий функціонал та забезпечувати підтримку системи у майбутньому.

Також варто врахувати переносимість та розгортання. Система повинна бути придатною для розгортання у хмарному середовищі, зокрема з використанням сучасних платформ, що забезпечують гнучкість і доступність.

Таким чином, нефункціональні вимоги визначають якість роботи системи та забезпечують її ефективність, стабільність і можливість подальшого розвитку.

3.2 Архітектура системи моніторингу

Під час проєктування архітектури важливо врахувати вимоги до продуктивності, надійності та масштабованості. Система повинна бути здатною

працювати з великою кількістю моніторів, обробляти результати перевірок без перевантаження та забезпечувати стабільну роботу навіть при зростанні навантаження. Саме тому доцільно використовувати модульний підхід, за якого окремі частини системи відповідають за конкретні задачі.

Архітектура системи моніторингу має бути побудована таким чином, щоб процес створення моніторів, виконання перевірок, збереження результатів, виявлення інцидентів та формування статистики не змішувалися між собою.

Важливим архітектурним рішенням є використання фонових процесів для виконання періодичних задач. Перевірки доступності не повинні залежати від активності користувача або виконуватися безпосередньо під час обробки HTTP-запиту. Тому окремі сервіси відповідають за планування перевірок, їх виконання та подальшу обробку результатів. Це дозволяє системі працювати безперервно та виконувати моніторинг у автоматичному режимі.

Окрему роль відіграє асинхронна обробка даних. Оскільки кількість моніторів може збільшуватись, виконання перевірок у послідовному режимі може призвести до затримок. Використання черг повідомлень дає можливість відокремити формування задач від їх виконання, а також більш рівномірно розподіляти навантаження між компонентами системи.

Також під час проєктування враховано необхідність збереження історії перевірок. Це важливо не лише для відображення поточного стану сервісу, але й для подальшого аналізу його роботи. На основі накопичених даних система може обчислювати статистичні показники, такі як відсоток доступності, середній час відповіді та кількість інцидентів за певний період.

Отже, проєктування архітектури системи спрямоване на створення структури, яка забезпечить ефективну, надійну та гнучку роботу системи моніторингу, а також дозволить у майбутньому розширювати її функціональність без суттєвих змін у базових компонентах. Проєктування архітектури програмних систем базується на принципах розділення відповідальності та модульності. Розподіл системи на окремі компоненти дозволяє підвищити її гнучкість і

спростити підтримку [6]. Використання шаблонів проєктування також сприяє повторному використанню перевірених рішень і підвищенню якості програмного забезпечення [7].

3.2.1 Загальна архітектура

Розроблена система моніторингу реалізована у вигляді серверного застосунку з монолітною архітектурою та логічним поділом на окремі проєкти відповідно до їхнього призначення. Такий підхід дозволяє поєднати простоту розгортання з достатнім рівнем структурованості коду, що є доцільним для системи даного типу. Архітектура побудована таким чином, щоб розділити бізнес-логіку, доступ до даних, API-рівень та фонову обробку задач.

Альтернативою монолітній архітектурі є мікросервісний підхід, який передбачає розподіл системи на незалежні сервіси. Мікросервіси дозволяють підвищити масштабованість і спростити розгортання окремих компонентів [4]. Водночас такий підхід ускладнює управління взаємодією між сервісами та вимагає додаткових механізмів координації [5].

Вибір монолітної архітектури зумовлений характером системи, яка, незважаючи на наявність декількох логічних компонентів, не потребує складної розподіленої взаємодії між сервісами. Такий підхід дозволяє спростити розгортання, зменшити кількість точок відмови та забезпечити зручність підтримки. Водночас логічне розділення на шари дозволяє зберегти переваги модульності та забезпечити можливість подальшого переходу до більш розподіленої архітектури у разі необхідності.

Для наочного представлення структури серверної частини системи та її взаємодії із зовнішніми сервісами побудовано архітектурну схему, наведену на рисунку 3.2. На схемі відображено основні логічні компоненти серверного застосунку, а також засоби збереження даних і асинхронної обробки задач.

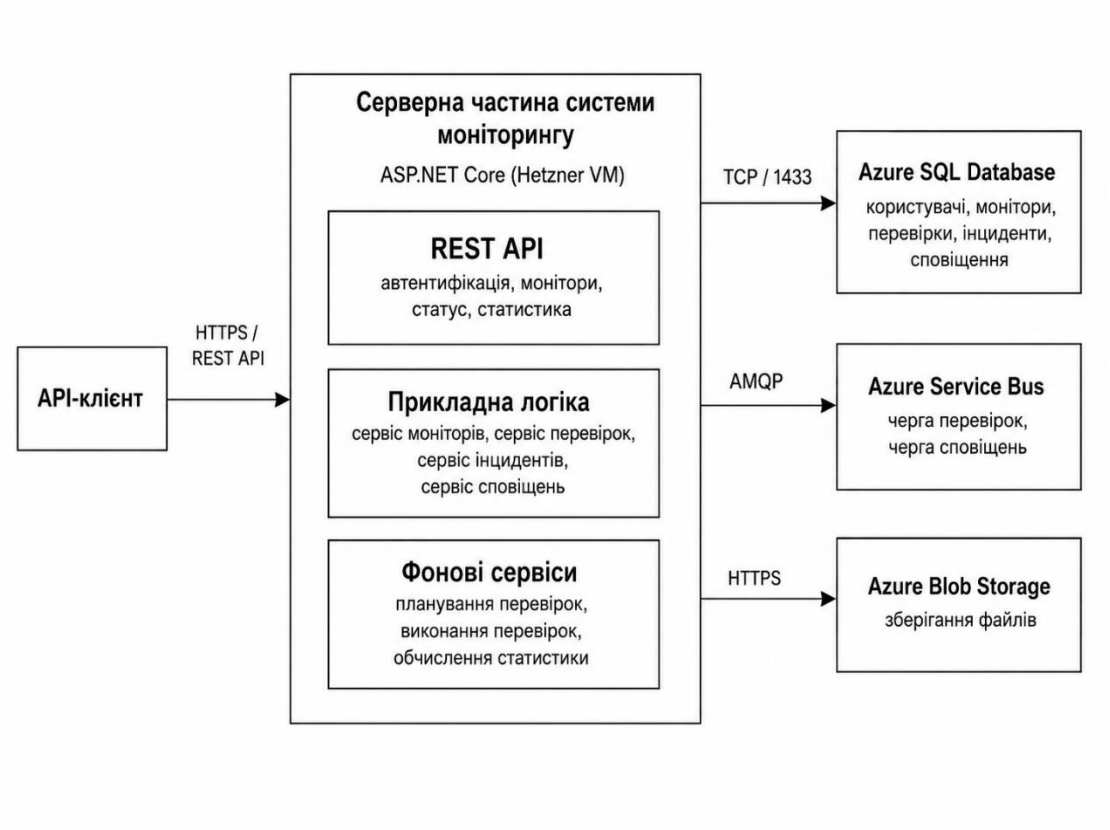


Рисунок 3.2 – Загальна архітектура системи

До складу рішення входять окремі проєкти, кожен з яких виконує визначену роль. Проєкт API виступає точкою входу до системи та відповідає за обробку HTTP-запитів, конфігурацію застосунку, маршрутизацію, автентифікацію, а також надання доступу до функціоналу через REST API. Саме через цей рівень клієнтські застосунки взаємодіють із системою.

Архітектурний стиль REST базується на принципах клієнт-серверної взаємодії, безстанності та використання уніфікованого інтерфейсу. REST дозволяє створювати масштабовані вебсервіси, які легко інтегруються з іншими системами.

Проєкт Application містить прикладну логіку системи. У ньому реалізуються сценарії роботи із моніторами, інцидентами, статистикою та іншими сутностями. Цей рівень виступає проміжною ланкою між API та доменною моделлю, координуючи виконання бізнес-операцій. Таким чином, прикладна логіка не

залежить напряму від деталей збереження даних чи зовнішніх технологій, що підвищує гнучкість системи.

Проєкт Domain містить основні доменні сутності, правила предметної області та базову бізнес-логіку. Саме на цьому рівні описуються ключові об’єкти системи, їхні властивості та взаємозв’язки. Виділення доменного шару дозволяє зосередити основні правила роботи системи в окремому місці та зменшити зв’язність між компонентами, що є важливим для підтримки та розвитку системи.

Проєкт Infrastructure відповідає за роботу із зовнішніми залежностями. До його задач належать доступ до бази даних, збереження інформації, налаштування міграцій, а також реалізація технічних механізмів, необхідних для функціонування системи. Саме на цьому рівні прикладна логіка отримує реалізації потрібних інтерфейсів для роботи з даними, що дозволяє ізолювати бізнес-логіку від конкретних технологій.

Окрему роль у системі відіграють фонові сервіси, які забезпечують виконання задач моніторингу без участі користувача. Зокрема, MonitorSchedulerBackgroundService відповідає за планування перевірок, WebsiteCheckerBackgroundService - за безпосереднє виконання перевірок вебресурсів, а CalculateStatisticsBackgroundService - за обчислення статистичних показників. Використання background services дозволяє винести довготривалі та періодичні процеси за межі основного потоку обробки HTTP-запитів, що підвищує продуктивність системи та забезпечує її стабільну роботу.

Взаємодія між компонентами системи організована таким чином, щоб мінімізувати прямі залежності між модулями. Для цього використовуються інтерфейси та принцип інверсії залежностей, що дозволяє легко змінювати реалізацію окремих компонентів без впливу на інші частини системи. Такий підхід сприяє підвищенню гнучкості архітектури та спрощує тестування.

Особливу увагу приділено асинхронній обробці задач. Система моніторингу повинна обробляти значну кількість перевірок, тому використання асинхронних механізмів дозволяє ефективно розподіляти навантаження та уникати блокування

потоків. Це забезпечує можливість масштабування системи та її стабільну роботу навіть при зростанні кількості моніторів.

Крім того, архітектура системи враховує можливість інтеграції з хмарною інфраструктурою. Використання зовнішніх сервісів, таких як база даних, черги повідомлень та сховище файлів, дозволяє розподілити навантаження між компонентами та підвищити відмовостійкість системи. Це особливо важливо для систем моніторингу, які повинні працювати безперервно та забезпечувати високу доступність.

Застосування принципів чистої архітектури дозволяє відокремити бізнес-логіку від зовнішніх залежностей. Така структура підвищує тестованість і підтримуваність системи [14].

Таким чином, загальна архітектура системи базується на поєднанні монолітного підходу та шарової організації проєкту. Це дозволяє забезпечити цілісність системи, спростити розгортання та підтримку, а також зберегти достатню гнучкість для подальшого розвитку функціоналу.

Для наочного представлення взаємодії між основними компонентами системи було побудовано загальну архітектурну схему, наведену на рисунку 3.2. На схемі відображено клієнтську частину застосунку, серверну частину, а також зовнішні сервіси, які використовуються для забезпечення роботи системи моніторингу.

Крім безпосередньої взаємодії між компонентами, важливим аспектом під час проєктування архітектури є забезпечення стабільної роботи системи при збільшенні кількості моніторів та перевірок. Оскільки система повинна працювати у безперервному режимі та виконувати перевірки через задані проміжки часу, необхідно було передбачити механізми, які дозволяють ефективно розподіляти навантаження між окремими компонентами. Саме тому значна частина операцій виконується у фоновому режимі без прямої участі користувача.

Окрему увагу приділено мінімізації залежностей між модулями системи. У випадку тісного зв’язку між компонентами будь-які зміни в одному модулі можуть

впливати на роботу інших частин застосунку, що ускладнює підтримку та розвиток системи. Для уникнення таких проблем у системі використовується підхід із розділенням відповідальності між окремими шарами. Завдяки цьому бізнес-логіка, робота з даними та механізми взаємодії із зовнішніми сервісами ізольовані один від одного.

Важливим фактором також є підтримка асинхронної обробки задач. У системах моніторингу одночасно може виконуватися велика кількість перевірок, а частина з них може тривати довше через мережеві затримки або нестабільність зовнішніх сервісів. Якщо виконувати всі операції послідовно в одному потоці, це може призвести до погіршення продуктивності та збільшення часу обробки. Використання асинхронних механізмів дозволяє уникнути блокування потоків виконання та ефективніше використовувати ресурси сервера.

Крім того, архітектура системи повинна враховувати можливість подальшого розширення функціоналу. У майбутньому система може бути доповнена новими типами перевірок, додатковими механізмами сповіщень або інтеграціями із зовнішніми сервісами. Саме тому при проєктуванні було обрано підхід, який дозволяє додавати нові компоненти без необхідності суттєвої зміни вже реалізованої логіки.

Не менш важливим аспектом є забезпечення відмовостійкості системи. Оскільки моніторинг передбачає постійний контроль доступності сервісів, сама система моніторингу також повинна працювати стабільно та безперервно. Для цього використовуються окремі зовнішні сервіси для зберігання даних, черг повідомлень та файлів. Такий підхід дозволяє розподілити навантаження між компонентами та зменшити ризик втрати даних або зупинки системи у випадку локальних збоїв.

Під час проєктування також враховувалась необхідність зручної взаємодії користувача із системою. Тому архітектура системи побудована таким чином, щоб забезпечити чітке розділення клієнтської та серверної частини, а також стандартизовану взаємодію через REST API.

3.2.2 Сценарій роботи системи

Робота системи моніторингу побудована як послідовність взаємопов’язаних етапів, що забезпечують повний цикл: від налаштування монітора до обробки результатів перевірки та інформування користувача. Основна логіка виконується у фоновому режимі, що дозволяє системі працювати безперервно та незалежно від активності користувачів.

На початковому етапі користувач через API створює монітор, задаючи параметри перевірки: адресу вебсервісу, інтервал виконання запитів та умови визначення успішності. Ці дані зберігаються у системі та використовуються для подальшого планування перевірок. Діаграма активності системи наведена в Додатку Б.

Далі у роботу вступає фоновий сервіс MonitorSchedulerBackgroundService, який відповідає за формування задач перевірки відповідно до заданих інтервалів. Він періодично аналізує конфігурацію моніторів та ініціює виконання перевірок, передаючи їх на обробку.

Безпосереднє виконання перевірок здійснює WebsiteCheckerBackgroundService. Він надсилає HTTP-запити до відповідних вебсервісів, фіксує отримані результати, зокрема статус відповіді та час відгуку, і передає ці дані для подальшої обробки.

Протокол HTTP функціонує за моделлю «запит-відповідь» і є основою взаємодії клієнта та сервера у вебсередовищі. При цьому надійність передачі даних забезпечується протоколом TCP, який гарантує доставку пакетів у правильному порядку [1].

Після отримання результатів система визначає поточний стан сервісу. Якщо виявлено відхилення від нормального стану, фіксується інцидент. Для зменшення кількості хибних спрацювань можуть використовуватися додаткові перевірки або аналіз декількох послідовних результатів.

Паралельно виконується обробка накопичених даних. Сервіс

CalculateStatisticsBackgroundService обчислює статистичні показники, такі як середній час відгуку або відсоток доступності, що дозволяє оцінити якість роботи сервісу у динаміці.

Отримані результати доступні користувачеві через API. Це дозволяє переглядати стан сервісів, історію перевірок та статистику. У випадку виникнення інцидентів система також може надсилати сповіщення відповідно до налаштованих параметрів.

Таким чином, сценарій роботи системи базується на поєднанні взаємодії користувача з API та автоматичної обробки даних у фоновому режимі. Це забезпечує безперервний моніторинг вебзастосунків та своєчасне виявлення проблем у їх роботі.

Важливо зазначити, що сценарій роботи системи має циклічний характер. Після створення монітора система не завершує роботу з ним, а постійно повертається до його перевірки відповідно до заданого інтервалу. Такий підхід дозволяє забезпечити безперервний контроль доступності вебсервісу без необхідності ручного запуску перевірок користувачем.

У межах цього процесу кожен компонент виконує окрему роль. API забезпечує взаємодію користувача із системою та приймає налаштування монітора. Фоновий планувальник відповідає за визначення моменту виконання перевірки, сервіс перевірки виконує запит до ресурсу, а модулі обробки результатів зберігають дані, визначають стан сервісу та, за потреби, створюють інцидент. Саме такий розподіл відповідальності дозволяє зробити роботу системи більш зрозумілою, стабільною та придатною до подальшого розширення.

3.2.3 Проектування бази даних

Для забезпечення збереження даних про користувачів, налаштовані монітори, результати перевірок, виявлені інциденти та сформовані сповіщення у

системі використовується реляційна база даних. Її структура побудована таким чином, щоб забезпечити збереження історії моніторингу, можливість аналізу доступності вебзастосунків та підтримку взаємозв'язків між основними об'єктами предметної області.

На рисунку 3.3 наведено ER-діаграму бази даних системи моніторингу доступності вебзастосунків, яка відображає основні сутності системи та зв'язки між ними.

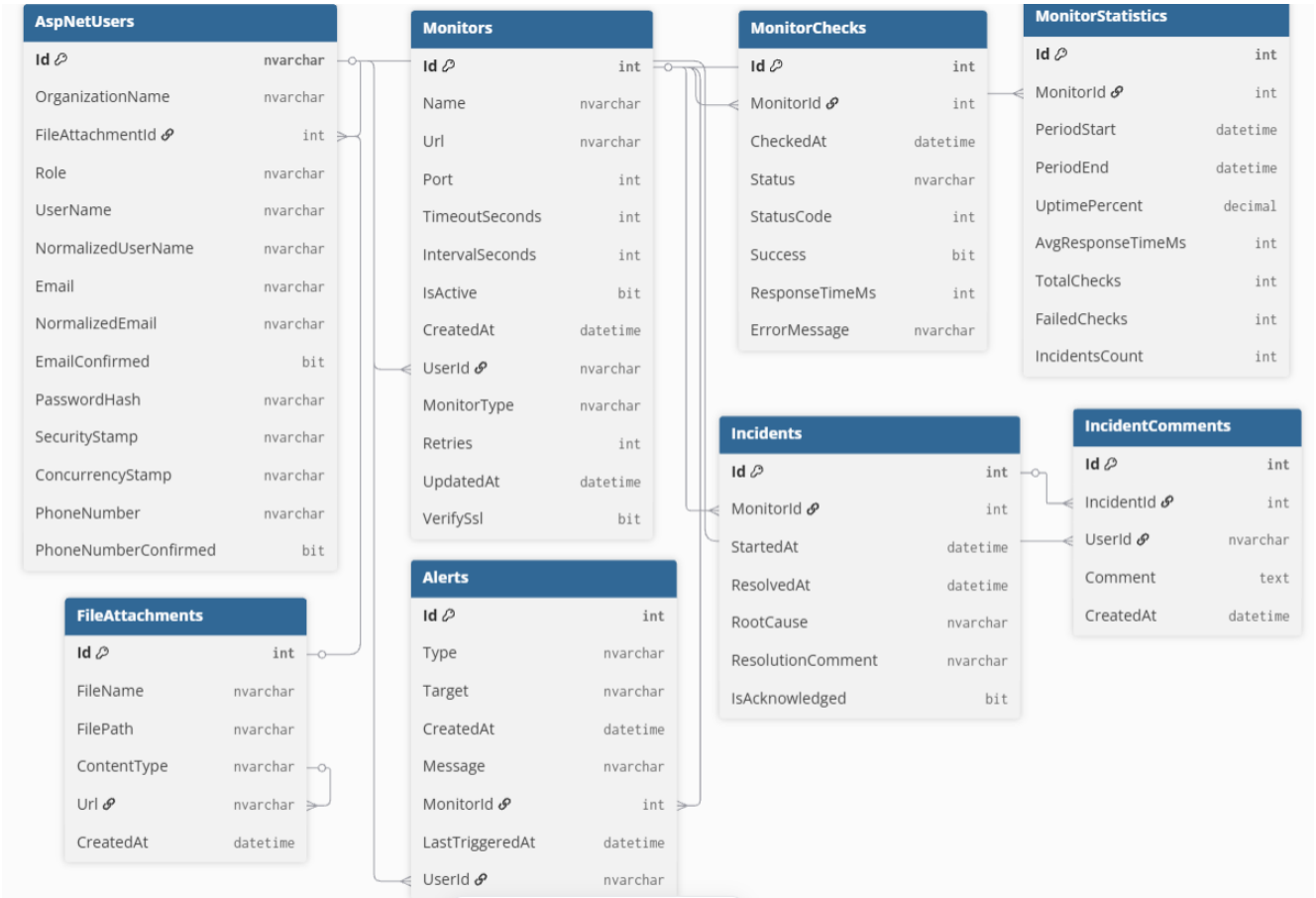


Рисунок 3.3 – ER-діаграма бази даних системи моніторингу доступності вебзастосунків

Центральною сутністю бази даних є таблиця Monitors, у якій зберігаються налаштування створених користувачем моніторів. Вона містить назву монітора, адресу контрольованого ресурсу, порт, допустимий час очікування відповіді,

інтервал виконання перевірок, ознаку активності, тип монітора, кількість повторних спроб та дату створення. Кожен монітор пов’язаний із користувачем, який його створив.

Результати виконаних перевірок зберігаються у таблиці MonitorChecks. Для кожної перевірки фіксуються дата та час виконання, отриманий статус, HTTP-код відповіді, ознака успішності, час відгуку та повідомлення про помилку, якщо перевірка завершилася невдало. Наявність цієї таблиці дозволяє зберігати історію роботи кожного монітора та використовувати накопичені дані для подальшого аналізу.

Для зберігання інформації про виявлені проблеми використовується таблиця Incidents. У ній фіксуються момент початку інциденту, час його завершення, основна причина виникнення, коментар щодо усунення проблеми та ознака підтвердження інциденту користувачем. Зв’язок із таблицею Monitors дозволяє визначити, для якого саме контрольованого сервісу було зафіксовано збій.

Окреме місце у структурі бази даних займає таблиця Alerts, яка містить інформацію про сформовані сповіщення. У ній зберігаються тип повідомлення, адресат або канал надсилання, текст сповіщення, дата створення, відповідний монітор та користувач. Така структура дозволяє відстежувати повідомлення, сформовані у випадку виникнення інцидентів або зміни стану контрольованого сервісу.

Для накопичення узагальнених показників роботи моніторів використовується таблиця MonitorStatistics. Вона містить період формування статистики, відсоток доступності, середній час відповіді, загальну кількість перевірок, кількість невдалих перевірок і кількість інцидентів. Використання окремої таблиці для статистичних даних дає змогу ефективно формувати звіти та відображати показники доступності без необхідності щоразу опрацьовувати всю історію перевірок.

Таким чином, спроєктована структура бази даних охоплює основні процеси системи: керування користувачами та моніторами, виконання і збереження

результатів перевірок, фіксацію інцидентів, формування сповіщень та розрахунок статистичних показників. Це забезпечує цілісне збереження інформації, необхідної для аналізу доступності вебзастосунків і контролю їх поточного стану.

3.3 Вибір засобів розробки

Вибір технологій для реалізації системи є важливим етапом, оскільки від нього залежить продуктивність, зручність розробки та можливість подальшого розвитку програмного продукту. При підборі засобів враховувалися вимоги до системи, необхідність обробки великої кількості запитів, підтримка асинхронної логіки та можливість інтеграції з хмарним середовищем.

Окрім технічних характеристик, важливу роль відігравав також практичний досвід використання обраних технологій. Наявність досвіду роботи з відповідним стеком дозволив більш обґрунтовано підійти до вибору інструментів, врахувати їх сильні та слабкі сторони, а також оптимально застосувати їх у процесі розробки. Це сприяло підвищенню ефективності реалізації системи та зменшенню ризиків, пов’язаних із впровадженням нових або недостатньо знайомих технологій.

Таким чином, вибір технологій базувався не лише на теоретичних перевагах, але й на практичному досвіді їх застосування, що забезпечило баланс між надійністю, продуктивністю та зручністю подальшої підтримки системи.

3.3.1 Мова програмування C#

Для реалізації серверної частини системи моніторингу було обрано мову програмування C#. Вибір цієї мови зумовлений її широким використанням у розробці вебзастосунків, високою продуктивністю та підтримкою сучасних підходів до побудови програмного забезпечення.

C# забезпечує зручні засоби для реалізації асинхронної обробки, що є важливим для системи моніторингу, яка виконує велику кількість перевірок у фоновому режимі. Використання механізмів async/await дозволяє ефективно працювати з мережевими запитами та уникати блокування потоків виконання.

Також важливою перевагою є тісна інтеграція мови з платформою .NET, яка надає широкий набір бібліотек та інструментів для розробки серверних застосунків. Це спрощує реалізацію таких задач, як обробка HTTP-запитів, робота з базами даних, конфігурація застосунку та організація залежностей.

Мова C# підтримує об’єктно-орієнтований підхід, що дозволяє будувати систему з чітким розподілом відповідальності між компонентами. Крім того, вона забезпечує високий рівень типізації, що зменшує кількість помилок під час розробки та спрощує підтримку коду.

Ще одним важливим фактором є активна підтримка та розвиток екосистеми .NET, що дозволяє використовувати сучасні технології, інструменти тестування та засоби розгортання.

Таким чином, вибір мови програмування C# є обґрунтованим з точки зору продуктивності, зручності розробки та відповідності вимогам системи моніторингу вебзастосунків.

3.3.2 Фреймворки та бібліотеки

Для реалізації серверної частини системи використано сучасні фреймворки та бібліотеки, які забезпечують зручність розробки, масштабованість та підтримку основних функціональних можливостей.

Основним фреймворком обрано ASP.NET Core, який використовується для побудови вебзастосунку та реалізації API. Він забезпечує високу продуктивність, підтримку асинхронної обробки запитів, вбудований механізм dependency injection та зручну конфігурацію застосунку. Це дозволяє ефективно організувати серверну

логіку та взаємодію з клієнтами.

Платформа ASP.NET Core забезпечує створення високопродуктивних вебзастосунків із підтримкою сучасних архітектурних підходів. Фреймворк дозволяє ефективно реалізовувати вебсервіси з використанням MVC-підходу [15], а також є гнучким і підтримує кросплатформеність [16].

Для реалізації доступу до даних використовується ORM-бібліотека Entity Framework Core. Вона дозволяє працювати з базою даних на рівні об’єктів, спрощує виконання запитів та керування структурою бази даних за допомогою міграцій. Це значно зменшує обсяг ручної роботи та підвищує зручність підтримки коду.

У системі також використовуються механізми фонових сервісів, реалізовані засобами .NET, що дозволяє виконувати періодичні задачі, такі як перевірка доступності сервісів або обчислення статистики, незалежно від обробки користувацьких запитів.

Додатково застосовуються допоміжні бібліотеки для мапінгу об’єктів, обробки помилок, конфігурації та організації структури коду. Це дозволяє зменшити дублювання логіки та зробити код більш зрозумілим і підтримуваним.

Таким чином, обраний набір фреймворків та бібліотек забезпечує ефективну реалізацію функціоналу системи, спрощує розробку та створює основу для її подальшого розвитку.

3.3.3 Хмарна платформа Microsoft Azure

Для розгортання системи обрано хмарну платформу Microsoft Azure, яка забезпечує необхідні можливості для стабільної роботи, масштабування та управління ресурсами. Використання хмарного середовища дозволяє розмістити застосунок без прив’язки до локальної інфраструктури та забезпечити його доступність з будь-якої точки мережі. Використання хмарних платформ дозволяє забезпечити масштабованість і високу доступність системи. Хмарні обчислення

забезпечують гнучке управління ресурсами та підвищують відмовостійкість систем за рахунок розподіленої інфраструктури [12].

Однією з ключових переваг обраної платформи є підтримка контейнеризації. Система розгортається за допомогою Docker, що дозволяє створити ізольоване середовище для виконання застосунку та спростити процес перенесення між різними середовищами. Використання docker-compose дає можливість керувати конфігурацією сервісів та швидко розгортати систему.

Окрему увагу приділено процесу автоматизації розгортання системи. Для цього використовується підхід CI/CD, реалізований за допомогою GitHub Actions. Такий підхід дозволяє автоматизувати процес збірки, тестування та доставки застосунку до середовища виконання.

Процес розгортання починається з внесення змін до репозиторію. Після цього автоматично запускається workflow у GitHub Actions, який виконує послідовність дій: збірку застосунку, виконання базових перевірок, створення Docker-образу та його публікацію в Docker Hub. Таким чином формується актуальна версія застосунку у вигляді контейнеру.

Після публікації образу відбувається оновлення застосунку на віртуальній машині, розміщеній на платформі Hetzner. Сервер виконує отримання (pull) нової версії Docker-образу та перезапуск контейнерів за допомогою docker-compose. Це дозволяє швидко оновлювати систему без необхідності ручного втручання та мінімізувати час простою.

Для управління контейнеризованими застосунками можуть використовуватися системи оркестрації, такі як Kubernetes. Такі системи дозволяють автоматизувати масштабування та управління навантаженням [11].

Використання CI/CD підходу забезпечує стабільність процесу розгортання, зменшує ймовірність помилок, пов’язаних із ручними діями, а також дозволяє швидко доставляти зміни у робоче середовище. Крім того, така схема забезпечує повторюваність процесу розгортання та спрощує підтримку системи.

Практики безперервної інтеграції та доставки (CI/CD) дозволяють

автоматизувати процес розгортання та підвищити стабільність системи. Автоматизація зменшує кількість помилок і прискорює випуск нових версій програмного забезпечення [10].

Хмарна платформа також забезпечує гнучке масштабування ресурсів залежно від навантаження. Це особливо важливо для системи моніторингу, яка може обробляти різну кількість перевірок у різні моменти часу. У разі збільшення навантаження система може бути масштабована без суттєвих змін у її архітектурі.

Крім того, використання хмарного середовища дозволяє забезпечити високу доступність та надійність системи, оскільки інфраструктура платформи підтримує резервування ресурсів та автоматичне відновлення у випадку збоїв.

Таким чином, обрана платформа для розгортання забезпечує зручність управління, гнучкість, масштабованість та стабільність роботи системи, що відповідає вимогам до сучасних вебзастосунків.

На рисунку 3.4 наведено загальну схему процесу розгортання застосунку із використанням підходу CI/CD. Діаграма демонструє послідовність етапів доставки нової версії системи: від внесення змін у репозиторій до автоматичного оновлення контейнерів на сервері. З рисунка видно, що GitHub Actions використовується для автоматичного запуску процесу збірки та створення Docker-образу, який після цього публікується у Docker Hub. Далі віртуальна машина на платформі Hetzner отримує актуальну версію образу та виконує перезапуск контейнерів за допомогою docker-compose.

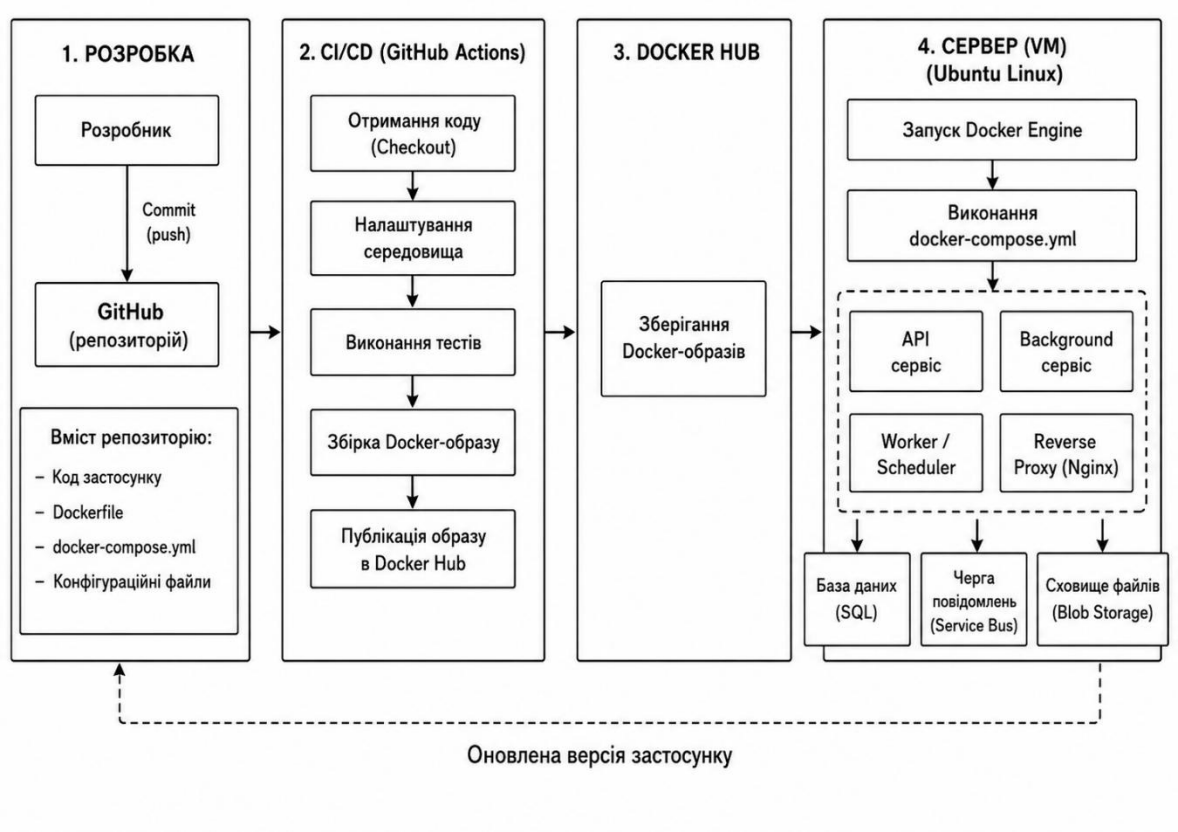


Рисунок 3.4 – Життєвий цикл розробки

Представлена схема дозволяє краще зрозуміти взаємодію між компонентами процесу розгортання та демонструє автоматизацію основних етапів доставки застосунку. Використання такого підходу значно спрощує оновлення системи, зменшує кількість ручних дій та підвищує стабільність роботи середовища. Крім того, автоматизоване розгортання дозволяє швидше доставляти зміни у робоче середовище та забезпечує повторюваність процесу оновлення застосунку.

4 ПРОГРАМНА РЕАЛІЗАЦІЯ МОДУЛІВ СИСТЕМИ МОНІТОРИНГУ

Після проєктування архітектури системи наступним кроком стала її практична реалізація. На цьому етапі розроблено основні програмні модулі, які забезпечують роботу системи та взаємодію між її компонентами. Реалізація виконана з урахуванням раніше визначеної структури проєкту та поділу на окремі рівні відповідальності.

Система побудована таким чином, що кожен модуль виконує конкретну функцію та взаємодіє з іншими через визначені інтерфейси. Це дозволяє уникнути жорсткої залежності між компонентами та спрощує подальше розширення функціоналу. Основна логіка зосереджена у прикладному шарі, тоді як доступ до даних та технічні деталі винесені в окремі частини системи.

У межах даного розділу розглядається реалізація ключових модулів, зокрема механізму перевірки доступності вебсервісів, обробки результатів перевірок та визначення інцидентів. Окремо описано роботу фонових сервісів, які виконують періодичні задачі без участі користувача, а також компоненти, що відповідають за взаємодію із системою через API.

Також буде показано, як реалізовано зберігання даних, обчислення статистики та організацію внутрішніх сервісів. Це дозволяє краще зрозуміти, яким чином система функціонує на практиці та як окремі частини поєднуються в єдине рішення.

4.1 Модуль автентифікації та авторизації користувачів

Модуль автентифікації та авторизації відповідає за перевірку користувачів, управління їх доступом до системи та забезпечення безпечної взаємодії з її

функціоналом. До його основних задач належать реєстрація користувачів, вхід у систему, зміна пароля, а також видача і оновлення токенів доступу.

У системі використовується підхід на основі JWT-токенів, що дозволяє реалізувати stateless-автентифікацію. Це означає, що сервер не зберігає інформацію про сесію користувача, а кожен запит перевіряється на основі токена, який передається разом із ним. Такий підхід спрощує масштабування системи та зменшує навантаження на сервер.

Після успішного входу користувачу видається access token, який містить набір даних, зокрема ідентифікаційну інформацію та роль користувача. Цей токен має обмежений термін дії, що підвищує безпеку у випадку його компрометації. Для підтримки безперервної роботи також використовується refresh token, який дозволяє отримати новий access token без повторної автентифікації.

Процес оновлення токена передбачає перевірку дійсності refresh token та відновлення інформації про користувача навіть у випадку, якщо access token вже прострочений. Це дозволяє забезпечити баланс між безпекою та зручністю використання системи.

Взаємодія з модулем здійснюється через API, яке надає можливість виконувати основні операції: реєстрацію нового користувача, вхід у систему, зміну пароля та оновлення токена. Усі запити проходять перевірку коректності даних, а також контроль доступу до захищених ресурсів.

Механізм авторизації базується на використанні ролей та інших атрибутів, що містяться у токені. Це дозволяє обмежувати доступ до окремих функцій системи залежно від прав користувача.

У таблиці 4.1 описано методи, що використовуються для реєстрації та аутентифікації користувача у системі.

Таблиця 4.1 – Кінцеві точки автентифікації та авторизації

Назва інтерфейсу	**Назва методу**	**HTTP метод**	**Опис запиту**	**Опис відповіді**	**Опис**
/auth	/token/refresh	POST	Токен оновлення	Нові токени	Оновлення токена доступу
	/signup	POST	Дані нового користувача	Результат реєстрації	Реєстрація користувача
	/login	POST	Логін і пароль	Токени доступу	Вхід до системи
	/change-password	POST	Поточний і новий пароль	Результат зміни пароля	Зміна пароля

Інтерфейс /auth забезпечує виконання основних операцій, пов’язаних з автентифікацією та керуванням обліковими даними користувача. Метод /signup використовується для створення нового облікового запису, а метод /login — для входу користувача до системи та отримання токенів доступу. Для продовження роботи користувача без повторної автентифікації застосовується метод /token/refresh, який формує новий токен доступу на основі токена оновлення. Метод /change-password надає можливість змінити пароль авторизованого користувача із зазначенням поточного та нового значення.

4.2 Модуль конфігурації моніторингу

Модуль конфігурації моніторингу відповідає за створення, налаштування та управління моніторами вебзастосунків. Саме через нього користувач визначає, які ресурси необхідно перевіряти, з якою частотою та за якими критеріями оцінювати їх стан.

Взаємодія з модулем здійснюється через API, яке надає можливість виконувати основні операції: створення нового монітора, отримання списку

моніторів, перегляд деталей конкретного монітора, оновлення його параметрів та видалення. Крім цього, передбачено можливість тимчасового призупинення монітора та його повторного запуску, що дозволяє гнучко керувати процесом моніторингу.

Під час створення монітора користувач задає основні параметри, такі як назва, адреса вебресурсу та інші налаштування. Перед збереженням виконується перевірка доступності вказаного ресурсу, що дозволяє уникнути додавання некоректних або недоступних адрес. Після цього монітор зберігається у системі та стає доступним для подальшого використання.

Система підтримує оновлення конфігурації монітора. Користувач може змінювати параметри перевірки, після чого ці зміни застосовуються до подальших перевірок. Усі операції виконуються з урахуванням належності монітора конкретному користувачу, що забезпечує ізоляцію даних.

Для оптимізації роботи реалізовано механізм кешування списку моніторів. Це дозволяє зменшити кількість звернень до бази даних при отриманні часто використовуваної інформації. У разі змін у конфігурації кеш оновлюється або очищується, щоб забезпечити актуальність даних.

Окрім базових операцій, модуль підтримує додатковий функціонал, зокрема імпорт та експорт моніторів у форматі CSV. Це дозволяє швидко додавати велику кількість записів або переносити конфігурацію між різними середовищами.

У таблиці 4.2 описано методи, що використовуються для створення, налаштування та перегляду результатів роботи моніторів доступності вебзастосунків.

Таблиця 4.2 – Кінцеві точки керування моніторами

Назва інтерфейсу	**Назва методу**	**HTTP метод**	**Опис запиту**	**Опис відповіді**	**Опис**
/monitors	/	POST	Дані монітора	Створений монітор	Створення монітора
	/	GET	-	Список моніторів	Перегляд моніторів
	/	DELETE	Ідентифікатор монітора	Результат видалення	Видалення монітора
	/{id}	GET	Ідентифікатор монітора	Дані монітора	Перегляд монітора
	/{id}	PUT	Ідентифікатор і нові дані	Оновлений монітор	Редагування монітора
	/pause	POST	Ідентифікатор монітора	Результат призупинення	Призупинення монітора
	/{id}/resume	POST	Ідентифікатор монітора	Результат відновлення	Відновлення монітора
	/{id}/export	GET	Ідентифікатор монітора	Файл експорту	Експорт монітора
	/{id}/history	GET	Ідентифікатор монітора	Історія перевірок	Перегляд історії
	/{id}/summary	GET	Ідентифікатор монітора	Загальні показники	Перегляд зведення
	/{id}/downtimes	GET	Ідентифікатор монітора	Періоди недоступності	Перегляд простоїв

Інтерфейс /api/monitors забезпечує виконання основних операцій із моніторами системи. За його допомогою користувач може створювати, переглядати, редагувати та видаляти монітори, а також тимчасово призупиняти або відновлювати виконання перевірок. Окремі методи інтерфейсу призначені для імпорту та експорту даних, перегляду історії перевірок, поточного статусу, періодів

недоступності, відсотка доступності та часу відгуку контрольованого сервісу.

Також модуль надає доступ до інформації про стан монітора, історію перевірок, показники доступності та інші статистичні дані. Це дозволяє не лише налаштовувати моніторинг, але й аналізувати результати роботи системи.

Таким чином, модуль конфігурації моніторингу забезпечує повний цикл управління моніторами - від їх створення до аналізу результатів. Він виступає ключовою частиною системи, оскільки саме через нього визначається, які сервіси будуть контролюватися та яким чином здійснюватиметься моніторинг.

4.3 Модуль перевірки доступності вебзастосунків

Модуль перевірки доступності вебзастосунків є одним із центральних у системі моніторингу, оскільки саме він відповідає за фактичне визначення поточного стану сервісів. Його робота побудована як окремий безперервний процес, що виконується у фоновому режимі та не залежить від активності користувача. Основним завданням модуля є періодичний запуск перевірок, отримання результатів, збереження інформації про виконані перевірки та передача цих даних для подальшого аналізу.

Реалізація модуля передбачає поділ процесу перевірки на кілька послідовних етапів. На першому етапі система аналізує наявні активні монітори та визначає, для яких із них настав час чергової перевірки. Для цього враховується попередній час виконання перевірки та інтервал, заданий у конфігурації монітора. Якщо умови виконані, система формує задачу на перевірку та передає її до черги повідомлень. Такий підхід дозволяє відокремити планування перевірок від їх безпосереднього виконання та зменшити навантаження на основну логіку застосунку.

На наступному етапі інший фоновий процес отримує сформовані задачі з черги та виконує перевірку конкретного ресурсу. Для цього використовується інформація про монітор, зокрема його ідентифікатор, адресу та тип перевірки.

Після завершення виконання результат обробляється та фіксується у базі даних. Така організація дозволяє зробити систему більш гнучкою та підготувати її до подальшого масштабування, оскільки процес планування і процес виконання перевірок працюють незалежно один від одного.

Під час виконання перевірки система визначає успішність доступу до ресурсу, зберігає час виконання, код відповіді або текст помилки, а також дату й час перевірки. Якщо ресурс виявився недоступним, додатково може бути сформовано повідомлення про проблему, а також створено запис про інцидент. Це дозволяє використовувати модуль не лише як інструмент перевірки, а і як джерело даних для всієї підсистеми виявлення збоїв та сповіщення користувачів.

Окрім звичайних перевірок, модуль забезпечує накопичення історії виконання, що надалі використовується для розрахунку статистики. Окремий фоновий процес із заданою періодичністю узагальнює накопичені результати перевірок, обчислює відсоток доступності, середній час відповіді, кількість невдалих перевірок та число інцидентів за певний проміжок часу. Завдяки цьому система не лише фіксує поточний стан ресурсу, але й дає можливість оцінювати його роботу в динаміці.

Таким чином, модуль перевірки доступності вебзастосунків реалізує повний цикл автоматизованого контролю: від визначення моменту запуску перевірки до збереження результатів і формування даних для статистичного аналізу. Його побудова на основі фонових процесів та черги повідомлень забезпечує безперервність роботи, зручність розширення та відповідність вимогам до сучасних систем моніторингу.

4.3.1 Перевірка HTTP/HTTPS запитами

Одним із основних способів перевірки доступності в системі є виконання HTTP/HTTPS-запитів до вебзастосунків. Саме цей підхід використовується як

базовий механізм контролю стану сервісів, оскільки він дозволяє перевірити, чи відповідає ресурс на мережевий запит та чи повертає коректну відповідь. У процесі перевірки до вказаної адреси надсилається запит, після чого аналізується факт отримання відповіді та її статус.

Успішність перевірки визначається на основі HTTP-статусу відповіді. Якщо сервер повертає успішний код відповіді, перевірка вважається успішною. Якщо ж ресурс не відповідає, повертає помилку або під час запиту виникає виняткова ситуація, система фіксує невдалу перевірку. Додатково зберігається код статусу або текст помилки, що дозволяє надалі аналізувати характер проблеми та використовувати ці дані під час обробки інцидентів.

Реалізація HTTP/HTTPS-перевірок є достатньо універсальною, оскільки підходить для більшості вебзастосунків незалежно від їх призначення. Такий підхід дозволяє швидко перевірити доступність ресурсу без необхідності складної інтеграції з його внутрішньою логікою. Водночас результати перевірок можуть бути використані не лише для визначення факту доступності, але і для побудови історії роботи сервісу.

У системі також підтримуються й інші типи перевірок, зокрема контроль TCP-з’єднання та перевірка доступності через пінгування. Це дає можливість використовувати модуль не лише для вебресурсів у вузькому розумінні, а і для більш загального моніторингу мережевих сервісів. Проте HTTP/HTTPS-запити залишаються основним способом перевірки для вебзастосунків, оскільки найкраще відповідають їх природі та сценаріям використання.

Таким чином, перевірка HTTP/HTTPS-запитами є базовим елементом системи моніторингу, що дозволяє в автоматичному режимі контролювати стан вебзастосунків.

4.3.2 Вимірювання часу відгуку та продуктивності

Окрім визначення факту доступності ресурсу, важливим завданням системи є оцінка швидкодії вебзастосунку. Для цього під час кожної перевірки виконується вимірювання часу, необхідного для отримання відповіді від сервісу. Такий показник дозволяє оцінити не лише наявність зв’язку із ресурсом, але й якість його роботи з точки зору користувача.

Час відгуку фіксується для кожної перевірки окремо та зберігається разом з іншими параметрами, зокрема результатом перевірки, кодом відповіді та можливими повідомленнями про помилки. Завдяки цьому формується історія продуктивності ресурсу, яку можна використовувати для подальшого аналізу. Накопичення таких даних дає змогу не лише виявляти критичні відмови, але й визначати поступове погіршення роботи сервісу, наприклад збільшення затримок при обробці запитів.

Для узагальнення отриманої інформації у системі передбачено окремий механізм розрахунку статистики. Через визначені проміжки часу обробляються результати перевірок за попередній період, після чого для кожного монітора обчислюються такі показники, як середній час відповіді, відсоток доступності, загальна кількість перевірок, кількість невдалих перевірок та кількість інцидентів. Це дозволяє перейти від аналізу окремих перевірок до оцінки загальної продуктивності сервісу за певний часовий інтервал.

З практичної точки зору вимірювання часу відгуку є важливим не менше, ніж перевірка самої доступності. Сервіс може формально залишатися доступним, але працювати настільки повільно, що це негативно впливатиме на користувачів. Саме тому фіксація показників продуктивності дозволяє виявляти проблеми на ранніх етапах та отримувати більш повну картину стану вебзастосунку.

Отже, модуль вимірювання часу відгуку та продуктивності доповнює перевірку доступності, забезпечуючи збір кількісних характеристик роботи сервісів.

4.4 Модуль виявлення інцидентів

Модуль виявлення інцидентів призначений для фіксації проблем у роботі моніторів та подальшого управління інформацією про такі події. Його задача полягає не лише у визначенні факту збою, але й у збереженні структурованих даних про інцидент, щоб у подальшому можна було аналізувати причини виникнення проблем, відслідковувати їхній життєвий цикл та надавати користувачу актуальну інформацію про стан сервісу.

Формування інциденту відбувається під час виконання перевірки доступності. Якщо перевірка завершується неуспішно, система додатково аналізує, чи не було вже створено інцидент для цього монітора протягом останнього проміжку часу. Такий підхід дозволяє уникати дублювання записів у випадках, коли одна і та сама проблема триває певний час і сервіс не проходить кілька перевірок поспіль. Якщо нещодавнього інциденту не існує, у системі створюється новий запис із зазначенням монітора, часу початку, ознаки підтвердження та можливої причини збою.

Окрім автоматичного створення, модуль забезпечує подальшу роботу з інцидентами після їх виникнення. Користувач може отримати список інцидентів із посторінковим виведенням, переглянути детальну інформацію про конкретний інцидент, а також виконувати операції, пов’язані з його життєвим циклом. До таких операцій належать підтвердження інциденту, його закриття, додавання коментарів та видалення. Завдяки цьому інцидент у системі виступає не просто як одноразовий запис про помилку, а як повноцінна сутність, яка супроводжує процес обробки проблеми від моменту її виявлення до остаточного вирішення.

Під час закриття інциденту система зберігає додаткову інформацію, зокрема час завершення, встановлену причину збою та коментар щодо виконаних дій. Це важливо з точки зору подальшого аналізу, оскільки дозволяє накопичувати історію відмов і рішень, що може бути використана для пошуку типових проблем або оцінки стабільності конкретних сервісів. Якщо інцидент уже був закритий раніше,

повторне закриття не допускається, що забезпечує коректність станів у межах системи.

Окремою частиною модуля є підтримка коментарів до інцидентів. Це дає можливість фіксувати додаткові пояснення, хід розслідування або службову інформацію, пов’язану з проблемою. Таким чином інцидент перетворюється на центр накопичення всієї супровідної інформації, яка може бути корисною під час командної роботи або подальшого аудиту. У реалізації також передбачено отримання детальної інформації про інцидент разом із пов’язаними коментарями та даними, необхідними для відображення їх у клієнтському інтерфейсі.

Для доступу до інцидентів у системі використовується окремий інтерфейс запитів, що дозволяє отримувати як список інцидентів користувача, так і дані про конкретний запис. Це забезпечує зручну інтеграцію модуля з клієнтською частиною та дає змогу будувати окремий інтерфейс для перегляду і обробки інцидентів.

Таким чином, модуль виявлення інцидентів виконує відразу кілька важливих функцій: фіксує проблеми під час перевірок, запобігає дублюванню однакових подій, підтримує повний життєвий цикл інциденту та надає інструменти для його подальшого аналізу. Завдяки цьому система моніторингу не обмежується лише виявленням недоступності сервісу, а дозволяє організувати більш повний процес обліку та супроводу проблем.

5 ВЗАЄМОДІЯ КОРИСТУВАЧА ІЗ СИСТЕМОЮ МОНІТОРИНГУ

У цьому розділі розглядається практичне використання розробленої системи моніторингу вебзастосунків. Після опису архітектури та реалізації програмних модулів важливо показати, як система працює у реальних умовах, як відбувається її запуск, налаштування та обробка перевірок.

Буде описано процес розгортання системи, а також наведено приклад її роботи, який демонструє основні етапи: створення монітора, виконання перевірок, виявлення інцидентів та отримання результатів користувачем.

5.1 Налаштування та розгортання системи

Для запуску системи моніторингу необхідно виконати базову конфігурацію середовища та підготувати інфраструктуру для її роботи. Оскільки система реалізована як серверний застосунок, вона розгортається у вигляді вебсервісу з підтримкою фонових процесів.

Першим етапом є налаштування конфігураційних параметрів. До них належать параметри підключення до бази даних, налаштування автентифікації, ключі для генерації JWT-токенів, а також параметри підключення до сервісу черг повідомлень. Усі ці дані задаються у конфігураційних файлах.

Наступним кроком є підготовка середовища виконання. Система контейнеризована за допомогою Docker, що дозволяє запускати її незалежно від операційної системи. Для цього використовується Dockerfile та docker-compose, які описують процес збірки образу та запуску сервісів.

Після цього система може бути розгорнута у хмарному середовищі, зокрема на платформі Microsoft Azure. Це забезпечує постійну доступність, можливість

масштабування та стабільну роботу фонових процесів, які виконують перевірки доступності.

5.2 Приклад роботи системи

Для демонстрації роботи системи розглянемо типовий сценарій використання моніторингу вебзастосунку. Система забезпечує повний цикл взаємодії з користувачем - від створення монітора до аналізу отриманих результатів.

На першому етапі користувач створює новий монітор, вказуючи адресу вебсервісу та параметри перевірки. На рисунку 5.1 представлено інтерфейс створення монітора. Користувач задає назву, адресу ресурсу, тип перевірки та інтервал виконання запитів. Після збереження монітор додається до системи та може бути активований для подальшого використання.

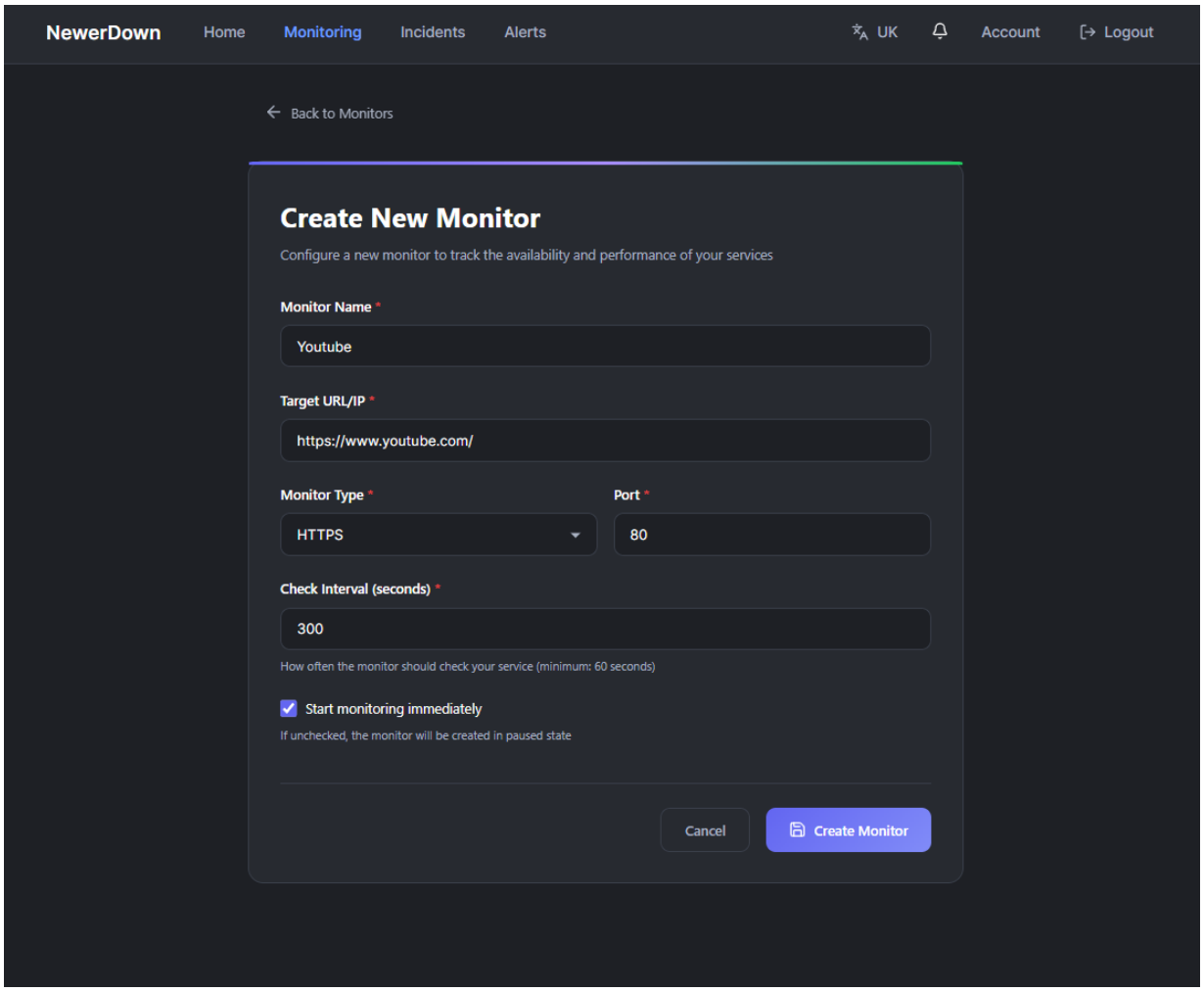


Рисунок 5.1 – Форма створення нового монітора

Після створення монітора користувач отримує доступ до списку всіх створених моніторів, де відображається їх поточний стан.

На рисунку 5.2 наведено список моніторів, які відображаються у системі. Для кожного монітора показано його стан, тип перевірки та основні параметри. Користувач може керувати моніторами, зокрема редагувати, призупиняти або видаляти їх.

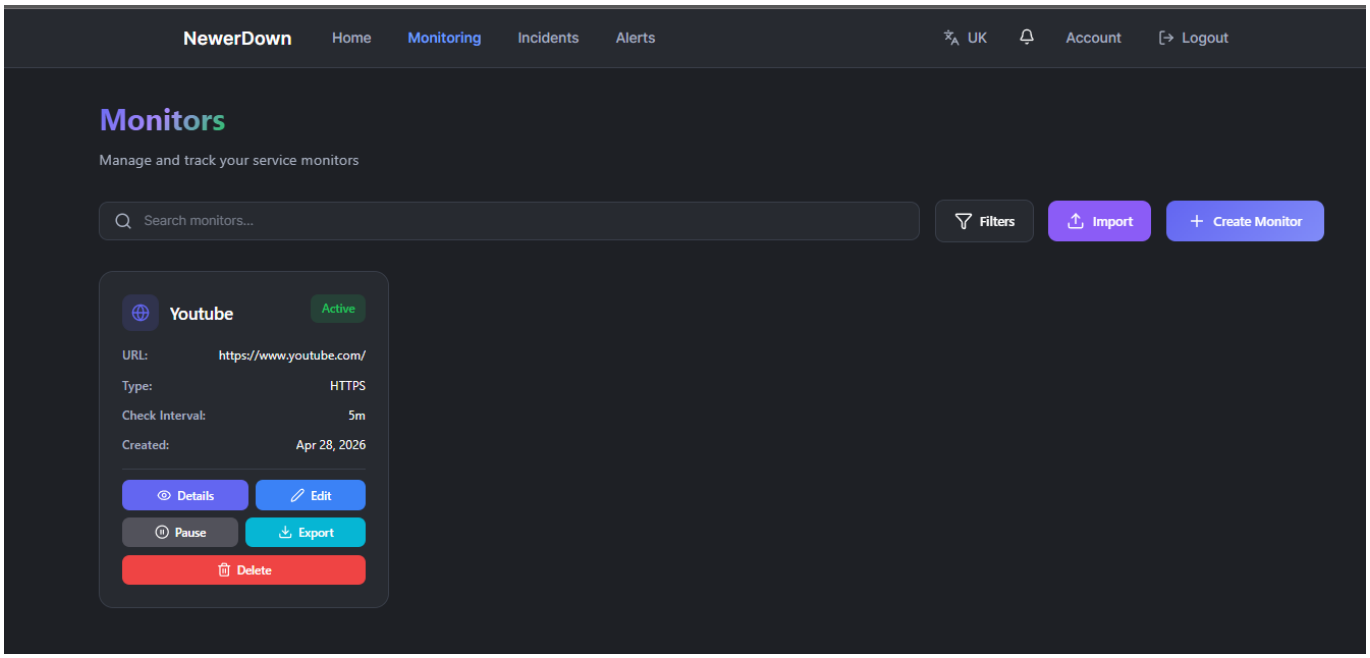


Рисунок 5.2 – Список створених моніторів

Після цього система у фоновому режимі автоматично виконує перевірки. Планувальник визначає, коли необхідно виконати наступну перевірку, формує відповідну задачу та передає її на виконання. Інший процес отримує цю задачу та здійснює перевірку доступності ресурсу.

Під час перевірки система надсилає HTTP-запит до вебсервісу, вимірює час відповіді та визначає, чи є ресурс доступним. Результат перевірки зберігається у базі даних разом із додатковою інформацією, такою як статус відповіді та можливі помилки.

У випадку, якщо ресурс недоступний або працює некоректно, система фіксує це як інцидент. При цьому виконується перевірка на наявність аналогічних інцидентів для уникнення дублювання.

На рисунку 5.3 представлено інтерфейс перегляду інцидентів. У разі виникнення помилки система фіксує її як інцидент та відображає інформацію про час виникнення та причину. Користувач може переглянути деталі або змінити статус інциденту.

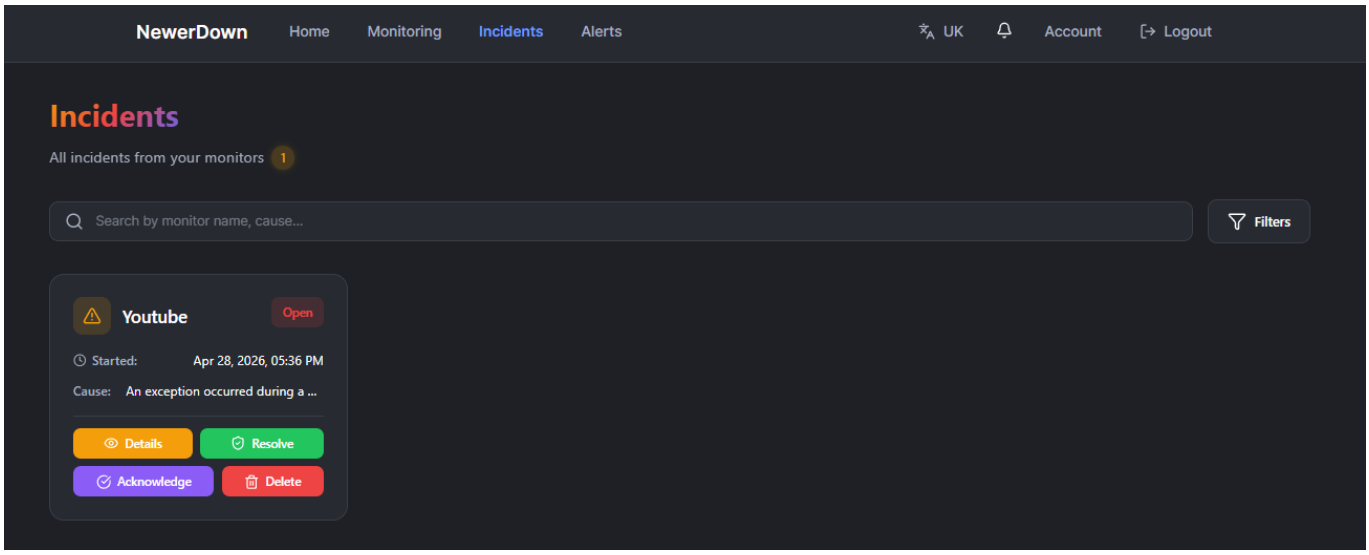


Рисунок 5.3 – Відображення інцидентів у системі

Після накопичення достатньої кількості даних система виконує їх обробку та формує статистичні показники. Обчислюються значення доступності сервісу, середній час відповіді та кількість збоїв, що дозволяє оцінити якість роботи ресурсу.

На рисунку 5.4 показано статистичні показники роботи монітора, зокрема відсоток доступності, середній час відповіді та кількість перевірок. Також представлено графік зміни часу відгуку, що дозволяє аналізувати поведінку сервісу у часі та виявляти можливі проблеми продуктивності.

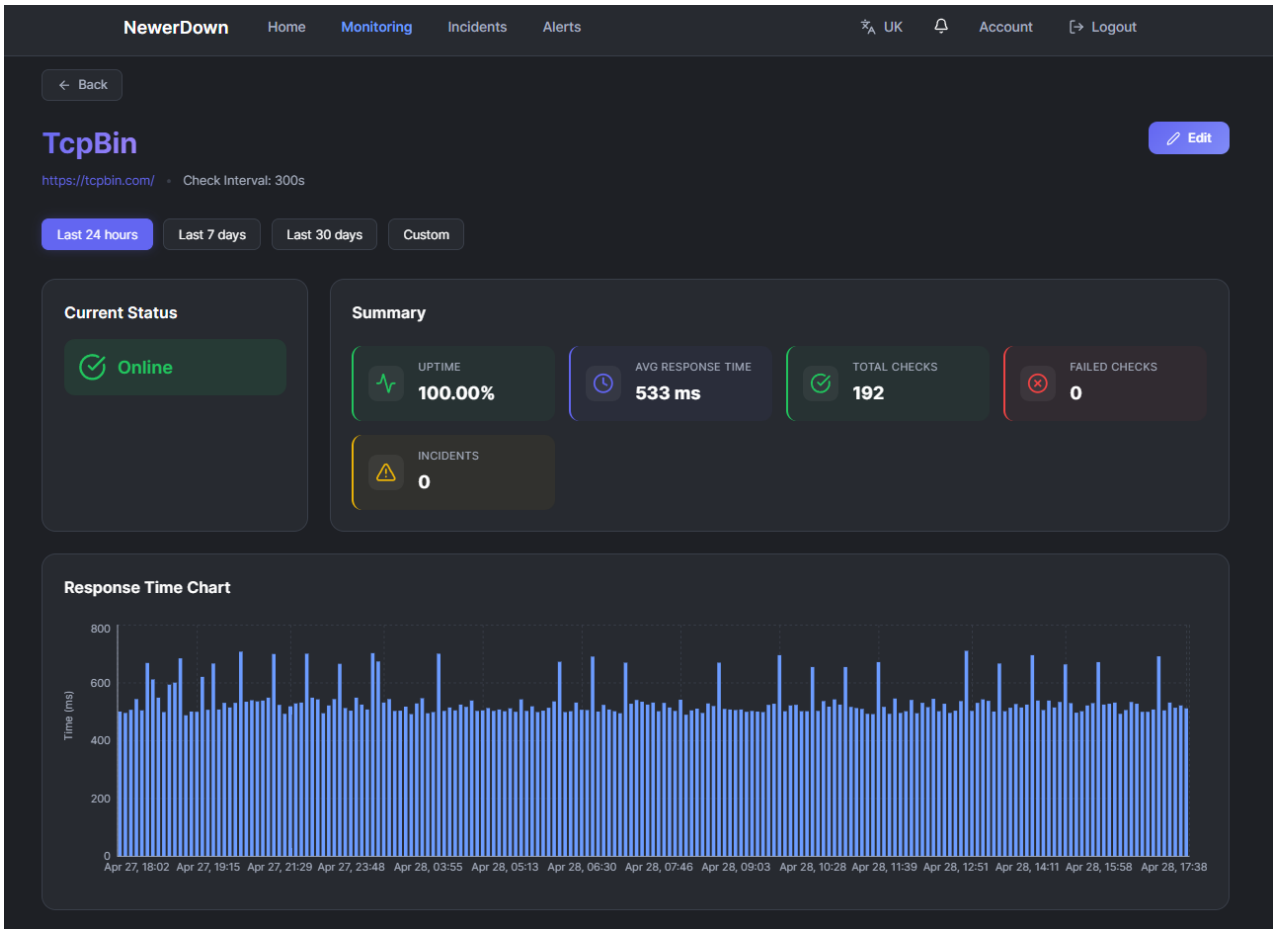


Рисунок 5.4 – Статистика роботи монітора

Крім агрегованих даних, користувач має можливість переглянути детальну історію перевірок.

На рисунку 5.5 наведено детальну історію перевірок, що містить інформацію про час виконання запиту, статус відповіді та час відгуку. Це дозволяє здійснювати глибший аналіз роботи сервісу та виявляти нестабільність у його функціонуванні.

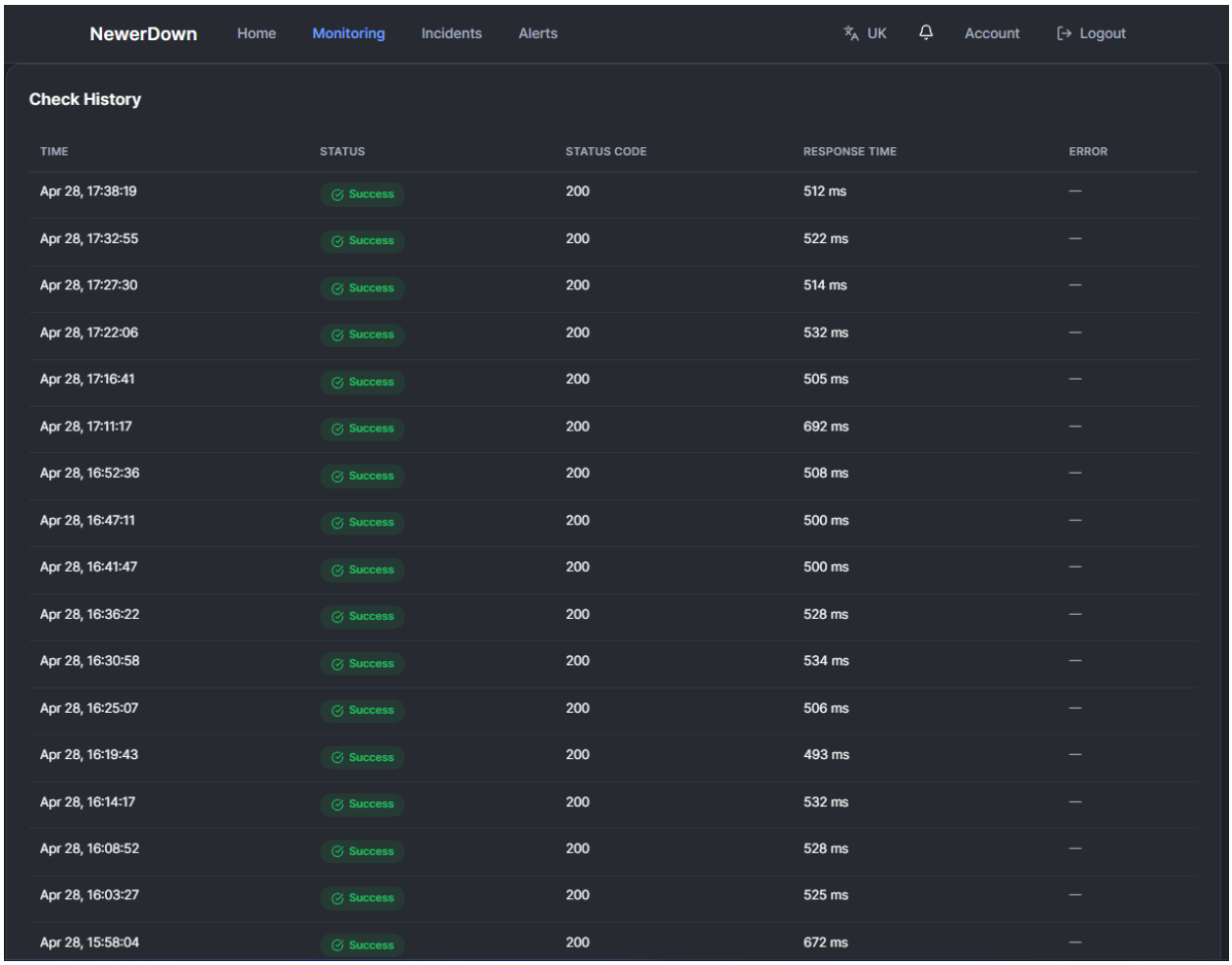


Рисунок 5.5 – Історія перевірок вебсервісу

Таким чином, система забезпечує повний цикл моніторингу – від налаштування до отримання результатів і подальшого аналізу роботи вебзастосунку. Представлений приклад демонструє практичну реалізацію основних функцій системи та підтверджує її працездатність у реальних умовах.

ВИСНОВКИ

Під час виконання дипломної роботи було вирішено задачу розробки серверної частини системи моніторингу доступності вебзастосунків. У результаті виконано повний цикл робіт: від аналізу предметної області до реалізації, тестування та розгортання системи. Основні результати роботи полягають у наступному:

1. Проведено аналіз підходів до моніторингу вебзастосунків, включаючи методи перевірки доступності, оцінку продуктивності та способи виявлення інцидентів. На основі цього було обґрунтовано використання активного моніторингу з виконанням HTTPS-запитів у поєднанні з аналізом часу відгуку та результатів перевірок для визначення стану сервісів.

2. Проаналізовано існуючі рішення у сфері моніторингу, як комерційні, так і з відкритим вихідним кодом, що дозволило визначити їх переваги та обмеження. У результаті було прийнято рішення про розробку власної системи, яка забезпечує гнучкість налаштування, контроль над даними та можливість інтеграції з іншими компонентами.

3. Розроблено програмну систему моніторингу доступності вебзастосунків, яка реалізує основний функціонал: створення та управління моніторами, автоматичне виконання перевірок, збереження результатів, вимірювання часу відгуку та обчислення статистики. Архітектура системи побудована за модульним принципом із розділенням на окремі рівні, що забезпечує зручність підтримки та можливість подальшого розвитку.

4. У системі реалізовано асинхронну обробку перевірок із використанням черг повідомлень та фонових сервісів. Це дозволило відокремити планування перевірок від їх виконання, підвищити продуктивність та забезпечити масштабованість системи при збільшенні кількості моніторів.

5. Реалізовано модуль виявлення інцидентів, який дозволяє автоматично фіксувати проблеми у роботі сервісів, уникати дублювання подій та підтримувати

життєвий цикл інциденту. Користувач має можливість отримувати список інцидентів, переглядати детальну інформацію та виконувати дії з їх обробки.

6. Забезпечено взаємодію із системою через REST API, що дозволяє отримувати дані про монітори, результати перевірок, статистику та інциденти. Для захисту доступу використано механізм автентифікації на основі JWT токену, що забезпечує масштабованість та безпечну роботу системи.

7. Здійснено розгортання системи у хмарному середовищі Microsoft Azure з використанням контейнеризації. Це дозволило забезпечити доступність, гнучкість налаштування та можливість масштабування системи відповідно до навантаження.

Розроблена система є повноцінним прототипом рішення для моніторингу доступності вебзастосунків та може бути використана як основа для подальшого розвитку. Перспективи вдосконалення включають розширення типів перевірок, додавання нових каналів сповіщень, а також покращення механізмів аналізу даних.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Tanenbaum A. S., Wetherall D. J. Computer Networks. 5th ed. Boston : Pearson, 2011. 960 p.

2. Kurose J. F., Ross K. W. Computer Networking: A Top-Down Approach. 7th ed. Boston : Pearson, 2017. 824 p.

3. Kleppmann M. Designing Data-Intensive Applications: The Big Ideas Behind Reliable, Scalable, and Maintainable Systems. Kyiv : O'Reilly Media, 2017. 616 p.

4. Newman S. Building Microservices: Designing Fine-Grained Systems. Kyiv: O'Reilly Media, 2015. 278 p.

5. Richardson C. Microservices Patterns: With Examples in Java. Shelter Island : Manning Publications, 2018. 520 p.

6. Fowler M. Patterns of Enterprise Application Architecture. Boston : Addison-Wesley Professional, 2002. 560 p.

7. Design Patterns: Elements of Reusable Object-Oriented Software / E. Gamma, R. Helm, R. Johnson, J. Vlissides. Reading : Addison-Wesley, 1994. 395 p.

8. Turnbull J. The Art of Monitoring. Turnbull Press, 2016**.** 524 p.

9. Site Reliability Engineering: How Google Runs Production Systems / B. Beyer, C. Jones, J. Petoff, N. R. Murphy. Kyiv: O'Reilly Media, 2016. 552 p.

10. Humble J., Farley D. Continuous Delivery: Reliable Software Releases through Build, Test, and Deployment Automation. Boston : Addison-Wesley Professional, 2010. 512 p.

11. Burns B., Beda J., Hightower K. Kubernetes: Up and Running: Dive into the Future of Infrastructure. 2nd ed. Kyiv: O'Reilly Media, 2019. 278 p.

12. Erl T., Mahmood Z., Puttini R. Cloud Computing: Concepts, Technology & Architecture. Upper Saddle River : Prentice Hall, 2013. 528 p.

13. van Steen M., Tanenbaum A. S. Distributed Systems. 3rd ed. [S. l.] : distributed-systems.net, 2017. 596 p.

14. Martin R. C. Clean Architecture: A Craftsman's Guide to Software Structure

and Design. Boston : Pearson, 2017. 432 p.

15. Freeman A. Pro ASP.NET Core MVC. Berkeley : Apress, 2016**.** 1018 p.

16. Esposito D. Programming ASP.NET Core. Redmond : Microsoft Press, 2018. 416 p.

ДОДАТОК А

РЕАЛІЗАЦІЯ МЕХАНІЗМУ ПЕРЕВІРКИ ДОСТУПНОСТІ ТА ОБРОБКИ РЕЗУЛЬТАТІВ МОНІТОРИНГУ

УКР.НТУУ”КПІ ім. Ігоря Сікорського”

Аркушів 3

Київ 2026

Програмні засоби:

- мова програмування - C#;
- фреймворк - .NET

```
  public async Task CheckWebsiteAsync(MonitorViewModel request,
CancellationToken cancellationToken)
    {
        bool isSuccess = false;
        var monitor = await _context.Monitors
            .Include(m => m.User)
            .FirstOrDefaultAsync(m => m.Id == request.Id,
cancellationToken: cancellationToken);

        if (monitor is null)
        {
            _logger.LogWarning("Monitor {MonitorId} not found",
request.Id);
            return;
        }

        var stopwatch = Stopwatch.StartNew();
        string? status = null;
        string? error = null;
        try
        {
            switch (monitor.MonitorType)
            {
                case MonitorType.Http:
                    (isSuccess, status, error) = await
CheckHttpAsync(monitor.Url, cancellationToken);
                    break;
                case MonitorType.TcpPort:
                    (isSuccess, status, error) = await
CheckTcpAsync(monitor.Url, monitor.Port.GetValueOrDefault(),
cancellationToken);
                    break;
                case MonitorType.Ping:
                    (isSuccess, status, error) = await
CheckPingAsync(monitor.Url, cancellationToken);
                    break;
            }
        }
        catch (Exception ex)
        {
            error = ex.Message;
            _logger.LogError(ex, "Error checking monitor {MonitorId}",
monitor.Id);
        }
        finally
```

```
        {
            stopwatch.Stop();
        }

        var check = new MonitorCheck
        {
            Id = Guid.NewGuid(),
            MonitorId = monitor.Id,
            CheckedAt = _dateTimeProvider.UtcNow(),
            ResponseTimeMs = (int)stopwatch.ElapsedMilliseconds,
            StatusCode = status,
            IsSuccess = isSuccess,
            ErrorMessage = error
        };

        _context.MonitorChecks.Add(check);
        await _context.SaveChangesAsync(cancellationToken);

        var alert = await _context.Alerts.FirstOrDefaultAsync(x =>
x.MonitorId == monitor.Id, cancellationToken);
        if (!isSuccess && alert is not null && alert.Type ==
AlertType.SignalR)
        {
            var payload = new
            {
                monitorId = monitor.Id,
                url = monitor.Url,
                isSuccess,
                status,
                error,
                checkedAt = check.CheckedAt
            };

            await _notificationHub
                .Clients
                .All
                .SendAsync("ReceiveAlert", payload, cancellationToken);
        }
        if (!isSuccess)
        {
            await TryCreateIncidentAsync(monitor.Id, error,
cancellationToken);
        }

        _logger.LogInformation("Website check for monitor {MonitorId}
completed, success={Success}", monitor.Id, isSuccess);
    }

    private async Task<(bool success, string? status, string? error)>
```

```
CheckHttpAsync(string url, CancellationToken ct)
    {
        try
        {
            var client = _httpClientFactory.CreateClient("MonitorClient");
            var response = await client.GetAsync(url, ct);
            return (response.IsSuccessStatusCode,
((int)response.StatusCode).ToString(), null);
        }
        catch (Exception ex)
        {
            return (false, null, ex.Message);
        }
    }

    private async Task TryCreateIncidentAsync(Guid monitorId, string?
error, CancellationToken cancellationToken)
    {
        var now = _dateTimeProvider.UtcNow();
        var fiveMinutesAgo = now.AddMinutes(-
IncidentConstants.DefaultCheckIntervalInMinutes);

        var recentIncident = await _context.Incidents
            .Where(i => i.MonitorId == monitorId)
            .Where(i => i.StartedAt >= fiveMinutesAgo)
            .OrderByDescending(i => i.StartedAt)
            .FirstOrDefaultAsync(cancellationToken);

        if (recentIncident is not null)
        {
            _logger.LogInformation(
                "Incident for monitor {MonitorId} skipped (exists recent
incident at {StartedAt})",
                monitorId,
                recentIncident.StartedAt);
            return;
```

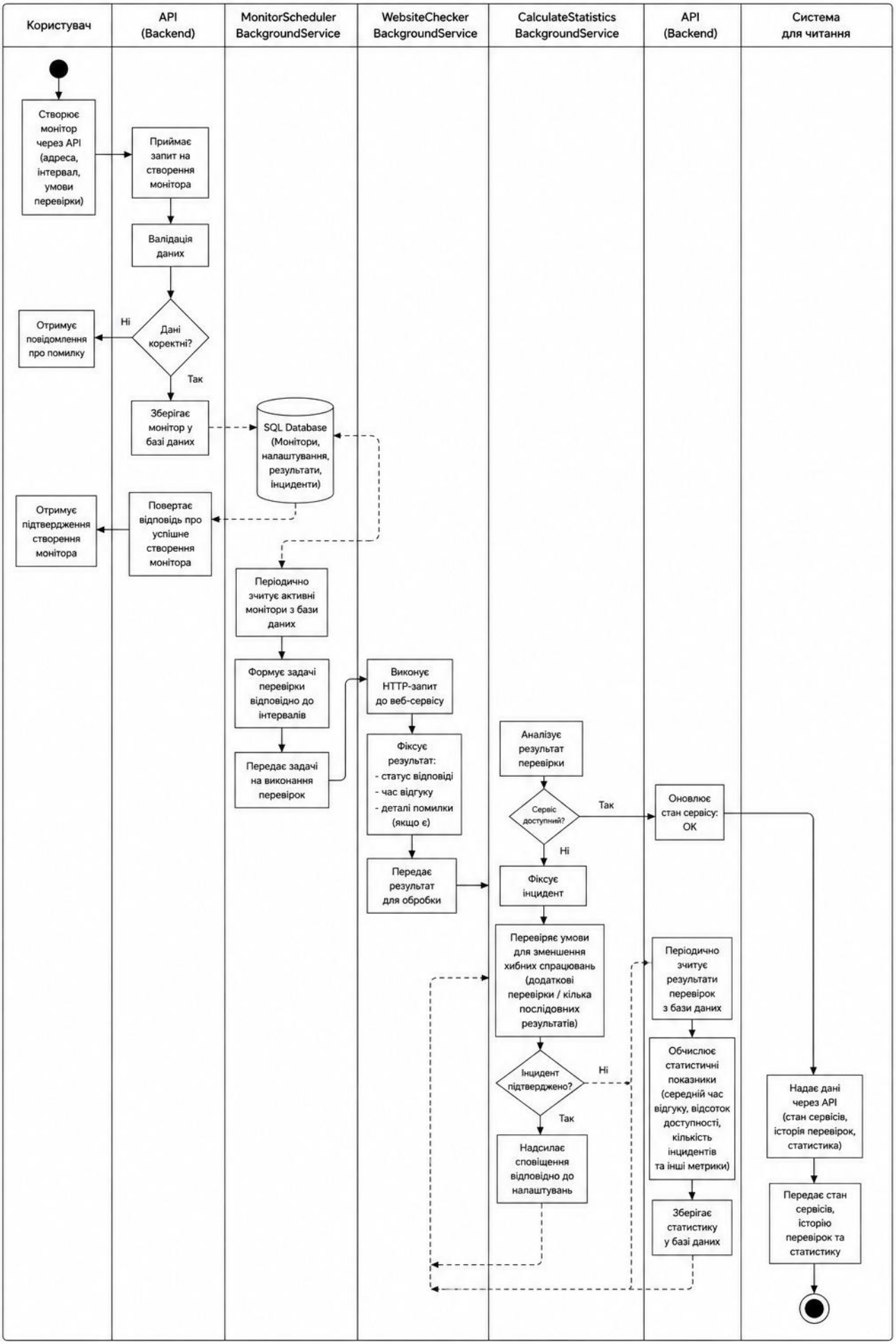
ДОДАТОК Б

ДІАГРАМА АКТИВНОСТІ

УКР.НТУУ”КПІ ім. Ігоря Сікорського”

Аркушів 1

Київ 2026


Користувач
API (Backend)
MonitorScheduler BackgroundService
WebsiteChecker BackgroundService
CalculateStatistics BackgroundService
API (Backend)
Система для читання
Створює монітор через API (адреса, інтервал, умови перевірки)
Приймає запит на створення монітора
Валідація даних
Дані коректні?
Ні
Так
Отримує повідомлення про помилку
Зберігає монітор у базі даних
SQL Database (Монітори, налаштування, результати, інциденти)
Повертає відповідь про успішне створення монітора
Отримує підтвердження створення монітора
Періодично зчитує активні монітори з бази даних
Формує задачі перевірки відповідно до інтервалів
Передає задачі на виконання перевірок
Виконує HTTP-запит до веб-сервісу
Фіксує результат:
- статус відповіді
- час відгуку
- деталі помилки (якщо є)
Передає результат для обробки
Аналізує результат перевірки
Сервіс доступний?
Так
Ні
Оновлює стан сервісу: OK
Фіксує інцидент
Перевіряє умови для зменшення хибних спрацювань (додаткові перевірки / кілька послідовних результатів)
Інцидент підтверджено?
Ні
Так
Надсилає сповіщення відповідно до налаштувань
Періодично зчитує результати перевірок з бази даних
Обчислює статистичні показники (середній час відгуку, відсоток доступності, кількість інцидентів та інші метрики)
Зберігає статистику у базі даних
Надає дані через API (стан сервісів, історія перевірок, статистика)
Передає стан сервісів, історію перевірок та статистику