Вінницький національний технічний університет
(повне найменування вищого навчального закладу)

Факультет інформаційних технологій та комп'ютерної інженерії
(повне найменування інституту, назва факультету (відділення))

Кафедра програмного забезпечення
(повна назва кафедри (предметної, циклової комісії))

МАГІСТЕРСЬКА КВАЛІФІКАЦІЙНІ РОБОТА

на тему:

«Розробка відмовостійких методів передавання повідомлень та розподіленої backend as a service платформи для мобільних і веб-застосувань»

Виконав: студент 2-го курсу групи 2ПІ-22м
спеціальності 121 «Інженерія програмного забезпечення»
Павлюк І. А.
(прізвище та ініціали)

Керівник: к.т.н., доц. каф. ПЗ
Ракитянська Г.Б.
(прізвище та ініціали)
«15» грудня 2023 р.

Опонент: к.т.н., доц. каф. ЗІ
Баришев Ю. В.
(прізвище та ініціали)
«15» грудня 2023 р.

Допущено до захисту
завідувач кафедри ПЗ
д.т.н. проф. Романюк О.Н.
(прізвище та ініціали)
«15» грудня 2023 р.

Вінниця ВНТУ – 2023

Вінницький національний технічний університет
Факультет інформаційних технологій та комп'ютерної інженерії
Кафедра програмного забезпечення
Рівень вищої освіти ІІ-й (магістерський)
Галузь знань – 12 Інформаційні системи
Спеціальність 121 – «Інженерія програмного забезпечення»
Освітньо-професійна програма – Інженерія програмного забезпечення

ЗАТВЕРДЖУЮ
Завідувач кафедри ПЗ
Романюк О.Н.
"19" вересня 2023 року

З А В Д А Н Н Я

НА МАГІСТЕРСЬКУ КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТУ

Павлюку Ігорю Анатолійовичу

1. Тема роботи: «Розробка відмовостійких методів передавання повідомлень та розподіленої backend as a service платформи для мобільних і веб-застосувань»
Керівник роботи: Ракитянська Ганна Борисівна, к.т.н., доцент кафедри ПЗ, затверджені наказом вищого навчального закладу від "18" вересня 2023 року № 247.
2. Строк подання студентом роботи: "05" грудня 2023 року.
3. Вихідні дані до роботи: модель розробки – ітеративна; метод передачі повідомлень між компонентами – REST API; метод балансування навантаження – раунд-робін; метод обробки даних – потокова обробка; застосування теорії управління для оптимізації процесів; вхідні дані – набір даних користувачів і транзакцій у форматі JSON; API ключі для інтеграції з зовнішніми сервісами; ідентифікатори сесій користувачів; вихідні дані – звіти та аналітичні дані, результати оптимізації процесів у відповідному форматі.
4. Зміст розрахунково-пояснювальної записки: вступ; аналіз стану питання та постановка задач дослідження, розробка адаптивного алгоритму для підвищення ефективності передачі повідомлень, розробка програмних засобів, тестування системи, економічна частина, висновки, список використаних джерел, додатки;
5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень): мета, об'єкт та предмет дослідження; завдання дослідження; аналіз стану питання; порівняння з аналогами; використані технології при розробці системи; тестування системи; економічне обґрунтування; наукова новизна одержаних результатів; практична цінність одержаних результатів, апробація та публікації.

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада Консультанта	Підпис, дата	
		завдання видав	виконання прийняв
1-4	Ракитянська Г.Б. к.т.н, доцент кафедри ПЗ	19.09.2023	05.12.2023
5	Кавецький В.В., к.е.н., доцент ЕПВМ	26.11.2023	28.11.2023

7. Дата видачі завдання: "19" вересня 2023 року.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів магістерської кваліфікаційної Роботи	Строк виконання етапів роботи	Примітка
1	Аналіз стану питання	20.09.20 – 25.09.23	вик
2	Аналіз існуючих аналогів та постановка задач дослідження	26.09.20 – 28.09.23	вик
3	Розробка адаптивного алгоритму для підвищення ефективності передачі повідомлень	29.09.20 – 12.10.23	вик
4	Написання програмних модулів для реалізації розроблених алгоритмів	13.10.20 – 28.10.23	вик
5	Розробка серверної частини BaaS-платформи	29.11.20 – 20.11.23	вик
6	Тестування розробленого програмного продукту	21.11.20 – 25.11.23	вик
7	Економічна частина	26.11.20 – 28.11.23	вик
8	Оформлення матеріалів до захисту МКР	29.11.20 – 1.12.23	вик

Студент (підпис) **Павлюк І.А.** (прізвище та ініціали)

Керівник магістерської кваліфікаційної роботи (підпис) **Ракитянська Г.Б.** (прізвище та ініціали)

АНОТАЦІЯ

УДК 519.7:004.89

Павлюк І. А. Розробка відмовостійких методів передавання повідомлень та розподіленої backend as a service платформи для мобільних і веб-застосувань. Магістерська кваліфікаційна робота зі спеціальності 121 – інженерія програмного забезпечення, освітня програма – інженерія програмного забезпечення. Вінниця: ВНТУ, 2023. – 137 с.

На укр. мові. Бібліогр.: 36 назв; рис.: 50; табл.: 17;

У магістерській кваліфікаційній роботі розроблено методи та алгоритми для адаптивного управління процесом повторної передачі повідомлень, що дозволяє зменшити навантаження на мережу і підвищити її ефективність. Результати інтегровані у розроблену BaaS-платформу, призначену для мобільних та веб-застосувань.

Особлива увага приділена розробці адаптивних механізмів, які динамічно визначають оптимальні інтервали між спробами повторної передачі, базуючись на реальному стані мережі.

У процесі розробки системи управління процесом повторної передачі повідомлень було використано мову програмування Python. BaaS-платформа розроблена як мікросервісна архітектура на платформі .NET, використовуючи серверні технології Microsoft та MongoDB в якості основного сховища даних.

Для моделювання та дизайну системи використовувався Visual Paradigm, а всі сервіси були реалізовані з підтримкою стандартів REST API, що забезпечує гнучку взаємодію з різними клієнтами. Розроблені алгоритми можна використовувати для оптимізації і підвищення ефективності BaaS-платформ.

Ключові слова: BaaS-платформа, теорія керування, AR, ARIMA, MAPE, C#, Python, MongoDB, Jupyter Notebook.

ABSTRACT

UDC 519.7:004.89

Pavliuk I. A. Development of fault-tolerant message transmission methods and distributed backend as a service platform for mobile and web applications. Master's qualification work in specialty 121 – Software Engineering, educational program – Software Engineering. Vinnytsia: VNTU, 2023. – 137 p.

In Ukrainian language. Bibliogr.: 36 titles; fig .: 50; tab .: 17.

In the master's qualification work, methods and algorithms for adaptive control of the message retransmission process were developed, which allows to reduce the network load and increase its efficiency. The results have been integrated into the developed BaaS platform, designed for mobile and web applications.

Special attention was paid to the development of adaptive mechanisms that dynamically determine the optimal intervals between retransmission attempts, based on the real state of the network.

During the development process of the message retransmission control system, the Python programming language was used. The BaaS platform was developed as a microservice architecture on the .NET platform, using Microsoft server technologies and MongoDB as the primary data storage. Visual Paradigm was used for system modeling and design, and all services were implemented with support for REST API standards, providing flexible interaction with various clients.

The developed algorithms can be used to optimize and enhance the efficiency of BaaS platforms.

Keywords: BaaS, Control theory, AR, ARIMA, MAPE, C#, Python, MongoDB, Jupyter Notebook.

ЗМІСТ

ВСТУП

Актуальність роботи. У сучасному світі інформаційних технологій зростає потреба у вдосконаленні методів передачі повідомлень і розробці надійних розподілених систем для підвищення ефективності та доступності веб- та мобільних додатків. Незважаючи на постійний розвиток хмарних технологій та безсерверної архітектури, існуючі підходи до розробки backend-систем часто зіштовхуються з викликами, які пов'язані з обробкою та передачею даних, особливо у контексті високої доступності та відмовостійкості [1].

Використання таких патернів як "Circuit Breaker"[2] і "Brownout"[3] у веб-серверах та балансувальниках навантаження дозволяє ефективніше керувати запитами під високим навантаженням, забезпечуючи більш гнучке зниження якості обслуговування. Особливість "Brownout" полягає у можливості динамічного регулювання відсотка запитів до певних критичних компонентів системи, що забезпечує стабільність та надійність BaaS-платформ і збалансовує потреби в надійності системи під високим навантаженням.

В останні роки здобула популярність мікросервісна архітектура для проектування та створення розподілених програмних систем [4]. Даний архітектурний стиль передбачає створення сукупності невеличких сервісів, які функціонують як незалежні процеси, що взаємодіють через повідомлення використовуючи прості та швидкі протоколи передачі даних, зазвичай HTTP та мають власне сховище даних.

У сучасних умовах, мікросервіси все частіше використовуються у різноманітних сферах - від IoT-додатків [5], [6] до високопродуктивних обчислень [7], важливим є розробка методів та інструментів для оптимізації конфігурації цих патернів. Це особливо актуально в динамічних середовищах, де часто відбуваються зміни, такі як збільшення навантаження, варіації у кількості реплік або випуск нових версій реплік з вищим рівнем використання ресурсів.

Дослідження спрямоване на вивчення та оптимізацію механізмів "Retry" та "Circuit Breaker" у контексті мікросервісних архітектур є актуальним і важливим завданням, що відкриває нові можливості для підвищення надійності в розподілених системах. Зокрема, потребою у розробці автоматизованих технік для оптимізації конфігурації, які враховують складні та часто нелінійні взаємодії між параметрами конфігурації цих систем.

Передача повідомлень у розподілених системах, особливо в Backend as a Service (BaaS) для мобільних та веб-застосувань, вимагає ретельного планування та інноваційних рішень для забезпечення стабільності, швидкості та ефективності. Розвиток адаптивних методів передачі повідомлень, здатних динамічно реагувати на зміни у мережевому середовищі та поведінці користувачів, є важливим кроком у напрямку підвищення продуктивності та надійності цих систем. Така оптимізація не тільки покращує загальну працездатність BaaS-платформ, але й забезпечує кращий досвід для кінцевих користувачів. У зв'язку з цим, розробка відмовостійких методів передавання повідомлень і розподіленої BaaS-платформи є актуальною та важливою задачею, яка вимагає досліджень і інноваційних підходів.

Мета та завдання дослідження. Метою даної роботи є підвищення ефективності обробки запитів у мобільних та веб-застосуваннях шляхом розробки та аналізу адаптивних алгоритмів для повторної передачі повідомлень у контексті BaaS платформи.

Основні завдання дослідження включають:

- провести аналіз існуючих методів передачі повідомлень у BaaS платформах;
- розробити адаптивні алгоритми для повторної передачі повідомлень;
- розробити дизайн архітектури BaaS-платформи;
- розробити програмні сервіси BaaS-платформи;
- провести інтеграцію створених алгоритмів у BaaS-платформу;
- провести оцінку ефективності запропонованих алгоритмів у різних сценаріях використання;

- провести тестування розробленої платформи.

Об’єкт дослідження – процес передачі повідомлень у мікросервісній архітектурі BaaS-платформ.

Предмет дослідження – алгоритми та методи повторної передачі повідомлень у BaaS-платформах, зокрема адаптивні механізми, що реагують на зміни у мережевому середовищі та навантаженні системи.

Методи дослідження. Для досягнення поставлених завдань використовувалися наступні методи дослідження:

- методи теорії керування для розробки і налаштування адаптивного алгоритму повторної передачі повідомлень;
- статистичні методи аналізу часових рядів для прогнозування та аналізу метрик системи, таких як час відповіді та невдачі запитів;
- методи комп'ютерного моделювання для симуляції роботи адаптивного алгоритму та аналізу його ефективності в різних умовах;
- методи тестування програмного забезпечення для забезпечення надійності та безпеки розробленої платформи;
- методи побудови розподілених інформаційних систем для розробки архітектури програмної системи.

Наукова новизна отриманих результатів.

1. Отримав подальшого розвитку адаптивний метод передачі повідомлень, що на відміну від традиційних методів, які зосереджуються на статичному управлінні повторною передачею повідомлень, використовує динамічне адаптування на основі часових рядів та статистичного аналізу, що дозволяє системі ефективно реагувати на зміну в мережі та запобігати втраті повідомлень.

2. Отримала подальший розвиток модель високорівневої адаптивної системи повторної відправки повідомлень, яка на відміну від існуючих інтегрує адаптивний механізм на основі ПІД-контролера для регулювання поведінки системи з урахуванням поточних умов роботи, що дозволяє їй самостійно адаптуватися до змін у навантаженні та ефективно керувати процесом передачі повідомлень.

Практична цінність отриманих результатів.

Практична цінність отриманих у ході магістерської кваліфікаційної роботи результатів полягає в розробці BaaS платформи, яка застосовує адаптивний алгоритм для повторних спроб передачі повідомлень. Впровадження цієї платформи може значно підвищити ефективність обробки запитів та зменшити відмови в системах у різних галузях, від фінансових технологій до хмарних обчислень.

Особистий внесок здобувача. У науковій роботі, опублікованій у співавторстві, автору належать такі результати: методи та моделі адаптивної системи передавання повідомлень у розподіленій BAAS-платформі [8].

Апробація матеріалів. Результати дослідження були представлені на науково-практичній конференції «Електронні інформаційні ресурси: створення, використання, доступ» – Вінниця 2023.

Публікації. Основні положення магістерської кваліфікаційної роботи опубліковані у збірнику матеріалів Міжнародної науково-практично Інтернет-конференції 2023 року «Електронні інформаційні ресурси: створення, використання, доступ» [8].

Структура та обсяг роботи. Магістерська кваліфікаційна робота складається зі вступу, п’яти розділів, висновків, списку літератури, що містить 36 найменувань, 4 додатки. Робота містить 18 ілюстрації, 17 таблиць.

1 АНАЛІЗ СТАНУ РОЗВИТКУ BACKEND AS A SERVICE ПЛАТФОРМ ТА ПОСТАНОВКА ЗАДАЧ ДОСЛІДЖЕННЯ

1.1 Аналіз стану питання розробки BaaS-платформи для мобільних та веб-застосувань

Велика кількість сучасних додатків побудовані на базі архітектур, що відстежують і забезпечують працездатність серверної та клієнтської сторін. Однак, з розвитком хмарних технологій та безсерверних архітектур (Serverless), з'являється можливість значно спростити та оптимізувати процес розробки та впровадження додатків [9].

Безсерверні архітектури надають можливість додаткам значною мірою залежати від сторонніх служб, які розміщуються "в хмарі" та управляють серверною логікою та станом. Основна перевага безсерверних систем полягає у здатності скоротити витрати, зменшити час та складність розробки за рахунок використання існуючих ресурсів і сервісів, розроблених провідними хмарними провайдерами, такими як Amazon, Google, і Facebook [10].

З погляду архітектури, класичний підхід до розробки додатків передбачає розділення на клієнтську (frontend) та серверну (backend) частини (рисунок 1.1) [11]. Клієнтська частина відповідає за взаємодію з користувачем, показуючи інтерфейс та отримуючи дані на пристрої користувача. Серверна частина, з іншого боку, складається з програм бізнес-логіки та компонентів для обробки та управління даними, забезпечуючи опрацювання ресурсів даних, таких як користувачі, геолокація, файли, медіа-потоки, підписки на повідомлення та інше [11].

Важливим є відокремлення бізнес-логіки від компонентів обробки даних. Це дозволяє створювати універсальні компоненти, які можуть бути використані в будь-якому клієнт-серверному додатку, не залежно від специфічних бізнес-цілей або предметної області застосування. Таким чином, функціональність, що пов'язана, наприклад, із роботою із базою даних, користувацькою авторизацією,

завантаженням файлів, відправкою push-повідомлень, залишається стандартною і може бути легко інтегрованою в будь-який новий проект. В результаті, ці функції можуть бути упаковані у вигляді повторно використовуваних послуг, а також шляхом об'єднання їх разом можна використовувати Baas [12]. У випадку коли такий сервіс використовується для надання послуг виключно мобільним пристроям, його називають mobile BaaS(mBaaS).

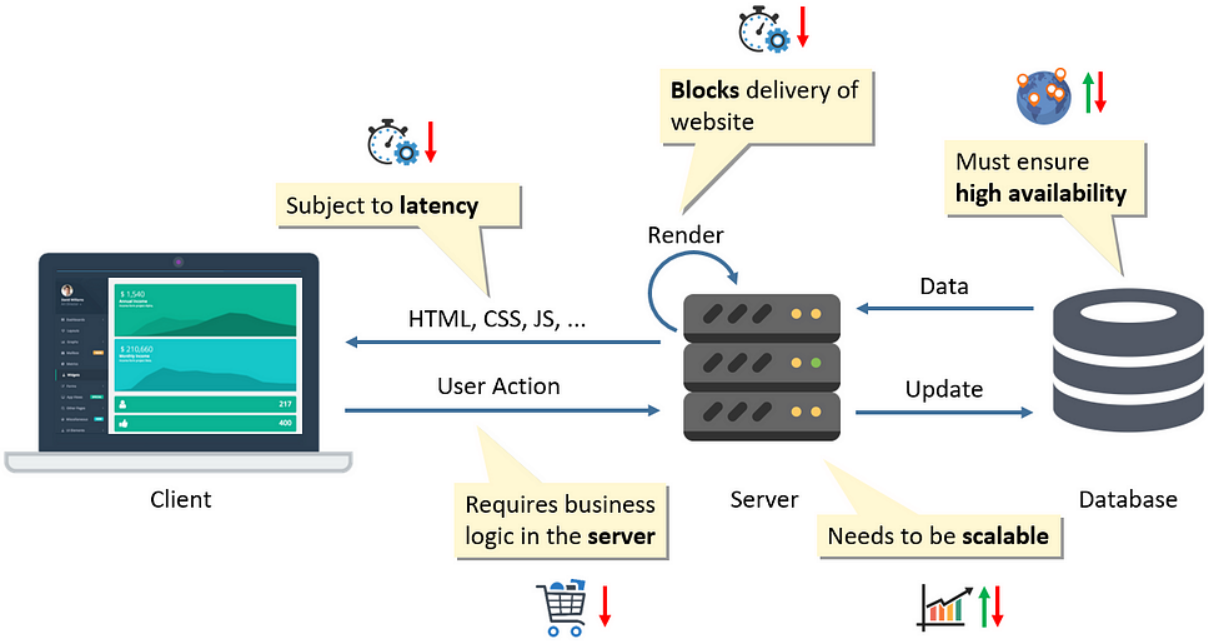


Рисунок 1.1 – Класична 3-рівнева архітектура

При розробці BaaS платформи для мобільних та веб-застосувань, слід уважно розглядати всі ці аспекти та їх взаємозв'язки. Враховуючи складність та множинність задач, які BaaS покликаний вирішувати, варто пам’ятати про важливість універсальності розроблених рішень, а також про можливість інтеграції з існуючими рішеннями на ринку [11].

Будь-який BaaS зазвичай складається з трьох рівнів. (рисунок 1.2)

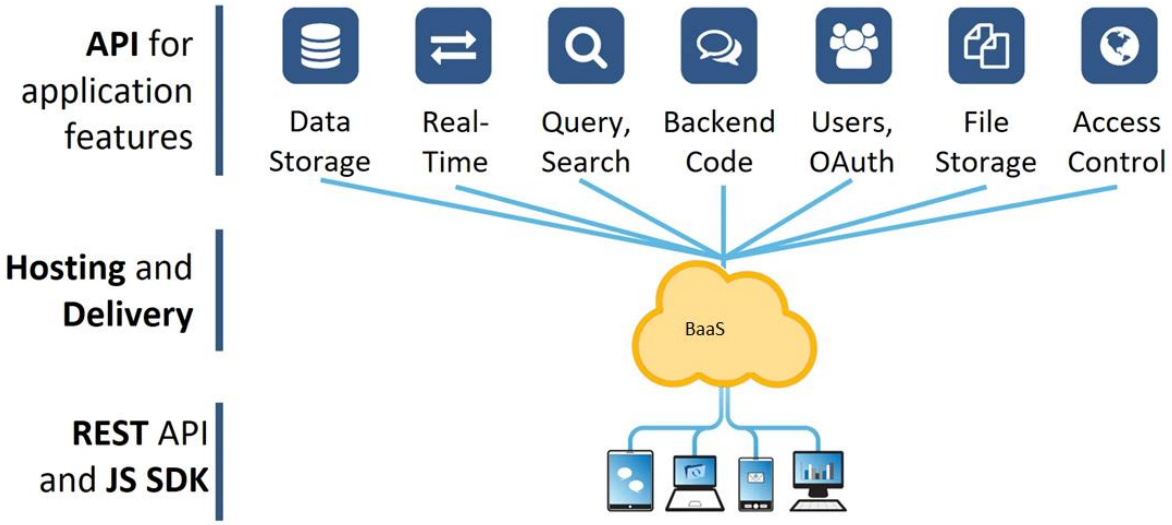


Рисунок 1.2 – Компоненти платформи Backend-as-a-Service.

API – це те, що робить BaaS корисним, оскільки він дозволяє зберігати та запитувати дані, запускати код на стороні сервера, автентифікувати користувачів тощо. Зазвичай це реалізується як багатокористувацький хмарний сервіс у постачальника інфраструктури як послуги (IaaS), наприклад AWS.

BaaS також піклується про хостинг і доставку . У випадку веб-додатків це означає, що він зберігатиме ваші HTML-файли та інші ресурси та доставлятиме їх користувачам [14].

Розробники створюють свій веб-сайт або мобільний додаток за допомогою REST API або SDK.

Використання BaaS дозволяє прискорити процес розробки додатків, а саме шляхом того, що клієнтські програми використовують API-інтерфейси для управління обробкою даних, які обробляються на стороні сервера. Проте, серверні послуги є загальними, а всі додатки мають специфічні вимоги до бізнес-правил, тому виникає питання про те, де розмістити бізнес-логіку. Існує декілька архітектурних варіантів вирішення даної проблеми:

1. Консоль розробника – більшість BaaS платформ включають консоль для розробки. Вона використовується для конфігурації та управління сервісом, а також є інтерфейсом між розробником клієнтського застосування та платформою. За допомогою консолі розробник може працювати з даними додатка, встановлювати політики безпеки, а також налаштувати деякі бізнес-правила.
2. Код користувача на стороні сервера – можливість виконання довільного коду користувача на сервері є найбільш гнучким рішенням, оскільки дозволяє додати в BaaS спеціалізований рівень логіки. Наприклад, певна бізнес-логіка, яка виконується кожен раз при вході користувача в систему, які відповідають певним унікальним сценаріям використання.
3. Клієнтська бізнес-логіка – основна ідея даного підходу в тому, що розробники додають спеціалізовану бізнес-логіку безпосередньо в клієнтський компонент. Виконання тих чи інших бізнес-правил відбувається за допомогою перевірки деяких спеціальних умов моделі даних у програмі.

Застосування BaaS має ряд переваг і недоліків. Розглянемо деякі з них.

Переваги:

1. Економія часу. Оскільки налаштування BaaS займає всього кілька хвилин, в той час як створення власного бекенду може зайняти кілька годин або навіть днів.
2. Економія коштів. BaaS усуває необхідність наймати додатковий персонал для розробки серверної частини.
3. Набір функціоналу з коробки. Клієнти забезпечуються основним набором мобільних функцій, такі як підтримка геолокаційними даними, інтеграція з соціальними мережами та сервісами, Push-повідомлення, зберігання даних, управління користувачами, аналітика, управління версіями та зберігання файлів.

4. Збільшення продуктивності бізнесу. Користувачі піклуються про реальний продукт, а не про розробку серверної частини. BaaS дозволяє розробникам зосередитися на тому, що важливо для їх бізнес моделі.

Незалежність протоколу даних. REST підтримка API через єдиний інтерфейс додає універсальності для використання в різних мобільних та web-платформах.

Недоліки:

1. Клієнти не мають повного контролю за інфраструктурою, яка управляється безпосередньо постачальником послуги, а клієнт контролює тільки дані і використовувані функції.
2. Зазвичай BaaS платформи базуються в одному певному центрі обробки даних або хмарної платформи, що може призводити до певної затримки у відповіді сервера, особливо якщо користувачі програми знаходяться на іншому боці земної кулі.
3. Постачальник BaaS послуг може вийти з бізнесу, що може вплинути на бізнес застосування, які використовують його ресурси.
4. BaaS прекрасно масштабується в співвідношенні ціна-якість для невеликих бізнес-моделей та стартапів, проте це не найкращий варіант для масштабного проекту з великою базою користувачів. Тому, із зростанням останніх, необхідно планувати створення власного серверного компоненту.
5. Неможливість мігрувати проект від одного BaaS провайдера до іншого або хмарної платформі третьої сторони.

1.2 Аналіз відмовостійких методів передачі повідомлень в розподілених системах

Розробка розподілених систем часто стикається із викликами, пов'язаними із забезпеченням надійності та стійкості під час передачі повідомлень між компонентами системи. Розглянемо декілька ключових стратегій та патернів, які захищають службу від збоїв низхідних залежностей [15].

1.2.1 Тайм-аути (Timeout)

Тайм-аути служать простим, але потужним методом виявлення та реагування на мережеві проблеми, зокрема зависання або відмови в роботі компонентів системи. Вони встановлюють граничний час, протягом якого клієнт готовий очікувати відповіді, тим самим, перериваючи потенційно безкінечні або довготривалі запити і захищаючи від витоків ресурсів. Проте, обрання оптимального часу тайм-ауту вимагає розуміння нормальних мережевих умов і патернів відповіді сервісу.

1.2.2 Стратегія повтору спроб (Retry)

Коли тайм-аут або інша помилка спричиняє збій у передачі повідомлення, стратегія повтору спроб може бути використана для підвищення шансів на успішну передачу, обмежуючи при цьому ризик подальших відмов. Правильно розроблена стратегія повтору спроб з урахуванням експоненційної затримки та "jitter" може мінімізувати негативний вплив повторних спроб на обидва, відправника та отримувача.

1.2.3 Експоненційна затримка (Exponential backoff)

Експоненційна затримка і jitter допомагають розсіювати повторні спроби в часі, запобігаючи одночасним спробам від множини клієнтів і, таким чином, зменшуючи ризик подальших проблем з доступністю.

Щоб установити затримку між повторними спробами, ми можемо використати обмежену експоненціальну функцію, де затримка виходить шляхом множення

початкової тривалості відстрочки на константу, яка експоненційно збільшується після кожної спроби, до деякого максимального значення (обмеження)

$$delay = \min(cap, initial\_backoff \cdot 2^{attempt}), \quad (1)$$

Де delay – час затримки,

initial_backoff- початкова тривалість відстрочки,

attempt- номер спроби

Наприклад, якщо обмеження встановлено на 8 секунд, а початкова тривалість відстрочки становить 2 секунди, тоді перша затримка повтору становитиме 2 секунди, друга – 4 секунди, третя – 8 секунд, і будь-яка подальша затримка буде обмежена до 8 секунд (1). Незважаючи на те, що експоненціальна затримка дійсно зменшує тиск на низхідну залежність, вона все ще має проблему. Коли низхідна служба тимчасово знижується, кілька клієнтів, ймовірно, побачать помилку своїх запитів приблизно в один і той же час. Це призведе до того, що клієнти будуть повторювати спроби одночасно, вражаючи низхідну службу через стрибки навантаження, які ще більше погіршують її, як показано на рисунок 1.3.

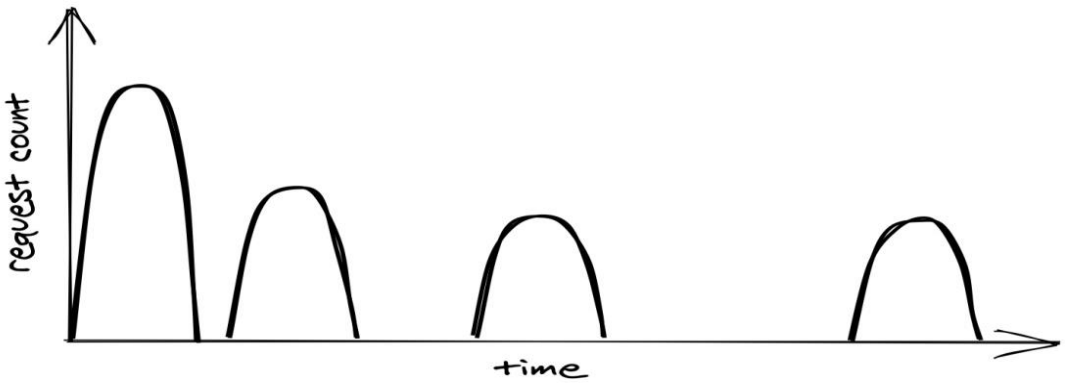


Рисунок 1.3 – Шторм повторів (Retry Storm)

1.2.4 Посилення спроби повтору (Retry Amplification)[16]

Коли повторення спроб відбуваються на кількох рівнях в ланцюгу залежностей, ефект "Retry Amplification" може призвести до збільшення навантаження на всю мережу. Це особливо проблематично в ситуаціях, коли сервіси, які вже переживають труднощі, стають предметом подальших спроб з’єднань від різних вузлів.

Припустимо, що обробка запиту користувача вимагає проходження ланцюжка з трьох сервісів. Клієнт користувача викликає службу A, яка викликає службу B, а та, у свою чергу, викликає службу C. Якщо проміжний запит від служби B до служби C не вдається, B повторити запит чи ні? Що ж, якщо B повторить спробу, A відчує довший час виконання для свого запиту, що підвищує ймовірність того, що він досягне тайм-ауту A. Якщо це станеться, A повторить запит, підвищуючи ймовірність того, що клієнт вичерпає час очікування та повторить спробу (рисунок 1.4).

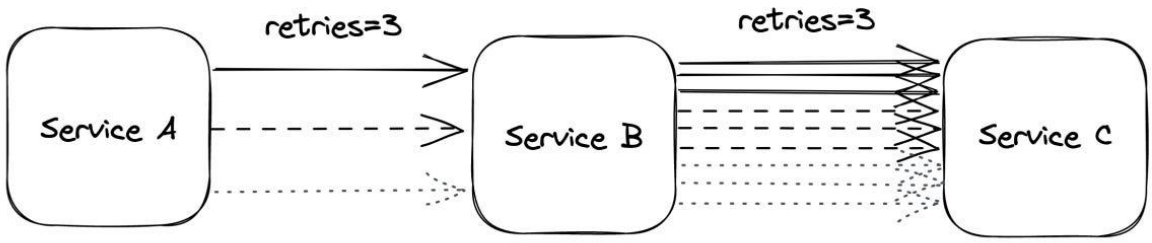


Рисунок 1.4 – Посилення спроби повтору *в дії*

1.2.5 Автоматичний вимикач (Circuit Breaker Pattern)[14]

Патерн "Circuit Breaker" служить захистом від задовгих або безкінечних намагань з’єднань, дозволяючи системі швидко "помилитися" та спробувати інший

підхід або підтримувати обмежене функціонування, коли зовнішній сервіс або залежність є недоступною.

1.2.6 Точки контролю і логіка компенсації (Checkpointing and Compensation Logic)

Стратегії відновлення даних, такі як точки контролю та логіка компенсації, дозволяють системам відновлювати операції після збою, забезпечуючи консистентність та стабільність.

1.2.7 Очікування та Ліміти (Timeouts and Limits)

Встановленням лімітів на ресурси, такі як пам'ять, мережа, і процесорне навантаження, і за допомогою очікувань, можна вплинути на якість обслуговування та захистити ресурси від перенавантаження.

1.3 Порівняльний аналіз аналогів

На ринку сучасних хмарних сервісів існує декілька потужних BaaS платформ, які в тій чи іншій мірі дозволяють вирішувати бізнес завдання предметної області. Розглянемо особливості провідних BaaS постачальників на ринку, на прикладі Firebase, AWS Amplify та Parse платформ.

Firebase — це платформа BaaS, яка підтримується Google та спеціалізується на синхронізації даних в реальному часі і допомагає розробникам створювати та надсилати програми для Android, iOS і Інтернету (рисунок. 1.5). Він був заснований у 2011 році з Realtime Database як його перший продукт. За ці роки він значно розвинувся і сьогодні пропонує широкий вибір інструментів, таких як автентифікація, база даних у реальному часі, хостинг, хмарні функції, хмарне сховище, хмарне сховище, аналітика та багато іншого. Основною перевагою є можливість завжди отримувати оновлених даних. Крім того, такий підхід усуває проблеми із затримкою оновлення, які є у інших постачальників. Ще однією сильною стороною Firebase є налаштування конфігурації безпеки системи при взаємодії з іншими соціальними сервісами. Для розробки Firebase надає всі

необхідні SDK, які покривають потреби в створені більшості сучасних мобільних та Web-застосувань: JavaScript (як веб-клієнти так і Node.js), Objective-C (IOS) і Java (Android) або HTTP REST API, також Firebase надає офіційну підтримку інтеграції з AngularJS, Backbone.js і EmberJS [17].

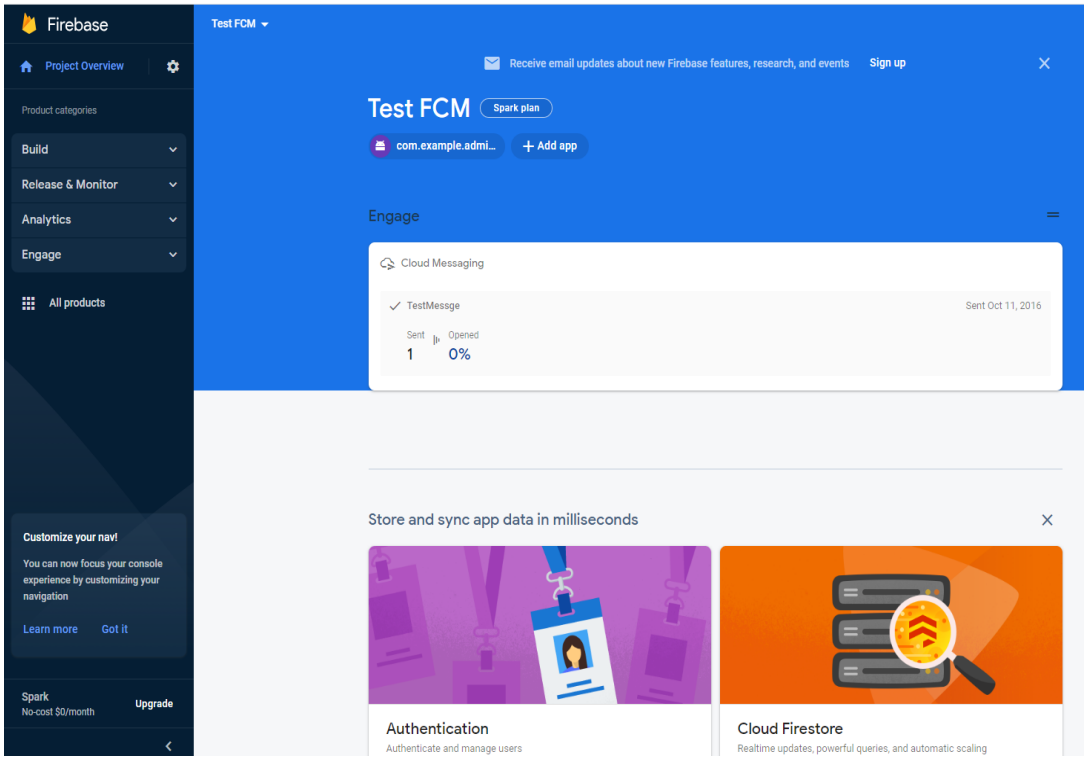


Рисунок 1.5 – Головна сторінка проекту Firebase

AWS Amplify — це платформа BaaS від Amazon, яка допомагає розробникам швидко створювати повний стек програм на AWS (рисунок. 1.5). Щоб зробити це можливим, Amplify надає набір інструментів, таких як Amplify Libraries , Amplify UI Components, Amplify Studio, Amplify CLI toolchain і Amplify Hosting.

За допомогою Amplify можна налаштувати бекенд веб-програми або мобільного додатка, підключити свою програму за лічені хвилини, візуально створити інтерфейс веб-інтерфейсу та легко керувати вмістом програми за межами консолі AWS. Доставляйте швидше та легко масштабуйте — без жодних знань у хмарі [18].

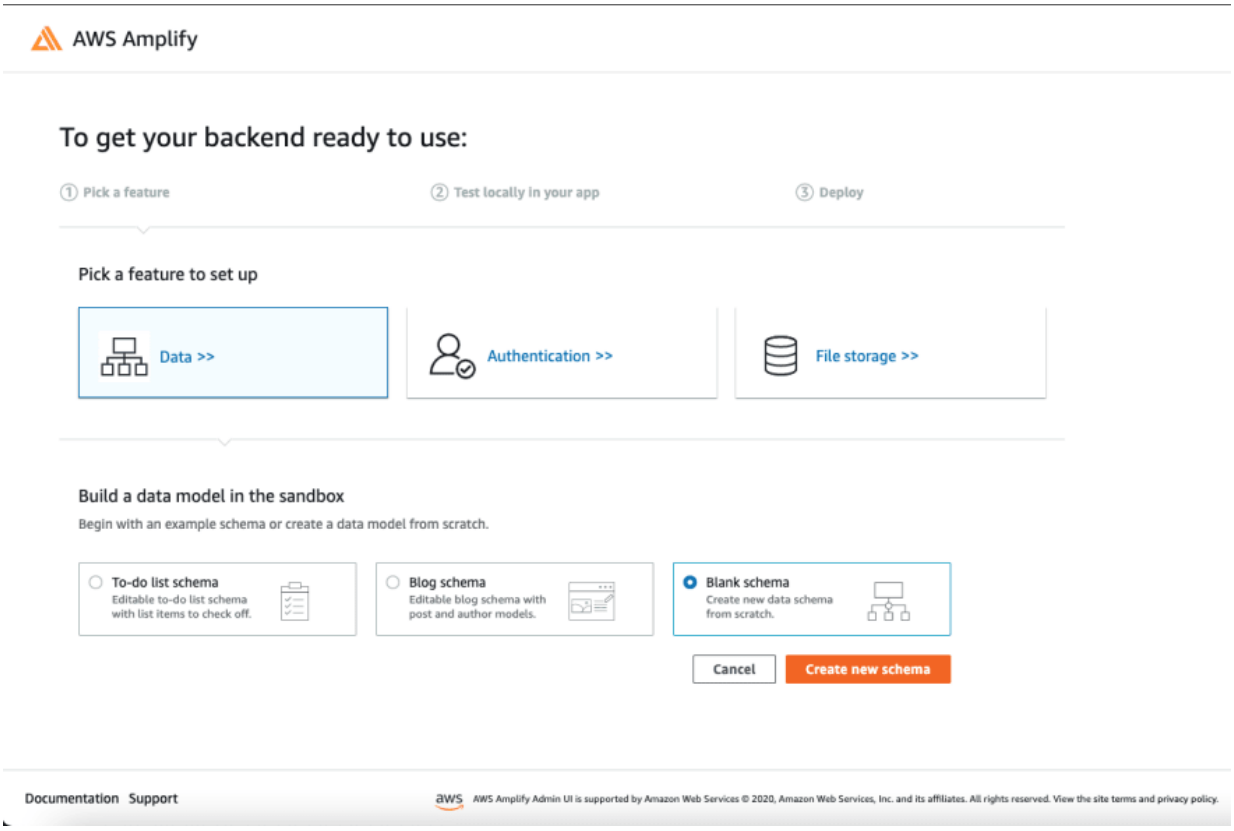


Рисунок 1.6 – Головна сторінка проекту AWS Amplify

Parse є однією з провідних серверних платформ як сервісних платформ у сучасному світі. Він був придбаний Facebook (тепер Meta) у 2013 році, а пізніше у 2016 році став відкритим кодом [19].

Його можна розгорнути в будь-якій інфраструктурі, яка підтримує Node.js і використовує MongoDB або Postgres як базу даних (рисунок. 1.7). Сервер аналізу можна налаштувати локально для розробки та розгорнути його пізніше в таких службах, як AWS, Heroku, Digital Ocean, Microsoft Azure тощо.

У Parse є величезна та активна спільнота з відкритим кодом і SDK для створення програм для будь-якої платформи, будь то веб, мобільний пристрій, настільний комп'ютер чи Інтернет речей. Такі компанії, як Prattle, Pinpo Tech, Rydite, Casperise тощо, використовують сервер Parse у своїх стеках технологій.

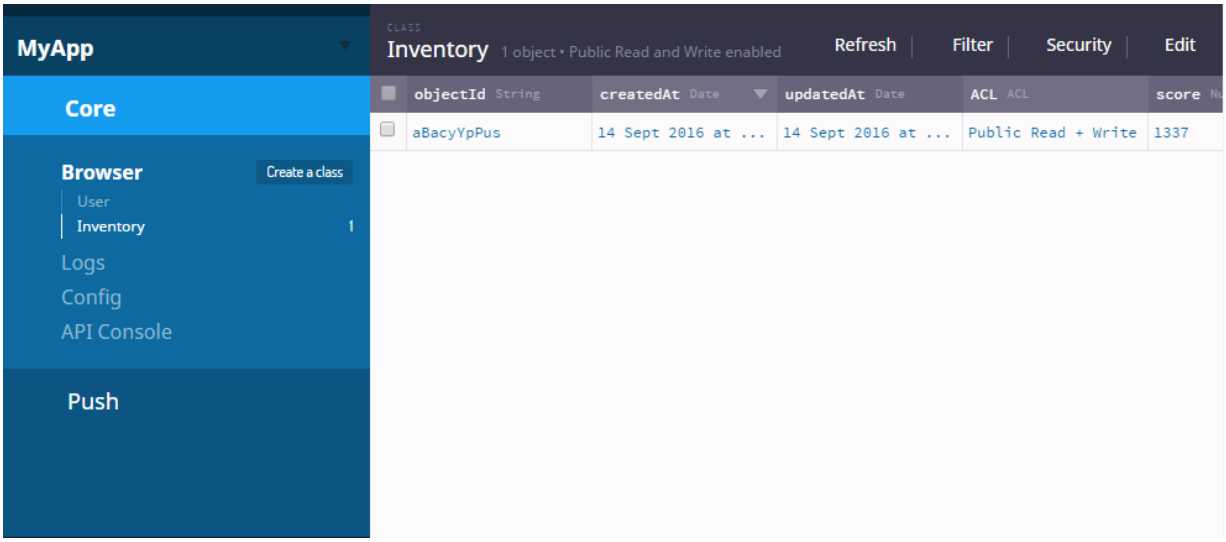


Рисунок 1.7 – Головна сторінка проекту Parse Dashboard

Провівши аналіз існуючих рішень на ринку можна виділити основні критерії для порівняння з розроблюваної системою. Результати порівняльного аналізу занесено в таблицю 1.1.

Таблиця 1.1 – Порівняльні характеристики програмних продуктів

Критерій	Firebase	Amplify	Parse	Власна реалізація
Адаптивний алгоритм для повторних спроб передачі повідомлень	-	-	-	+
Динамічне налаштування в залежності від навантаження системи	+	+	-	+
Підтримка різних хостинг платформ	-	-	+	+
Відкрите ПЗ	-	-	+	+
Вартість	-	-	+	+
Кросплатформеність	+	+	+	-
Масштабованість	+	+	-	-
Результат	3	3	4	5

Отже, виходячи з отриманих даних, можна зробити висновок, що розробка BaaS-платформи з динамічним визначенням оптимальних інтервалів між спробами повторної передачі є доцільною. В результаті отримаємо систему, що покриває недоліки існуючих рішень та забезпечує кращу відмовостійкість у порівнянні з аналогами.

1.4 Постановка задачі та обґрунтування актуальності дослідження

У сучасному світі інформаційних технологій, концепція розподілених систем грає вирішальну роль у багатьох сферах. Особливо це стосується платформ Backend as a Service (BaaS), які вимагають високої надійності, масштабованості та ефективності управління даними. Проте виникають складнощі, пов'язані з передачею повідомлень у таких розподілених системах, які включають проблеми надмірного навантаження на мережу, великі затрати енергії, і збої у передачі даних через нестабільність мережі.

Актуальність даного дослідження обумовлена необхідністю розробки інтелектуальних алгоритмів, які адаптують стратегії повторної передачі повідомлень на основі контексту мережі та характеристик даних. Такий підхід дозволить знизити навантаження на мережу та споживання енергії. Особлива увага приділяється механізмам, що динамічно визначають оптимальні інтервали між спробами повторної передачі, базуючись на реальному стані мережі та поведінці користувачів у реальному часі. Це включає в себе дослідження таких аспектів як стратегії повторення спроб (Retry), Посилення спроби повтору (Retry Amplification), використання таймаутів, принципів побудови схеми обробки повідомлень і захисту від збоїв, зокрема за допомогою патерну "Circuit Breaker" та ін.

Ця задача вимагає глибокого аналізу існуючих методів передачі повідомлень у розподілених системах, з метою виявлення їхніх слабких місць та можливостей для оптимізації. Розробка нових адаптивних методів передачі повідомлень потребує

інтеграції знань з областей розподілених систем, штучного інтелекту та машинного навчання.

Таким чином, основними задачами цього дослідження є:

- Аналіз та оцінка існуючих методів і технік передачі повідомлень в розподілених системах.
- Розробка інтелектуальних алгоритмів для адаптації стратегій повторної передачі повідомлень.
- Впровадження адаптивних механізмів, що динамічно визначають оптимальні інтервали між спробами повторної передачі.
- Тестування та перевірка запропонованих методів у розроблювальній BaaS-платформі з подальшою можливістю використання в реальних умовах розподілених систем.

Успішне вирішення цих задач не тільки підвищить ефективність та надійність передачі повідомлень в розподілених системах, але й відкриє нові можливості для оптимізації роботи BaaS платформ, поліпшуючи якість сервісу для кінцевих користувачів. Також, результати даного дослідження можуть бути використані для підвищення ефективності існуючих та розробки нових розподілених систем через оптимізацію стратегій обробки та передачі повідомлень, підвищення їхньої надійності, стійкості та безпеки. Це, в свою чергу, може сприяти покращенню якості сервісів, забезпеченню високої доступності та захисту даних користувачів в умовах розподілених обчислень.

1.5 Висновки

У розділі 1 було проведено детальний аналіз сучасного стану методів передачі повідомлень у розподілених системах. Було виявлено, що існуючі методики все ще мають ряд обмежень, особливо щодо ефективності використання мережевих ресурсів та енергії.

У контексті розробки платформи Backend as a Service (BaaS), це відкриває значні можливості для інновацій. Інтеграція інтелектуальних алгоритмів, які здатні адаптувати стратегії повторної передачі повідомлень на основі контексту мережі та характеристик даних, є критично важливою для підвищення загальної продуктивності системи та забезпечення високої якості обслуговування користувачів.

Особливо актуальним є впровадження адаптивних механізмів, що динамічно визначають оптимальні інтервали між спробами повторної передачі, враховуючи реальний стан мережі та поведінку користувачів. Це дозволить не тільки оптимізувати використання мережевих ресурсів, але й значно зменшити споживання енергії.

У результаті, це дослідження підкреслює важливість продовження робіт у напрямку розробки новітніх методів передачі повідомлень в розподілених системах, які будуть базуватися на сучасних досягненнях у галузі штучного інтелекту та машинного навчання. Враховуючи швидкий розвиток цих технологій, можна передбачити, що майбутні інновації не тільки значно підвищать ефективність передачі даних, але й відкриють нові перспективи для розширення функціональних можливостей розподілених систем.

2 РОЗРОБКА МЕТОДУ ТА МОДЕЛІ АДАПТИВНОЇ СИСТЕМИ ІЗ ПОВТОРНОЮ ПЕРЕДАЧЕЮ ПОВІДОМЛЕНЬ У BAAS

2.1 Розробка адаптивного методу повторної передачі повідомлень

2.1.1 Аналіз теорії керування для розробки адаптивних методу передачі повідомлень

Сучасні методи у сфері програмного забезпечення включають різноманітні підходи для розробки адаптивного програмного забезпечення. Ці підходи базуються на принципах машинного навчання та теорії керування [20], що дозволяє створювати програмне забезпечення, здатне самостійно контролювати та адаптувати свою поведінку для досягнення поставлених цілей.

Теорія керування - це галузь інженерії та математики, яка займається поведінкою динамічних систем. Основна мета - розробка моделей та алгоритмів для керування системами таким чином, щоб досягти бажаного стану або поведінки. У контексті інформаційних систем, таких як BaaS, теорія керування дозволяє розробити механізми для ефективного реагування на зміни в навантаженні, помилки системи, та інші зовнішні впливи.

На відміну від машинного навчання, яке часто вимагає великих даних для тренування моделей та може бути чутливим до змін у вхідних даних, теорія керування пропонує математично обґрунтоване та стабільне рішення для адаптації систем у реальному часі. Це особливо важливо для критичних систем, де непередбачуваність поведінки алгоритмів машинного навчання може призвести до небажаних наслідків.

Самоадаптивне програмне забезпечення реалізується, як замкнута система із зворотним зв’язком (closed-loop system), спрямованою на адаптацію до змін під час роботи, для забезпечення непорушності заданих цілей [20]. Цією замкнутою

системою керує менеджер адаптації, який відповідає за обробку петлі зворотного зв’язку.

У програмній інженерії поширеним дизайном петлі зворотного зв'язку є цикл "Monitor-Analyse-Plan-Execute" (MAPE), зі спільною базою знань (рисунок 2.1). Фаза моніторингу збирає та корелює дані з системи та її оточення; фаза аналізу аналізує дані моніторингу разом з історичними даними; фаза планування визначає, які зміни потрібні та оптимальний спосіб їх реалізації; фаза виконання застосовує визначені дії. Всі етапи використовують спільну базу знань, що призводить до акроніму MAPE-K.

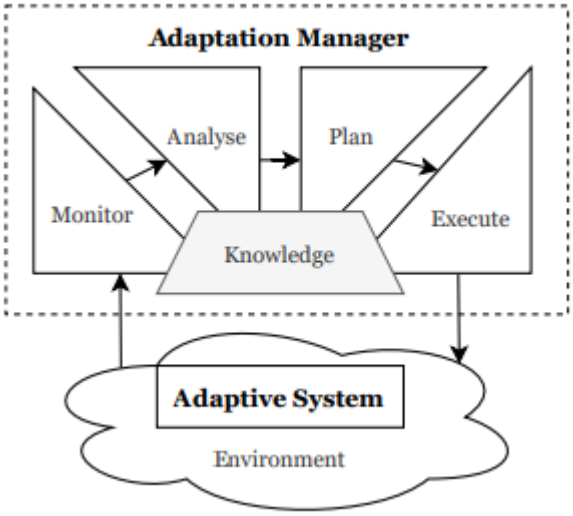


Рисунок 2.1 – Адаптивна система на основу циклу MAPE-K

У теорії керування самоадаптивна система реалізується за допомогою контрольної петлі, яка складається з об'єкта керування (тобто адаптивної системи) та контролера, як показано на рисунку 2.2. Інтуїтивно, враховуючи дискретизований крок часу k, контролер використовує помилку e(k) між вихідними даними об'єкта $\bar{y}(k)$ та цілями об'єкта y(k), щоб вирішити, як змінити u(k) системи. Контрольні петлі також можуть мати справу із порушеннями, позначеними як d(k), які є зовнішніми факторами, що можуть вплинути на вихідні дані.

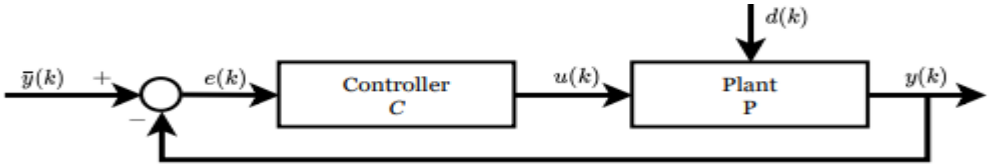


Рисунок 2.2 – Контрольна петля в адаптивній системі

Проектування стратегії керування зворотним зв'язком для існуючої програмної системи вимагає дотримання певних кроків. Нижче наведено деталізований процес розробки самоадаптивної системи заснованими на теорії керування:

1. Визначення цілей системи. Початковим етапом є встановлення чітких цілей для системи, які можуть включати визначення ключових показників ефективності, значень або параметрів, які потрібно мінімізувати чи максимізувати.
2. Визначення параметрів керування. Наступний крок полягає у визначенні змінних управління, які можуть бути використані для модифікації поведінки програмного забезпечення. Це можуть бути параметри, такі як розподіл ресурсів або налаштування конфігурацій.
3. Розробка моделі об'єкта керування. Після визначення параметрів необхідно розробити модель об'єкта керування. Модель має відображати взаємозв'язок між цілями системи та впливом ручок керування.
4. Проектування Контролера. На цьому етапі проектується контролер, який буде реалізовувати стратегію управління на основі моделі об'єкта. Контролер може бути реалізований різними способами, включаючи часові або на знаннях контролери, залежно від специфіки системи.
5. Реалізація та перевірка контролера. Після проектування контролер реалізується та інтегрується з системою. На завершальному етапі система тестується для переконання у відповідності встановленим цілям та ефективності керування.

2.1.2 Розробка адаптивного алгоритму повторної передачі повідомлень

Для створення адаптивного алгоритму повторної передачі повідомлень у BaaS, спочатку розробимо точну модель системи. Ця модель повинна відображати динамічні властивості системи, такі як час відповіді на запити, помилки запитів, та частоту повторних спроб.

Використовуємо лінійну модель стану для опису поведінки BaaS- платформи, де кожен компонент стану x(t) включає час відповіді на запит, невдачі запитів та частоту повторних спроб (2.1).

$$\dot{x}(t) = Ax(t) + Bu(t), \tag{2.1}$$

де $x(t)$ – стан системи (наприклад, загальну продуктивність, час відповіді, тощо); $u(t)$ – вхідні параметри системи (наприклад, кількість активних запитів в системи); A та B є матрицями системи, що визначають взаємодію між станами на входами.

Наступним кроком необхідно визначити модель даних про стан системи, які необхідні для подальшого аналізу і впровадження у алгоритм для оптимізації процесу повторної передачі повідомлень. У контексті аналітичної служби BaaS-платформи будемо збирати наступні метрики:

- Час відповіді (response_time). Цей показник відображає, скільки часу проходить від моменту відправки запиту до моменту отримання відповіді.
- Рівень невдалих запитів *(*FR) - відсоток невдалих запитів від загальної кількості запитів за певний період часу. Щоб розрахувати рівень невдалих запитів, необхідно кількість невдалих запитів (наприклад ті, що повертають помилку сервера) N_{failed} та поділити її на загальну кількість запитів за цей період часу N (2.2).
- Кількість повторних спроб ($number\_of\_retries$). Даний показник відображає скільки разів система спробувала повторити запит після його передачі. Для розрахунку кількості повторних спроб, необхідно

простежити кожен запит та порахувати, скільки разів він був повторений до успішної обробки або остаточної відмови.

$$FR(t) = \frac{N_{failed}}{N} \cdot 100\%, \quad (2.2)$$

Отже, визначаємо множину даних M(t), зібраних у часі, необхідних для аналізу і визначенні стратегії, на основі яких алгоритм буде адаптуватися у випадку помилок при передачі повідомлень у систему (2.3). Ці дані будуть зібрані та збережені у часових рядах у відповідній базі даних (Time Series DB).

$$M(t) = \{response\_time, FR, number\_of\_retries\}, \quad (2.3)$$

Наступним кроком визначаємо контролер, який буде визначати стратегію управління. Для розробки алгоритму обираємо ПІД-контролер (пропорційно-інтегрально-диференціальний контролер), який використовується як частина алгоритму в адаптивних системах для оптимізації відповіді на змінні умови середовища. ПІД-контролери є популярною часовою технікою управління. Вони не потребують явної моделі: натомість, їх можна налаштувати на основі експериментування та евристик [21]. Використання ПІД-контролера дозволяє визначати оптимальної кількості повторних спроб на основі поточного стану системи та використовуючи різницю між поточним станом системи та бажаним станом. З таким підходом вхідні параметри системи обчислюється наступним чином:

$$u(t) = u(t-1) + K_p e(t) + K_i \int_0^t e(\tau)d\tau + K_d \frac{de(t)}{dt}, \quad (2.4)$$

де $e(t)$ - різниця між поточним і цільовим станом (наприклад, час відповіді на запит), а Kp, Ki і Kd - налаштування ПІД -контролера.

Розрізняються наступні основних компоненти регулятора:

- Пропорційний компонент Kp пропорційний помилці між заданим значенням та фактичним значенням, генеруючи коригувальну відповідь при наявності помилки.
- Інтегральний компонент Ki інтегрує минулі значення помилки, прискорюючи збіжність до заданого значення та зменшуючи сталу помилку від виключно використання пропорційного компоненту, але може викликати перевищення фактичного значення над заданим.
- Диференційний компонент Kd використовує нахил помилки з часом для поліпшення часу стабілізації та зменшення перевищення.

Варто зазначити, що інтегральний та диференційний компоненти можуть бути виключені[22], якщо інші компоненти достатні для того, щоб контролер досяг заданого значення з мінімальним перевищенням і коливанням.

Отже, загальний алгоритм буде складатися з наступних кроків:

1. Збір даних та метрик (M(t)). Система збирає дані про час відповіді, невдачі запитів та частоту повторних спроб.
2. Аналіз даних. Використання статистичних методів для виявлення шаблонів у зібраних метриках.
3. Визначення параметрів контролера. На основі аналізу даних, налаштовуємо параметри ПІД-контролера для оптимізації поведінки системи.
4. Впровадження змін. Система вносить зміни у поведінку повторних спроб згідно з вказівками контролера.
5. Зворотний зв'язок. Система постійно моніторить свій стан та корегує дії згідно з отриманими даними для підтримання бажаного рівня продуктивності.

2.2 Розробка методу аналізу даних передачі повідомлень в BaaS-платформі на основі часових рядів

Зібрані дані та метрики аналізуються для виявлення тенденцій, проблем або можливостей для оптимізації. Для цього використовується методи аналіз часових рядів. Часовий ряд — це серія точок даних, зібраних за певний час. Метрики, зібрані протягом часу в BaaS-платформі, можуть бути представлені, як часовий ряд, а потім аналізуються за допомогою статистичних підходів.

Для даного процесу можуть використовуватися методи статистичного аналізу, такі як лінійна регресія (Least Squares Estimation) або авторегресійне моделювання (AR). Розглянемо їх детальніше та визначимо метод, який найкраще підходить для адаптивного алгоритму в BaaS-платформі.

Метод найменших квадратів використовується при наявності декількох прогнозованих значень (наприклад, набір параметрів, спостережуваних у часі). Метод підбирає коефіцієнти для цих параметрів, щоб мінімізувати суму квадратів помилок для прогнозу (2.5) [23]. Коефіцієнти $\beta_0, \beta_1, \ldots, \beta_k$ для параметрів $x_0, x_1, \ldots, x_k$ вибираються для мінімізації, де нижній індекс t вказує на значення в часовому ряду, індексованому від t = 1 до T.

$$\sum_{t=1}^{T} \varepsilon_t^2 = \sum_{t=1}^{T} (y_t - \beta_0 - \beta_1 x_{1,t} - \beta_2 x_{2,t} - \cdots - \beta_k x_{k,t})^2, \quad (2.5)$$

Авторегресивне моделювання прогнозує змінну на основі лінійної комбінації попередніх значень цієї змінної. Авторегресивне моделювання може захоплювати патерни в часових рядах, наприклад, як сплеск використання ЦПУ викликаний попередніми сплесками. Авторегресивна модель порядку p, AR(p), визначається за формулою 2.6, де $\varphi_1, \varphi_2, \ldots, \varphi_p$ - параметри моделі, c - стала, а ϵ_t - білим шумом[24].

$$X_t = c + \sum_p \varphi_i X_{t-i} + \epsilon_t \quad (2.6)$$

ARIMA - це інший підхід до прогнозування часових рядів, де авторегресивне моделювання комбінується з моделюванням ковзного середнього, який схожий на

AR(p), але використовує минулі помилки прогнозу замість минулих значень прогнозу.

Для аналізу даних M(t), що включають час відповіді, невдачі запитів та кількість повторних спроб у BaaS, оптимальним вибором буде авторегресійне (AR) моделювання. Це пояснюється тим, що дані M(t) являють собою часовий ряд, де минулі події (наприклад, невдачі запитів) можуть впливати на майбутні (наприклад, потребу в повторних спробах).

У контексті BaaS- платформи, аналіз даних дозволяє системі більш інтелектуально управляти повторними спробами запитів. Наприклад, якщо аналіз показує, що в певні години доби невдачі запитів зростають через високе навантаження на сервери, система може автоматично налаштовувати інтервали між повторними спробами або вводити тимчасові обмеження для певних видів запитів, знижуючи тим самим загальне навантаження та покращуючи загальну продуктивність системи.

Отже, визначимо основні кроки алгоритму аналізу даних часових рядів за допомогою AR моделювання:

1. Збір Даних. Система збирає метрики M(t), які включають час відповіді на запити, частоту невдач запитів, та кількість повторних спроб. Дані зберігаються у вигляді часових рядів для кожної метрики окремо. Періодичність збору даних може бути налаштована в залежності від потреб системи та обсягу даних.
2. Перевірка на Стаціонарність. Застосування тестів на стаціонарність, таких як Дікі-Фуллера [25], для перевірки, чи є часовий ряд M(t) стаціонарним. Якщо дані не стаціонарні, застосовуються методи перетворення (наприклад, диференціювання), щоб зробити їх стаціонарними.
3. Вибір Порядку Моделі (p). Використання критеріїв вибору моделі, таких як критерій Акаіке (AIC) або Байесівський критерій (BIC) [26], для визначення оптимального порядку p для AR моделі. Оцінка різних

моделей з різними значеннями p та вибір найбільш адекватної моделі на основі статистичних показників.

4. Оцінка Параметрів Моделі. Застосування методів оцінювання параметрів, таких як метод найменших квадратів або максимальної вірогідності, для визначення значень параметрів $\phi 1$, $\phi 2$, ..., ϕp. Перевірка адекватності моделі, відбувається зокрема за допомогою аналізу залишків.
5. Прогнозування та Аналіз. Використання оціненої AR моделі для прогнозування майбутніх значень метрик M(t). Аналіз зв'язків між різними метриками, щоб виявити можливі причинно-наслідкові зв'язки. Використання результатів аналізу для адаптації поведінки системи, наприклад, для налаштування стратегії повторних спроб або для оптимізації відповідей серверів.

На основі аналізу може бути виявлено, що при певних умовах (наприклад, підвищеному навантаженні) система має тенденцію до збільшення часу відповіді чи кількості невдач. Відповідно до цих даних, можна регулювати параметри системи, такі як розмір черги запитів або інтервали між повторними спробами. Отримані результати використовуються у ПІД-контролері для динамічного регулювання цих параметрів на основі виявлених тенденцій, що допомагає підтримувати оптимальний рівень продуктивності та надійності системи.

2.3 Модель високорівневої адаптивної системи із повторною передачею повідомлень

Вибір мікросервісної архітектури для BaaS платформи обумовлений потребою у гнучкості, масштабованості та забезпеченні високої доступності сервісів. Використання мікросервісів дозволяє розділити систему на окремі незалежні компоненти, що спрощує процес розробки, тестування, впровадження змін та масштабування [27][28]. Кожен мікросервіс може бути оптимізований для

виконання певної функції, що забезпечує високу продуктивність і знижує ризики виникнення помилок у системі [29][30].

Основні компоненти мікросервісної архітектури наведені на рисунку 2.3 та складаються з наступних сервісів:

- API Gateway. Центральний вхідний вузол для всіх запитів до платформи BaaS. Він маршрутизує запити до відповідних мікросервісів, забезпечуючи безпеку, авторизацію та аутентифікацію.
- BaaS Core Services: Включає основні сервіси, такі як User Service, Database Service та File Storage Service. Ці сервіси надають фундаментальні функції BaaS, обробляючи запити від користувачів та інших сервісів.
- Metric Collection Service: Відповідає за збір метрик, таких як час відповіді, невдачі запитів, кількість повторних спроб. Цей сервіс є ключовим для аналізу роботи системи та її оптимізації.
- Analysis Service: Аналізує зібрані дані, виявляючи тенденції та закономірності, необхідні для адаптивного регулювання поведінки системи.
- Adaptive Retry Service: Управління логікою повторних спроб на основі даних від Analysis Service. Цей сервіс відповідає за динамічне налаштування параметрів повторних спроб, враховуючи поточний стан системи.
- Circuit Breaker Service: Моніторинг високих показників невдач запитів та реалізація механізму обмеження чи призупинення запитів для забезпечення стабільності системи.
- Feedback Loop Controller: Координує роботу всієї адаптивної системи, забезпечуючи зворотний зв'язок між різними компонентами та адаптацію до змін.

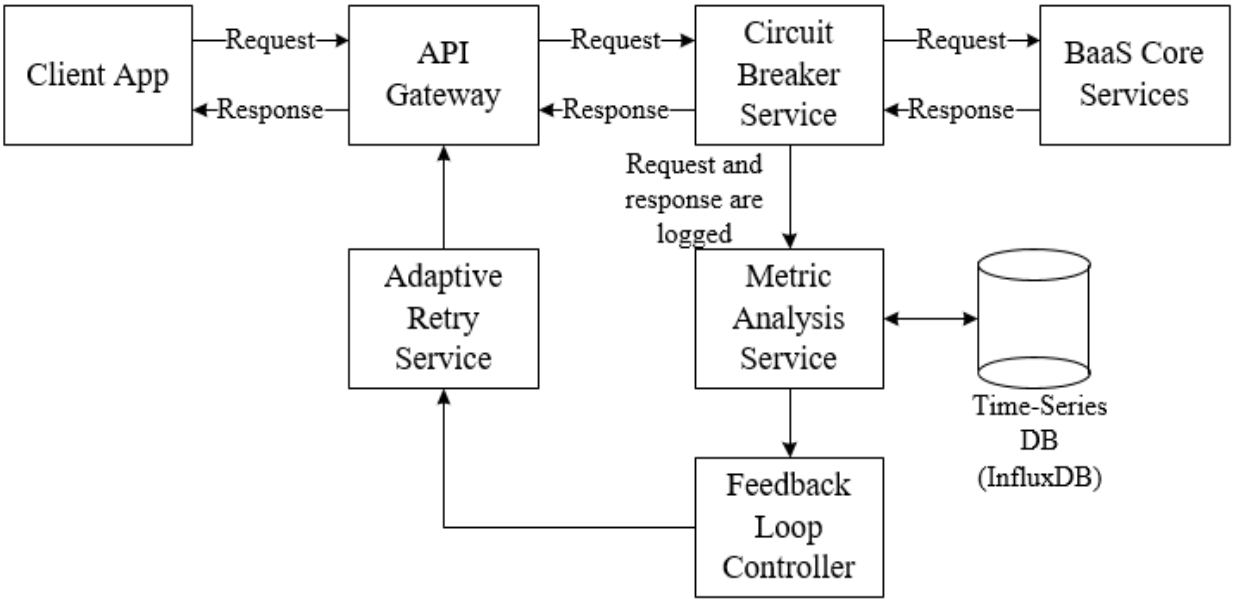


Рисунок 2.3 – Модель високорівневої адаптивної системи із повторною передачею повідомлень

ПІД-контролер (пропорційно-інтегрально-диференційний контролер) є частиною Feedback Loop Controller. Цей контролер відіграє важливу роль у регулюванні поведінки системи, особливо у контексті адаптації параметрів повторних спроб у Adaptive Retry Service. ПІД-контролер аналізує дані від Analysis Service, такі як відхилення часу відповіді від очікуваного рівня, і регулює поведінку системи, щоб мінімізувати це відхилення. Компоненти ПІД-контролера - пропорційний, інтегральний та диференційний - дозволяють точно налаштувати реакцію системи на різні типи відхилень.

Процес роботи системи зображений в діаграмі послідовності на рисунку 2.4. Основні запити системи складаються з наступних кроків:

1. Запити від користувачів. Всі запити спочатку надходять через API Gateway, який служить вхідним вузлом до платформи BaaS.
2. Перевірка Circuit Breaker Service. Перш ніж запити досягають BaaS Core Services, вони проходять через Circuit Breaker Service. Цей сервіс виконує

моніторинг і, за потреби, обмеження навантаження, перешкоджаючи перевантаженню системи.

3. Маршрутизація до BaaS Core Services. Запити, які проходять Circuit Breaker Service, направляються до відповідних сервісів у BaaS Core Services для обробки.
4. Обробка та збір метрик. Metric Collection Service збирає важливі метрики, такі як час відповіді, невдачі запитів, кількість повторних спроб, під час обробки запитів.
5. Аналіз даних у Analysis Service. Зібрані метрики передаються в Analysis Service, де вони аналізуються для виявлення закономірностей та потенційних проблем з використанням статистичних або машинних методів.
6. Адаптація стратегії повторних спроб. На основі отриманих даних з Analysis Service, Adaptive Retry Service адаптує параметри повторних спроб, оптимізуючи їх залежно від поточного стану системи.
7. Моніторинг та реагування Circuit Breaker Service. Circuit Breaker Service продовжує моніторинг загального стану системи і, у разі виявлення проблем з навантаженням або високого рівня помилок, вживає заходів для зменшення навантаження.
8. Зворотний зв'язок та адаптація через Feedback Loop Controller. Feedback Loop Controller, в якому інтегровано ПІД-контролер, забезпечує зворотний зв'язок між усіма компонентами системи, координуючи їх адаптацію для забезпечення стабільності та ефективності системи.

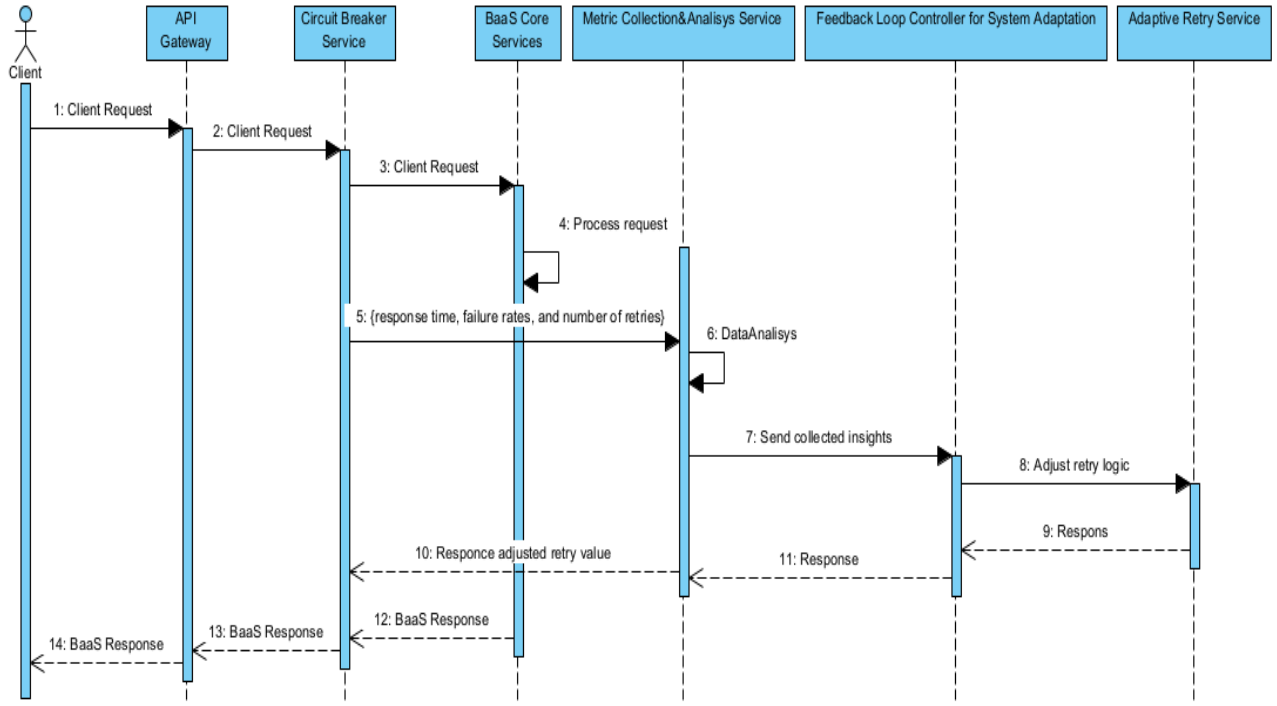


Рисунок 2.4 –Діаграма послідовності адаптивного алгоритму повторної передачі повідомлень

2.4 Розробка алгоритмів роботи системи

Першим етапом розробки системи є проектування загального адаптивного алгоритму повторної передачі повідомлень. Розроблювана система буде складатися з сукупності мікросервісів, де кожен мікросервіс функціонує як самостійний веб-сервіс, спеціалізований на конкретному аспекті загальної функціональності системи. Ключовою особливістю системи є модульність і незалежність її компонентів. Взаємодія між сервісами здійснюється через HTTP-запити, що дозволяє системі бути гнучкою та масштабованою. Кожен мікросервіс реагує на запити, обробляючи їх згідно із власною бізнес-логікою та відповідаючи на них відповідно до своїх функціональних обов'язків. Високорівневий алгоритм повторної передачі повідомлень зображено на рисунку 2.5, ілюструє інтеграцію мікросервісів в єдину координовану систему.

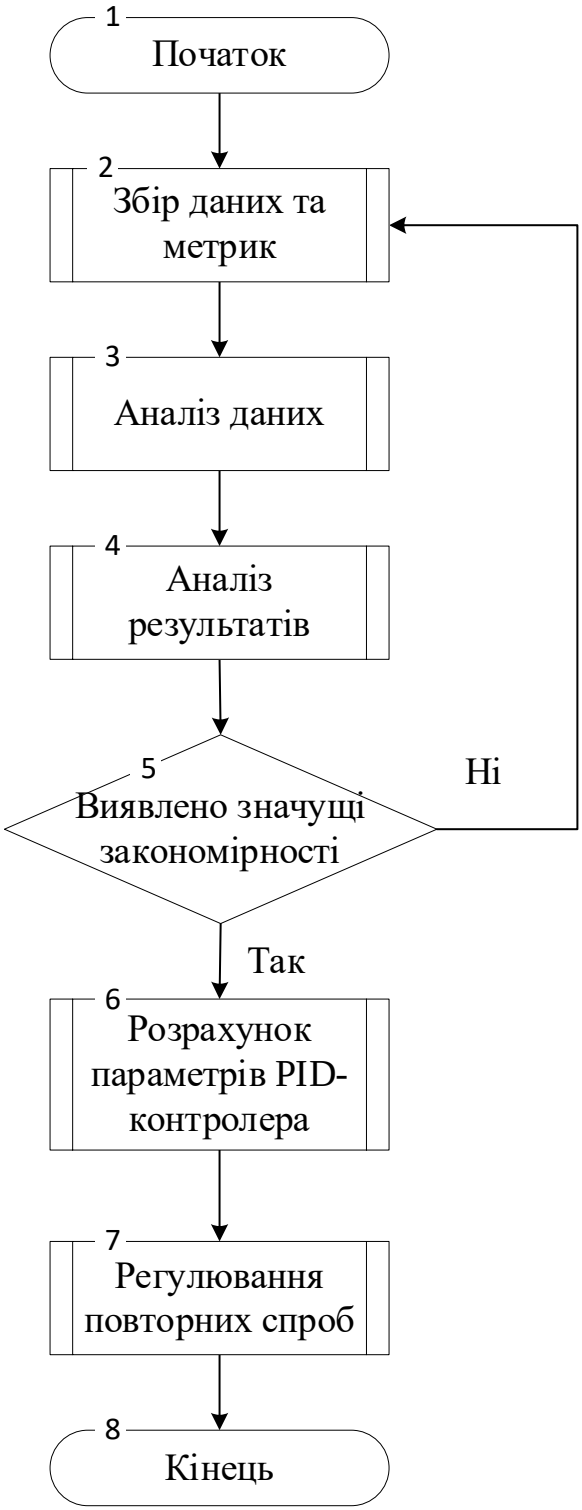


Рисунок 2.5 – Блок-схема адаптивного алгоритму повторної передачі повідомлень

На рисунку 2.6 зображено алгоритм аналізу часових даних на основі зібраних метрик.

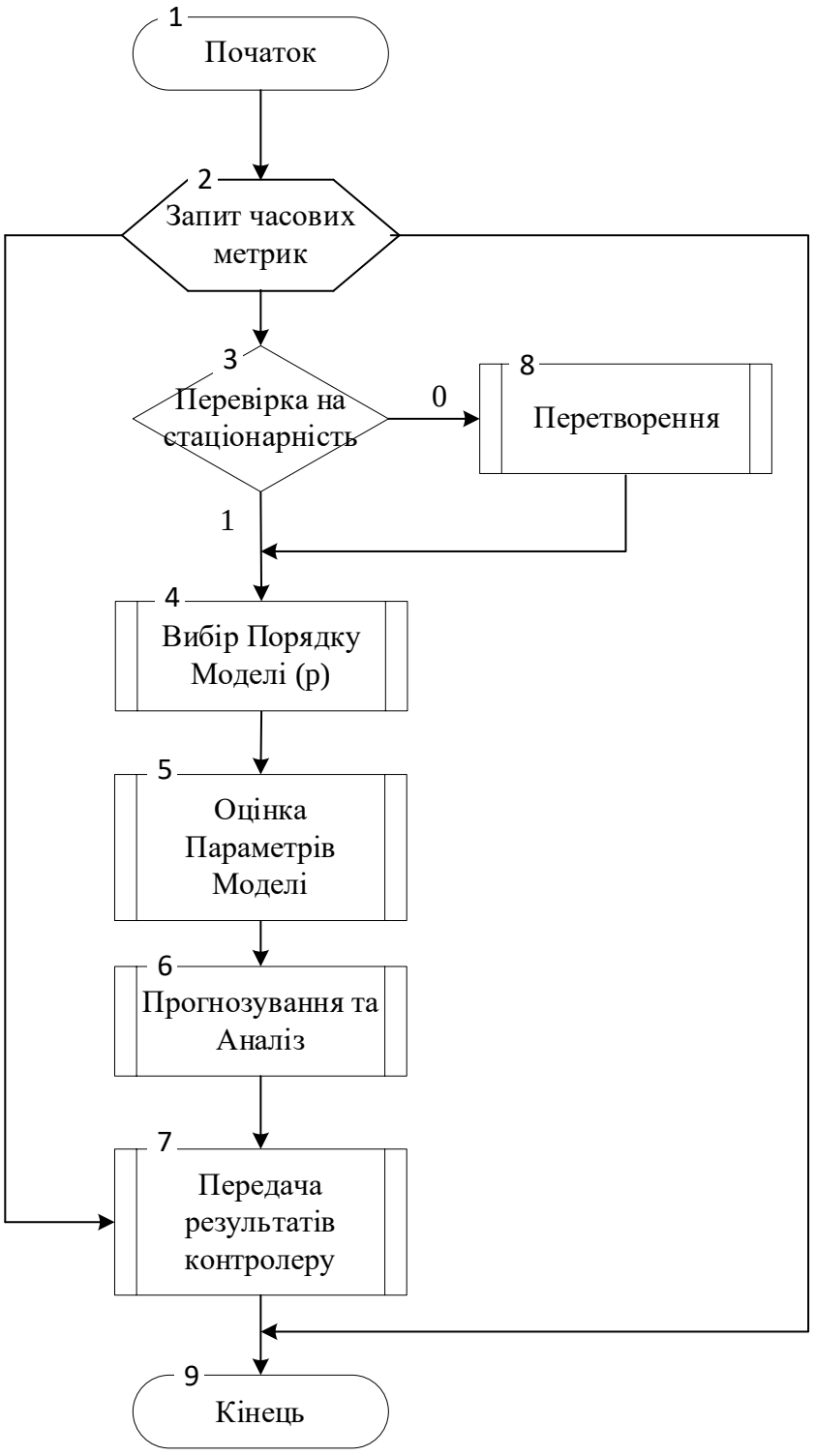


Рисунок 2.6 – Блок-схема алгоритму аналізу часових метрик

2.4 Висновки

У другому розділі було розроблено адаптивний алгоритму для повторних спроб у BaaS. Було визначено основних кроків алгоритму у вирішенні завдання оптимізації поведінки системи з урахуванням динамічних змін у мережі та поведінці користувачів. Основною перевагою запропонованого підходу є його здатність до самоналаштування та адаптації, заснована на контрольно-теоретичних принципах та статистичному аналізі даних.

Ключовим компонентом алгоритму є збір та аналіз метрик M(t), які включають час відповіді, частоту невдач запитів та кількість повторних спроб. Це дозволяє системі точно оцінювати свій поточний стан та ефективно реагувати на виклики.

Розроблено алгоритм аналізу даних, який дозволяє оцінювати та виявляти закономірності у зібраних метриках. Використання методів часових рядів, таких як модель авторегресії (AR), надає можливість прогнозувати майбутню поведінку системи та вчасно адаптувати стратегію повторних спроб.

Застосування ПІД-контролера забезпечує збалансовану регуляцію повторних спроб, де кожен компонент (пропорційний, інтегральний, диференціальний) відіграє свою роль у досягненні оптимального рішення.

Важливим висновком є те, що адаптивний алгоритм дозволяє BaaS системі не просто реагувати на проблеми, але й передбачати можливі виклики, адаптуючись до змінних умов мережі та поведінки користувачів. Це знижує загальне навантаження на систему та покращує користувацький досвід за рахунок зменшення часу відповіді та збільшення надійності обробки запитів.

3 РОЗРОБКА ПРОГРАМНИХ КОМПОНЕНТ BACKEND AS A SERVICE ПЛАТФОРМИ ДЛЯ МОБІЛЬНИХ І ВЕБ-ЗАСТОСУВАНЬ

3.1 Варіантний аналіз та обґрунтування вибору засобів реалізації програмного засобу

Прийняття рішення щодо вибору технологій та інструментів є ключовим етапом у процесі розробки BaaS платформи, оскільки це безпосередньо впливає на ефективність та гнучкість системи. У контексті нашого проекту потрібно обрати мову програмування для реалізації мікросервісів адаптивного алгоритму, яка має включати широкий спектр підтримки бібліотек для наукових обчислень та відповідне середовище для проведення симуляції даних для визначення відповідних коефіцієнтів ПІД-контролера.

Наступним кроком буде аналіз та вибір мови програмування та фреймворків для реалізації BaaS Core сервісів. Потрібно врахувати головні критерії, для розробки основної частини BaaS, такі як: високу продуктивність, підтримку мікросервісної архітектури та безпеку. Також обрана мова програмування повинна надавати гнучкість у розробці розподілених систем і надавати можливість швидко масштабуватись.

3.1.2 Обґрунтування вибору мови програмування для адаптивного алгоритму

У процесі вибору мови програмування для реалізації адаптивного алгоритму, основними критеріями є можливості в аналізі часових рядів, зокрема AR (авторегресійне) моделювання, а також зручність переведення алгоритму в мікросервісну архітектуру. Основними кандидатами є Python та ML.NET.

Python - мова з великою кількістю бібліотек для наукових обчислень та машинного навчання, що робить її ідеальною для розробки алгоритмів, зокрема для аналізу часових рядів. Python також популярний для прототипування завдяки зручності використання інструментів, таких як Jupyter Notebook [31].

ML.NET - фреймворк машинного навчання від Microsoft, що інтегрується з .NET. Хоча ML.NET пропонує великі можливості для розробки машинного навчання, але він все ще має обмеження у порівнянні з Python, особливо щодо аналізу часових рядів і гнучкості в прототипуванні. В таблиці 3.1 наведено порівняння можливостей Python та ML.NET.

Таблиця 3.1 − Порівняння можливостей Python та ML.NET

Критерій	Python	ML.NET
Аналіз часових рядів (AR)	+	-
Підтримка машинного навчання	+	+
Гнучкість та швидкість прототипування (Jupyter Notebook, ін.)	+	-
Інтеграція в мікросервісну архітектуру	+	+
Спільнота та підтримка	+	-
Широта використання бібліотек	+	-
Результат	6	2

Отже, Python у порівнянні з ML.NET надає більше можливостей, особливо в контексті гнучкості та аналізу часових рядів. Вибір Python також забезпечує зручність в прототипуванні та інтеграції з іншими інструментами, що є важливим для розвитку адаптивного алгоритму у BaaS середовищі.

При виборі конкретних бібліотек та інструментів для реалізації алгоритму в Python, особливу увагу приділено таким, як Pandas для обробки та аналізу даних, NumPy для математичних обчислень та Matplotlib для візуалізації результатів. Ці бібліотеки мають широку підтримку у науковому середовищі та високу ефективність, що є важливим для забезпечення точності та швидкості аналізу в межах нашої системи.

Веб-сервіси, які використовують адаптивний алгоритм використовують Flask фреймворк. Flask - легкий і гнучкий веб-фреймворк. Flask дозволяє швидко

розробляти REST API, забезпечуючи простоту та зручність у роботі. Основні переваги Flask включають:

- Мінімалістичність та легкість у розробці.
- Гнучкість у побудові веб-сервісів з можливістю розширення функціоналу через різноманітні розширення.
- Підтримка REST API для легкої інтеграції з клієнтськими додатками.
- Широка спільнота та наявність численних ресурсів для навчання та підтримки.

3.1.2 Обґрунтування вибору мови програмування

Під час вибору мови програмування та її фреймворків для реалізації BaaS Core сервісів було розглянуто декілька варіантів, а саме: .NET з мовою програмування C#, Spring з Java, Django з Python та Golang [32]. Використання кожної з цих технологій мало свої переваги та особливості, які були враховані при виборі системи. В таблиці 3.2 наведено порівняння деяких популярних мов програмування за різними критеріями.

Таблиця 3.2 − Порівняння популярних мов програмування

Мова програмування	Фреймворк	Переваги	Недоліки
C#	.NET	Велика екосистема, надійність, продуктивність	Вимагає платформи .NET для роботи
Java	Spring	Велика підтримка, розширюваність	Більша витривалість в розробці
Python	Django	Простота, широкі можливості	Швидкість виконання може бути нижчою
Golang	-	Висока продуктивність, простота	Обмежена екосистема, менша підтримка

Після ретельного аналізу та порівняння, ми обрали .NET з мовою програмування C# для реалізації проекту. Основні аргументи, що підтримують цей вибір, включають:

- Широка екосистема: .NET має велику спільноту розробників та багатий набір інструментів та бібліотек для розробки програмного забезпечення. Це сприяє швидкому розвитку та реалізації функціональних вимог системи.
- Надійність та продуктивність: .NET з C# є платформою, що надає надійність та високу продуктивність. Вона дозволяє ефективно керувати ресурсами системи та забезпечує високу швидкість виконання.]
- Легкість інтеграції: .NET може легко інтегруватись з іншими технологіями та платформами, що дозволяє розширити функціональність системи та спростити її взаємодію з іншими системами.
- Підтримка C#: Мова програмування C# є ефективною та експресивною мовою, яка дозволяє швидку розробку та підтримку коду. Вона має багатий набір функцій та зручний синтаксис, що полегшує роботу розробників.
- Масштабованість: .NET забезпечує гнучкість та масштабованість, що є важливими факторами для розробки сервісів BaaS Core, яка може потребувати збільшення обсягу даних та кількості користувачів.

Отже, .NET 6 з мовою програмування C#, новітня версія популярного фреймворку від Microsoft, була обрана для розробки основної частини BaaS через її високу продуктивність, підтримку мікросервісної архітектури та безпеку. .NET 6 забезпечує велику гнучкість у розробці розподілених систем і є оптимальним рішенням для створення надійної та масштабованої BaaS платформи.

Для розробки веб-сервісів BaaS платформи, використаємо ASP.NET 6. Цей фреймворк підтримує створення як комплексних веб-застосунків, так і REST API сервісів, забезпечуючи гнучкість та високу продуктивність [33]. Основні переваги ASP.NET 6 включають:

- Активну підтримку від Microsoft.

- Підтримку Web API для легкої інтеграції з різними клієнтськими додатками.
- Інтегрований веб-сервер Kestrel, що забезпечує високу продуктивність та надійність.
- Можливості масштабування, які важливі для обробки великої кількості запитів у BaaS.
- Вищу продуктивність порівняно з популярними аналогами.

Для моделювання та проектування системи використовується Visual Paradigm. Visual Paradigm пропонує широкий набір функціональності, включаючи аналіз вимог, моделювання бази даних, розробку графічного інтерфейсу та програмування на різних мовах, включаючи C#. Він має зрозумілий інтерфейс, що полегшує роботу з ним, а також забезпечує широку підтримку та доступну документацію. Visual Paradigm також популярний серед розробників та використовується в багатьох проектах, що свідчить про його ефективність та надійність [34].

3.1.3 Обґрунтування вибору системи управління базою даних

Однією з основних вимог до системи зберігання даних платформи є її розширюваність, масштабування у відповідності до кількості користувачів та здатність обробляти дані довільної схеми. Тому в якості засобу зберігання даних перевагу віддають NoSQL рішенням.

MongoDB є NoSQL базою даних, яка пропонує високу гнучкість і швидкість при роботі з великими обсягами неструктурованих даних. З іншого боку, MS SQL Server є реляційною базою даних, яка пропонує сильну підтримку транзакцій та структурованих запитів.

В таблиці 3.3 наведено результати порівняння MongoDB з реляційною базою даних MS SQL Server.

Таблиця 3.3 − Порівняння MongoDB та MS SQL Server

Критерій	MongoDB	MS SQL Server
Робота з неструктурованими даними	+	-
Транзакційність	-	+
Масштабованість	+	-
Висока продуктивність	+	+
Результат	3	2

Отже, виходячи з вимог до системи та проведеного аналізу обираємо MongoDB, як основну систему зберігання даними BaaS-Core сервісів. Для зберігання та аналізу часових рядів даних використовуємо InfluxDB. InfluxDB спеціалізується на зберіганні та аналізі часових рядів, пропонуючи оптимізовані можливості для швидкої обробки і запитів на великі обсяги даних, що змінюються з часом.

3.2 Розробка архітектури BaaS Core сервісів платформи

3.2.1 Компоненти системи.

Архітектура BaaS-core платформи спроектована як трирівнева клієнт-серверна архітектура. Де серверна частина містить бізнес-логіку та взаємодіє з середовищем збереження даних. В основу взаємодії між сервісами та сервером покладені принципи REST архітектури, та використовує протокол HTTP для передачі даних між ними. На рисунку 3.1 зображено компонентну діаграму, яка включає основні компоненти системи та інтерфейси їх взаємодії.

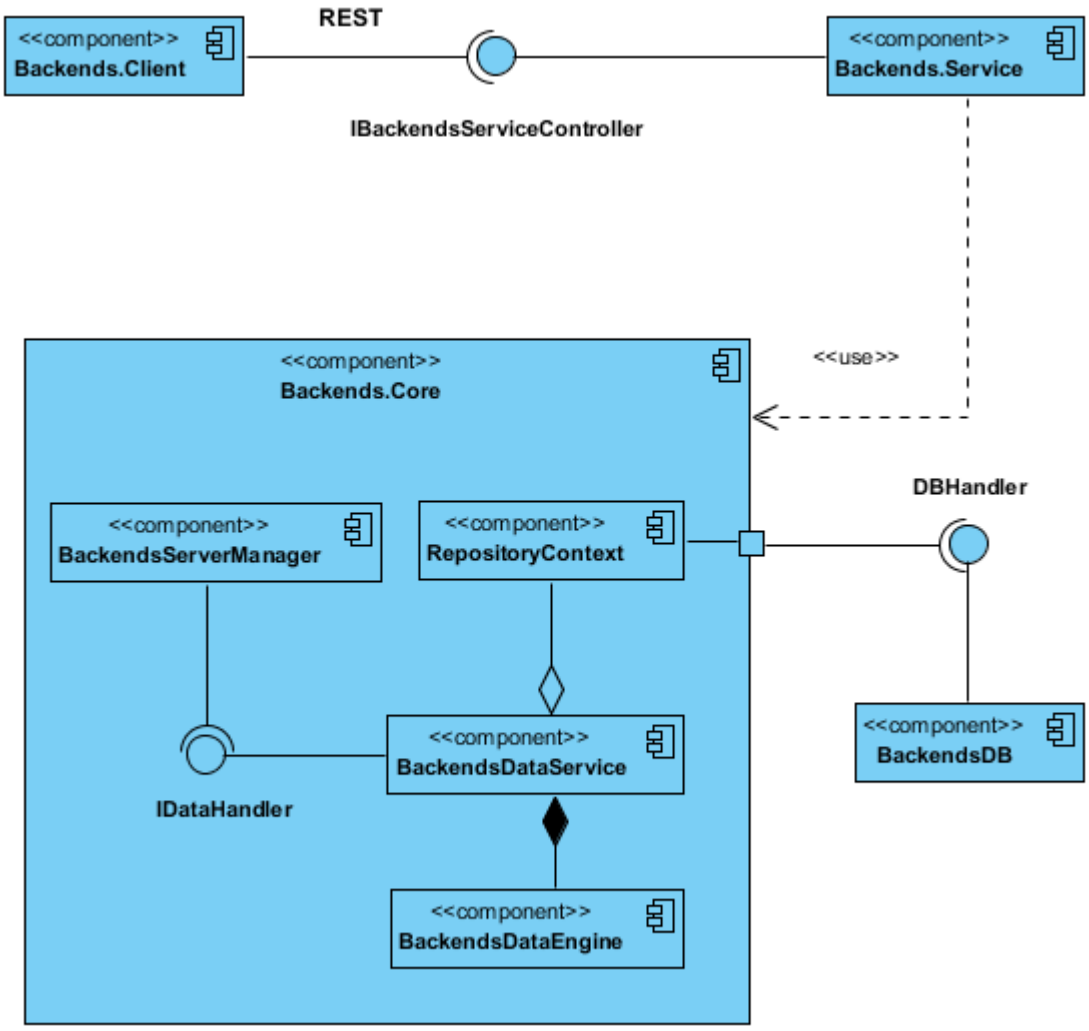


Рисунок 3.1 – Компонентна діаграма платформи BaaS

Розглянемо основні компоненти діаграми. Backends.Client — клієнтське застосування (веб або мобільне), що взаємодіє з сервісом. Backends.Service — сервіс, що є фасадом бізнес-логіки, який взаємодіє з клієнтом через інтерфейс IBackendsServiceController. Згідно з представленою компонентною діаграмою, бізнес-логіка нашого проекту розміщуватиметься у компоненті Backends.Core. RepositoryContext – компонент системи, через який забезпечується доступ до даних BackendsDB в базі даних системи. BackendsServerManager, BackendsDataService та DataEngine є компонентами бізнес-логіки системи.

3.2.2. Проектування серверної частини платформи.

Серверна частина платформи буде розроблена, як RESTful Web-сервіс. Це накладає певні умови при проектуванні системи: всі методи API повинні бути ідемпотентними, а сервер повинен масштабуватися. Саме тому обрано модель stateless для веб-архітектури. Спроектовані відповідно до зазначених вимог бізнес об’єкти системи будуть з використанням. Діаграма класів серверної частини наведена на рисунку 3.2.

Клас BackendServerController реалізує інтерфейс IBaseController та є фасадним об’єктом системи. В ньому використовуєтсья п’ять класів DTO (data transfer object) для повернення інформації клієнту, яка формується на основі даних отриманих від компонента бізнес-логіки Backends.Core (рисунок 2.4). Об’єкт фасаду взаємодіє з Backends.Core через клас BackendsServerManager, який є сінглонтоном, та виконує управління всіма бізнес процесами в платформи. Для цього BackendsServerManager містить класи BackendsUserManager, який здійснює управління користувачами та BackendsDataService, який відповідає за бізнес-логікою операцій з даними платформи. Обидва цих класи використовують інтерфейс IRepository для виконання операцій з базою даних. Даний інтерфейс — IRepository створює рівень абстракції над доступом до бази даних системи, що дозволяє у випадку зміни провайдера даних дуже легко переключитись від одного провайдера до іншого, створивши відповідну реалізації цього інтерфейсу та передавши її в BackendsServerManager. В даному випадку реалізація інтерфейсу BacksRepository — BacksRepository використовує MongoDB C# драйвер для доступу до бази даних. Також BacksRepository містить списки базових об’єктів даних платформи BacksEntity, BacksUser BacksSession та BacksObject.

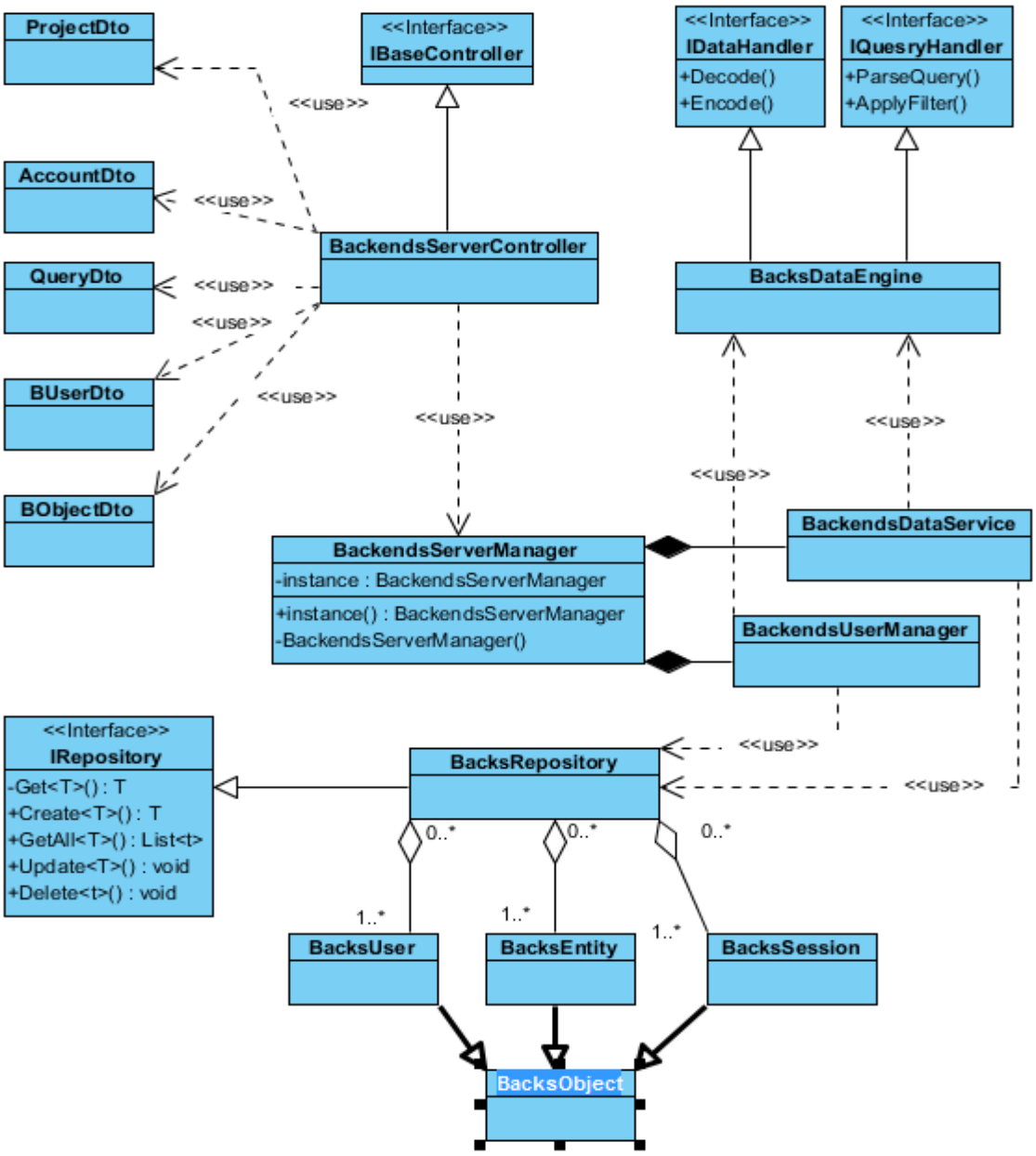


Рисунок 3.2 – Діаграма класів серверу

3.2.3 Проектування бази даних

На основі моделі предметної області виділимо наступні сутності:

- «Користувач платформи» (Accounts) – містить персональні дані користувача платформи
- «Проекти» (Projects) – проекти, що використовується користувачем платформи

- «Користувач проекту» (_Users) – персональні дані користувачів проекту
- «Сесії користувачів» (_Sessions) – сеанси користувача проекту
- «Об’єкти даних користувача» (BackendsEntities) – об’єкти довільної схеми, створені користувачем платформи
- «Схема даних» (_Schema) – схема даних проекту

Для кожної з сутностей додаємо атрибути та їх описання. (таблиця 3.4)

Таблиця 3.4 – Сутності та атрибути предметної області проекту

Назва сутностей	Атрибут	Опис атрибута
Accounts	FirstName	Ім’я користувача
	LastName	Прізвище користувача
	ScreenName	Логін
	Password	Пароль
	Email	Електронна адреса
Projects	Id	Унікальний ідентифікатор проекту
	ProjectName	Назва проекту
	ApiKeyAccess	Унікальний ключ доступу до API проекту
	MasterKeyAccess	Головний ключ проекту (дозволяє змінювати проект)
	Settings	Налаштування проекту
	CreatedAt	Час створення проекту

Продовження таблиці 3.4

Назва сутностей	Атрибут	Опис атрибута
_Users	UserName	Унікальне ім’я користувача проекту
	Password	Пароль
	CreatedAt	Час створення запису
	UpdateAt	Час оновлення запису
_Sessions	Id	Унікальний ідентифікатор сесії користувача проекту
	SessionToken	Токен сесії
	ExpiresAt	Час до якого існує сесія
	CreatedAt	Час створення запису
	UpdatedAt	Час оновлення запису
_Schema	_entityId	Унікальний ідентифікатор сутності проекту
	EntityValueType Mapping	Опис схеми даних (змінних - типів)
BackendsEntities	Id	Унікальний ідентифікатор об’єкту даних користувача проекту
	EntityName	Назва об’єкту даних
	CreatedAt	Час створення запису
	UpdatedAt	Час оновлення запису
	UserDataObject	Об’єкт довільних даних типу даних
_Roles	Name	Назва ролі
	ReadPermission	Дозвіл читання
	WritePermission	Дозвіл запису
	_acl	ACL (Access Control List)

Встановимо зв’язки між описаними сутностями. Відповідно до специфікації вимог, кожен зареєстрований користувач може створювати проекти. Таким чином, існує відношення «один до багатьох» між сутностями «Користувач платформи» (Account) та «Проекти» (Projects). В межах кожного проекту передбачається можливість управління користувачами і їхніми сесіями, а також надання прав доступу до проекту та створення даних користувача. Тому встановлюємо зв’язок «один до багатьох» між сутністю «Проекти» (Projects) та «Користувач проекту» (_Users) та «Об’єкти даних користувача» (BackendsEntities). Взаємозв’язок між користувачами проекту та їхніми сесіями «багато до багатьох». Після визначення сутностей та зв’язків між ними побудуємо концептуальну діаграму (рисунок 3.3) використовувався CASE-засіб Visual Paradigm v17 [30].

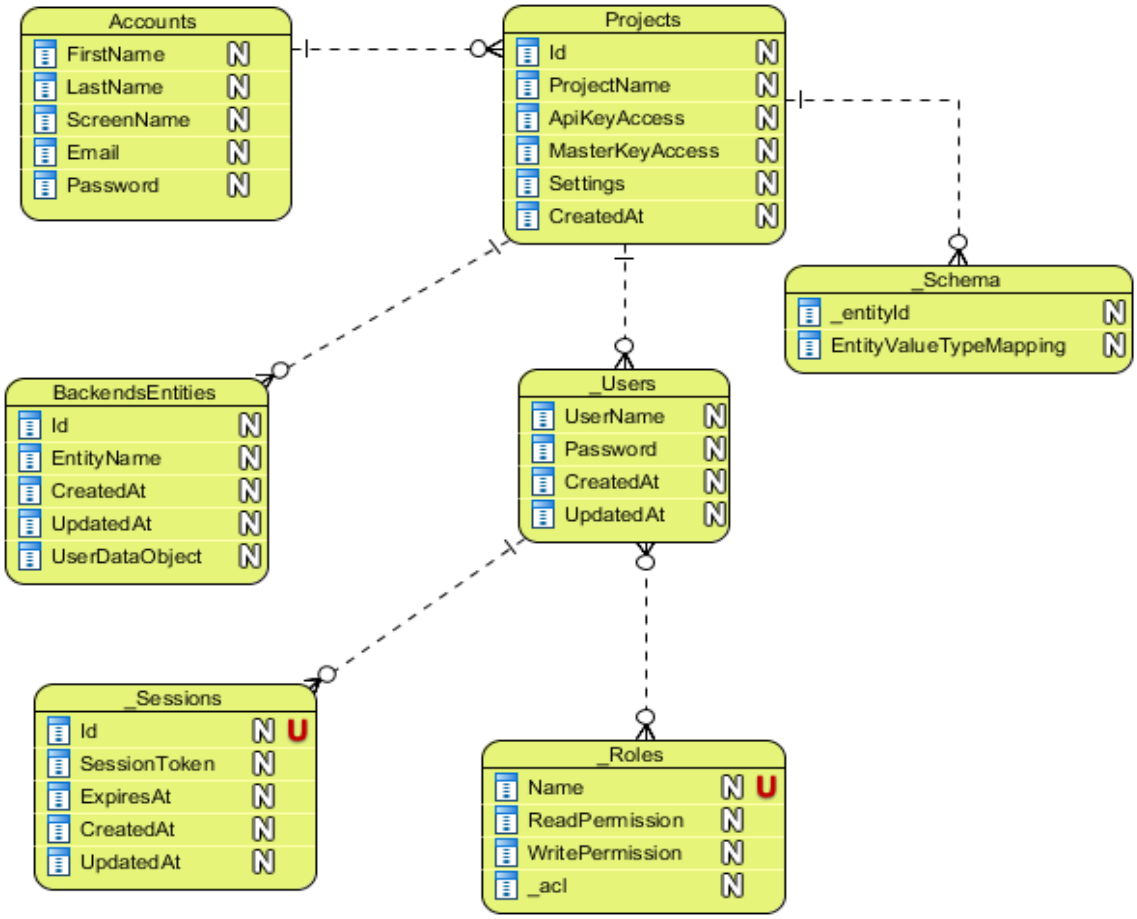


Рисунок 3.3 – Концептуальна ER-діаграма бази даних BaaS-Core

Проведемо аналіз розробленої концептуальної моделі та її нормалізацію. Встановимо типи атрибутів та ключі до кожної сутностей. Після нормалізації сутностей _Users з _Roles створюємо нову сутність _Users__Roles, що містить зовнішні ключі до _Users та _Roles зі зв’язком «один до багатьох».

Виходячи з нормалізованої логічної моделі побудуємо логічну модель. Так як фізична модель має відображати реальну структуру бази даних згідно з вибраною системою управління базами даних, необхідно адаптувати логічну модель до фізичної в MongoDB.

Під час проектування бази даних MongoDB, зайва абстракція може бути недоцільною, оскільки MongoDB не вимагає заздалегідь визначеної фізичної схеми. Хоча створення абстракцій зменшує гнучкість, використання MongoDB передбачає еластичний підхід до дизайну. Втім, абстракція в MongoDB може бути корисною для інтеграції певних концепцій. У контексті платформи BaaS, важливо підтримувати гнучкість для абстрактних сутностей, щоб забезпечити їхнє розширення за потребою та контроль цілісності даних. Обраний дизайн повністю відповідає вимогам фізичної моделі MongoDB.

3.2.4 Проектування REST API

Згідно з компонентною структурою платформи, взаємодія між клієнтом та сервером здійснюється через REST API, що забезпечує сумісність з мобільними та веб-додатками. Ми розробимо ключові методи API, а також формати запитів та відповідей, які клієнти використовуватимуть для комунікації з BaaS платформою.

Для безпеки, всі запити до платформи реалізуються через HTTPS. При використанні HTTP методів POST та PUT, тіло запиту повинно бути у форматі JSON із вказаним заголовком Content-Type ‘application/json’. Автентифікація відбувається через заголовки HTTP: X-Backends-AppId для ідентифікації додатку та X-Backends-REST-API-Key для автентифікації точки доступу.

Відповідно до зазначеного вище, спроектовані API наведені в таблицях 3.5-3.6, кожна з яких відображає ідентифікатор ресурсу, HTTP метод, та призначення операції.

Таблиця 3.5 – Основні API методи для здійснення операцій з об’єктами платформи

URL	HTTP метод	Призначення
/1/entities/<entityName>	POST	Створення об’єкту
/1/entities/<entityName>/<objectId>	GET	Отримання об’єкту
/1/entities/<entityName>/<objectId>	PUT	Оновлення об’єкту
/1/entities/<entityName>	GET	Запит даних
/1/entities/<entityName>/<objectId>	DELETE	Видалення об’єкту

Таблиця 3.6 – User API методи для управління користувачами в платформі

URL	HTTP метод	Призначення
/1/users	POST	Створення користувача
/1/login	GET	Логін
/1/logout	POST	Вихід
/1/users/<objectId>	GET	Отримання користувача
/1/users/<objectId>	PUT	Оновлення даних користувача
/1/users	GET	Запит даних користувачів
/1/users/<objectId>	DELETE	Видалення користувача
/1/requestPasswordReset	POST	Скидання пароля

Результати запитів визначаються за допомогою HTTP статус-кодів та кодів платформи. Код статусу 2xx свідчить про успішне виконання, тоді як 4xx вказує на клієнтські помилки у запиті. Як у випадку помилки так і в результаті успішного виконання, тіло відповіді буде JSON об’єкт, та міститиме код і опис помилки, які можуть бути використаним для відлагодження програми:

{"errorCode": 105, "errorMsg": "Invalid field name: name"}

3.2.5 Проектування запитів об’єктів даних платформи

Вказані раніше методи API надають можливість здійснювати основні дії з даними проектів, такі як створення, модифікація, оновлення, та видалення елементів. Однак для досягнення більшої гнучкості в обробці даних, важливо забезпечити функціонал фільтрації отриманої від сервера інформації. Тому ми плануємо розробити інтерфейс, що дозволить формувати запити для відбору даних за певними критеріями.

Для цього розробимо протокол для запитів даних. Оскільки для запитів використовується HTTP GET метод, параметри фільтрації будуть включені у URL-рядок запиту. Основні параметри фільтрації представлені у таблиці 3.7.

Таблиця 3.7 – Параметри фільтрації запитів

Параметр	Використання
where	Фільтрація об’єкту запиту за операторами умов.
take	Визначає кількість записів об’єкту, які потрібно повернути на запит
skip	Кількість записів об’єкту, які потрібно пропустити на запит.
join	Додає об’єкт в поле посилання на інший об’єкт
keys	Визначає поля об’єкту, які будуть результатами відповіді сервера.
order	Визначає поле об’єкту сортування. Прямий порядок сортування.
dorder	Визначає поле об’єкту сортування. Зворотній порядок сортування.
count	Отримує загальну кількість записів об’єкту.

Параметр where дозволяє встановлювати обмеження результату пошуку. Значення параметра where повинні бути закодовані в JSON. Для виконання умов використовуються операції наведені в таблиці 3.8.

Таблиця 3.8 – Умови які підтримує параметр where

Операції	Використання
$gt	Більше за значення
$gte	Більше або рівне значенню
$lt	Менше за значення
$lte	Менше або рівне значенню
$ne	Не дорівнює
$select	Дозволяє створити вкладений запит
$in	Значення знаходиться в списку
$nin	Значення не знаходиться в списку
$or	Оператор “або”

Шаблон базової форми для будь-якого запиту буде мати наступний вигляд у форматі JSON:

"_query”:{"className": <Name>, (<operation>, ...)},

в якому <op> = {<field>:{<_cond>}}, де

className – ключ в якому задається назва сутності,

<field> — значення поля сутності,

<_cond> —умови, які підтримує оператор

<operation> — це параметри запиту такі як "where", "key" і т.д.

3.3 Програмна реалізація додатку

3.3.1. Реалізація доступу до даних.

Без врахування специфіки обраної системи управління базами даних (СУБД) та використовуваних бібліотек доступу, сформуємо абстракцію для взаємодії з базою даних. Це забезпечить гнучкість у рівні взаємодії та спростить процес зміни однієї СУБД на іншу.

Взаємодія з базою даних реалізована за допомогою двох шаблонів проектування: «Одиниця роботи» (Unit of Work) та «Репозиторій» (Repository)[30][35]. Ці шаблони дозволяють взаємодіяти з базою даних у межах єдиного контексту, і для кожного типу даних буде створено відповідний репозиторій із всіма необхідними методами для взаємодії.

На рисунку 3.4 наведений метод GetEntity, для отримання даних з бази даних, реалізовану в класі BacksRepository, який і є репозиторієм.

```
9 references
public async Task<BacksObject> GetEntity(string appId, string entityName, string entityId)
{
    try
    {
        var filter = Builders<BacksObject>.Filter.Eq("Id", entityId);
        return await _context.Get_Objects(entityName + "_" + appId)
                        .Find(filter)
                        .FirstOrDefaultAsync();
    }
    catch (Exception e)
    {
        _log.Error("Exception in GetEntity", e);
        throw;
    }
}
```

Рисунок 3.4 – Метод GetEntity доступу до сховища даних в MongoDB

Шаблон «Одиниця роботи» буде реалізовано в класі BackendsContext, який буде описувати сховище даних. Цей клас забезпечуватиме доступ до бази даних MongoDB. Для цього потрібно включити в проект Nuget пакет під назвою MongoDB.Driver, що є офіційним драйвером для .NET з повною підтримкою

ASP.NET застосунків. Приклад реалізації методу доступу до колекції даних MongoDB реалізований у класі BackendsContext наведено на рисунку 3.5

```
5 references
public IMongoCollection<BacksObject> Get_Objects(string collection)
{
    return _db.GetCollection<BacksObject>(collection);
}
```

Рисунок 3.5 – Метод Get_Objects для доступу до колекції MongoDB

Інтерфейс IRepositoryAsync описує реалізацію шаблону проектування “Репозиторію”. Клас що реалізує даний інтерфейс містить функціонал необхідний для отримання даних платформи. Лістинги реалізації операцій даного інтерфейсу наведені в додатку В.

3.3.2. Реалізація бізнес-логіки

Логіка бізнес-процесів платформи відокремлена на окремий абстрактний рівень і реалізована через сервісні класи, які виконують операції з сутностями платформи. Розробка бізнес-функціоналу поділена відповідно до специфіки домену та необхідної логіки виконання. У процесі реалізації бізнес-логіки були створені такі класи:

- BacksObjectService — забезпечує логіку роботи з об’єктами проекту.
- BacksUsersService — відповідає за логіку управління користувачами.
- BacksDashboardService — виконує операції з об’єктами проектів та розробниками на платформі.

Клас BacksObjectService надає інтерфейс для виконання зчитування, редагування, видалення та оновлення об’єктів проекту. На рисунку 3.6 показано реалізацію методу GetEntity класу BacksObjectService для реалізації бізнес логіки зчитування об’єктів проекту.

```
public async Task<Tuple<BacksErrorCodes, ObjectsDto>> GetEntity(string appId, string name, string entityId)
{
    var error = BacksErrorCodes.Ok;
    var obj = new ObjectsDto();
    try
    {
        BacksObject entity = await _repo.GetEntity(appId, name, entityId).ConfigureAwait(false);
        if (entity == null)
        {
            error = BacksErrorCodes.EntityNotFound;
            return new Tuple<BacksErrorCodes, ObjectsDto>(error, null);
        }

        obj = new ObjectsDto()
        {
            Id = entity.Id,
            Name = entity.Name,
            Data = entity.Data,
            CreatedAt = entity.CreatedAt,
            UpdatedAt = entity.UpdatedAt
        };
    }
    catch (Exception e)
    {
        _log.Error("GetEntity exception : ", e);
        error = BacksErrorCodes.SystemError;
    }
    return new Tuple<BacksErrorCodes, ObjectsDto>(error, obj);
}
```

Рисунок 3.6 – Метод GetEntity класу BacksObjectService для реалізації бізнес логіки зчитування об’єктів проекту.

Взаємодія рівня бізнес-логіки з базою даних здійснюється через інтерфейс IRepositoryAsync, що дозволяє забезпечити доступ до сховища даних, відповідно до схеми представленої на рисунку 3.7.

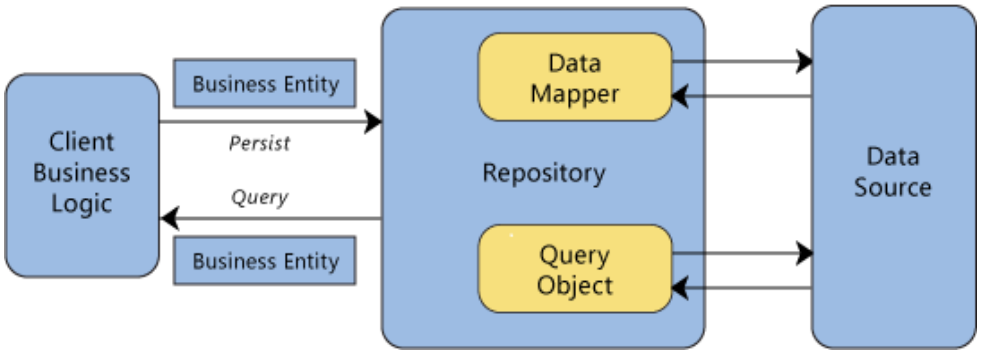


Рисунок 3.7 – Взаємодія з сховищем даних

Лістинги бізнес операцій наведених вище класів наведені в додатку В.

3.3.3 Реалізація сервісу аналізу часових даних адаптивного алгоритму системи.

Сервіс аналізу даних AnalysisService є ключовим компонентом у реалізації адаптивного алгоритму системи BaaS. Цей сервіс, розроблений на Python, дозволяє збирати, обробляти та аналізувати часові дані використовуючи моделі AR (AutoReg). Розглянемо детально основні функції класу AnalysisService.

Функція check_stationarity для визначення стаціонарності часових рядів зображена на рисунку 3.8.

```python
def check_stationarity(self, timeseries, alpha=0.05):
    """Check if the provided timeseries is stationary."""
    result = adfuller(timeseries, autolag='AIC')
    logging.info(f'ADF Statistic: {result[0]}')
    logging.info(f'p-value: {result[1]}')
    return result[1] < alpha
```

Рисунок 3.8 – Функція визначення стаціонарності

В ній використовується функція adfuller класу stattools з модуля statsmodels.tsa, що являє собою реалізацію тесту Дікі–Фуллера.

Функція fit_ar_model зображена на рисунку 3.9, та використовує для авторегресивного моделювання функцію AutoReg з модуля statsmodels.tsa.

```python
def fit_ar_model(self, timeseries, max_lag=10):
    """Fit an AutoRegressive model to the provided timeseries."""
    if not self.check_stationarity(timeseries):
        logging.warning("Series is not stationary, differencing might be required.")
        timeseries = timeseries.diff().dropna()
    if not self.check_stationarity(timeseries):
        raise ValueError("Series is still not stationary after differencing.")

    order_selection = arma_order_select_ic(timeseries, max_ar=max_lag, ic=['aic'])
    lags = order_selection['aic_min_order'][0]
    model = AutoReg(timeseries, lags=lags)
    return model.fit()
```

Рисунок 3.9 – Функція авторегресивного моделювання

3.4 Висновки

У третьому розділі роботи було розглянуто ключові аспекти розробки та імплементації платформи Backend as a Service (BaaS), враховуючи важливість інтеграції адаптивного алгоритму для повторних спроб. Аналіз та вибір технологій та інструментів для розробки платформи були зосереджені на забезпеченні високої гнучкості, продуктивності та надійності системи.

Рішення про використання Python для симуляції адаптивного алгоритму в Jupyter Notebook та .NET 6 для розробки основних компонентів BaaS платформи було прийняте з урахуванням потреб у багатофункціональності, ефективності та легкості інтеграції різних компонентів системи.

В процесі розробки архітектури BaaS сервісів було приділено особливу увагу мікросервісній архітектурі, що забезпечує високу масштабованість та легкість управління сервісами. Дана архітектура дозволяє розділити відповідальність між різними сервісами, що сприяє кращій підтримці та розвитку системи. Ключові компоненти системи, включаючи API Gateway, BaaS Core Services, Metric Collection Service, Analysis Service, Adaptive Retry Service, Circuit Breaker Service та Feedback Loop Controller, були ретельно сплановані та реалізовані для забезпечення ефективної взаємодії та високої продуктивності системи.

4 ТЕСТУВАННЯ ПРОГРАМНОГО ДОДАТКУ

4.1 Тестування адаптивного алгоритму BaaS

Тестування та симуляція адаптивного алгоритму є ключовими компонентами перевірки його ефективності та надійності. Симуляція допомагає передбачити реакцію системи на різні вхідні дані та налаштувати параметри алгоритму для досягнення оптимальної продуктивності. Для зручності тестування будемо створювати симуляції в Jupyter Notebook. Програмний код симуляції алгоритму зображений на рисунку 4.1.

```
# Load the dataset
df = pd.read_csv("metrics_data.csv")

# Choose a metric for analysis, e.g., response time
metric = df['response_time']

# Check for stationarity and fit the AR model
result = adfuller(metric)
if result[1] < 0.05:
    print("Series is stationary")
    # Step 3: Model Order Selection
    order = arma_order_select_ic(metric, ic=['aic', 'bic'], trend='c')
    p = order.aic_min_order[0]
    print(f"Optimal order p: {p}")

    model = AutoReg(metric, lags=p, trend='c').fit()
    preds = model.predict(start=len(metric), end=len(metric) + 20)

    # Assume the setpoint is the average response time you want to achieve
    setpoint = df['response_time'].mean()
    print(f"setpoint = {setpoint}")
    # Initialize PID controller with a setpoint (example value)
    pid = PIDController(kp=-1.634, ki=0.001, kd=-0.07, setpoint=0.2)

    # Simulate PID adjustments based on AR model predictions
    adjustments = [pid.update(val) for val in preds]
```

Рисунок 4.1 – Програмний код симуляції алгоритму в Jupyter Notebook

На першому етапі створимо віртуальне середовище, яке імітує реальні умови роботи BaaS системи. Визначимо вхідні даних, таких як час відповіді, кількість

невдач і повторних спроб. Варто зазначити, що початкові дані для тестування, були згенеровані таким чином, щоб імітувати реальні умови експлуатації системи BaaS Приклад згенерованих даних зображений на рисунку 4.2. Дані містили в собі час відповіді системи на запити користувачів, а також відомості про невдачі та повторні спроби обробки запитів. Значення часу відповіді були визначені з урахуванням можливих затримок та коливань у роботі системи, а кількість невдач і повторних спроб відображала ступінь надійності та стійкості системи до помилок.

```
timestamp,response_time,failure_rate,num_retries
2023-11-08 01:22:46.552372,0.5939321535345923,0.33890826839811505,2
2023-11-08 01:21:46.552377,0.7436704297351775,0.13500398659608243,4
2023-11-08 01:20:46.552379,0.6424870384644795,0.36759701106129744,3
2023-11-08 01:19:46.552380,0.5903948646972071,0.4810942725587191,3
2023-11-08 01:18:46.552381,0.4812893194050143,0.12437657175997902,1
2023-11-08 01:17:46.552383,0.6813047017599905,0.28807866720891845,1
2023-11-08 01:16:46.552384,0.49382849013642327,0.2960209656359195,1
2023-11-08 01:15:46.552385,0.9025957007038717,0.2861259528954367,1
2023-11-08 01:14:46.552386,0.9672964844509263,0.11154081632030916,2
2023-11-08 01:13:46.552387,0.4450973669431999,0.4763745057584925,0
2023-11-08 01:12:46.552389,0.8125525342743981,0.22356268930881368,3
2023-11-08 01:11:46.552390,0.5760054277776141,0.4232043362355639,1
2023-11-08 01:10:46.552391,0.6112401049845391,0.34973963765875216,4
2023-11-08 01:09:46.552392,0.933036974463395,0.14871847542756683,3
2023-11-08 01:08:46.552393,0.16393245237809825,0.4068989098512386,1
2023-11-08 01:07:46.552394,0.17841636973138664,0.19825287042349232,1
2023-11-08 01:06:46.552395,0.11819655769629316,0.4405515985555808,2
2023-11-08 01:05:46.552396,0.8493578609931441,0.29063643631792935,0
2023-11-08 01:04:46.552398,0.8003410758548655,0.4408676809274264,0
2023-11-08 01:03:46.552399,0.8830109334221372,0.34626579503888294,2
2023-11-08 01:02:46.552400,0.9807565080094875,0.36262713990982026,4
2023-11-08 01:01:46.552401,0.8192427077950513,0.25066219096335113,0
2023-11-08 01:00:46.552403,0.5153314260276387,0.47804181736161194,4
```

Рисунок 4.2 – Початкові вхідні дані

Визначаємо цільове значення (setpoint) ПІД-контролера. Нехай для симуляції адаптивного алгоритму обираємо параметр часу відповіді на запит. Приймемо його за 200мс, що є оптимальним значенням для більшості сучасних розподілених систем.

Далі виконаємо ручне налаштування коефіцієнтів ПІД- контролера (Kp, Ki, Kd) за допомогою методики Зіглера-Нікольса[20]. Який полягає в тому, що спочатку визначається так звана «критична точка» системи, коли при певному значенні коефіцієнта зворотного зв'язку система починає осцилювати з постійною амплітудою. Початок із нульових значень Ki та Kd та поступове збільшення Kp до отримання сталої осциляції. Результат симуляції зображений на рисунку 4.3.

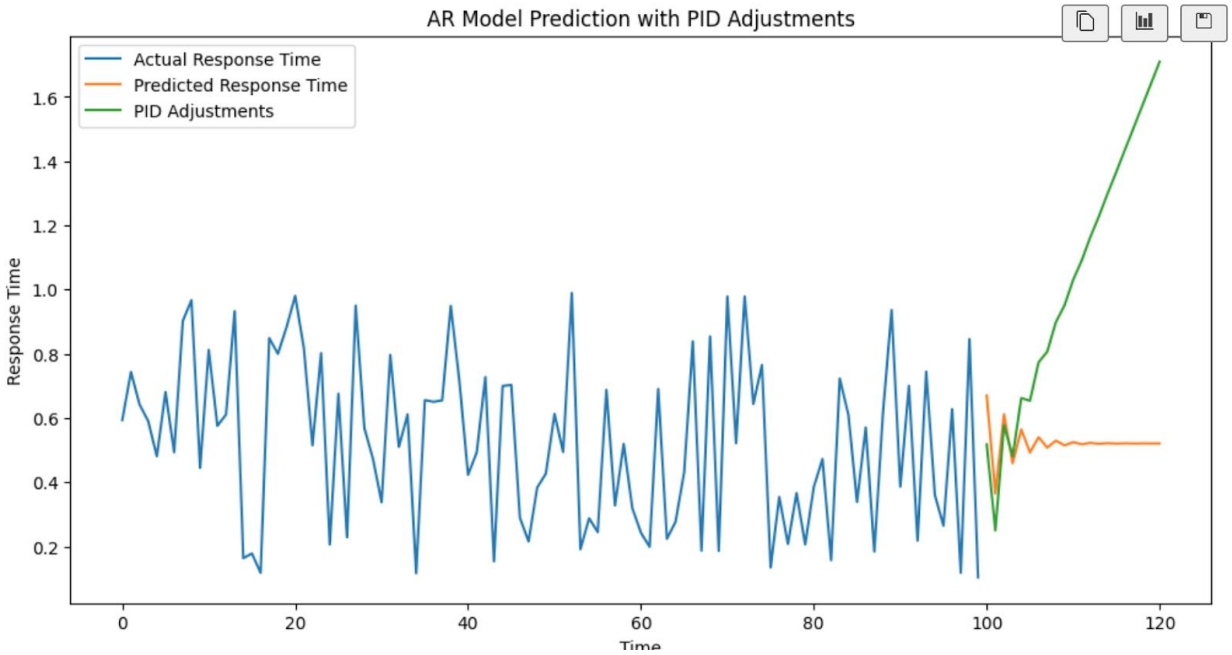


Рисунок 4.3 – Результат налаштування коефіцієнтів ПІД-регулятора за допомогою методики Зіглера-Нікольса

На даному графіку, зелена лінія (PID Adjustments) показує адаптації, зроблені ПІД-контролером у відповідь на реальний час відгуку системи (синя лінія) та прогнозований час відгуку (помаранчева лінія). Існує кілька ключових моментів, які варто відзначити з цього графіку:

1. ПІД-контролер намагається скорегувати систему, щоб відповідати прогнозованому часу відгуку, але корекції відбуваються з великими затримками і перевищеннями.

2. За винятком різкого зростання наприкінці, зелена лінія здається стабілізованою відносно помаранчевої лінії прогнозу. Це може свідчити про те, що ПІД-контролер ефективно реагує на відхилення від цільового значення.
3. Різке зростання зеленої лінії в кінці графіку є аномалією, яка може бути викликана неправильними налаштуваннями контролера або несподіваними збуреннями в системі.

Для більш тонкого налаштування ПІД-регулятора використаємо методику оптимізації за допомогою функції minimize з бібліотеки scipy.optimize. Програмний код використання якої зображений на рисунку 4.4

```python
# Load CSV data
df = pd.read_csv('metrics_data.csv')
# Assume we're tuning PID to adjust the 'response_time' to a setpoint
response_time = df['response_time'].values
setpoint = np.mean(response_time)  # This could be a target value you want to achieve
# Define the PID function to be tuned
def pid_controller(Kp, Ki, Kd, setpoint, response_time):
    pid = PID(Kp, Ki, Kd, setpoint=setpoint)
    pid.sample_time = 0.01  # Update every 0.01 seconds
    control = []
    for rt in response_time:
        control_output = pid.update(rt)  # Use the update method to get the control output
        control.append(control_output)

    return control
# Define the cost function for optimization
def pid_cost_function(params, setpoint, response_time):
    Kp, Ki, Kd = params
    control = pid_controller(Kp, Ki, Kd, setpoint, response_time)
    # Here we calculate the mean squared error
    mse = np.mean((np.array(control) - np.array(response_time))**2)
    return mse
# Initial guesses for Kp, Ki, Kd
initial_guess = [0.1, 0.001, 0.01]
bounds = [(-5, 5), (-5, 5), (-5, 5)]
# Run the optimization
result = minimize(pid_cost_function, initial_guess, args=(setpoint, response_time), bounds=bounds)
# The optimal PID parameters
optimal_Kp, optimal_Ki, optimal_Kd = result.x
print("Optimal Kp, Ki, Kd:", optimal_Kp, optimal_Ki, optimal_Kd)
# Apply the tuned PID controller
optimized_control = pid_controller(optimal_Kp, optimal_Ki, optimal_Kd, setpoint, response_time)
```

Рисунок 4.4 – Програмний код налаштування за допомогою функції minimize

Даний підхід дозволяє знайти оптимальні значення коефіцієнтів (Kp, Ki, Kd) шляхом мінімізації зазначеної цільової функції, яка у нашому випадку визначалася як середньоквадратична помилка між вихідними даними контролера та актуальними значеннями часу відповіді. Ця оптимізація дозволяє автоматично підібрати коефіцієнти для ПІД-контролера, які забезпечують найкращу відповідність встановленому рівню (setpoint). Результати симуляції зображено на рисунку 4.5.

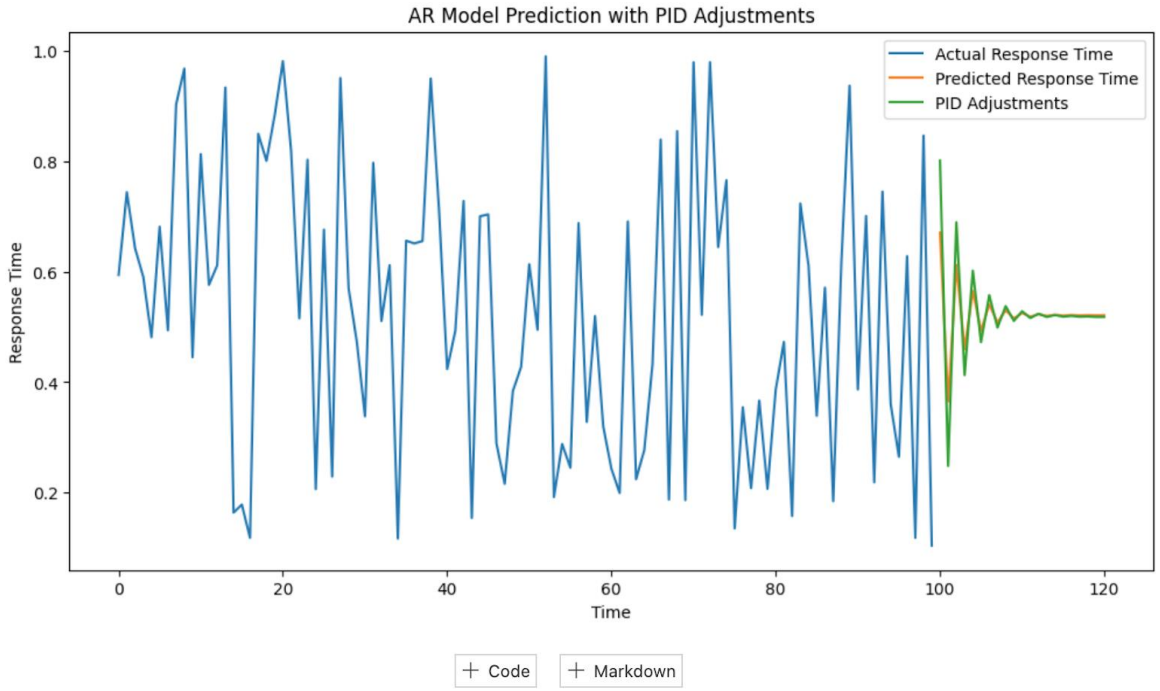


Рисунок 4.5 – Результат налаштування коефіцієнтів за допомогою функції minimize

На зображенні представлено графік з трьома наборами даних: фактичний час відповіді (синя лінія), прогнозований час відповіді (помаранчева лінія), і коригування за допомогою ПІД-контролера (зелена лінія).

Видно, що прогнозований час відповіді тісно слідує за фактичним часом відповіді, що свідчить про те, що модель AR (авторегресивна модель) здійснила

достатньо точний прогноз. Однак, зелена лінія, що представляє коригування за допомогою ПІД, показує різкий сплеск близько 100-го часового кроку, після чого вона швидко стабілізується.

Цей сплеск може бути результатом надмірного коригування, яке часто трапляється в ПІД -контролерах, а також може вказувати на те, що контролер був занадто чутливий до помилки в прогнозі в той момент часу.

Після сплеску, ПІД коригування швидко знижуються і стабілізуються, що може свідчити про те, що контролер ефективно зменшує помилку і приводить систему до бажаного стану (setpoint).

Отже, на основі проведених симуляцій можна зробити висновок, що для досягнення оптимальної роботи системи потрібно більш точне та обережне налаштування ПІД-контролера, що може включати ручне налаштування, автоматизовану оптимізацію або використання додаткових методів фільтрації для згладжування вхідних даних перед їх подачею в контролер.

4.2 Тестування системи

Тестування BaaS платформи є вирішальним етапом у забезпеченні якості та надійності сервісів перед їх впровадженням у виробництво. Цей процес включає в себе ряд дій, спрямованих на виявлення та усунення потенційних проблем, оцінку продуктивності системи та забезпечення відповідності функціональних та нефункціональних вимог.

BaaS-платформа створена, мікросервісна архітектура, тому для тестування бекенд сервісів використовуємо «піраміди тестування» [36]. Такий підхід допомагає організувати процес тестування відповідно до рівнів: від модульних тестів до системних та інтеграційних.

Під час розробки платформи було розроблено 50 функціональних тестових випадки, які покривають варіанти використання платформи (таблиця 4.1). Лістинги реалізації функціональних тестів основних сервісів BaaS наведені в додатку А.

Таблиця 4.1 – Розподіл функціональних тестових випадків за варіантами використання

Варіанти використання	Тестові випадки
Реєстрація	7
ДодаванняПроекту.	10
Проект.Створення	8
Проект.Сутність.Створення	7
Проект.Сутність.Редагування	8
Проект.Сутність.Видалення	5
Проект.Сутність.Запит	5
Загалом	**50**

Результат виконання функціонального тестування зображено на рисунку 4.6

В результаті всі 50 тестів були виконані успішно, отже можна розглядати функціональне тестування, як успішне – 100%.

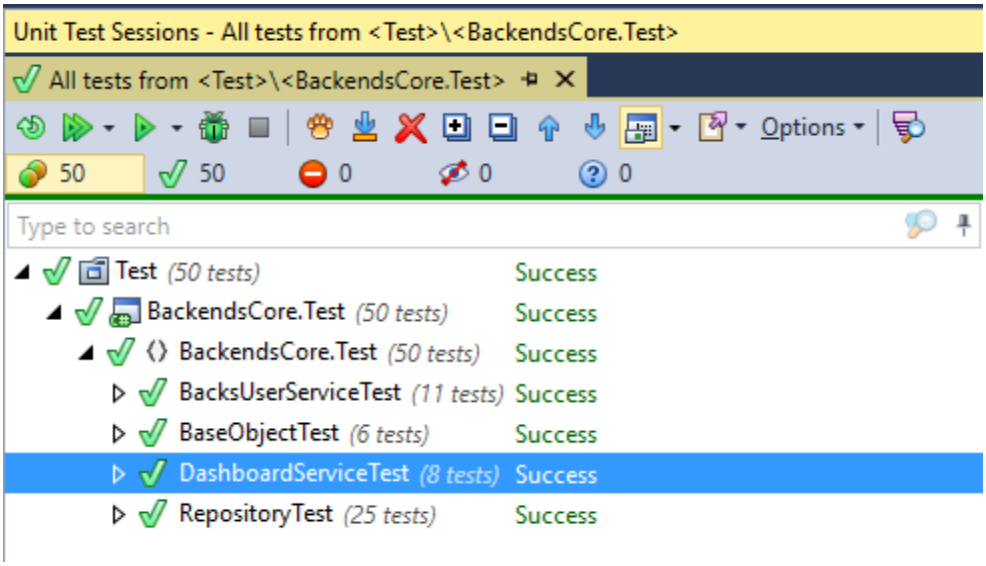


Рисунок 4.6 – Вікно виконання функціональних тестових випадків

Варто зазначити, що покриття тестами коду платформи складає 78% (рисунок 4.7)

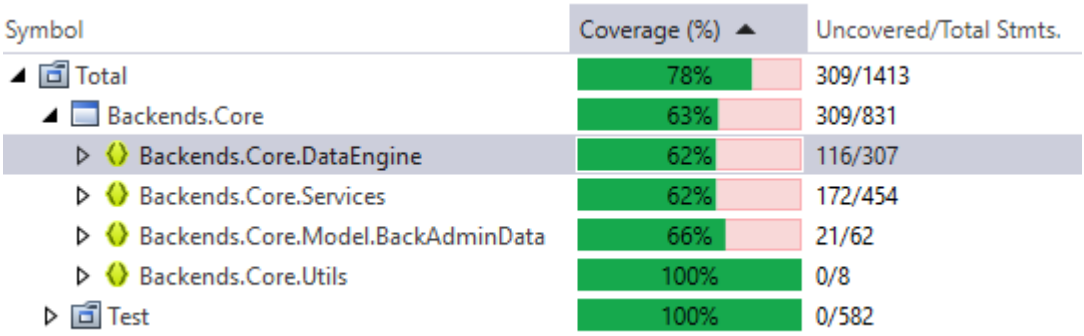


Рисунок 4.7 – Покриття коду BaaS-платформи тестами

Для проведення тестування консолі розробника використовуємо веб-сторінку Backends Dashboard для створення нового проекту (рисунку 4.8).

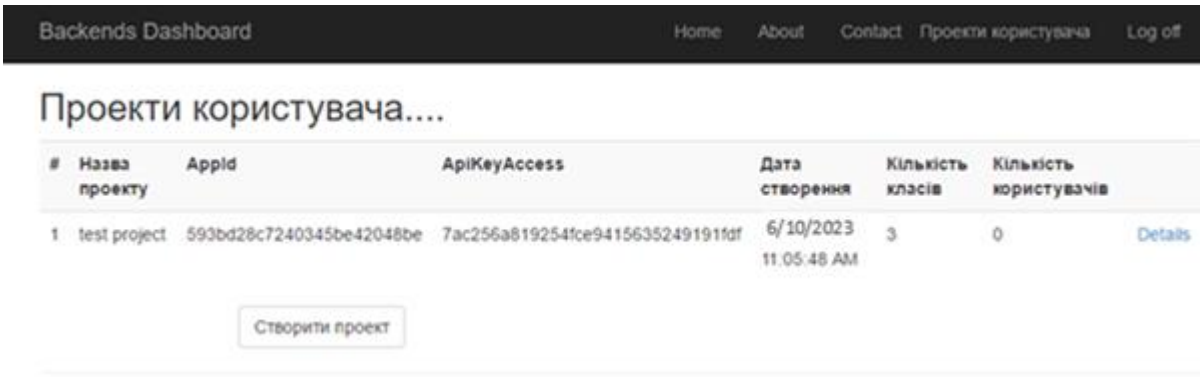


Рисунок 4.8 – Веб-сторінка проектів консолі розробника

Використовуючи веб-сторінку проекту здійснюймо додавання наступних об’єктів GameResult. Результат додавання даних до проекту зображено на рисунку 4.9. Оскільки BaaS-платформа надає REST API інтерфейс для інтеграції у мобільні та веб-додатки додатки, використовуємо веб-сторінку документованого REST API сервісу (рисунок 4.10) для додавання та редагування об’єктів.

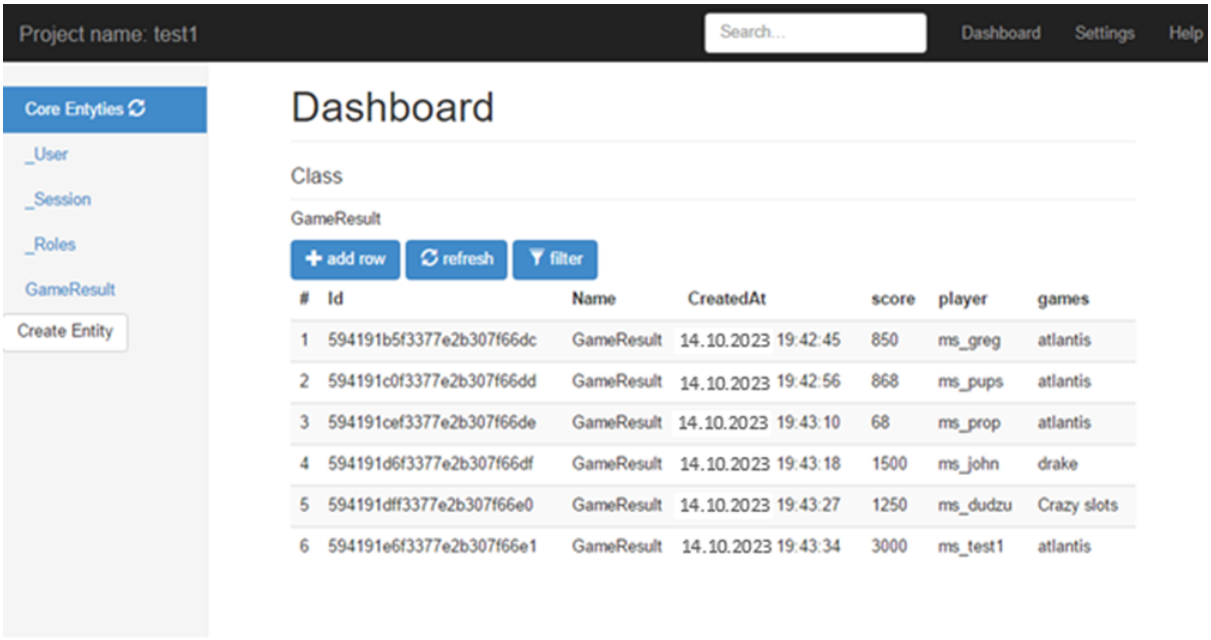


Рисунок 4.9 – Веб-сторінка керування об’єктами проекту

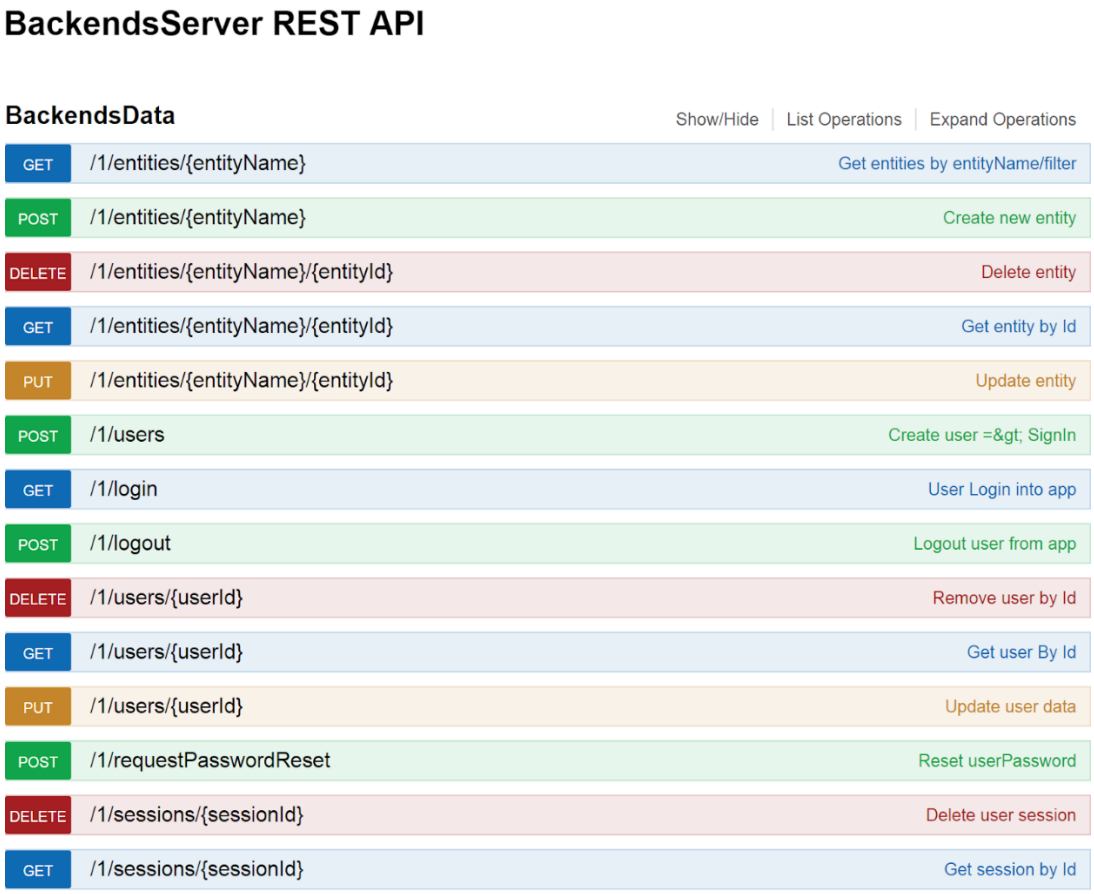


Рисунок 4.10 – Веб-сторінка документованого REST API сервісу

4.3 Розробка інструкції користувача

Платформа BaaS розроблена на мові програмування C# 8 з використанням бібліотек asp.net в середовищі Microsoft Visual Studio 2022. Робота сервісної частини здійснюється на web-сервері IIS під управлінням операційній системі Windows. Платформа спроектована, як мікросервісна застосування з використанням HTTP протоколу для взаємодії з клієнтськими додатками.

Мінімальні характеристики сервера платформи повинні відповідати наступним значення16 GB RAM, 4 CPU i7 2.5GHz, 250 GB HDD. Для коректної роботи платформи також необхідною умовою є встановлення бібліотеки .Net 6, Windows Server 2016. Перелік необхідних файлів платформи наведено рисунку 4.11.

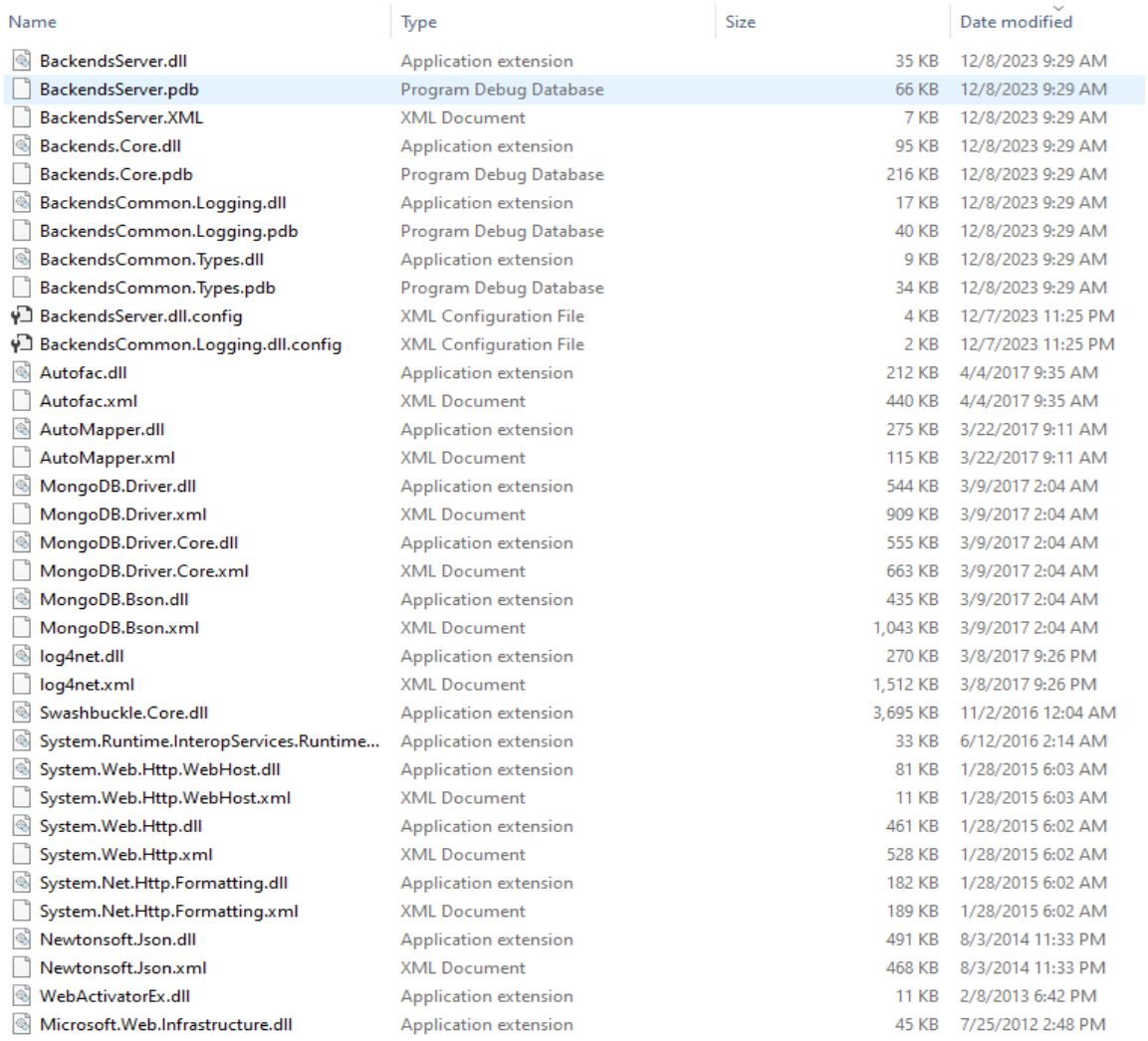
Для встановлення платформи необхідно виконати наступні кроки

1. Встановити MongoDB 3.4
2. Встановити веб-сервер IIS 7.5
3. Встановити Web Deploy 3.6 [18]
4. Імпортувати в IIS Web Deploy пакет BackendsServer.zip, як зазначено нижче.

Імпорт пакету:

1. Зробити резервну копію системи IIS 7.5 перед встановленням виконавши наступну команду в консолі:
   %windir%\system32\inetsrv\appcmd add backup "PreMsDeploy"
2. Відкрити IIS Manager через Start > Run командою inetmgr.
3. У IIS Manager вибрати Default Web Site у вузлі Server > Sites.
4. У правій панелі Actions, виберіть Import Application посилання для запуску майстра установки.
5. Вибрати пакет BackendsServer.zip
6. На сторінці Parameters введіть нове ім'я додатка.
7. Натисніть кнопку Next, щоб встановити пакет.

8. Після закінчення установки відкриється сторінка Summary, яка покаже список встановлених елементів з пакета з деталями на вкладці Details.



Name	Type	Size	Date modified
BackendsServer.dll	Application extension	35 KB	12/8/2023 9:29 AM
BackendsServer.pdb	Program Debug Database	66 KB	12/8/2023 9:29 AM
BackendsServer.XML	XML Document	7 KB	12/8/2023 9:29 AM
Backends.Core.dll	Application extension	95 KB	12/8/2023 9:29 AM
Backends.Core.pdb	Program Debug Database	216 KB	12/8/2023 9:29 AM
BackendsCommon.Logging.dll	Application extension	17 KB	12/8/2023 9:29 AM
BackendsCommon.Logging.pdb	Program Debug Database	40 KB	12/8/2023 9:29 AM
BackendsCommon.Types.dll	Application extension	9 KB	12/8/2023 9:29 AM
BackendsCommon.Types.pdb	Program Debug Database	34 KB	12/8/2023 9:29 AM
BackendsServer.dll.config	XML Configuration File	4 KB	12/7/2023 11:25 PM
BackendsCommon.Logging.dll.config	XML Configuration File	2 KB	12/7/2023 11:25 PM
Autofac.dll	Application extension	212 KB	4/4/2017 9:35 AM
Autofac.xml	XML Document	440 KB	4/4/2017 9:35 AM
AutoMapper.dll	Application extension	275 KB	3/22/2017 9:11 AM
AutoMapper.xml	XML Document	115 KB	3/22/2017 9:11 AM
MongoDB.Driver.dll	Application extension	544 KB	3/9/2017 2:04 AM
MongoDB.Driver.xml	XML Document	909 KB	3/9/2017 2:04 AM
MongoDB.Driver.Core.dll	Application extension	555 KB	3/9/2017 2:04 AM
MongoDB.Driver.Core.xml	XML Document	663 KB	3/9/2017 2:04 AM
MongoDB.Bson.dll	Application extension	435 KB	3/9/2017 2:04 AM
MongoDB.Bson.xml	XML Document	1,043 KB	3/9/2017 2:04 AM
log4net.dll	Application extension	270 KB	3/8/2017 9:26 PM
log4net.xml	XML Document	1,512 KB	3/8/2017 9:26 PM
Swashbuckle.Core.dll	Application extension	3,695 KB	11/2/2016 12:04 AM
System.Runtime.InteropServices.Runtime...	Application extension	33 KB	6/12/2016 2:14 AM
System.Web.Http.WebHost.dll	Application extension	81 KB	1/28/2015 6:03 AM
System.Web.Http.WebHost.xml	XML Document	11 KB	1/28/2015 6:03 AM
System.Web.Http.dll	Application extension	461 KB	1/28/2015 6:02 AM
System.Web.Http.xml	XML Document	528 KB	1/28/2015 6:02 AM
System.Net.Http.Formatting.dll	Application extension	182 KB	1/28/2015 6:02 AM
System.Net.Http.Formatting.xml	XML Document	189 KB	1/28/2015 6:02 AM
Newtonsoft.Json.dll	Application extension	491 KB	8/3/2014 11:33 PM
Newtonsoft.Json.xml	XML Document	468 KB	8/3/2014 11:33 PM
WebActivatorEx.dll	Application extension	11 KB	2/8/2013 6:42 PM
Microsoft.Web.Infrastructure.dll	Application extension	45 KB	7/25/2012 2:48 PM

Рисунок 4.11 – Перелік файлів BaaS-платформи необхідний файлів для коректної роботи

Аналогічні процес імпорту потрібно провести з пакетом BackendsDashboard.zip web-застосування для управління даними та проектами платформи.

Перед запуском сервісів платформи необхідно провести необхідні налаштування встановлених програм та бібліотек.

1. Конфігурація доступу до бази даних.

Необхідно задати доступ до файлів бази MongoDB через файл конфігурації mongod.cfg, який зберігається за шляхом X:

systemLog:

destination: file

path: "C:\\data\\db\\log\\mongo.log"

logAppend: true

storage:

dbPath: "C:\\data\\db"

Виконати наступну команду для запуску сервера з відповідною конфігурацією:

“<Шлях установки MongoDB>/mongod.exe”--config X\mongod.cfg

2. Встановити URL бази даних для web-сервісів.

Файл конфігурації web.config містить задані за замовчуванням параметри шляху до MongoDB. Тому у випадку встановлення серверу бази даних та BaaS платформи на одному і тому ж сервері конфігурація не потребує змін. Якщо ж база даних була створена на іншому сервері, тоді необхідно змінити URL відповідного сервера в ключі конфігурації <connectionStringMongoDB> файлу web.config. Також необхідно переконатися, що зазначений в URL DNS ім’я та порт є доступними з сервера який їх викликає. У випадку відсутності доступу необхідно створити відповідний запис на DNS-сервері або використовувати IP-адресу сервера та відкрити доступ до визначеного порту.

3. Встановити шлях до файлів логування

За замовчуванням вся інформація для логування, а також рівень логування визначаються в секції <log4net> файлу web.config.

Значення, які задані за замовчуванням:

- спосіб логування в файл;
- шлях до лог-файлів “D:/Backends/BackendsServer.<date-time>.log”;
- рівень логування —“INFO”;

При необхідності створення іншої конфігурації потрібно змінити відповідні значення.

4.4 Висновки

Уданому розділі було проведено симуляцію та тестування адаптивного алгоритму повторної передачі повідомлень в BaaS-платформі. Проведено функціональне тестування серверної частини, засноване на принципах «піраміди тестування». Визначено інструкції користувача та системні вимоги для розгортання сервісів BaaS-платформи.

5 ЕКОНОМІЧНА ЧАСТИНА

5.1 Оцінювання комерційного потенціалу розробки

Метою проведення комерційного аудиту є оцінювання комерційного потенціалу впровадження відмовостійких методів передавання повідомлень та розподіленої BaaS-платформи для мобільних та веб-застосувань.

Для проведення технологічного аудиту було залучено 3-х незалежних експертів Вінницького національного технічного університету к.т.н., доцента Майданюка В. П., к.т.н., доцента Ракитянську Г. Б., к.т.н., доцента Рейди О.М з кафедри програмного забезпечення.

Аудит науково-технічної розробки та її комерційного потенціалу проведено за допомогою таблиці 5.1, застосовуючи п’ятибальну шкалу оцінювання за 12-ма критеріями оцінки.

Таблиця 5.1 – Рекомендовані критерії оцінювання комерційного потенціалу розробки та їх можлива бальна оцінка

Критерії оцінювання та бали (за 5-ти бальною шкалою)					
#	0	1	2	3	4
1	2	3	4	5	6
Технічна здійсненність концепції:					
1	Достовірність концепції не підтверджена	Концепція підтверджена експертними висновками	Концепція підтверджена розрахунками	Концепція перевірена на практиці	Перевірено роботоздатність продукту в реальних умовах
Ринкові переваги (недоліки):					
2	Багато аналогів на малому ринку	Мало аналогів на малому ринку	Кілька аналогів на великому ринку	Один аналог на великому ринку	Продукт не має аналогів на великому ринку
3	Ціна продукту значно вища за ціни аналогів	Ціна продукту дещо вища за ціни аналогів	Ціна продукту приблизно дорівнює цінам аналогів	Ціна продукту дещо нижче за ціни аналогів	Ціна продукту значно нижче за ціни аналогів
4	Технічні та споживчі властивості продукту значно гірші, ніж в аналогів	Технічні та споживчі властивості продукту трохи гірші, ніж в аналогів	Технічні та споживчі властивості продукту на рівні аналогів	Технічні та споживчі властивості продукту трохи кращі, ніж в аналогів	Технічні та споживчі властивості продукту значно кращі, ніж в аналогів

Продовження таблиці 5.1.

1	2	3	4	5	6
5	Експлуатаційні витрати значно вищі, ніж в аналогів	Експлуатаційні витрати дещо вищі, ніж в аналогів	Експлуатаційні витрати на рівні експлуатаційних витрат аналогів	Експлуатаційні витрати трохи нижчі, ніж в аналогів	Експлуатаційні витрати значно нижчі, ніж в аналогів
Ринкові перспективи					
6	Ринок малий і не має позитивної динаміки	Ринок малий, але має позитивну динаміку	Середній ринок з позитивною динамікою	Великий стабільний ринок	Великий ринок з позитивною динамікою
7	Активна конкуренція великих компаній на ринку	Активна конкуренція	Помірна конкуренція	Незначна конкуренція	Конкурентів немає
Практична здійсненність					
8	Відсутні фахівці як з технічної, так і з комерційної реалізації ідеї	Необхідно наймати фахівців або витрачати значні кошти та час на навчання наявних фахівців	Необхідне незначне навчання фахівців та збільшення їх штату	Необхідне незначне навчання фахівців	Є фахівці з питань як з технічної, так і з комерційної реалізації ідеї
9	Потрібні значні фінансові ресурси, які відсутні. Джерела фінансування ідеї відсутні	Потрібні незначні фінансові ресурси. Джерела фінансування відсутні	Потрібні значні фінансові ресурси. Джерела фінансування є	Потрібні незначні фінансові ресурси. Джерела фінансування є	Не потребує додаткового фінансування
10	Необхідна розробка нових матеріалів	Потрібні матеріали, що використовуються у військово-промисловому комплексі	Потрібні дорогі матеріали	Потрібні досяжні та дешеві матеріали	Всі матеріали для реалізації ідеї відомі та давно використовуються у виробництві
11	Термін реалізації ідеї більший за 10 років	Термін реалізації ідеї більший за 5 років. Термін окупності інвестицій більше 10-ти років	Термін реалізації ідеї від 3-х до 5-ти років. Термін окупності інвестицій більше 5-ти років	Термін реалізації ідеї менше 3-х років. Термін окупності інвестицій від 3-х до 5-ти років	Термін реалізації ідеї менше 3-х років. Термін окупності інвестицій менше 3-х років
12	Необхідна розробка регламентних документів та отримання великої кількості дозвільних документів на виробництво та реалізацію продукту	Необхідно отримання великої кількості дозвільних документів на виробництво та реалізацію продукту, що вимагає значних коштів та часу	Процедура отримання дозвільних документів для виробництва та реалізації продукту вимагає незначних коштів та часу	Необхідно тільки повідомлення відповідним органам про виробництво та реалізацію продукту	Відсутні будь-які регламентні обмеження на виробництво та реалізацію продукту

В таблиці 5.2 наведено результати оцінювання науково-технічного рівня і комерційного потенціалу розробки.

Таблиця 5.2 – Результати оцінювання комерційного потенціалу розробки

Критерії	Прізвище, ініціали, посада експерта		
	1. Майданюк В. П.	2. Ракитянська Г. Б.	3. Рейда О. М.
	Бали, виставлені експертами:		
1	3	4	3
2	2	3	3
3	4	3	3
4	2	3	1
5	3	4	3
6	3	3	3
7	4	2	3
8	3	4	2
9	2	3	4
10	4	4	5
11	3	5	4
12	5	4	3
Сума балів	$СБ_1$=38	$СБ_2$=42	$СБ_3$=37
Середньоарифметична сума балів $\overline{СБ}$	$\overline{СБ} = \frac{\sum_1^3 СБ_i}{3} = \frac{38+42+37}{3} = 39$		

В таблиці 5.3 наведено шкалу оцінки комерційного потенціалу розробки.

Таблиця 5.3 – Рівні комерційного потенціалу розробки

Середньоарифметична сума балів СБ, розрахована на основі висновків експертів	Рівень комерційного потенціалу розробки
0-10	Низький
11-20	Нижче середнього
21-30	Середній
31-40	Вище середнього
41-48	Високий

За результатами розрахунків, наведених в таблиці 5.2 та шкалою оцінки наведеної в таблиці 5.3 можна зробити висновок щодо рівня комерційного потенціалу розробки. Середньоарифметична сума балів, виставлених експертами склала 39, що відповідає рівню «вище середнього».

Впровадження розробки відмовостійких методів передавання повідомлень та розподіленої Backend as a Service платформи для мобільних та веб-застосувань сприяє значному зниженню витрат на утримання та управління IT-інфраструктурою. Це особливо важливо для великих організацій, таких як корпорації та державні установи, які залежать від надійних та ефективних систем обробки даних. Завдяки цій розробці можливо істотно скоротити витрати на технічне обслуговування, утримання серверів, а також зменшити енергоспоживання, пов'язане з роботою великих дата-центрів. Використання розподіленої BaaS платформи також сприяє оптимізації робочого часу розробників, оскільки вони отримують доступ до готових до використання рішень. Це дозволяє зосередитись на специфічних задачах, пов'язаних з розвитком бізнесу, а не на технічному утриманні інфраструктури.

5.2 Прогнозування витрат на виконання науково-дослідної роботи

Витрати, пов’язані з проведенням науково-дослідної роботи можна розрахувати за наступними статтями:

- Витрати на оплату праці;
- Витрати на інструментальне та інфраструктурне програмне забезпечення;
- Амортизаційні відрахування;
- Енергія для науково-виробничих цілей.

5.2.1 Основна заробітна плата

Заробітна плата кожного із залучених осіб визначається за формулою (5.1)

$$З_0 = \sum_{i=1}^{K} \frac{M_{ni} * t_i}{T_p} \text{ (грн)} \qquad (5.1)$$

Де k – кількість посад працівників, залучених до процесу дослідження і розробки;

M_{ni} – місячний посадовий оклад конкретного розробника (інженера, дослідника, науковця тощо), грн.;

$Т_р$– число робочих днів в місяці; приблизно $Т_р = 22$;

t - кількість робочих днів роботи працівника.

Для проектування і розробки розподіленої BaaS-платформи для мобільних та веб-застосувань було залучено наступних робітників: Solution Architect, Senior Software Engineer, DevOps Engineer, та Quality Assurance Engineer. Посадові оклади, число днів роботи та витрати на компенсацію наведено в таблиці 5.4

Таблиця 5.4 – Компенсація спеціаліста в дослідницькій установі

Найменування посади	Місячний посадовий оклад, грн.	Оплата за робочий день, грн.	Число днів роботи	Витрати на заробітну плату грн.
Solution Architect	57600	2880	10	28800
Senior Software Engineer	48000	2400	80	192000
DevOps Engineer	44960	2248	60	134880
Quality Assurance Engineer	28960	1448	30	43440
Всього				399120

5.2.2 Розрахунок додаткової заробітної плати робітників

Додаткова заробітна плата $З_д$ всіх робітників, які приймали участь в розробці нового технічного рішення розраховується за формулою (5.2) як 10 - 15 % від основної заробітної плати робітників як премія.

На даному підприємстві додаткова заробітна плата нараховується в розмірі 10% від основної заробітної плати.

$$З_д = (З_о + З_р) * \frac{Н_{дод}}{100\%} \quad (5.2)$$

$$З_д = 0{,}1 * 399120 = 39912 \text{ (грн)}$$

5.2.3 Нарахування на заробітну плату

Нарахування на заробітну плату $Н_{ЗП}$ робітників, які брали участь у виконанні роботи, розраховуються за формулою (5.3):

$$З_н = (З_о + З_р + З_д) * \frac{Н_{ЗП}}{100} \text{ (грн)} \quad (5.3)$$

де $З_о$ – основна заробітна плата розробників, грн.;

$З_д$ – додаткова заробітна плата всіх розробників та робітників, грн.;

$Н_{ЗП}$ – ставка єдиного внеску на загальнообов'язкове державне соціальне страхування, % .

Основна ставка єдиного внеску на загальнообов'язкове державне соціальне страхування на 2023 рік – 22 %, тоді:

$$З_н = (399120 + 39912) * 0.22 = 96587(грн)$$

5.2.4 Витрати на матеріали та комплектуючі вироби

Дана стаття витрат включає витрати на матеріали, пристрої, засоби, які використовують при виготовленні одиниці продукції. Розраховуються, згідно їх номенклатури, за формулою:

$$К=\sum_{i=1}^{n} Н_i \cdot Ц_i \cdot К_i , \quad (5.4)$$

де $Н_i$ – кількість комплектуючих *i*-го виду, шт.;

$Ц_i$ – покупна ціна комплектуючих *i*-го найменування, грн.;

$К_i$ – коефіцієнт транспортних витрат (1,1…1,15).

Потрібно закладати витрати на доставку у вигляді коефіцієнту транспортних витрат – 1.1. Інформацію про використані матеріали та комплектуючі наведено у таблиці 5.5.

Таблиця 5.5 – Матеріали, що використані на розробку

Найменування матеріалу	Ціна за одиницю, грн.	Витрачено	Вартість витраченого матеріалу, грн.
Дошка магнітно-маркерна Buromax 90x60 см	548	1	548
Набір маркерів кольорових Xiaomi 3 шт.	239	1	239
Папір офісний A4 Xerox ECF 500 аркушів	205	1	205
Тонер ColorWay для принтера HP LaserJet 1018	129	1	129
Мережеве сховище Synology DS620Slim	21023	1	21023
Всього			22144
З врахуванням коефіцієнта транспортування			24358

5.2.5 Витрати на програмне забезпечення

До даної статті входять витрати на програмне забезпечення, необхідне для проектування та розробки відмовостійких методів передавання повідомлень та розподіленої BaaS-платформи для мобільних та веб-застосувань. З метою оптимізації витрат на інфраструктуру, було прийнято рішення використовувати платформу хмарних – MS Azure. Балансову вартість програмного забезпечення розраховують за формулою 5.5:

$$В_{прг} = \sum_{i=1}^{k} Ц_{іпрг} \cdot С_{пргі} \cdot К_і, \quad (5.5)$$

де $Ц_{іпрг}$ – ціна придбання/використання одиниці програмного засобу цього виду, грн;

$С_{пргі}$ – кількість одиниць програмного забезпечення відповідного найменування, які придбані для проведення досліджень, шт.;

$К_і$ – коефіцієнт, що враховує інсталяцію, налагодження програмного засобу тощо ($К_і$ = 1,10…1,12).

k – кількість найменувань програмних засобів.

Отримані результати наведено в таблиці 5.6.

Таблиця 5.6 – Витрати на використання програмних

Найменування устаткування	Час використання, місяців	Ціна за місяць, грн	Вартість, грн
Підписка Visual Paradigm Enterprise	1	3660	3660
Підписка Microsoft Visual Studio Professional 2022	3	1647	4941
Azure SQL Server Standard	3	5000	15000
Azure App Service	3	2000	6000
Всього			29601

5.2.6 Амортизація обладнання

До даної статті включаються амортизаційні відрахування по кожному виду обладнання, устаткування яке використовувалось для проектування та розробки BaaS-платформи.

$$А_{обл} = \frac{Ц_б}{Т_в} * \frac{t_{вик}}{12} \quad (5.6)$$

де $Ц_б$ – балансова вартість даного виду обладнання (приміщень), грн.;

$t_{вик}$ – час користування;

$Т_в$ – термін використання обладнання (приміщень), цілі місяці.

Для розробки продукту використовувався персональний комп'ютер вартістю 60000 грн. Для тестування використовувався ноутбук вартістю 30000 грн. Амортизаційні відрахування наведено в таблиці 5.7.

Таблиця 5.7 – Амортизаційні відрахування по кожному виду обладнання

Найменування обладнання	Балансова вартість, грн	Строк корисного використання, років	Термін використання обладнання, місяців	Амортизаційні відрахування, грн
Комп'ютер	60000	5	3	3000
Ноутбук	30000	4	3	1875
Офісне приміщення	1600000	30	3	13333
Всього				18208

5.2.7 Енергія для науково-виробничих цілей

До даної статті відносяться витрати на всі види палива й енергії, що безпосередньо використовуються з технологічною метою на проведення досліджень.

$$В_е = \sum_{i=1}^{n} \frac{W_{yt} \cdot t_i \cdot Ц_е \cdot К_{впі}}{\eta_i} \quad (5.7)$$

де W_{yt} – встановлена потужність обладнання на певному етапі розробки, кВт;

t_i – тривалість роботи обладнання, год;

$Ц_е$ – вартість 1 кВт-години електроенергії, грн;

$К_{впі}$ – коефіцієнт, що враховує використання потужності, $К_{впі} < 1$;

$\eta_і$ – коефіцієнт корисної дії обладнання, $\eta_і < 1$.

Вартість електроенергії становить 7.6 грн за кВт в 2023 році. При розробці системи використовувалось комп'ютер та ноутбук з сумарною потужністю 1.5 кВт, освітлення, вентиляція та кондиціювання має сумарну потужність 0.5 кВт. Сумарні витрати на електроенергію становлять:

$$В_е = \frac{(1.5 + 0.5) \cdot 720 \cdot 7.6 \cdot 0.45}{0.76} = 6480 \text{ грн}$$

5.2.8 Службові відрядження

Стаття включає витрати на відрядження штатних працівників, працівників організацій, які працюють за договорами цивільно-правового характеру, аспірантів, зайнятих розробленням досліджень, відрядження, пов'язані з проведенням випробувань машин та приладів, а також витрати на відрядження на наукові з'їзди, конференції, наради, пов'язані з виконанням конкретних досліджень. Витрати за статтею «Службові відрядження» розраховуються як 20…25% від суми основної заробітної плати дослідників та робітників за формулою 5.8:

$$В_{св} = \left(З_о + З_р\right) \cdot \frac{Н_{св}}{100} \qquad (5.8)$$

де $Н_{св}$ – норма нарахування за статтею «Інші витрати».

$$В_{св} = 399120 * 0.2 = 79824 \text{ грн}$$

5.2.9 Витрати на роботи, які виконують сторонні підприємства, установи і організації

Дана стаття витрат включає витрати на проведення досліджень, що не можуть бути виконані штатними працівниками або наявним обладнанням організації, а виконуються на договірній основі іншими підприємствами, установами і організаціями незалежно від форм власності та позаштатними працівниками. Такі витрати розраховуються як 30…45% від суми основної заробітної плати дослідників та робітників за формулою 5.9.

$$В_{сп} = \left(З_о + З_р\right) \cdot \frac{Н_{сп}}{100} \quad (5.9)$$

де $Н_{пв}$ – норма нарахування за статтею «Витрати на роботи, які виконують сторонні підприємства, установи і організації».

$$В_{сп} = 399120 * 0.3 = 119736 \text{ грн}$$

5.2.10 Інші витрати

До статті «Інші витрати» належать витрати, які не знайшли відображення у попередніх статтях витрат і можуть бути віднесені безпосередньо на собівартість розробки за прямими ознаками.

Витрати за статтею «Інші витрати» розраховуються як 50…100% від суми основної заробітної плати робітників за формулою 5.10

$$І_в = \left(З_о + З_р\right) \cdot \frac{Н_{ів}}{100} \quad (5.10)$$

де $Н_{ів}$ – норма нарахування за статтею «Інші витрати».

$$І_в = 399120 * 0.5 = 199560 \text{ грн}$$

5.2.11 Загальновиробничі витрати

Дана стаття витрат охоплює витрати на управління організацією, оплату службових відряджень, витрати на утримання, ремонт та експлуатацію основних

засобів, витрати на опалення, освітлення, водопостачання, охорону праці тощо. Накладні (загальновиробничі) витрати $B_{нзв}$ можна прийняти як 120% від суми основної заробітної плати розробників та робітників, які виконували дану МКНР, тобто:

$$\text{В}_{\text{нзв}} = \left(\text{З}_{\text{о}} + \text{З}_{\text{р}}\right) \cdot \frac{\text{Н}_{\text{нзв}}}{100}, \quad (5.11)$$

де $\text{Н}_{\text{нзв}}$ – норма нарахування за статтею «Загальновиробничі витрати».

$$\text{В}_{\text{нзв}} = 399120 * 1.2 = 478944 \text{ грн}$$

5.2.12 Загальні витрати

Сума всіх статей витрат дає в результаті витрати на проведення дослідження та розробку розподіленої BaaS-платформи для мобільних та веб-застосувань розраховується за формулою:

$$\text{В} = \text{З}_{\text{о}} + \text{З}_{\text{р}} + \text{З}_{\text{дод}} + \text{З}_{\text{н}} + \text{М} + \text{К}_{\text{в}} + \text{В}_{\text{спец}} + \text{В}_{\text{прг}} + \text{А}_{\text{обл}} + \text{В}_{\text{е}} + \text{В}_{\text{св}} + \text{В}_{\text{сп}} + \text{І}_{\text{в}} + \text{В}_{\text{нзв}} \quad (5.12)$$

$$\text{В} = 399120 + 24358 + 96587 + 29601 + 18208 + 6480 + 79824 + 119736 + 199560 + 478944 = 1452418 \text{ грн}$$

Загальні витрати ЗВ на завершення роботи та оформлення її результатів розраховуються за формулою:

$$\text{ЗВ} = \frac{\text{В}}{\eta}, \quad (5.13)$$

де η – коефіцієнт, який характеризує стадію виконання даної НДР.

Оскільки, робота знаходиться на стадії дослідного зразка, то коефіцієнт β = 0.5.

Звідси:

$$\text{ЗВ} = \frac{1452418}{0.5} = 2904836 \text{ грн.}$$

5.3 Розрахунок економічної ефективності науково-технічної розробки

Економічна ефективність дозволяє спрогнозувати чистий прибуток, який може бути отриманий від впровадження розробленої системи.

При розрахунку економічної ефективності потрібно обов'язково враховувати зміну вартості грошей у часі, оскільки від вкладення інвестицій до отримання прибутку минає чимало часу.

Для розрахунку збільшення чистого прибутку підприємства $\Delta П_i$, для кожного із років, протягом яких очікується отримання позитивних результатів від впровадження розробки використовується формула формулою 5.14:

$$\Delta П_i = \sum_{1}^{n} \left(\Delta Ц_o \cdot N + Ц_o \cdot \Delta N \right)_i \cdot \lambda \cdot \rho \cdot \left(1 - \frac{\upsilon}{100} \right) \quad (5.14)$$

де $\Delta Ц_0$ – покращення основного оціночного показника від впровадження результатів розробки у даному році.

N – основний кількісний показник, який визначає діяльність підприємства у даному році до впровадження нової розробки;

ΔN – покращення основного кількісного показника діяльності підприємства від впровадження результатів розробки:

Цо – основний оціночний показник, який визначає діяльність підприємства у даному році після впровадження результатів розробки;

n – кількість років, протягом яких очікується отримання позитивних результатів від впровадження розробки:

λ – коефіцієнт, який враховує сплату податку на додану вартість. Ставка податку на додану вартість дорівнює 20%, а коефіцієнт *л* = 0,8333.

ρ – коефіцієнт, який враховує рентабельність продукту. р = 0,25;

ϑ – ставка податку на прибуток. У 2023 році – 18%.

Впровадження розроблених відмовостійких методів передавання повідомлень та розподіленої BaaS-платформи для мобільних та веб-застосувань може суттєво вплинути на роботу та доходи бізнесу, який активно використовує

цифрові технології. Розглянемо впровадження цих розробок на прикладі середнього ІТ-компанії, яка спеціалізується на розробці мобільних додатків.

Припустимо, що компанія працює з клієнтами, чия середня плата за проект становить 1825000 грн. Розробка відмовостійких методів забезпечить стабільність та надійність передачі даних, що є критично важливим для клієнтів, особливо в галузях електронної комерції та фінансових технологій.

З впровадженням BaaS-платформи, розробка стає швидшою та ефективнішою, що знижує витрати на розробку на 20%. При середній вартості проекту у 1825000 грн, економія складе 365000 грн. Припустимо, що компанія реалізує 40 проектів на рік, загальна економія складе 14600000 грн.

Завдяки підвищеній ефективності та надійності, компанія може привабити більше клієнтів. Якщо впровадження цих розробок збільшить кількість проектів на 10% в перший рік, це додаткові 4 проекти, або 7300000 грн доходу. З урахуванням зростання на 15% у другий рік (6 проектів) і на 20% у третій (8 проектів).

$$\Delta П_1 = [365000 \cdot 40 + (1825000 + 365000) \cdot 4] \cdot 0{,}833 \cdot 0{,}25 \cdot \left(1 - \frac{18}{100}\right)$$

$$= 23360000 \cdot 0.170765 = 3989070 = 3.98 \text{ млн}$$

$$\Delta П_2 = [365000 \cdot 40 + (1825000 + 365000) \cdot (4 + 6)] \cdot 0{,}833 \cdot 0{,}25 \cdot \left(1 - \frac{18}{100}\right)$$

$$= 36500000 \cdot 0.170765 = 6232922 = 6.23 \text{ млн}$$

$$\Delta П_3 = [365000 \cdot 40 + (1825000 + 365000) \cdot (4 + 6 + 8)] \cdot 0{,}833 \cdot 0{,}25$$

$$\cdot \left(1 - \frac{18}{100}\right) = 54020000 \cdot 0.170765 = 9224725 = 9.22 \text{ млн}$$

Далі за формулою 5.15 розраховують приведену вартість збільшення всіх чистих прибутків ПП, що їх може отримати потенційний інвестор від можливого впровадження та комерціалізації розробки:

$$ПП = \sum_{i=1}^{T} \frac{\Delta П_i}{(1+\tau)^t} \quad (5.15)$$

де ΔПі – збільшення чистого прибутку у кожному з років, протягом яких виявляються результати впровадження науково-технічної розробки, грн;

Т – період часу, протягом якого очікується отримання позитивних результатів від впровадження та комерціалізації розробки, роки;

τ – ставка дисконтування, за яку можна взяти щорічний прогнозований рівень інфляції в країні, τ =0,05…0,15. В Україні рівень інфляції за підсумком 2023 року склав 10.6%, прогнозований рівень на 2024 рік – 8.5% ;

t – період часу (в роках) від моменту початку впровадження розробки до моменту отримання потенційним інвестором додаткових чистих прибутків у цьому році.

$$ПП = \frac{3989070}{1+0.1} + \frac{6232922}{(1+0.085)^2} + \frac{9224725}{(1+0.085)^3} = 3626427 + 5294588 + 7222111 =$$

16143126 грн

За формулою 5.16 необхідно розрахувати величину початкових інвестицій PV, які потенційний інвестор має вкласти для впровадження і комерціалізації розробки.

$$PV = k_{інв} \cdot ЗВ \quad (5.16)$$

де $k_{інв}$ – коефіцієнт, що враховує витрати інвестора на впровадження науково-технічної розробки та її комерціалізацію. Це можуть бути витрати на підготовку приміщень, розробку технологій, навчання персоналу, маркетингові заходи тощо; зазвичай $k_{інв}$ =2...5, але може бути і більшим;

ЗВ – загальні витрати на проведення науково-технічної розробки та оформлення її результатів, грн.

$$PV = 2 \cdot 2904836 = 5809672 \text{ грн}$$

Тоді абсолютний економічний ефект $E_{абс}$ або чистий приведений дохід (NPV, Net Present Value) для потенційного інвестора від можливого впровадження та комерціалізації розробки становитиме:

$$E_{абс} = ПП - PV \quad (5.17)$$

де ПП – приведена вартість зростання всіх чистих прибутків від можливого впровадження та комерціалізації науково-технічної розробки, грн;

PV – теперішня вартість початкових інвестицій, грн.

$$E_{\text{абс}} = 16143126 - 5809672 = 10333454 \text{ грн}$$

Оскільки величина економічного ефекту має велике додатне значення, це свідчить про потенційну зацікавленість інвесторів у впровадженні та комерціалізацію розробки.

5.4 Розрахунок ефективності вкладених інвестицій та періоду їх окупності

Для остаточного прийняття рішення про впровадження розробки та виведення її на ринок необхідно розрахувати внутрішню економічну дохідність $E_в$ або показник внутрішньої норми дохідності (IRR, Internal Rate of Return) вкладених інвестицій та порівняти її з бар'єрною ставкою дисконтування, яка визначає ту мінімальну внутрішню економічну дохідність, нижче якої інвестиції в будь-яку розробку вкладати буде економічно недоцільно.

Розрахуємо відносну (щорічну) ефективність вкладених в наукову розробку інвестицій $E_в$. Для цього користуються формулою 5.18:

$$E_в = \sqrt[T_ж]{1 + \frac{E_{абс}}{PV}} - 1, \tag{5.18}$$

де $E_{абс}$ – абсолютний економічний ефект вкладених інвестицій, грн; PV – теперішня вартість початкових інвестицій, грн;

$T_ж$ – життєвий цикл науково-технічної розробки, тобто час від початку її розробки до закінчення отримування позитивних результатів від її впровадження, роки.

$$E_в = \sqrt[3]{1 + \frac{10333454}{5809672}} - 1 = \sqrt[3]{2{,}77} - 1 = 1.40 - 1 = 0.40 = 40\%$$

Визначимо мінімальну ставку дисконтування, яка у загальному вигляді визначається за формулою 5.19:

$$\tau = d + f, \tag{5.19}$$

де d – середньозважена ставка за депозитними операціями в комерційних банках; в 2023 році в Україні d= (0,14…0,2);

f − показник, що характеризує ризикованість вкладень; зазвичай, величина f = (0,05...0,1).

$$\tau_{min} = 0.17 + 0.07 = 0.24$$

Так як $E_в > \tau_{min}$ то інвестор може бути зацікавлений у фінансуванні даної наукової розробки.

Далі розраховується період окупності інвестицій, вкладених у реалізацію проекту за формулою 5.20.

$$T_{ок} = \frac{1}{E_в} \tag{5.20}$$

де Ев – внутрішня економічна дохідність вкладених інвестицій.

$$\text{Т}_{\text{ок}} = \frac{1}{0.40} = 2.5 \text{ роки} = 30 \text{ міс} = 2 \text{ роки та } 6 \text{ міс}$$

Так як $\text{Т}_{\text{ок}} \leq 3$ років, то це свідчить про комерційну привабливість науково-технічної розробки і може спонукати потенційного інвестора профінансувати впровадження цієї розробки та виведення її на ринок.

5.5 Висновки

В даному розділі було проведено оцінку комерційного потенціалу розробки програмного забезпечення відмовостійких методів передавання повідомлень та розподіленої BaaS-платформи для мобільних та веб-застосувань. Економічний потенціал склав значення, яке є вище середнього.

Також було зпрогнозовано витрати на проектування та реалізацію системи за необхідними статтями витрат, які склали 1452418 грн. Загальна величина витрат склала 2904836 грн. Обрахунок економічної ефективності та терміну окупності вкладених інвестицій показали, що розробка є економічно доцільною та привабливою для потенційних інвесторів. Термін окупності інвестицій складає 2 роки та 6 місяців а прогнозований прибуток за 3 роки 16143126 грн.

ВИСНОВКИ

В даній магістерській кваліфікаційної роботі було розроблено комплексний підхід до створення адаптивного алгоритму для повторних спроб у BaaS-платформі, який включав аналіз даних, проектування архітектури, вибір технологій та інструментів, а також імплементацію та тестування системи. Також розроблено BaaS сервіси, які надають API для управління даними та користувачами розробникам мобільних та web-додатків без необхідності створення серверної частини.

Проведений детальний аналіз предметної області, та розглянуто основні аспекти мікросервісної архітектури та адаптивних алгоритмів з використанням підходів теорії керування. Визначено, що використання мови програмування C# та .NET 6 є оптимальним для розробки BaaS платформи, а Python зі спеціалізованими бібліотеками ідеально підходить для розробки, симуляції та аналізу адаптивного алгоритму.

Розроблена архітектура BaaS сервісів забезпечила надійну основу для подальшої імплементації адаптивного алгоритму. Зокрема, було реалізовано такі сервіси як Metric Collection Service, Analysis Service, та Adaptive Retry Service, які у сукупності формують механізм зворотного зв'язку та адаптації системи до змінюваних умов використання.

Тестування програми продемонструвало її ефективність та здатність до адаптації, підтверджуючи правильність вибраних методів та підходів. Результати тестування показали, що розроблений алгоритм здатний правильно реагувати на зміни в навантаженні та параметрах системи, забезпечуючи оптимальну працездатність та високу доступність сервісів.

Розроблена інструкція користувача забезпечує належне введення в експлуатацію та використання системи, роблячи процес інтеграції зі сторонніми сервісами зрозумілим та доступним для кінцевих користувачів.

У цілому, результати роботи відкривають шлях для подальшого розвитку та оптимізації BaaS платформ, а також знайшли практичне застосування у вигляді працездатного програмного продукту, готового до широкого впровадження.

У майбутньому розвиток BaaS платформи може бути спрямований на подальше вдосконалення адаптивного алгоритму з метою підвищення його точності та швидкодії. Зокрема, можливе впровадження більш складних механізмів машинного навчання та штучного інтелекту для прогнозування та автоматичного реагування на зміни у поведінці користувачів та навантаженні системи.

Також передбачається інтеграція з іншими хмарними платформами та сервісами для розширення можливостей BaaS, що забезпечить більш широкий спектр сервісів та функцій для кінцевих користувачів. Це може включати розробку API для легкої інтеграції з різними хмарними сервісами, такими як Amazon Web Services, Google Cloud та Microsoft Azure.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Building Standardized Systems Across an Engineering Organization. In: Production-Ready Microservices: Building Standardized Systems Across an Engineering Organization. URL: https://www.oreilly.com/library/view/productionready-microservices/9781491965962/ch04.html (дата звернення: 02.09.2023)

2. Richardson C. Circuit Breaker. URL: https://microservices.io/patterns/reliability/circuit-breaker.html (дата звернення: 02.09.2023)

3. Klein C, Maggio M, Arzén KE, Hernández-Rodriguez F. Brownout: Building more robust cloud applications. International Conference on Software Engineering, 2014, 711p

4. N. Dragoni, S. Giallorenzo, A. L. Lafuente, M. Mazzara, F. Montesi, R. Mustafin, and L. Safina, “Microservices: Yesterday, Today, and Tomorrow”, 2017, 216p.

5. A. Krylovskiy, M. Jahn, and E. Patti, “Designing a Smart City Internet of Things Platform with Microservice Architecture” 2015, 328 p.

6. B. Butzin, F. Golatowski, and D. Timmermann, “Microservices approach for the internet of things”, 2016, 7p.

7. J. Gao, W. Li, Z. Zhao, and Y. Han, “Provisioning big data applications as services on containerized cloud: a microservices-based approach”, 2020, 167p.

8. Павлюк І.А. Розробка відмовостійких методів передавання повідомлень та розподіленої BAAS-платформи для мобільних та веб-застосувань. Міжнародна науково-практична Інтернет-конференція «Електронні інформаційні ресурси: створення, використання, доступ». URL: https://drive.google.com/file/d/1eAb_QXS6CvA-e1ikKxerTqvw6P1IPRhc/view (дата звернення: 05.12.2023)

9. Lessons Learned Building a Backend-as-a-Service: A Technical Deep Dive. URL: https://medium.baqend.com/how-to-develop-a-backend-as-a-service-from-scratch-lessons-learned-a9fac618c2ce (дата звернення: 02.09.2023)

10. What is BaaS? Backend-as-a-Service vs. serverless. URL: https://www.cloudflare.com/learning/serverless/glossary/backend-as-a-service-baas/ (дата звернення: 02.09.2023).

11. What is backend as a service. URL: https://backendless.com/what-is-backend-as-a-service (дата звернення: 05.09.2023).

12. What is BaaS – Backend-as-a-Service? https://blog.back4app.com/backend-as-a-service-baas/ (дата звернення: 05.09.2023).

13. Backend as a Service (BaaS). URL: https://www.geeksforgeeks.org/backend-as-a-service-baas/ (дата звернення: 05.09.2023).

14. Resiliency in Distributed Systems. URL: https://newsletter.pragmaticengineer.com/p/resiliency-in-distributed-systems-74c (дата звернення: 10.09.2023).

15. Patterns of Distributed Systems. URL: https://martinfowler.com/articles/patterns-of-distributed-systems/ (дата звернення: 05.09.2023).

16. Roberto Vitillo. Understanding Distributed Systems: What every developer should know about large distributed applications, 2021, 253 p.

17. AWS Amplify. URL: https://aws.amazon.com/amplify/ (дата звернення: 05.09.2023).

18. Top Backend as a Service (BaaS) Platforms in 2023. URL: https://www.commoninja.com/blog/top-baas-platforms#Top-Backend-as-a-Service-Platforms-2023 (дата звернення: 05.09.2023).

19. Backend as a Service List. URL: https://george-51059.medium.com/backend-as-a-service-list-9104bdce845e (дата звернення: 05.09.2023).

20. Filieri A, Maggio M, Angelopoulos K, D'Ippolito N, Gerostathopoulos I, Hempel AB, et al. Software Engineering Meets Control Theory, 2015, 72 p.

21. Salehie M, Tahvildari L. Self-adaptive software: Landscape and research challenges, 2009, 40 p.

22. Filieri A, Maggio M, Angelopoulos K, D'Ippolito N, Gerostathopoulos I, Hempel AB, Control strategies for self-adaptive software systems, 2017, 31p.

23. Least squares estimation. In: Forecasting. URL: https://otexts.com/fpp2/least-squares.html (дата звернення: 12.10.2023).

24. How To Forecast Time-Series Using Autoregression. URL: https://towardsdatascience.com/how-to-forecast-time-series-using-autoregression-1d45db71683 (дата звернення: 12.10.2023).

25. Stationarity: Defining, Detecting, Types, and Transforming Time Series. URL: https://blog.quantinsti.com/stationarity/ (дата звернення: 12.10.2023).

26. Probabilistic Model Selection with AIC, BIC, and MDL. URL: https://machinelearningmastery.com/probabilistic-model-selection-measures/ (дата звернення: 12.10.2023).

27. Sam Newman. Building Microservices: Designing Fine-Grained Systems 2nd Edition, 2021, 612 p.

28. Martin Kleppmann. Designing Data-Intensive Applications: The Big Ideas Behind Reliable, Scalable, and Maintainable Systems 1st Edition, 2017, 611 p.

29. Паламарчук Є.А. Методи побудови мікросервісної архітектури систем електронного навчання, 2022. 43–54.

30. Коваленко О.О. Методологія реалізації інтеграції ІТ-систем. 2022, 171-172.

31. 7 Reasons Why You Should Use Jupyterlab for Data Science. URL: https://towardsdatascience.com/7-reasons-why-you-should-use-jupyterlab-for-data-science-7c2a3db8755a (дата звернення: 25.09.2023).

32. Golang, C#, or Java: Which Language Is Best for Building Microservices? URL: https://www.apriorit.com/dev-blog/652-virtualization-golang-c-java-for-building-microservices. (дата звернення: 25.10.2023).

33. ASP.NET Core 8 Pros and Cons. URL: https://ukad-group.com/blog/aspnet-core-8-pros-and-cons/ (дата звернення: 15.10.2023).

34. Visual Paradigm. URL: https://bpms.com.ua/visual-paradigm (дата звернення: 6.09.2023).

35 Є. А. Васянович, Д. І. Катєльніков. Використання шаблонів проектування у сучасному підході до розробки програмного забезпечення. 2021, 4с.

36. The Practical Test Pyramid. URL: https://martinfowler.com/articles/practical-test-pyramid.html (дата звернення: 26.10.2023).

ДОДАТКИ

96

Додаток А – Технічне завдання

Міністерство освіти і науки України

Вінницький національний технічний університет

Факультет інформаційних технологій та комп'ютерної інженерії

ЗАТВЕРДЖУЮ

д.т.н., проф. О. Н. Романюк

"19" вересня 2023 р.

Технічне завдання

на магістерську кваліфікаційну роботу «Розробка відмовостійких методів передавання повідомлень та розподіленої backend as a service платформи для мобільних і веб-застосувань» за спеціальністю 121 – Інженерія програмного забезпечення

Керівник магістерської кваліфікаційної роботи:

_____ доц. каф. ПЗ Г.Б. Ракитянська

"19" 09 2023 р.

Виконав:

_____ студент гр.2ПІ-22м І.А. Павлюк

"19" 09 2023 р.

Вінниця – 2023 року

1. Найменування та галузь застосування

Магістерська кваліфікаційна робота: «Розробка відмовостійких методів передавання повідомлень та розподіленої backend as a service платформи для мобільних і веб-застосувань».

Галузь застосування – розробка веб-та мобільних додатків.

2. Підстава для розробки.

Підставою для виконання магістерської кваліфікаційної роботи (МКР) є індивідуальне завдання на МКР та наказ № 247 ректора по ВНТУ про закріплення тем МКР.

3. Мета та призначення розробки.

Метою даної роботи є підвищення відмовостійкості та ефективності обробки запитів у мобільних та веб-застосуваннях шляхом розробки та аналізу адаптивних алгоритмів для повторної передачі повідомлень у контексті BaaS платформи.

Призначення роботи – розробка методів і засобів реалізації відмовостійких методів передавання повідомлень та розподіленої backend as a service платформи для мобільних і веб-застосувань

3 Вихідні дані для проведення НДР

Перелік основних літературних джерел, на основі яких буде виконуватись МКР.

1. Filieri A, Maggio M, Angelopoulos K, D’Ippolito N, Gerostathopoulos I, Hempel AB, et al. Software Engineering Meets Control Theory, 2015, 72 p.
2. Salehie M, Tahvildari L. Self-adaptive software: Landscape and research challenges, 2009, 40 p.
3. Filieri A, Maggio M, Angelopoulos K, D’Ippolito N, Gerostathopoulos I, Hempel AB, Control strategies for self-adaptive software systems, 2017, 31p.
4. Sam Newman. Building Microservices: Designing Fine-Grained Systems 2nd Edition, 2021, 612 p.

4. Технічні вимоги

Вихідні дані до роботи: модель розробки – ітеративна; метод передачі повідомлень між компонентами – REST API; метод балансування навантаження – раунд-робін; метод обробки даних – потокова обробка; застосування теорії управління для оптимізації процесів; вхідні дані – набір даних користувачів і транзакцій у форматі JSON; API ключі для інтеграції з зовнішніми сервісами; ідентифікатори сесій користувачів; вихідні дані – звіти та аналітичні дані, результати оптимізації процесів у відповідному форматі.

5. Конструктивні вимоги.

Конструкція пристрою повинна відповідати естетичним та ергономічним вимогам, повинна бути зручною в обслуговуванні та керуванні.

Графічна та текстова документація повинна відповідати діючим стандартам України.

6. Перелік технічної документації, що пред'являється по закінченню робіт:

– пояснювальна записка до МКР;

– технічне завдання;

– лістинги програми.

7. Вимоги до рівня уніфікації та стандартизації

При розробці програмних засобів слід дотримуватися уніфікації і ДСТУ.

8. Стадії та етапи розробки:

№ з/п	Назва етапів магістерської кваліфікаційної Роботи	Строк виконання етапів роботи
1	Аналіз стану питання	20.09.20 – 25.09.23
2	Аналіз існуючих аналогів та постановка задач дослідження	26.09.20 – 28.09.23
3	Розробка адаптивного алгоритму для підвищення ефективності передачі повідомлень	29.09.20 – 12.10.23
4	Написання програмних модулів для реалізації розроблених алгоритмів	13.10.20 – 28.10.23
5	Розробка серверної частини BaaS-платформи	29.11.20 – 20.11.23
6	Тестування розробленого програмного продукту	21.11.20 – 25.11.23
7	Економічна частина	26.11.20 – 28.11.23
8	Оформлення матеріалів до захисту МКР	29.11.20 – 1.12.23

9. Порядок контролю та прийняття.

Виконання етапів магістерської кваліфікаційної роботи контролюється керівником згідно з графіком виконання роботи. Прийняття магістерської кваліфікаційної роботи здійснюється ДЕК, затвердженою зав. кафедрою згідно з графіком

100

Додаток Б – Протокол перевірки на плагіат

ПРОТОКОЛ ПЕРЕВІРКИ НАВЧАЛЬНОЇ (КВАЛІФІКАЦІЙНОЇ) РОБОТИ

Назва роботи: **Розробка відмовостійких методів передавання повідомлень та розподіленої backend as a service платформи для мобільних і веб-застосувань**
Тип роботи: кваліфікаційна робота

Підрозділ : кафедра програмного забезпечення, ФІТКІ, 2ПІ – 22м

Науковий керівник: к.т.н. доц. Ракитянська Г. Б.

Unicheck	
Оригінальність	**93,7 %**
Схожість	6,3 %

Аналіз звіту подібності

■ **Запозичення, виявлені у роботі, оформлені коректно і не містять ознак плагіату.**
□ Виявлені у роботі запозичення не мають ознак плагіату, але їх надмірна кількість викликає сумніви щодо цінності роботи і відсутності самостійності її автора. Роботу направити на доопрацювання.
□ Виявлені у роботі запозичення є недобросовісними і мають ознаки плагіату та/або в ній містяться навмисні спотворення тексту, що вказують на спроби приховування недобросовісних запозичень.

Особа, відповідальна за перевірку _____ Черноволик Г. О.

Опис прийнятого рішення: **допустити до захисту**

Ознайомлені з повним звітом подібності, який був згенерований системою Unicheck

Автор роботи _____ Павлюк І. А.

Керівник роботи _____ Ракитянська Г. Б.

Додаток В – Лістинг програми

AnalysisService.py

```
import pandas as pd
import numpy as np
from statsmodels.tsa.stattools import adfuller, arma_order_select_ic
from statsmodels.tsa.ar_model import AutoReg
import logging

class AnalysisService:
    def __init__(self, column_name='response_time'):
        self.data = pd.DataFrame()
        self.column_name = column_name
        logging.basicConfig(level=logging.INFO)

    def add_data(self, new_data):
        """Add new data to the service for analysis."""
        if not isinstance(new_data, pd.DataFrame):
            raise ValueError("New data must be a pandas DataFrame.")
        if self.column_name not in new_data.columns:
            raise ValueError(f"Column {self.column_name} not found in new data.")
        self.data = pd.concat([self.data, new_data], ignore_index=True)

    def check_stationarity(self, timeseries, alpha=0.05):
        """Check if the provided timeseries is stationary."""
        result = adfuller(timeseries, autolag='AIC')
        logging.info(f'ADF Statistic: {result[0]}')
        logging.info(f'p-value: {result[1]}')
        return result[1] < alpha

    def fit_ar_model(self, timeseries, max_lag=10):
        """Fit an AutoRegressive model to the provided timeseries."""
        if not self.check_stationarity(timeseries):
            logging.warning("Series is not stationary, differencing might be
required.")
            timeseries = timeseries.diff().dropna()
        if not self.check_stationarity(timeseries):
            raise ValueError("Series is still not stationary after differencing.")

        order_selection = arma_order_select_ic(timeseries, max_ar=max_lag, ic=['aic'])
        lags = order_selection['aic_min_order'][0]
        model = AutoReg(timeseries, lags=lags)
        return model.fit()

    def analyze(self):
        """Main method to analyze the stored data."""
```

```
        if self.column_name not in self.data.columns:
            raise ValueError("Required data column is not available for analysis.")

        timeseries = self.data[self.column_name]
        try:
            model_fit = self.fit_ar_model(timeseries)
            logging.info(model_fit.summary())
            return model_fit
        except Exception as e:
            logging.error(f"Analysis failed: {e}")
            return None
```

PIDController.py

```
class PIDController:
    def __init__(self, kp, ki, kd, setpoint):
        self.kp = kp
        self.ki = ki
        self.kd = kd
        self.setpoint = setpoint
        self.integral = 0
        self.previous_error = 0
        self.previous_time = None

    def calculate(self, current_value, current_time):
        """
        Calculate the PID control output.
        :param current_value: The current value of the parameter being controlled.
        :param current_time: The current time.
        :return: The control output.
        """
        if self.previous_time is None:
            self.previous_time = current_time

        # Calculate time difference
        time_diff = current_time - self.previous_time

        # Proportional term
        error = self.setpoint - current_value
        proportional = self.kp * error

        # Integral term
        self.integral += error * time_diff
        integral = self.ki * self.integral

        # Derivative term
        derivative = self.kd * (error - self.previous_error) / time_diff
```

```
        # Update previous values
        self.previous_error = error
        self.previous_time = current_time

        # PID output
        return proportional + integral + derivative

    def reset(self):
        """
        Reset the PID controller state.
        """
        self.integral = 0
        self.previous_error = 0
        self.previous_time = None
```

AdaptiveRetryService.py

```
class AdaptiveRetryService:
    def __init__(self, feedback_loop_controller):
        """
        Initializes the Adaptive Retry Service.
        :param feedback_loop_controller: An instance of the FeedbackLoopController.
        """
        self.feedback_loop_controller = feedback_loop_controller
        self.retry_interval = 0.2  # Default retry interval in seconds
        self.max_retries = 5  # Default maximum number of retries

    def adjust_retry_parameters(self, adjustment):
        """
        Adjusts the retry parameters based on the feedback from the controller.
        :param adjustment: Adjustment value from the feedback loop controller.
        """
        self.retry_interval = max(0.1, self.retry_interval + adjustment)
        self.max_retries = min(10, max(1, self.max_retries + int(adjustment)))

    def should_retry(self, attempt_number):
        """
        Determines if a retry should be attempted.
        :param attempt_number: The current attempt number.
        :return: True if retry is allowed, False otherwise.
        """
        return attempt_number < self.max_retries

    def get_retry_interval(self):
        """
        Gets the current retry interval.
```

```
        :return: The retry interval in seconds.
        """
        return self.retry_interval
```

FeedbackLoopController.py

```
import pandas as pd

class FeedbackLoopController:
    def __init__(self, pid_controller, analysis_service):
        """
        Initializes the Feedback Loop Controller.
        :param pid_controller: An instance of the PID controller.
        :param analysis_service: An instance of the Analysis Service.
        """
        self.pid_controller = pid_controller
        self.analysis_service = analysis_service
        self.setpoint = None

    def set_setpoint(self, setpoint):
        """
        Sets the desired setpoint for the system.
        :param setpoint: The target setpoint value.
        """
        self.setpoint = setpoint

    def process_data(self, data):
        """
        Process incoming data, perform analysis and adjust using PID controller.
        :param data: Incoming data (typically as a DataFrame).
        """
        if self.setpoint is None:
            raise ValueError("Setpoint not defined.")

        # Add data to analysis service
        self.analysis_service.add_data(data)

        # Perform analysis
        self.analysis_service.analyze()

        # Here you would extract relevant metrics from the analysis
        # For this example, let's assume we get a response time
        response_time = self.analysis_service.get_latest_response_time()

        # Use PID controller to adjust
        adjustment = self.pid_controller.update(response_time)
```

```
        # Here, you can use 'adjustment' to modify system parameters or take actions
        # For example, adjusting retry intervals, thresholds, etc.

        return adjustment
```

IRepositoryAsync.cs

```
public interface IRepositoryAsync
{
      void DropDB(string name);
      void DropCollection(string name);
      #region BacksAPIObjects
      //users
      Task AddUser(string appId, BacksUsers user);
      Task<BacksUsers> Authenticate(string appId, string username, string pwd);
      Task<BacksUsers> GetUser(string appId, string userId);
      Task<List<BacksUsers>> GetUsers(string appId, string query);
      Task UpdateUserPasswrod(string appId, string userId, string password);
      Task UpdateUserData(string appId, string userId, Dictionary<string, object>
data);
      Task RemoveUser(string appId, string userId);
      Task AddSession(string appId, BacksSessions session);
      Task<BacksSessions> GetSession(string appId, string sessionId);
      Task UpdateSession(string appId, string sessionId, Dictionary<string, object>
data);
      Task<List<BacksSessions>> GetAllSessions(string appId, string userId);
      Task<List<BacksSessions>> GetSessions(string appId);
      Task RemoveSession(string appId, string sessionId);
      Task AddEntity(string appId, BacksObject entity);
      Task<BacksObject> GetEntity(string appId, string entityName, string entityId);
      Task<List<BacksObject>> GetAllEntity(string appId, string entityName);
      Task UpdateEntity(string appId, string entityName, string entityId,
Dictionary<string, object> data);
      Task RemoveEntity(string appId, string entityName, string entityId);
    #endregion
}
```

BacksRepository_Clients.cs

```
public partial class BacksRepository
{
      public async Task AddUser(string appId, BacksUsers user)
      {
            try
            {
                  await _context.Get_Users("_User_" + appId).InsertOneAsync(user);
            }
            catch (Exception e)
            {
                  _log.Error("Exception in AddUser", e);
                  throw;
            }
```

```
	}
	public async Task<BacksUsers> Authenticate(string appId, string username, string
pwd)
	{
		try
		{
			var filter = Builders<BacksUsers>.Filter.Eq(s => s.AppId, appId) &
					Builders<BacksUsers>.Filter.Eq(s => s.UserName,
username) &
					Builders<BacksUsers>.Filter.Eq(s => s.Password, pwd);

			return await _context.Get_Users("_User_" + appId)
				.Find(filter)
				.FirstOrDefaultAsync();
		}
		catch (Exception e)
		{
			_log.Error("Exception in Authenticate", e);
			throw;
		}
	}
	public async Task<BacksUsers> GetUser(string appId, string userId)
	{
		try
		{
			var filter = Builders<BacksUsers>.Filter.Eq("Id", userId);
			return await _context.Get_Users("_User_" + appId)
						.Find(filter)
						.FirstOrDefaultAsync();
		}
		catch (Exception e)
		{
			_log.Error("Exception in GetUser", e);
			throw;
		}
	}

	public async Task<List<BacksUsers>> GetUsers(string appId, string query)
	{
		try
		{
			return await _context.Get_Users("_User_" + appId).Find(_ =>
true).ToListAsync();
		}
		catch (Exception e)
		{
			_log.Error("Exception in GetUser", e);
			throw;
		}
	}

	public async Task UpdateUserPasswrod(string appId, string userId, string
password)
	{
		try
```

```
            {
                  var filter = Builders<BacksUsers>.Filter.Eq(s => s.Id, userId);
                  var update = Builders<BacksUsers>.Update
                                          .Set(s => s.Password, password)
                                          .CurrentDate(s => s.UpdatedAt);

                  await _context.Get_Users("_User_" + appId).UpdateOneAsync(filter,
update);
            }
            catch (Exception e)
            {
                  _log.Error("Exception in UpdateUser", e);
                  throw;
            }
      }

      public async Task UpdateUserData(string appId, string userId, Dictionary<string,
object> data)
      {
            try
            {
                  var filter = Builders<BacksUsers>.Filter.Eq(s => s.Id, userId);

                  var update = Builders<BacksUsers>.Update
                                          .Set(s => s.Data, data)
                                          .CurrentDate(s => s.UpdatedAt);
                  await _context.Get_Users("_User_" + appId).UpdateOneAsync(filter,
update);
            }
            catch (Exception e)
            {
                  _log.Error("Exception in UpdateUser", e);
                  throw;
            }
      }

      public async Task RemoveUser(string appId, string userId)
      {
            try
            {
                  await _context.Get_Users("_User_" +
appId).DeleteOneAsync(Builders<BacksUsers>.Filter.Eq("Id", userId));
            }
            catch (Exception e)
            {
                  _log.Error("Exception in RemoveUser", e);
                  throw;
            }
      }


      //session
      public async Task AddSession(string appId, BacksSessions session)
      {
            try
```

```
            {
                  await _context.Get_Sessions("_Session_" +
appId).InsertOneAsync(session);
            }
            catch (Exception e)
            {
                  _log.Error("Exception in AddSession", e);
                  throw;
            }

      }
      public async Task<BacksSessions> GetSession(string appId, string sessionId)
      {
            try
            {
                  var filter = Builders<BacksSessions>.Filter.Eq("Id", sessionId);
                  return await _context.Get_Sessions("_Session_" + appId)
                                    .Find(filter)
                                    .FirstOrDefaultAsync();
            }
            catch (Exception e)
            {
                  _log.Error("Exception in GetSession", e);
                  throw;
            }
      }

      public async Task UpdateSession(string appId, string sessionId,
Dictionary<string, object> data)
      {
            try
            {
                  var filter = Builders<BacksSessions>.Filter.Eq(s => s.Id,
sessionId);

                  var update = Builders<BacksSessions>.Update
                                    .Set(s => s.Data, data)
                                    .CurrentDate(s => s.UpdatedAt);


                  await _context.Get_Sessions("_Session_" +
appId).UpdateOneAsync(filter, update);
            }
            catch (Exception e)
            {
                  _log.Error("Exception in UpdateSession", e);
                  throw;
            }
      }

      public async Task<List<BacksSessions>> GetAllSessions(string appId, string
userId)
      {
            try
            {
```

```
                  var filter = Builders<BacksSessions>.Filter.Eq(s => s.AppId, appId)
&
                        Builders<BacksSessions>.Filter.Eq(s => s.PUser, userId);

                  return await _context.Get_Sessions("_Session_" +
appId).Find(filter).ToListAsync();
            }
            catch (Exception e)
            {
                  _log.Error("Exception in GetAllSessions", e);
                  throw;
            }
      }

      public async Task<List<BacksSessions>> GetSessions(string appId)
      {
            try
            {
                  var filter = Builders<BacksSessions>.Filter.Eq(s => s.AppId,
appId);

                  return await _context.Get_Sessions("_Session_" +
appId).Find(filter).ToListAsync();
            }
            catch (Exception e)
            {
                  _log.Error("Exception in GetAllSessions", e);
                  throw;
            }
      }

      public async Task RemoveSession(string appId, string sessionId)
      {
            try
            {
                  await _context.Get_Sessions("_Session_" +
appId).DeleteOneAsync(Builders<BacksSessions>.Filter.Eq("Id", sessionId));
            }
            catch (Exception e)
            {
                  _log.Error("Exception in RemoveSession", e);
                  throw;
            }
      }

      public async Task AddEntity(string appId, BacksObject entity)
      {
            try
            {
                  await _context.Get_Objects(entity.Name + "_" +
appId).InsertOneAsync(entity);
            }
            catch (Exception e)
            {
                  _log.Error("Exception in AddEntity", e);
```

```
                throw;
            }
        }
        public async Task<BacksObject> GetEntity(string appId, string entityName, string
entityId)
        {
            try
            {
                var filter = Builders<BacksObject>.Filter.Eq("Id", entityId);
                return await _context.Get_Objects(entityName + "_" + appId)
                                     .Find(filter)
                                     .FirstOrDefaultAsync();
            }
            catch (Exception e)
            {
                _log.Error("Exception in GetEntity", e);
                throw;
            }
        }

        public async Task<List<BacksObject>> GetAllEntity(string appId, string
entityName)
        {
            try
            {
                var filter = Builders<BacksObject>.Filter.Eq("AppId", appId);
                return await _context.Get_Objects(entityName + "_" +
appId).Find(filter).ToListAsync();
            }
            catch (Exception e)
            {
                _log.Error("Exception in GetAllEntity", e);
                throw;
            }
        }

        public async Task UpdateEntity(string appId, string entityName, string entityId,
Dictionary<string, object> data)
        {
            try
            {
                var filter = Builders<BacksObject>.Filter.Eq(s => s.Id, entityId);

                var update = Builders<BacksObject>.Update
                                     .Set(s => s.Data, data)
                                     .CurrentDate(s => s.UpdatedAt);


                await _context.Get_Objects(entityName + "_" +
appId).UpdateOneAsync(filter, update);
            }
            catch (Exception e)
            {
                _log.Error("Exception in UpdateEntity", e);
                throw;
```

```
            }
      }

      public async Task RemoveEntity(string appId, string entityName, string entityId)
      {
            try
            {
                  await _context.Get_Objects(entityName + "_" +
appId).DeleteOneAsync(Builders<BacksObject>.Filter.Eq("Id", entityId));
            }
            catch (Exception e)
            {
                  _log.Error("Exception in RemoveSession", e);
                  throw;
            }
      }
      }
```

BacksRepository.cs

```
public partial class BacksRepository : IRepositoryAsync
{
      private readonly BackendsContext _context = null;
      private ILog _log = new Log(typeof(BacksRepository));
      public BacksRepository(Configuration config)
      {
            _context = new BackendsContext(config.ConnectionString, config.Database);
      }
      public async Task<IList<Account>> GetAllAccounts()
      {
            try
            {
                  return await _context.Accounts.Find(_ => true).ToListAsync();
            }
            catch (Exception e)
            {
                  _log.Error("Exception in GetAllAccounts", e);
                  throw;
            }
      }
      public async Task AddAccount(Account acc)
      {//Validate screenname
            try
            {
                  await _context.Accounts.InsertOneAsync(acc);
            }
            catch (Exception e)
            {
                  _log.Error("Exception in AddAccount" , e);
                  throw;
            }
      }
      public async Task UpdateAccount(string id, string pwd, string firstName, string
lastName, string email)
      {
```

```
            try
            {
                  var filter = Builders<Account>.Filter.Eq(s => s.Id, id);
                  var updater = Builders<Account>.Update;
                  var update = Builders<Account>.Update
                                          .Set(s => s.FirstName, firstName)
                                          .Set(s => s.LastName, lastName)
                                          .Set(s => s.Email, email);
                  await _context.Accounts.UpdateOneAsync(filter, update);
            }
            catch (Exception e)
            {
                  _log.Error("Exception in UpdateAccount", e);
                  throw;
            }

      }
      public async Task<Account> GetAccount(string accId)
      {
            try
            {
                  var filter = Builders<Account>.Filter.Eq("Id", accId);
                  return await _context.Accounts
                                          .Find(filter)
                                          .FirstOrDefaultAsync();
            }
            catch (Exception e)
            {
                  _log.Error("Exception in GetAccount", e);
                  throw;
            }
      }
      public async Task<Account> Login(string login, string pwd)
      {
            try
            {
                  var filter = Builders<Account>.Filter.Eq(s => s.ScreenName, login)
&
                        Builders<Account>.Filter.Eq(s => s.Pasword, pwd);
                  return await _context.Accounts
                                          .Find(filter)
                                          .FirstOrDefaultAsync();
            }
            catch (Exception e)
            {
                  _log.Error("Exception in Login", e);
                  throw;
            }
      }
      public async Task RemoveAccount(string accId)
      {
            try
            {
                  await
_context.Accounts.DeleteOneAsync(Builders<Account>.Filter.Eq("Id", accId));
```

```
        }
        catch (Exception e)
        {
                _log.Error("Exception in RemoveAccount", e);
                throw;
        }

}
public async Task<IList<Account>> FindDuplicates(string login)
{
        try
        {
                var filter = Builders<Account>.Filter.Eq("ScreenName", login);
                return await _context.Accounts.Find(filter).ToListAsync();
        }
        catch (Exception e)
        {
                _log.Error("Exception in FindDuplicates", e);
                throw;
        }
}
public async Task AddProject(Project proj)
{
        try
        {
                await _context.Projects.InsertOneAsync(proj);
        }
        catch (Exception e)
        {
                _log.Error("Exception in AddProject", e);
                throw;
        }

}
public async Task<Project> GetProject(string projId)
{
        try
        {
                var filter = Builders<Project>.Filter.Eq("Id", projId);
                return await _context.Projects
                                        .Find(filter)
                                        .FirstOrDefaultAsync();
        }
        catch (Exception e)
        {
                _log.Error("Exception in GetProject", e);
                throw;
        }
}
public async Task UpdateProject(string projId)
{

}
public async Task RemoveProject(string projId)
{
```

```
            try
            {
                  await
_context.Projects.DeleteOneAsync(Builders<Project>.Filter.Eq("Id", projId));
            }
            catch (Exception e)
            {
                  _log.Error("Exception in RemoveProject", e);
                  throw;
            }
      }
      public async Task<IList<Project>> GetAccountProjects(string accId)
      {
            try
            {
                  var filter = Builders<Project>.Filter.Eq("P_AccountId", accId);
                  return await _context.Projects.Find(filter).ToListAsync();
            }
            catch (Exception e)
            {
                  _log.Error("Exception in GetAccountProjects", e);
                  throw;
            }
      }
      public async Task<IList<Project>> GetAllProject()
      {
            try
            {
                  return await _context.Projects.Find(_ => true).ToListAsync();
            }
            catch (Exception e)
            {
                  _log.Error("Exception in GetAllProject", e);
                  throw;
            }
      }
      public async Task Add_Schema(BacksProjectSchema schema)
      {
            try
            {
                  await _context._Schema.InsertOneAsync(schema);
            }
            catch (Exception e)
            {
                  _log.Error("Exception in GetAllProject", e);
                  throw;
            }
      }
      public async Task<BacksProjectSchema> GetSchema(string appId)
      {
            try
            {
                  var filter = Builders<BacksProjectSchema>.Filter.Eq(s=>s.AppId,
appId);
                  return await _context._Schema.Find(filter).FirstOrDefaultAsync();
```

```
            }
            catch (Exception e)
            {
                  _log.Error("Exception in GetSchema", e);
                  throw;
            }
      }

      public async Task Update_Schema(string appId, Dictionary<string, EntitiesSchema>
data)
      {
            try
            {
                  var filter = Builders<BacksProjectSchema>.Filter.Eq(s => s.AppId,
appId);

                  var update = Builders<BacksProjectSchema>.Update.Set(s =>
s.EntityColumnTypeMapping, data);
                        //.CurrentDate(s => s.UpdatedAt);


                  await _context._Schema.UpdateOneAsync(filter, update);

            }
            catch (Exception e)
            {
                  _log.Error("Exception in GetSchema", e);
                  throw;
            }
      }

      public void CreateCollection(string name, BsonDocument validator)
      {
            try
            {
                  _context.CreateCollection(name, validator);
            }
            catch (Exception e)
            {
                  _log.Error("Exception in CreateCollection", e);
                  throw;
            }
      }
      public void DropDB(string name)
      {
            _context.DropDB(name);
      }

      public void DropCollection(string name)
      {
            _context.DropCollection(name);
      }
      }
```

BackendsContext.cs

```
public class BackendsContext
{
	private readonly IMongoDatabase _db = null;
	public BackendsContext(string connectionString, string database)
	{
		var client = new MongoClient(connectionString);
		if (client != null)
			_db = client.GetDatabase(database);
	}
	public IMongoCollection<BsonDocument>  GetCollection(string collection)
	{
		return _db.GetCollection<BsonDocument>(collection);
	}
	public IMongoCollection<BacksUsers> Get_Users(string collection)
	{
		return _db.GetCollection<BacksUsers>(collection);
	}
	public IMongoCollection<BacksSessions> Get_Sessions(string collection)
	{
		return _db.GetCollection<BacksSessions>(collection);
	}
	public IMongoCollection<BacksObject> Get_Objects(string collection)
	{
		return _db.GetCollection<BacksObject>(collection);
	}
	public IMongoCollection<Account> Accounts
	{
		get
		{
		return _db.GetCollection<Account>("Accounts");
		}
	}
	public IMongoCollection<Project> Projects
	{
		get
		{
		return _db.GetCollection<Project>("Projects");
		}
	}

	public IMongoCollection<BacksProjectSchema> _Schema
	{
		get
		{
			return _db.GetCollection<BacksProjectSchema>("_Schema");
		}
	}


	internal void DropDB(string name)
    {
	    _db.Client.DropDatabase(name);
```

```
    }

        public void DropCollection(string name)
        {
                _db.DropCollection(name);
        }

        public void CreateCollection(string name, BsonDocument validator)
        {
                _db.CreateCollectionAsync(name);
                _db.RunCommand<BsonDocument>(new
BsonDocumentCommand<BsonDocument>(validator));

        }
        }
```

BacksObjectService.cs

```
public class BacksObjectService
{
        private readonly SchemaHandler _handler;
        private readonly IRepositoryAsync _repo;
        private ILog _log = new Log(typeof(BacksUsersService));

        public BacksObjectService(IRepositoryAsync repository)
        {
                _repo = repository;
                _handler = new SchemaHandler(_repo);
                Mapper.Initialize(cfg =>
                {
                        cfg.CreateMap<BacksObject, ObjectsDto>();
                });
        }

        public ObjectsDto CreateEntity(string appId, string name,
Dictionary<string,object> data, out BacksErrorCodes error)
        {
                error = BacksErrorCodes.Ok;
                try
                {

                        var entity = new BacksObject()
                        {
                                AppId = appId,
                                Data = data,
                                CreatedAt = DateTime.UtcNow,
                                Name = name

                        };
                         _repo.AddEntity(appId, entity);

                        if (entity.Id == null)
                        {
                                error = BacksErrorCodes.SystemError;
                                return null;
```

```
                }

                return new ObjectsDto()
                {
                        Id = entity.Id,
                        CreatedAt = entity.CreatedAt.Value
                };
        }
        catch (Exception e)
        {
                _log.Error("CreateEntity exception : ", e);
                error = BacksErrorCodes.SystemError;
        }

        return null;
    }

    public ObjectsDto GetEntity(string appId, string name, string entityId, out
BacksErrorCodes error)
    {
        error = BacksErrorCodes.Ok;
        try
        {
                BacksObject entity = _repo.GetEntity(appId, name, entityId).Result;
                if (entity == null)
                {
                        error = BacksErrorCodes.EntityNotFound;
                        return null;
                }

                var mappedEntity = Mapper.Map<BacksObject, ObjectsDto>(entity);

                return mappedEntity;

        }
        catch (Exception e)
        {
                _log.Error("GetEntity exception : ", e);
                error = BacksErrorCodes.SystemError;
        }
        return null;
    }

    public ObjectsDto UpdateEntity(string appId, string entityName, string entityId,
                        Dictionary<string, object> data, out BacksErrorCodes error)
    {
        error = BacksErrorCodes.Ok;
        try
        {

                BacksObject entity = _repo.GetEntity(appId, entityName,
entityId).Result;
                if (entity == null)
                {
                        error = BacksErrorCodes.EntityNotFound;
```

```
                        return null;
                }

                var updatedData = entity.Data;
                foreach (var pair in data)
                {
                        updatedData.CreateNewOrUpdateExisting(pair.Key, pair.Value);
                }

                _repo.UpdateEntity(appId, entityName, entityId, updatedData);

                return new ObjectsDto() {UpdatedAt = DateTime.UtcNow};
            }
            catch (Exception e)
            {
                _log.Error("UpdateEntity exception : ", e);
                error = BacksErrorCodes.SystemError;
            }
            return null;
        }

        public ObjectsDto QueryEntity(string appId, string entityId, out BacksErrorCodes
error)
        {
            error = BacksErrorCodes.Ok;
            try
            {

            }
            catch (Exception e)
            {
                _log.Error("QueryEntity exception : ", e);
                error = BacksErrorCodes.SystemError;
            }
            return null;
        }

        public ObjectsDto RemoveEntity(string appId, string entityName, string entityId,
out BacksErrorCodes error)
        {
            error = BacksErrorCodes.Ok;
            try
            {
                _repo.RemoveEntity(appId, entityName, entityId).Wait();
            }
            catch (Exception e)
            {
                _log.Error("RemoveEntity exception : ", e);
                error = BacksErrorCodes.SystemError;
            }
            return null;
        }
        }
```

BacksUsersService.cs

```
public class BacksUsersService
{
	private readonly SchemaHandler _handler;

	private readonly IRepositoryAsync _repo;

	private ILog _log = new Log(typeof(BacksUsersService));

	public BacksUsersService(IRepositoryAsync repository)
	{
		_repo = repository;
		_handler = new SchemaHandler(_repo);
		Mapper.Initialize(cfg =>
		{
			cfg.CreateMap<BacksUsers, UserDto>();
		});
	}


	public UserDto SignUp(string appId, string email, string userName, string pwd,
out BacksErrorCodes error) // create user
	{
		error = BacksErrorCodes.Ok;
		try
		{
			var user = new BacksUsers()
			{
				AppId = appId,
				UserName = userName,
				Password = pwd.CreateMD5Hash(),
				Email = email,
				CreatedAt = DateTime.UtcNow
			};

			if (_repo.FindDuplicates(userName).Result.Any())
			{
				error = BacksErrorCodes.DuplicateLogin;
				return null;
			}

			_repo.AddUser(appId, user).Wait();
			if (user.Id == null)
			{
				error = BacksErrorCodes.SignUpError;
				return null;
			}

			//create session
			var session = new BacksSessions()
			{
				PUser = user.Id,
				CreatedAt = DateTime.UtcNow,
				ExpiresAt = DateTime.UtcNow.AddMinutes(30),
```

```
                        AppId = appId,
                    };
                    _repo.AddSession(appId, session).Wait();

                    if (session.Id == null)
                    {
                        error = BacksErrorCodes.SignUpError;
                        return null;
                    }
                    var mappedUser = Mapper.Map<BacksUsers, UserDto>(user);
                    mappedUser.SessionId = session.Id;

                    return mappedUser;

                }
                catch (Exception e)
                {
                    _log.Error("SignIn exception : ", e);
                    error = BacksErrorCodes.SystemError;
                }

                return null;
            }

            public UserDto Login(string appId, string userName, string pwd, out
BacksErrorCodes error)
            {
                error = BacksErrorCodes.Ok;
                try
                {

                    var user = _repo.Authenticate(appId, userName,
pwd.CreateMD5Hash()).Result;
                    if (user.Id == null)
                    {
                        error = BacksErrorCodes.AuthFailed;
                        return null;
                    }

                    var session = new BacksSessions()
                    {
                        PUser = user.Id,
                        CreatedAt = DateTime.UtcNow,
                        ExpiresAt = DateTime.UtcNow.AddMinutes(30),
                        AppId = appId,
                    };
                    _repo.AddSession(appId, session);

                    if (session.Id == null)
                    {
                        error = BacksErrorCodes.SignUpError;
                        return null;
                    }
                    var mappedUser = Mapper.Map<BacksUsers, UserDto>(user);
                    mappedUser.SessionId = session.Id;
```

```
                return mappedUser;


            }
            catch (Exception e)
            {
                _log.Error("Login exception : ", e);
                error = BacksErrorCodes.SystemError;
            }

            return null;
        }
        //KillSession

        public void Logout(string appId, string userId, string sessionId, out
BacksErrorCodes error)
        {
            error = BacksErrorCodes.Ok;
            try
            {
                if(ValidateSession(appId, userId, sessionId, out error) && error ==
BacksErrorCodes.Ok)
                {
                    _repo.RemoveSession(appId, sessionId).Wait();
                }

            }
            catch (Exception e)
            {
                _log.Error("Logout exception : ", e);
                error = BacksErrorCodes.SystemError;
            }
        }

        public void RemoveUser(string appId, string userId, out BacksErrorCodes error)
        {
            error = BacksErrorCodes.Ok;
            try
            {
                _repo.RemoveUser(appId, userId).Wait();

            }
            catch (Exception e)
            {
                _log.Error("RemoveUser exception : ", e);
                error = BacksErrorCodes.SystemError;
            }
        }

        public bool ValidateSession(string appId, string userId, string sessionId, out
BacksErrorCodes error)
        {
            error = BacksErrorCodes.Ok;
            try
```

```
            {
                  BacksSessions session = _repo.GetSession(appId, sessionId).Result;
                  return string.IsNullOrEmpty(userId) && session !=null ? session !=
null : session.PUser == userId;
            }
            catch (Exception e)
            {
                  _log.Error("ValidateSession exception : ", e);
                  error = BacksErrorCodes.SystemError;
            }
            return false;
      }

      public UserDto GetUser(string appId, string userId, out BacksErrorCodes error)
      {
            error = BacksErrorCodes.Ok;
            try
            {
                  BacksUsers user = _repo.GetUser(appId, userId).Result;
                  if (user == null)
                  {
                        error = BacksErrorCodes.UserIsNotFound;
                        return null;
                  }

                  var mappedUser = Mapper.Map<BacksUsers, UserDto>(user);

                  return mappedUser;
            }
            catch (Exception e)
            {
                  _log.Error("GetUser exception : ", e);
                  error = BacksErrorCodes.SystemError;
            }
            return null;
      }

      public void UpdateUser(string appId, string userId, string sessionToken,
Dictionary<string, object> customFields, out BacksErrorCodes error)
      {
            error = BacksErrorCodes.Ok;
            try
            {
                  if (!ValidateSession(appId, userId, sessionToken, out error))
                  {
                        error = BacksErrorCodes.SessionIsNotFound;
                        return;
                  }

                  //get User and  update
                  BacksUsers user = _repo.GetUser(appId, userId).Result;
                  if (user == null)
                  {
                        error = BacksErrorCodes.UserIsNotFound;
                        return;
```

```
                }

                var updatedData = user.Data;
                foreach (var pair in customFields)
                {
                        updatedData.CreateNewOrUpdateExisting(pair.Key, pair.Value);
                }


                _repo.UpdateUserData(appId, userId, updatedData).Wait();
            }
            catch (Exception e)
            {
                _log.Error("UpdateUser exception : ", e);
                error = BacksErrorCodes.SystemError;
            }
        }
        public void QueryUserData()
        {

        }

        public void PasswrodReset(string appId, string userId, string pwd, out
BacksErrorCodes error)
        {
            error = BacksErrorCodes.Ok;
            try
            {
                _repo.UpdateUserPasswrod(appId, userId,
pwd.CreateMD5Hash()).Wait();
                //send email
            }
            catch (Exception e)
            {
                _log.Error("PasswrodReset exception : ", e);
                error = BacksErrorCodes.SystemError;
            }

        }

        public void UpdateSession(string appId, string sessionToken, Dictionary<string,
object> customFields, out BacksErrorCodes error)
        {
            error = BacksErrorCodes.Ok;
            try
            {
                BacksSessions session = _repo.GetSession(appId,
sessionToken).Result;

                //if (ValidateSession(appId, null, sessionToken, out error))
                if(session == null)
                {
                    error = BacksErrorCodes.SessionIsNotFound;
                    return;
                }
```

```
                //Get Session and Update


                var updatedData = session.Data;
                foreach (var pair in customFields)
                {
                        updatedData.CreateNewOrUpdateExisting(pair.Key, pair.Value);
                }

                _repo.UpdateSession(appId, sessionToken, updatedData).Wait();
            }
            catch (Exception e)
            {
                _log.Error("UpdateUser exception : ", e);
                error = BacksErrorCodes.SystemError;
            }
        }
        public void RemoveSession(string appId, string sessionId, out BacksErrorCodes
error)
        {
            error = BacksErrorCodes.Ok;
            try
            {
                _repo.RemoveSession(appId, sessionId).Wait();

            }
            catch (Exception e)
            {
                _log.Error("RemoveSession exception : ", e);
                error = BacksErrorCodes.SystemError;
            }
        }

        public SessionDto GetSession(string appId, string sessionId, out BacksErrorCodes
error)
        {
            error = BacksErrorCodes.Ok;
            try
            {
                var session = _repo.GetSession(appId, sessionId).Result;
                if (session == null)
                {
                    error = BacksErrorCodes.SessionIsNotFound;
                    return null;
                }


                var sessionDto = new SessionDto()
                {
                    Id = session.Id,
                    PUser = session.PUser,
                    CreatedAt = session.CreatedAt,
                    ExpiresAt = session.ExpiresAt,
                    Data = session.Data
                };
```

```
                return sessionDto;
            }
            catch (Exception e)
            {
                _log.Error("GetSession exception : ", e);
                error = BacksErrorCodes.SystemError;
            }

            return null;
        }

        public List<SessionDto> GetUserSession(string appId, string userId, out
BacksErrorCodes error)
        {
            error = BacksErrorCodes.Ok;
            try
            {
                var sessions = _repo.GetAllSessions(appId, userId).Result;
                if (sessions == null)
                {
                    error = BacksErrorCodes.SessionIsNotFound;
                    return null;
                }

                var sessionsDto = new List<SessionDto>();

                foreach (var item in sessions)
                {
                    var sessionDto = new SessionDto()
                    {
                        Id = item.Id,
                        PUser = item.PUser,
                        CreatedAt = item.CreatedAt,
                        ExpiresAt = item.ExpiresAt,
                        Data = item.Data
                    };

                    sessionsDto.Add(sessionDto);
                }


                return sessionsDto;
            }
            catch (Exception e)
            {
                _log.Error("GetUserSession exception : ", e);
                error = BacksErrorCodes.SystemError;
            }

            return null;
        }
}
```

SchemaHandler.cs

```
public class SchemaHandler
{
    public static BacksProjectSchema CreateDefaultSchema(string projectId)
    {
        return new BacksProjectSchema()
        {
            AppId = projectId,
            EntityColumnTypeMapping = new Dictionary<string, EntitiesSchema>()
                {
                    {
                        "_User", new EntitiesSchema()
                        {
                            ColumnTypeMapping = new Dictionary<string, string>()
                            {
                                { "Id", BacksDataType.BString},
                                                { "AppId", BacksDataType.BString},
                                                { "UserName",
BacksDataType.BString},
                                { "Password", BacksDataType.BString},
                                { "CreatedAt", BacksDataType.BTime},
                                { "UpdatedAt", BacksDataType.BTime},
                            }
                        }
                    },
                    {
                        "_Session", new EntitiesSchema()
                        {
                            ColumnTypeMapping = new Dictionary<string, string>()
                            {
                                { "Id", BacksDataType.BString},
                                                { "AppId", BacksDataType.BString},
                                                { "Token", BacksDataType.BString},
                                { "CreatedAt", BacksDataType.BTime},
                                { "UpdatedAt", BacksDataType.BTime},
                                { "ExpiresAt", BacksDataType.BTime},
                                { "PUser", BacksDataType.BPointer}
                            }
                        }
                    },
                    {
                        "_Roles", new EntitiesSchema()
                        {
                            ColumnTypeMapping = new Dictionary<string, string>()
                            {
                                { "Id", BacksDataType.BString},
                                                { "AppId", BacksDataType.BString},
                                                { "Name", BacksDataType.BString},
                                { "Paswword", BacksDataType.BString},
                                { "CreatedAt", BacksDataType.BTime},
                                { "UpdatedAt", BacksDataType.BTime},
                             }
                        }
                    }
```

```
                }

        };
    }

    public EntitiesSchema GetSchema(Dictionary<string, object> data)
    {
          var columnTypeMapping = new EntitiesSchema()
          {
                ColumnTypeMapping = new Dictionary<string, string>()
                  {
                        { "Id", BacksDataType.BString},
                        { "Name", BacksDataType.BString},
                        { "CreatedAt", BacksDataType.BTime}

                  }
            };

            foreach (var item in data)
            {
                  columnTypeMapping.ColumnTypeMapping[item.Key]=
TypeConverter(item.Value);
            }

          return columnTypeMapping;
    }

      private string TypeConverter(object param)
    {
            Type t = param.GetType();
          if (t.Equals(typeof(string)))
                return BacksDataType.BString;

          else if (t.Equals(typeof(long)))
                  return BacksDataType.BInt;
          else if (t.Equals(typeof(DateTime)))
                return BacksDataType.BTime;
          else
                return BacksDataType.BString;

      }

    public SchemaHandler(IRepositoryAsync repo)
    {
        _repo = repo;
    }
}
```

Додаток Г – Ілюстративна частина

Розробка відмовостійких методів передавання повідомлень та розподіленої backend as a service платформи для мобільних і веб-застосувань

Виконав:
Павлюк І. А.

Керівник:
Ракитянська Г. Б.

Рисунок Г.1 – Назва роботи

Розробка відмовостійких методів передавання повідомлень та розподіленої backend as a service платформи для мобільних і веб-застосувань

- **Метою роботи** є підвищення ефективності обробки запитів у мобільних та веб-застосуваннях шляхом розробки та аналізу адаптивних алгоритмів для повторної передачі повідомлень у контексті BaaS платформи.
- **Об'єктом дослідження** є процес передачі повідомлень у мікросервісній архітектурі BaaS-платформ.
- **Предмет дослідження** – алгоритми та методи повторної передачі повідомлень у BaaS-платформах, зокрема адаптивні механізми, що реагують на зміни у мережевому середовищі та навантаженні системи.

Рисунок Г.2 – Мета, об'єкт і предмет дослідження

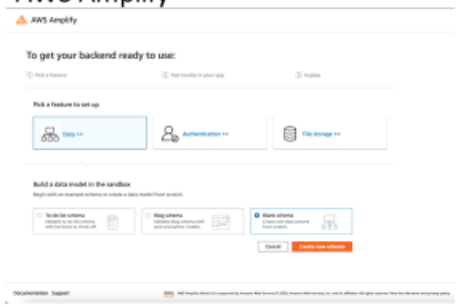
Задачі

- провести аналіз існуючих методів передачі повідомлень у BaaS платформах;
- розробити адаптивні алгоритми для повторної передачі повідомлень;
- розробити дизайн архітектури BaaS-платформи;
- розробити програмні сервіси BaaS-платформи
- провести інтеграцію створених алгоритмів у BaaS-платформу;
- провести оцінку ефективності запропонованих алгоритмів у різних сценаріях використання;
- провести тестування розробленої платформи.

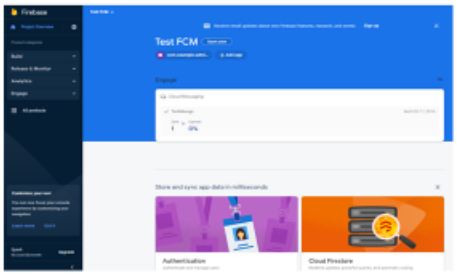
Рисунок Г.3 – Задачі

Аналоги

AWS Amplify



Google Firebase



Parse

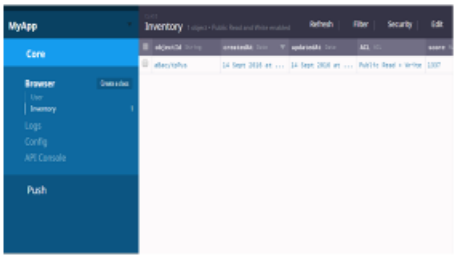


Рисунок Г.4 – Порівняння аналогів

Адаптивний метод повторної передачі повідомлень

1. **Визначення цілей системи.** Початковим етапом є встановлення чітких цілей для системи, які можуть включати визначення ключових показників ефективності, значень або параметрів, які потрібно мінімізувати чи максимізувати.
2. **Визначення параметрів керування.** Наступний крок полягає у визначенні змінних управління, які можуть бути використані для модифікації поведінки програмного забезпечення.
3. **Розробка моделі об'єкта керування.** Після визначення параметрів необхідно розробити модель об'єкта керування.
4. **Проектування Контролера.** На цьому етапі проектується контролер, який буде реалізовувати стратегію управління на основі моделі об'єкта.
5. **Реалізація та перевірка контролера.** Після проектування контролер реалізується та інтегрується з системою. На завершальному етапі система тестується для переконання у відповідності встановленим цілям та ефективності керування.

$$u(t) = u(t-1) + K_p e(t) + K_i \int_0^t e(\tau)d\tau + K_d \frac{de(t)}{dt}$$

де $e(t)$ - різниця між поточним і цільовим станом (наприклад, час відповіді на запит), а Kp, Ki і Kd - налаштування ПІД - контролера.

Рисунок Г.5 – Адаптивний метод повторної передачі повідомлень

Адаптивний алгоритм повторної передачі повідомлень

1. **Збір даних та метрик**. Система збирає дані про час відповіді, невдачі запитів та частоту повторних спроб.
2. **Аналіз даних.** Використання статистичних методів для виявлення шаблонів у зібраних метриках.
3. **Визначення параметрів контролера.** На основі аналізу даних, налаштовуємо параметри ПІД-контролера для оптимізації поведінки системи.
4. **Впровадження змін.** Система вносить зміни у поведінку повторних спроб згідно з вказівками контролера.
5. **Зворотний зв'язок.** Система постійно моніторить свій стан та корегує дії згідно з отриманими даними для підтримання бажаного рівня продуктивності.

Рисунок Г.6 – Адаптивний алгоритм повторної передачі повідомлень

Методу аналізу даних передачі повідомлень в BaaS

1. **Збір Даних.** Система збирає метрики M(t), які включають час відповіді на запити, частоту невдач запитів, та кількість повторних спроб. Дані зберігаються у вигляді часових рядів для кожної метрики окремо.
$$M(t) = \{response\_time, FR, number\_of\_retries\}; FR(t) = \frac{N_{failed}}{N} \cdot 100\%$$
2. **Перевірка на Стаціонарність.** Застосування тестів на стаціонарність, таких як Дікі-Фуллера , для перевірки, чи є часовий ряд M(t) стаціонарним. Якщо дані не стаціонарні, застосовуються методи перетворення (наприклад, диференціювання), щоб зробити їх стаціонарними.
3. **Вибір Порядку Моделі (p).** Використання критеріїв вибору моделі, таких як критерій Акаіке (AIC) або Байесівський критерій (BIC) , для визначення оптимального порядку p для AR моделі. Оцінка різних моделей з різними значеннями p та вибір найбільш адекватної моделі на основі статистичних показників.
4. **Оцінка Параметрів Моделі.** Застосування методів оцінювання параметрів, таких як метод найменших квадратів або максимальної вірогідності, для визначення значень параметрів φ1, φ2, ..., φp. Перевірка адекватності моделі, відбувається зокрема за допомогою аналізу залишків.
5. **Прогнозування та Аналіз.** Використання оціненої AR моделі для прогнозування майбутніх значень метрик M(t). Аналіз зв'язків між різними метриками, щоб виявити можливі причинно-наслідкові зв'язки. Використання результатів аналізу для адаптації поведінки системи, наприклад, для налаштування стратегії повторних спроб або для оптимізації відповідей серверів.
$$X_t = c + \sum_p \varphi_i X_{t-i} + \epsilon_t$$

Рисунок Г.7 – Метод аналізу даних передачі повідомлень у BaaS

Модель високорівневої адаптивної системи із повторною передачею повідомлень

1. **API Gateway**. Центральний вхідний вузол для всіх запитів до платформи BaaS. BaaS Core Services: Включає основні сервіси BaaS, для обробки запитів від користувачів та інших сервісів.
2. **Metric Collection Service.** Відповідає за збір метрик системи (час відповіді, невдачі запитів, кількість повторних спроб).
3. **Analysis Service.** Аналізує зібрані дані, виявляючи тенденції та закономірності, необхідні для адаптивного регулювання поведінки системи.
4. **Adaptive Retry Service.** Управління логікою повторних спроб на основі даних від Analysis Service.
5. **Circuit Breaker Service.** Моніторинг високих показників невдач запитів та реалізація механізму обмеження чи призупинення запитів для забезпечення стабільності системи.
6. **Feedback Loop Controller.** Координує роботу всієї адаптивної системи, забезпечуючи зворотний зв'язок між різними компонентами та адаптацію до змін.

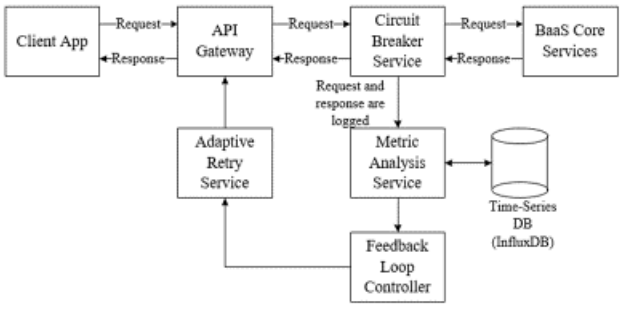


Рисунок Г.8 – Модель високорівневої адаптивної системи із повторною передачею повідомлень

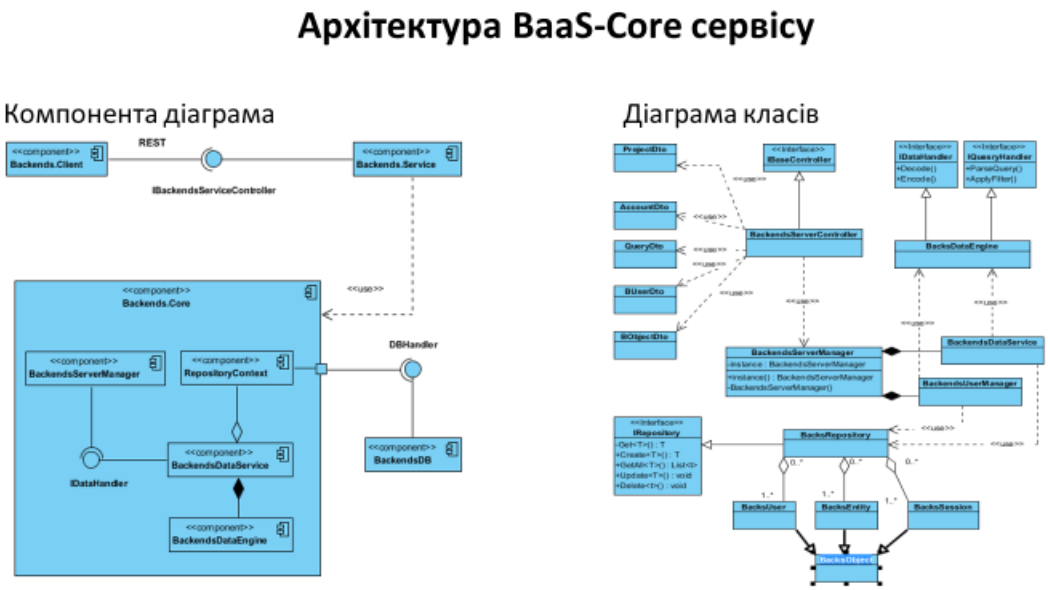


Рисунок Г.9 – Архітектура BaaS-Core сервісу

Рисунок Г.10 – Модель взаємозв'язків у базі даних

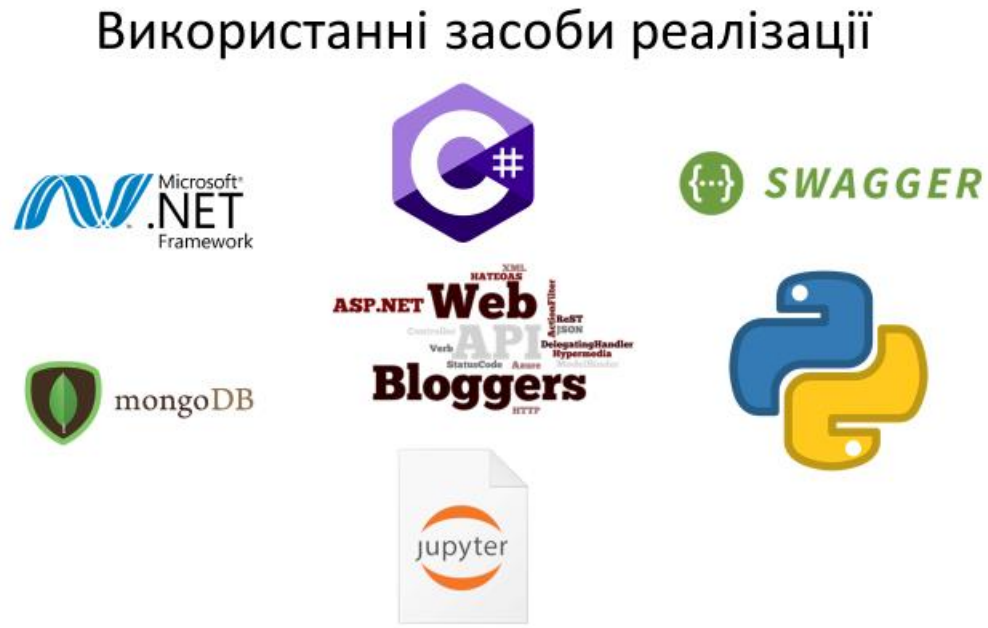


Рисунок Г.11 – Використанні засоби реалізації

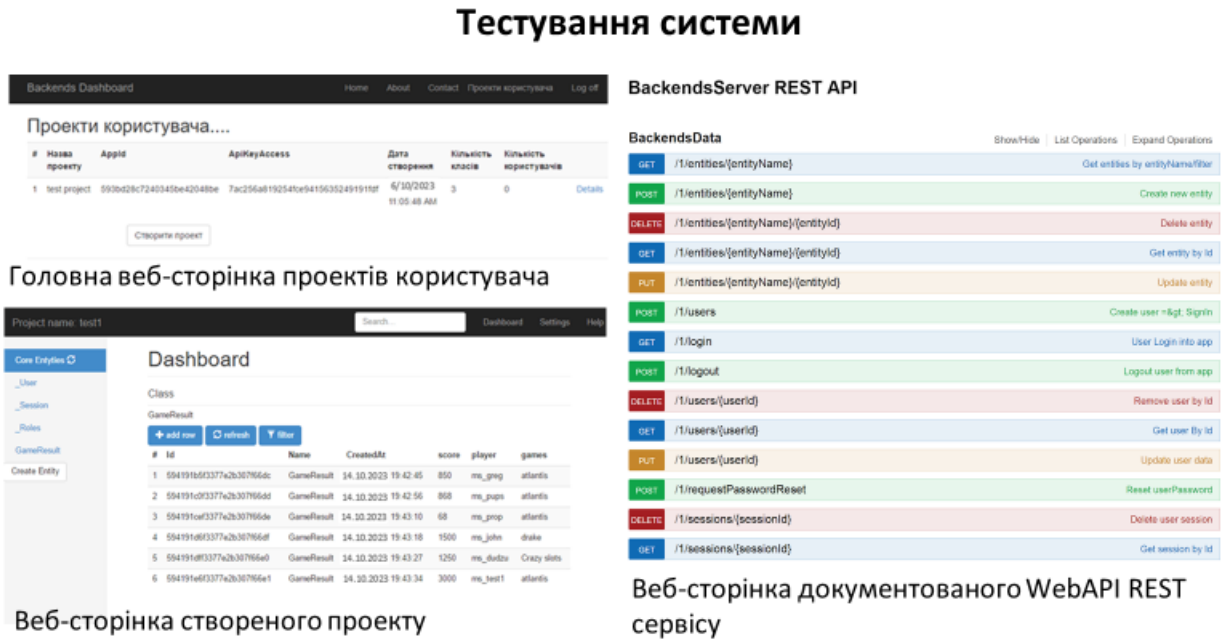
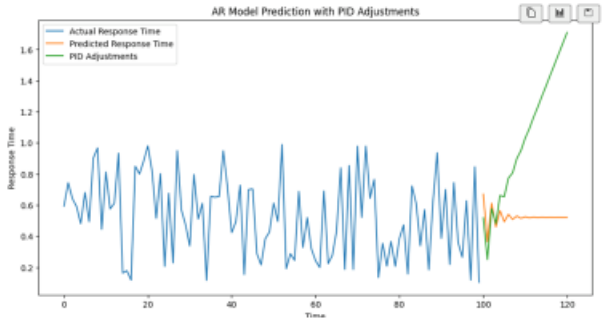
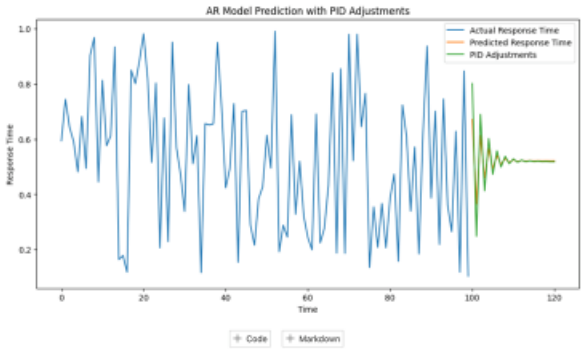


Рисунок Г.12 – Тестування створення даних користувача

Тестування системи



Результат налаштування коефіцієнтів ПІД-контролера за допомогою методики Зіглера-Нікольса



Результат налаштування коефіцієнтів ПІД-контролера за допомогою функції minimize

Рисунок Г.13 – Тестування адаптивного алгоритму

Тестування системи

Функціональне тестування:

- 30 юніт-тестів
- 20 інтеграційних тестів.

100% - завершилися успішно

Покриття тестами програмних вимог — 100%

Покриття тестами коду платформи — 78%

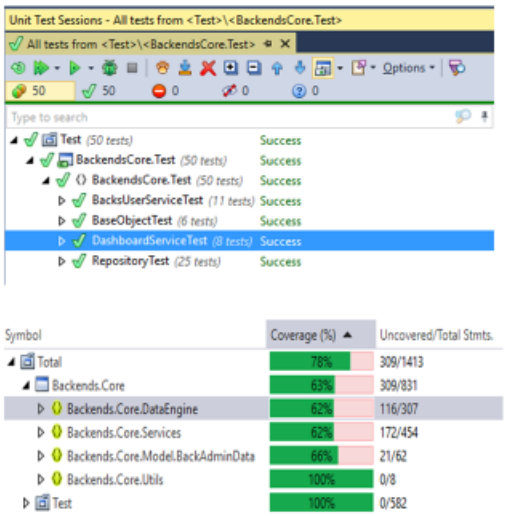


Рисунок Г.14 – Тестування сервісів

Наукова новизна отриманих результатів:

- Отримав подальшого розвитку адаптивний метод передачі повідомлень, що на відміну від традиційних методів, які зосереджуються на статичному управлінні повторною передачею повідомлень, використовує динамічне адаптування на основі часових рядів та статистичного аналізу, що дозволяє системі ефективно реагувати на зміну в мережі та запобігати втраті повідомлень.

- Отримала подальший розвиток модель високорівневої адаптивної системи повторної відправки повідомлень, яка на відміну від існуючих інтегрує адаптивний механізм на основі ПІД-контролера для регулювання поведінки системи з урахуванням поточних умов роботи, що дозволяє їй самостійно адаптуватися до змін у навантаженні та ефективно керувати процесом передачі повідомлень.

Практична цінність отриманих результатів:

- Практична цінність отриманих у ході магістерської кваліфікаційної роботи результатів полягає в розробці BaaS-платформи, яка застосовує адаптивний алгоритм для повторних спроб передачі повідомлень. Впровадження цієї платформи може значно підвищити ефективність обробки запитів та зменшити відмови в системах у різних галузях, від фінансових технологій до хмарних обчислень.

Рисунок Г.15 – Наукова новизна та практична цінність

Висновки

У процесі виконання магістерської кваліфікаційної роботи було зроблено:

- проведено аналіз існуючих методів передачі повідомлень у BaaS платформах;
- розроблено адаптивний алгоритми для повторної передачі повідомлень;
- розроблено дизайн архітектури BaaS-платформи;
- розробити програмні сервіси BaaS-платформи;
- проведено інтеграцію створених алгоритмів у BaaS-платформу;
- проведено оцінку ефективності запропонованих алгоритмів у різних сценаріях використання;
- проведено тестування розробленої платформи.

Рисунок Г.16 – Висновки

Апробації

Результати магістерської кваліфікаційної роботи доповідалися та обговорювалися на:

- науково-практичній конференції «Електронні інформаційні ресурси: створення, використання, доступ» (Вінниця 2023)

Рисунок Г.17 – Апробації

Публікації

Основні результати дослідження опубліковані в наступній науковій роботі:

- тези доповіді на науково-практичній конференції «Електронні інформаційні ресурси: створення, використання, доступ» (Вінниця 2023)

Рисунок Г.18 – Публікації