Міністерство освіти і науки України
Харківський національний університет радіоелектроніки

Факультет *Комп'ютерних наук*
(повна назва)
Кафедра *Системотехніки*
(повна назва)

КВАЛІФІКАЦІЙНА РОБОТА
Пояснювальна записка

Рівень вищої освіти другий (магістерський)
(освітньо-кваліфікаційний рівень)

Дослідження методів реалізації мікросервісної архітектури для розподілу функцій проекту системи обліку івент-послуг
(тема роботи)

Виконав:
студент 2 курсу, групи *СПРм-22-1*
Ходирєв Є.О.
(прізвище, ініціали)

Спеціальність 122 – «Комп'ютерні науки»
(код і повна назва спеціальності)

Тип програми освітньо-наукова
Освітня програма Системне проектування
(повна назва освітньої програми)

Керівник *доц. Коваленко А.І*
(прізвище, ініціали)

Допускається до захисту

Зав. кафедри СТ _____ *проф. Гребеннік І.В.*
(підпис) (прізвище, ініціали)

2024 р

Я, як студент ХНУРЕ, розумію і підтримую політику закладу із академічної доброчесності. Я не надавав і не одержував недозволену допомогу під час підготовки кваліфікаційної роботи. Використання ідей, результатів і текстів інших авторів мають посилання на відповідне джерело.

20.06.2024 Ходирєв Є.О.

Атестаційна робота не містить відомостей заборонених до відкритого опублікування.

Атестаційна робота виконана у відповідності до стандартів, що діють в Україні.

Попередній захист проведений «20» червня 2024 р.

Харківський національний університет радіоелектроніки

(назва вищого навчального закладу)

Факультет *Комп'ютерних наук*

Кафедра *Системотехніки*

Рівень вищої освіти *другий(магістерський)*

Спеціальність *122 «Комп'ютерні науки»*

Тип програми *освітньо-наукова*

Освітня програма *Системне проектування*

ЗАТВЕРДЖУЮ:

Зав. кафедри *СТ проф. Гребеннік І.В.*

(підпис)

" _____ " _____ *2024 р.*

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

Студентові *Ходирєву Єгору Олеговичу*

(прізвище, ім'я, по батькові)

1. Тема роботи *Дослідження методів реалізації мікросервісної архітектури для розподілу функцій проекту системи обліку івент-послуг*

затверджена наказом по університету від " *13* **"** *червня* **2024р. №** *695Ст*

2. Термін подання студентом роботи *20 червня 2024 р.*

3. Вихідні дані до роботи (проекту) Провести аналіз існуючих методів реалізації мікросервісної архітектури, та застосувати їх для оптимизації та покращення системи обліку івент-послуг. Побудувати модель мікросервісної архітектури для системи обліку івент-послуг, яка буде описувати компоненти системи, їхню взаємодію, принципи розгортання та підтримки системи.

4. Зміст пояснювальної записки (перелік питань, що потрібно розробити)

4.1 Вступ. 4.2 Аналіз предметної области. 4.2.1 Актуальність теми. 4.2.2 Огляд і аналіз сучасного стану роботи обліку івент-послуг фірми «White room". 4.2.3 Постановка задачі. 4.3 Аналіз методів будування інформаційної системи обліку івент-послуг на базі принципів мікросервісної архітектури. 4.3.1 Огляд основних принципів мікросервісної архітектури. 4.3.2 Аналіз технологій та інструментів для реалізації мікросервісної архітектури в інформаційній системі обліку івент-послуг. 4.3.3 Аналіз патернів проектування мікросервісів для реалізації інформаційної системи обліку івент-послуг. 4.4 Реалізація інформаційної системи обліку івент-послуг на базі принципів мікросервісної архітектури. 4.4.1 Проектування та розробка мікросервісів. 4.4.2 Інтеграція та комунікація між мікросервісами. 4.4.3 Розгортання та підтримка системи. 4.5 Висновки. 4.6 Перелік джерел посилання. 4.7 Додатки

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень, плакатів)

5.1 Модель реалізації MVC в інформаційній системі обліку замовлень івент-послуг «White Room» (аркуш формату А4); 5.2 Приклад структури реляційної бази даних (аркуш формату А4); 5.3 Приклад структури документо-орієнтованої NoSQL бази даних (аркуш формату А4); 5.4 Приклад структури NoSQL бази даних, що зберігає дані в форматі ключ-значення (аркуш формату А4); 5.5 Схема з різницею між синхронною та асинхронною комунікацією між

сервісами (аркуш формату А4); 5.6 Схема роботи контейнеризації в випадку використання Docker (аркуш формату А4).

6. Консультанти з роботи із зазначенням розділів роботи, що їх стосуються

Найменування розділу	Консультант (посада, прізвище, ім'я, по батькові)	Позначка консультанта про виконання розділу	
		(підпис)	(дата)
Аналіз предметної області	*Доц. Коваленко А.І.*		*05.05.24*
Опис прийнятих проектних рішень	*Доц. Коваленко А.І.*		*08.06.24*

7. Дата видачі завдання *28 квітня 2024 р.*

КАЛЕНДАРНИЙ ПЛАН

Пор. №	Назва етапів атестаційної роботи	Термін виконання етапів роботи	Примітка
1.	*Отримання завдання атестаційної роботи*	*28.04.24*	
2.	*Аналіз завдання, літератури та аналогів з теми атестаційної роботи*	*28.04 — 05.05.24*	
3.	*Вибір засобів для розробки технічних вимог до програми*	*05.05 — 10.05.24*	
4.	*Структурне проектування*	*10.05 – 15.05.24*	
5.	*Вибір середовища розробки програми*	*15.05 – 20.05 24*	
6.	*Розробка програми*	*20.05 — 25.05.24*	
7.	*Тестування програми*	*25.05 — 30.05.24*	
8.	*Розробка «Посібника користувача»*	*30.05 — 05.06.24*	
9.	*Оформлення пояснювальної записки та програмної документації*	*05.06 — 08.06.24*	
10.	*Оформлення графічної частини та презентаційних матеріалів комп'ютерного захисту*	*08.06.24-15.06.2024*	
11.	*Представлення на рецензування*	*18.06.2024*	
12.	*Представлення атестаційної роботи в ДЕК*	*20.06.2024*	

Студент _____ Ходирєв Є.О.
(підпис)

Керівник роботи _____ доцент Коваленко А.І.
(підпис)

РЕФЕРАТ

Пояснювальна записка до кваліфікаційної роботи: 56 стор., 20 рис, 2 додатки, 29 джерел. Графічний матеріал атестаційної роботи містить 6 аркушів.

МІКРОСЕРВІСНА АРХІТЕКТУРА, РОЗПОДІЛЕНІ СИСТЕМИ, СИСТЕМА ОБЛІКУ ІВЕНТ-ПОСЛУГ, ОБ'ЄКТНО-ОРІЄНТОВАНЕ ПРОГРАМУВАННЯ, JAVA, МАСШТАБОВАНІСТЬ, CI/CD

Об'єкт дослідження – процес реалізації мікросервісної архітектури інформаційної системи, розробки, розгортання, масштабування та обслуговування мікросервісів.

Предмет дослідження – інформаційні технології та методи розробки інформаційних систем на базі мікросервісної архітектектури та визначення проблемних питань, які виникають під час реалізації мікросервісів.

Мета роботи – дослідження методів реалізації мікросервісної архітектури для розподілу функцій проекту системи обліку івент-послуг.

Методи дослідження – методи мікросервісного проектування: декомпозиція застосунку на сервіси, визначення межі функціональності сервісів, комунікація між ними, методи контейнеризації та оркстрації для забеспечення зручності розгортання та масштабування мікросервісів, принципи DevOps для безперервної інтеграції та безперервного розгортання(CI/CD) мікросервісів.

Результати роботи – модель мікросервісної архітектури для системи обліку івент-послуг, яка буде описувати компоненти системи, їхню взаємодію, принципи розгортання та підтримки системі.

Область застосування – розробка та оптимізація програмного забеспечення для систем обліку івент послуг.

ABSTRACT

Qualification work: 56 p., 20 pic., 29 source, 2 applications. Graphic material attestation work contains 6 poster.

MICROSERVICE ARCHITECTURE, DISTRIBUTED SYSTEMS, EVENT SERVICE ACCOUNTING SYSTEM, OBJECT-ORIENTED PROGRAMMING, JAVA, SCALABILITY, CI/CD

The object of research is the process of implementing the microservice architecture of the information system, development, deployment, scaling and maintenance of microservices.

The subject of the research is information technologies and methods of developing information systems based on microservice architecture and identifying problematic issues that arise during the implementation of microservices.

The purpose of the work is to study the methods of implementing microservice architecture for the distribution of functions of the event-service accounting system project.

Research methods - microservice design methods: decomposition of the application into services, definition of service functionality limits, communication between them, containerization and orchestration methods to ensure ease of deployment and scaling of microservices, DevOps principles for continuous integration and continuous deployment (CI/CD) of microservices.

The results of the work are a microservice architecture model for the event-service accounting system, which will describe the system components, their interaction, the principles of system deployment and support.

The field of application is the development and optimization of software for event accounting systems.

ЗМІСТ

ВСТУП

Сьогодні у світі інформаційних технологій з динамічними бізнес-процесами постійно ростуть вимоги до гнучкості та масштабованості систем, які розроблюються, мікросервісна архітектура – це одне з ключових рішень для розробки та підтримки складних застосунків. Таке рішення дозволяє будувати системи як набір незалежних, легко масштабованих та підтримуваних сервісів, що є особливо актуальним для систем з високими вимогами до адаптивності та продуктивності, наприклад, інформаційна система обліку івент-послуг.

Аналіз існуючих інформаційних систем обліку івент-послуг дозволяє зробити висновок, що такі застосунки мають бути достатньо динамічними та мати можливість застосовувати індивідуальний підхід до кожного святкового заходу, такі вимоги можуть бути задовільнені тільки впровадженням ефективних IT-рішень для управління, підтримки та аналізу святкових заходів. Таким рішенням може бути впровадження мікросервісної архітектури, розгалуження монолітної системи на групу незалежних сервісів дозволить мати необхідну гнучкість, масштабованість та надійність системи, що в свою чергу буде мати позитивний вплив на якість обслуговування та рівень задоволеності клієнтів.

Усе розглянуте вище обумовлює актуальність теми кваліфікаційної роботи «Дослідження методів реалізації мікросервісної архітектури для розподілу функцій проекту системи обліку івент-послуг».

Метою кваліфікаційної роботи є дослідження методів реалізації мікросервісної архітектури для розподілу функцій проекту системи обліку івент-послуг.

Об’єктом дослідження є процес реалізації мікросервісної архітектури інформаційної системи, розробки, розгортання, масштабування та обслуговування мікросервісів.

Предметом дослідження є інформаційні технології та методи розробки інформаційних систем на базі мікросервісної архітектектури та визначення проблемних питань, які виникають під час реалізації мікросервісів.

1 АНАЛІЗ ТА ВИБІР ПРЕДМЕТНОЇ ОБЛАСТІ ДЛЯ ДОСЛІДЖЕННЯ МЕТОДІВ ПОБУДОВИ ІНФОРМАЦІЙНИХ СИСТЕМ НА БАЗІ МІКРОСЕРВІСНОЇ АРХІТЕКТУРИ

1.1 Огляд і аналіз недоліків монолітної архітектури

Монолітна архітектура – це традиційний метод розробки інформаційних систем, яка використовує технологію взаємодії «клієнт – сервер» для глобальної мережі Інтернет. Всі компоненти системи взаємодіють між собою і розгортаються на двох серверах (web-сервері та сервері баз даних). Приклад структури системи на базі монолітної архітектури наведено на рис 1.1.

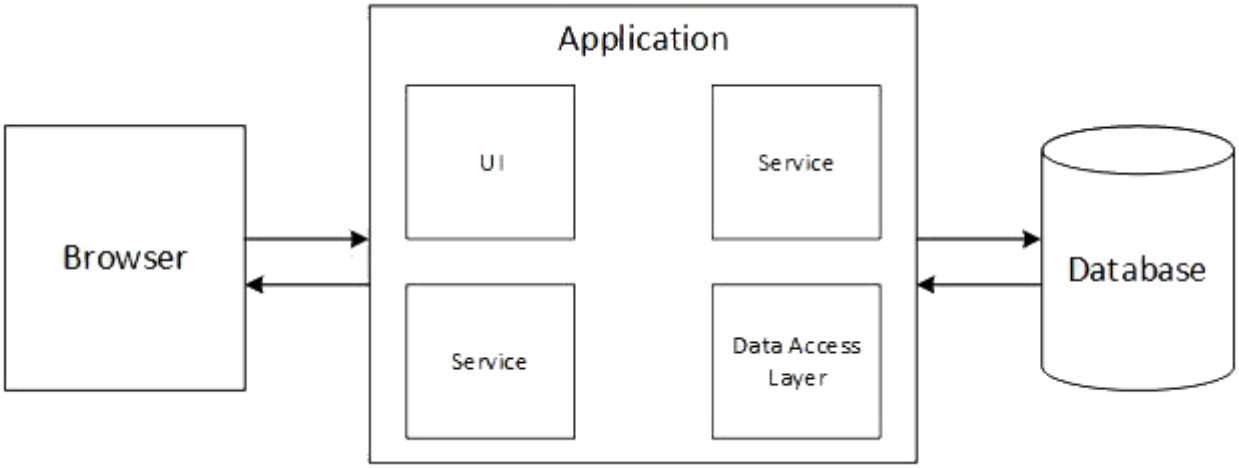

Рисунок 1.1 – Схема структури інформаційної системи побудованої на базі монолітної архітектури

Монолітна архітектура має ряд переваг. Інформаційні системи на основі моноліту зазвичай легші у розробці та тестуванні, оскільки всі компоненти додатку безпосередньо взаємодіють один з одним всередині єдиної системи, що дозволяє швидко вирішувати проблеми, що можуть виникати під час розробки та підтримки таких систем. Крім того, монолітні додатки найчастіше меньш складні в управлінні, оскільки все програмне забезпечення працює в рамках одного технологічного стеку.

Втім, монолітні додатки можуть викликати труднощі під час масштабування, оскільки розширення системи може бути ускладнено через взаємозалежність усіх компонентів. Також складна структура монолітних додатків може ускладнювати їх розробку та підтримку.

1.2 Вибір предметної області для проведення досліджень методів реалізації інформаційних систем на базі мікросервісної архітектури

Для дослідження було обрано проект моєї кваліфікаційної роботи бакалавра [2], в якій було побудовано інформаційну систему обліку святкових заходів(івентів-послуг) на базі монолітної архітектури, приклад діаграми потоків даних системи обліку замовлень з організації святкових заходів(івент-послуг) наведено на рис. 1.2.

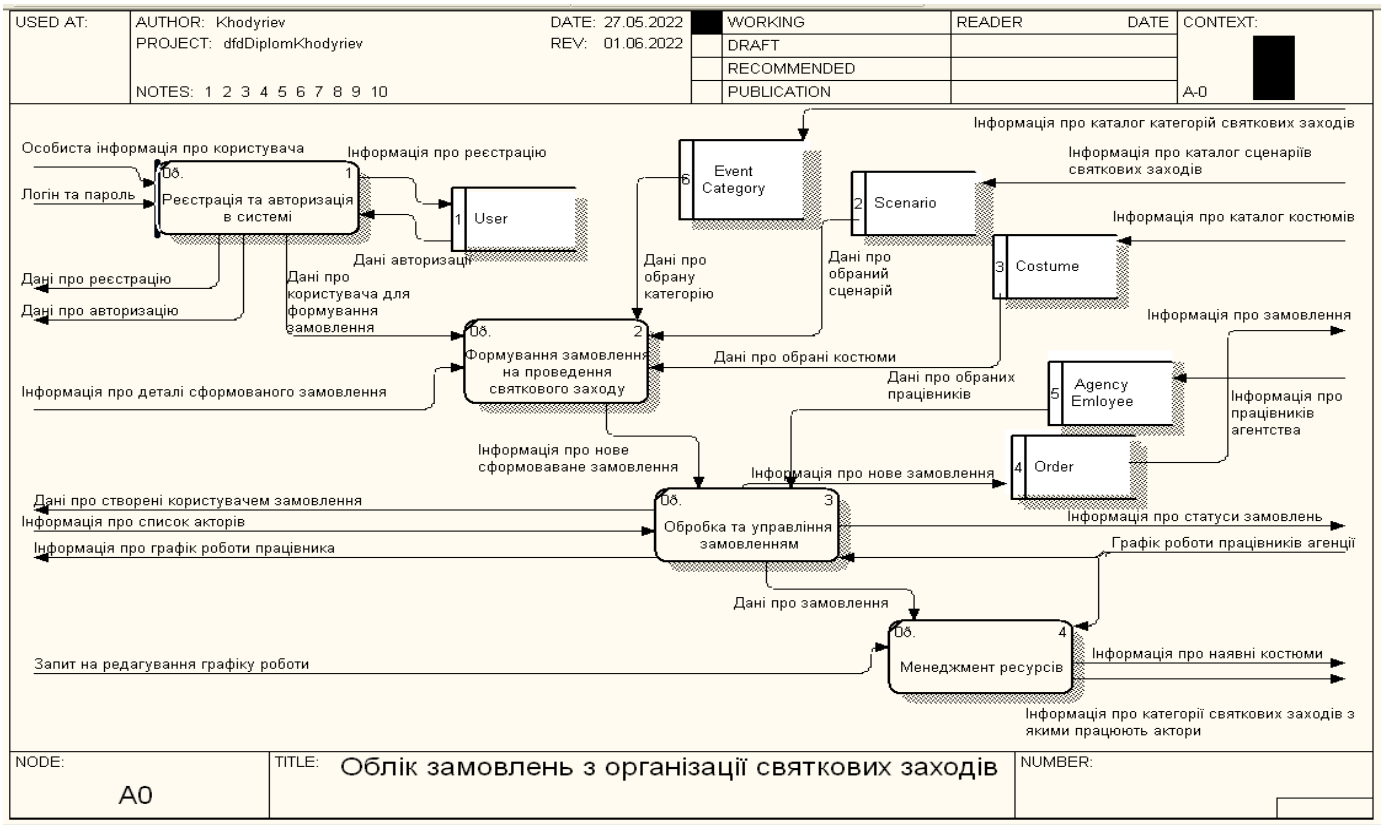

Рисунок 1.2 – Діаграма потоків даних системи обліку замовлень з організації святкових заходів(івент-послуг)

Предметна область івент-послуг – це надання послуг з організації широкого спектру заходів, кожне з яких має свої унікальні вимоги та характеристики. Ці заходи можуть приймати форму невеликих приватних зустрічей або великих міжнародних конференцій та виставок. Основною метою івент-агентств полягає в організації заходів, які будуть відповідати вимогам та очікування клієнтів.

Основні типи заходів, які можуть бути організовані за допомогою інформаційної системи обліку івент-послуг:

– корпоративні події – поєднують в собі конференції, тренінги. тімбілдинги, ювілеї компаній та корпоративні вечірки. Основна мета таких заходів – згуртування команди, обмін досвідом та знаннями, а також святкування важливих дат та досягнень компанії;

– приватні заходи – до цього типу заходів відносять весілля, ювілеї, дні народження та інші приватні святкування. Мета організаторів – створити подію, що запам’ятається клієнту, та врахує його індивідуальні побажання;

– публічні заходи – фестивалі, концерти, виставки, спортивні події, які можуть збирати навколо себе тисячі учасників. Організація таких заходів вимагає особливої уваги до логістики, безпеки та маркетингу.

Вимоги до організації та проведенню заходів:

– логістика – ефективне управління логістикою – ключ до успіху будь-якого заходу. Це поєднує між собою транспортування, розміщення, управління великою кількістю людей та матеріально-технічне забезпечення місця проведення;

– бюджет – створення та дотримання бюджету вимагає детального планування та контролю за витратами. Важливо вміти оптимізувати витрати, не знижуючи якості заходу.

Поточні тенденції та виклики:

– діджиталізація – впровадження цифрових технологій, наприклад віртуальні або гібридні заходи, стали більш популярними, після карантинних часів та зараз у під час війни. Впровадження нових типів заходів вимагає від івент-агентства гнучкості та можливості адаптуватися до нових форматів [3];

– персоналізація – учасники події очікують більш персоналізованого підходу та унікального досвіду. Організаторам необхідно враховувати індивідуальні вподобання та інтереси своєї аудиторії.

Отже, предметна область івент-послуг динамічно розвивається та вимагає від інформаційної системи обліку івент-послуг надання можливості користвачам слідувати поточним трендам та використати новітні технології. Успіх в цієї предметної галузі безпосередньо залежить від здатності системи адаптуватись до змін та від якості послуг, які повинні відповідати очікуванням клієнтів [4].

Інформаційна система обліку івент-послуг базується на принципах монолітної архітектури з використанням мови програмування Java (рис 1.3), що забезпечує централізоване управління всіма аспектами організації заходів. Система об’єднує в собі функціонал для планування, координації, управління ресурсами та комунікації з клієнтами. Система дозволяє вести облік заходів, управління замовленнями та взаємодію з постачальниками послуг.

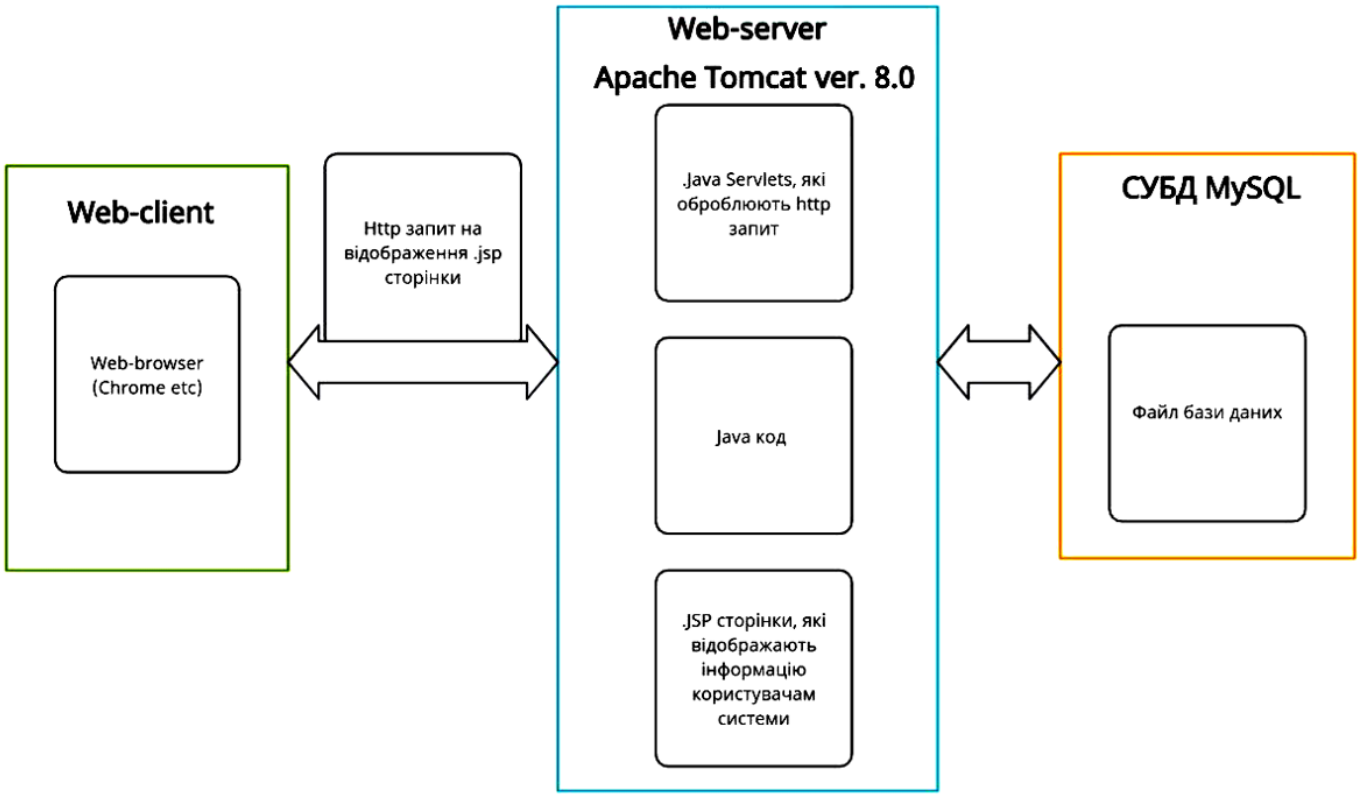

Рисунок 1.3 – Структура інформаційної системи обліку івент-послуг на базі монолітної архітектури

Основні компоненти системи мають в своєму складі:

– «модуль управління клієнтами» – зберігає інформацію про клієнтів, їхні замовлення та історію взаємодії з системою [5];

– «модуль планування заходів» – інструменти для створення, планування та координації заходів, включаючи розподіл ресурсів та управління графіком;

– «модуль менеджменту персоналу» – зберігає інформацію про працівників івент-агентства, категорії заходів з якими вони працюють;

– «модуль авторизації» – необхідний для надання можливості створювати замовлення клієнтам, або працівникам для роботи з системою;

– «модуль MVC (Model View Controller)» – реалізація однойменного дизайн паттерну програмування [6], необхідний для передачі та відображення даних в вигляді веб-сторінки відповідно до запиту користувача інформаційної системи, приклад моделі реалізації в застосунку фірми «White room» зображено на рис 1.4;

– «модуль управління інвентарем» – зберігає інформацію про наявні костюми та інший реквізит для проведення заходів [7].

Незважаючи на наявність широкого спектру функціональності, система стикається з низкою викликів, пов'язаних з її монолітною архітектурою та процесами розгортання:

– складність внесення змін та оновлень – кожна окрема зміна в системі вимагає перекомпіляції та перезапуску всього додатку, що може призвести до простоїв у роботі та збільшити ризик появлення помилок [8];

– мануальне розміщення додатку на хостінгу – під час процесу розгортання проходить ручне копіювання скомпільованого WAR-файлу на сервер, на таку процедуру витрачається багато часу, це призводить до збільшеного ризику виникнення помилок. Відсутність автоматизованого процесу безперервної інтеграції та безперервної доставки (Continuous Integration/Continuous Delivery) знижує ефективність розробки та ускладнює управління версіями [9];

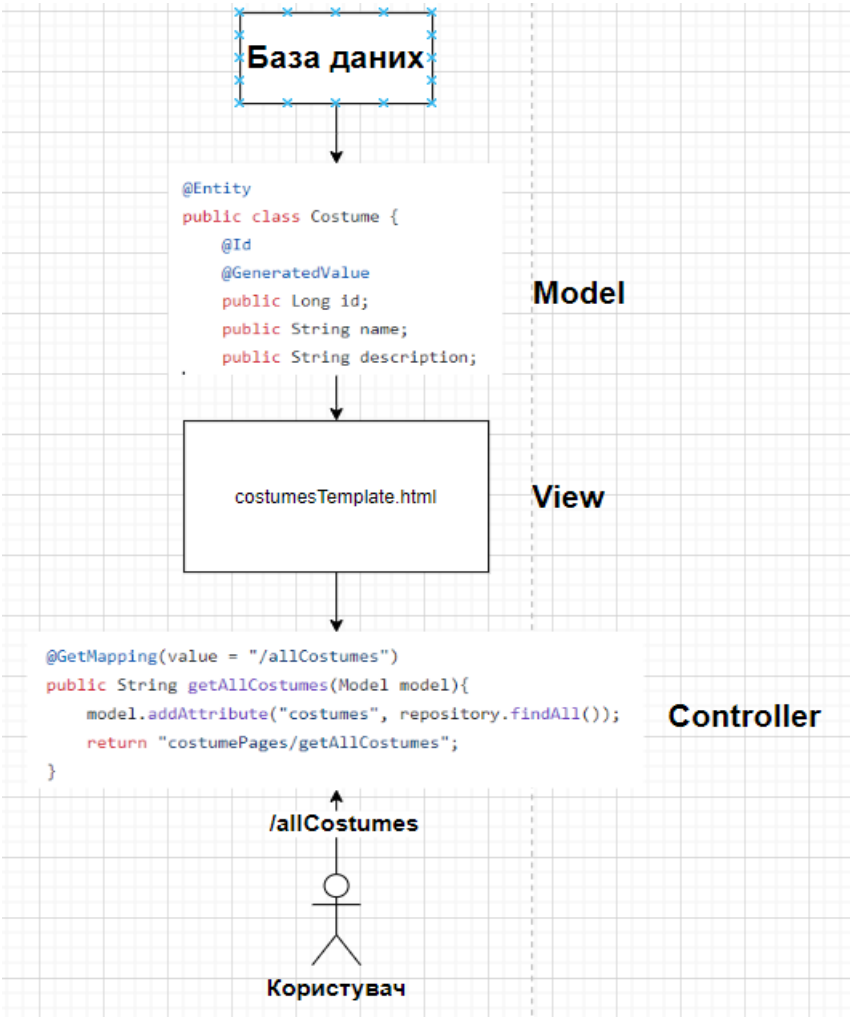

Рисунок 1.4 – Модель реалізації MVC в інформаційній системі обліку замовлень івент-послуг «White room»

– складність оновлення компонентів – оскільки всі компоненти тісно пов'язані між собою, зміни в одній частині системи можуть вимагати змін в інших частинах, що ускладнює процес оновлення та розширення функціоналу;

– обмежена гнучкість – через використання однієї мови програмування монолітна архітектура може обмежувати можливості інтеграції з новими сервісами та технологіями, що є критично важливим для інновацій та підтримки конкурентоспроможності в галузі івент-послуг [10];

– проблеми з надійністю – в монолітних системах помилка в одному компоненті може призвести до збоїв у роботі всієї системи, що ставить під загрозу стабільність обслуговування клієнтів та проведення заходів [11];

– відсутність використання контейнерів – не використовуються технології контейнеризації, такі як Docker, які могли б спростити процес розгортання, забезпечити консистентність середовищ та поліпшити ізоляцію компонентів.

1.3 Постановка задачі для проведення досліджень методів реалізації систем на базі мікросервісної архітектури

Мета роботи є дослідження методів реалізації мікросервісної архітектури для розподілу функцій проекту системи обліку івент-послуг.

Розробка та впровадження системи обліку івент-послуг для фірми «White room», побудованої за принципами мікросервісної архітектури, дозволить підвищити ефективність управління заходами, оптимізувати процеси роботи та покращити якість обслуговування клієнтів.

Для проведення дослідження необхідно виконати наступні завдання:

– огляд основних методів реалізації мікросервісної архітектури та проблемних питань, які виникають під час розробки сервісів. Метод декомпозиції системи, метод незалежного розгортання, метод реалізації децентралізованого управління даними та метод автономної розробки окремих компонентів системи. ;

– аналіз технологій та засобів необхідних для реалізації методів реалізації мікросервісної архітектури інформаційної системи обліку івент-послуг та вирішення проблемних питань розробки сервісів. Огляд мов програмування для розробки сервісів системи, аналіз баз даних для зберігання даних сервісів, порівняння використання синхронної та асинхронної комунікації між сервісами, огляд використання контейнеризації для ізоляції сервісів, аналіз засобів для реалізаії безперервної інтеграції та безперервного розгортання сервісів;

– аналіз використання патернів проектування мікросервісів для реалізації інформаційної системи обліку івент-послуг. Огляд типів патернів, патерни декомпозиції, патерни інтеграції, патерни реалізації відмовостійкості системи;

– дослідження визначеного варіанту розробки інформаційної системи на базі мікросервісної архітектури. Визначення бізнес-функцій, які будуть реалізовувати

сервіси, реалізація патернів API Gateway та Service Registry, визначення даних, які будуть зберігатись в базах даних окремих сервісів;

– дослідження визначеного варіанту реалізації взаємодії між мікросервісами, проектування архітектури та взаємодії між мікросервісами, планування процесів комунікації. Визначення використання REST API для комунікації між сервісами, включаючи вибір відповідних HTTP-методів («GET», «POST», «PUT», «DELETE») для кожного з сервісів, а також формат даних для обміну (наприклад, JSON). Проектування кінцевих точок (endpoints) сервісів, що включає визначення URL-структури, необхідних параметрів запиту та типів даних. Реалізація патерну Retry для вирішення проблем відмовостійкості сервісів під час комунікації, що включає встановлення умов для повторних спроб (умови для повторної спроби), максимальну кількість спроб, інтервал між спробами (зокрема, використання фіксованого інтервалу або експоненційного зростання інтервалу), а також обробку успішних та невдалих спроб. Написання коду реалізації патерну Retry з використанням відповідних бібліотек та інструментів, таких як Spring Retry у середовищі Spring Boot;

– дослідження розгортання та підтримки систем, використання Docker для контейнеризації мікросервісів. Написання Dockerfile для створення образів сервісів для подальшого розгортання контейнерів, огляд функціоналу операторів коду Dockerfile, використання Docker Compose для розгортання системи, написання файлу compose для створення образу системи для подальшого розгортання масиву контейнерів, огляд функціоналу операторів коду compose-файлу;

– дослідження ефективності та переваг реалізованої системи обліку івент послуг на базі мікросервісної архітектурі, оцінка масштабованості системи, відзначення гнучкості та модульності системи, зазначення переваг незалежності розгортання сервісів, оцінка переваг від можливості системи бути технологічно різноманітною, зазначення покращення відмовостійкості системи за рахунок ізоляції сервісів.

2 АНАЛІЗ МЕТОДІВ РЕАЛІЗАЦІЇ ІНФОРМАЦІЙНОЇ СИСТЕМИ ІВЕНТ-ПОСЛУГ НА БАЗІ МІКРОСЕРВІСНОЇ АРХІТЕКТУРИ

2.1 Огляд основних методів реалізації мікросервісної архітектури та проблемних питань, які виникають під час розробки сервісів

Мікросервісна архітектура представляє собою підхід до розробки програмного забезпечення, який полягає у створенні додатків як набору незалежних сервісів, що спілкуються між собою за допомогою протоколів HTTP або REST, або через системи обміну повідомленнями. Кожен мікросервіс відповідає за виконання добре визначеної, конкретної бізнес-функції і може бути розроблений, розгорнутий, запущений і масштабований незалежно від інших сервісів [13].

Декомпозиція на сервіси є фундаментальним методом мікросервісної архітектури. Цей метод дозволяє розбити комплексну, монолітну інформаційну систему обліку івент-послуг на менші, керовані частини, що спрощує розробку, тестування, розгортання та масштабування, порівняння монолітної та мікросервісної структури системи представлено на рис. 2.1. Цей підхід також сприяє паралельній розробці різними командами, що може значно прискорити процес розробки та впровадження нових функцій.

Метод незалежного розгортання сервісів забезпечує високу гнучкість у внесенні змін і швидке впровадження змін в інформаційній системі без необхідності зупиняти роботу всієї системи [14]. Це також полегшує процес безперервної інтеграції та безперервної доставки (Continuous Integration/Continuous Delivery), дозволяючи автоматизувати тестування та розгортання застосунку.

Метод реалізації децентралізованого управління даними в мікросервісній архітектурі дозволяє кожному сервісу використовувати власну базу даних або сховище даних, оптимізоване під конкретні потреби цього сервісу. Це сприяє

підвищенню продуктивності та масштабованості, але також ставить під загрозу забезпечення консистентності даних між сервісами [15].

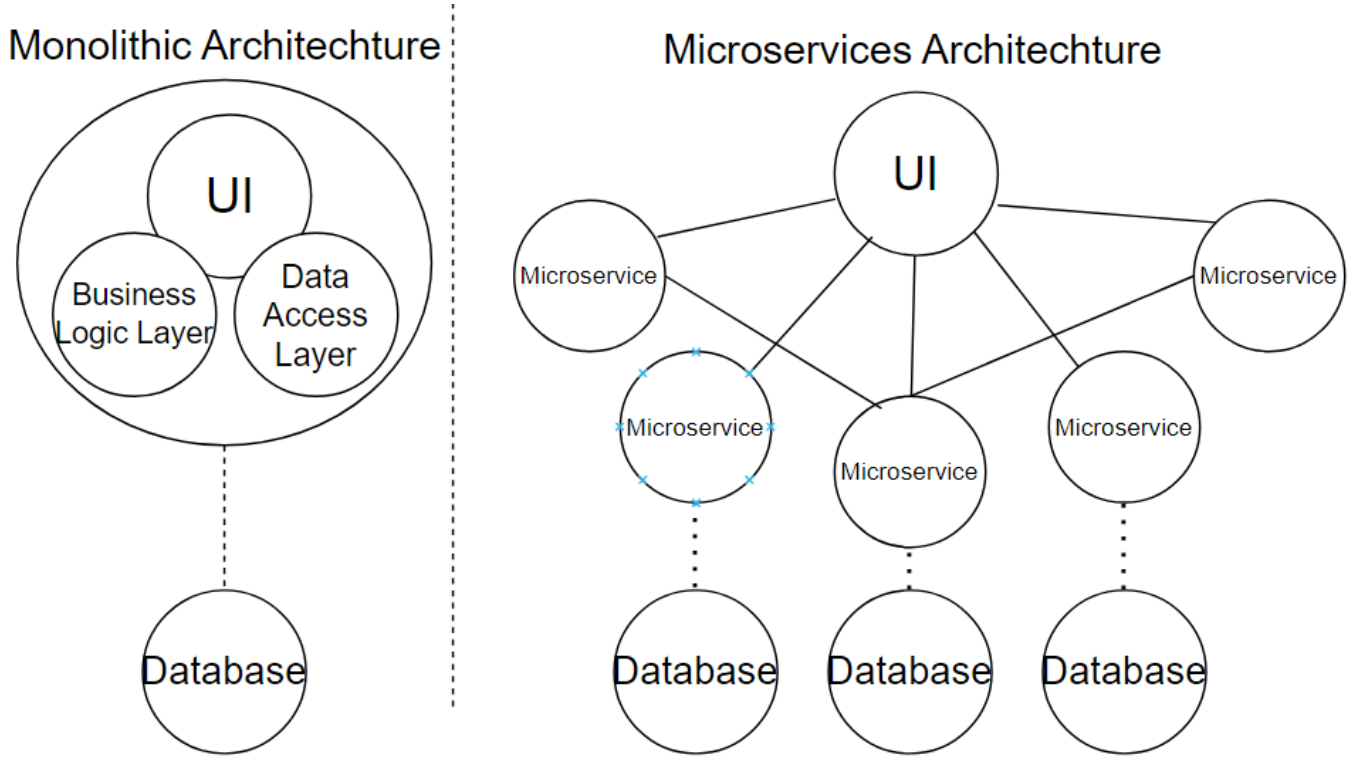

Рисунок 2.1 – Порівняння монолітної та мікросервісної структури системи

Метод автономної розробки окремих компонентів системи дозволяє кожній команді може обирати найбільш доречні технології та інструменти для свого мікросервісу – такий принцип сприяє інноваціям та використанню найкращих рішень для вирішення конкретних задач інформаційної системи обліку івент-послуг [16].

Порівняння процесу розробки сервісів інформаційних систем побудованих на базі монолиту та мікросервісів представлено на рис. 2.2.

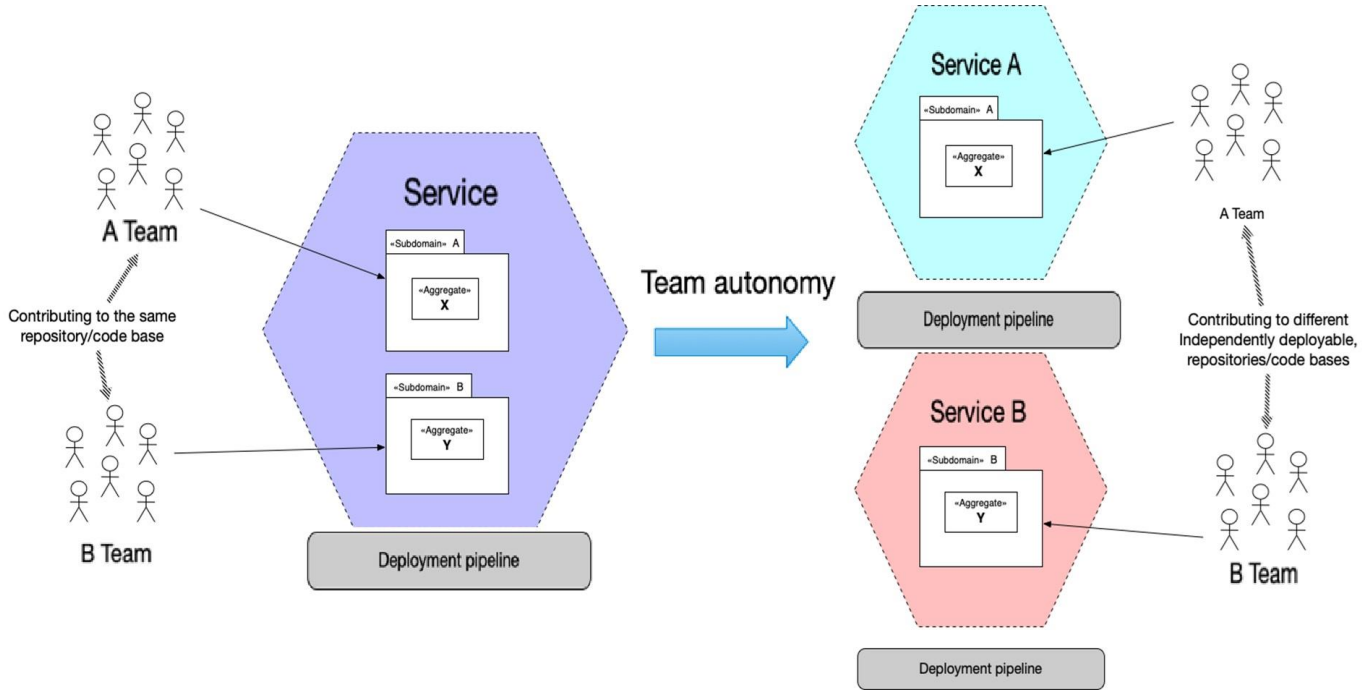

Рисунок 2.2 – Порівняння процесу розробки сервісів монолітної та мікросервісної системи різними командами

RESTful API є найпопулярнішим вибором для синхронної комунікації між мікросервісам, в свою чергу для асинхронної комунікації найчастіше використовуються системи обміну повідомленнями, такі як RabbitMQ або Kafka [17].

Масштабованість мікросервісів може бути досягнуто шляхом горизонтального масштабування, тобто додаванням більшої кількості екземплярів сервісу, що дозволяє системі ефективно обробляти зростаючий обсяг запитів.

Стійкість до відмов є важливою перевагою мікросервісних архітектур. На відміну від монолітних систем, завдяки ізоляції сервісів, в мікросервісній архітектурі, відмова одного сервісу не призводить до збоїв у роботі всієї системи. Патерни, такі як Circuit Breaker, допомагають запобігати поширенню помилок.

Важливим доповненням до зазначених принципів мікросервісної архітектури є автоматизація.

Автоматизація в контексті мікросервісної архітектури — це процес впровадження різноманітних інструментів та методик, спрямованих на зменшення ручної праці та підвищення ефективності розробки, розгортання, моніторингу та управління мікросервісами, охоплює широкий спектр діяльності:

– автоматизація розробки - поєднує в собі використання інструментів для автоматичного генерування коду, стандартизації кодування, автоматичного виконання тестів (модульних, інтеграційних, навантажувальних) та аналізу якості коду. Це дозволяє розробникам зосередитися на бізнес-логіці, зменшуючи час на рутинні операції та підвищуючи якість продукту.

– автоматизація розгортання - забезпечує можливість швидкого та безпомилкового розгортання мікросервісів у різних середовищах (розробка, тестування, продакшн) з мінімальним втручанням людини. Включає в себе автоматичне створення інфраструктури, налаштування середовищ, розгортання сервісів та їх конфігурацію.

– автоматизація моніторингу та логування - полягає у впровадженні систем, які автоматично збирають, агрегують та аналізують логи та метрики з усіх мікросервісів. Це дозволяє оперативно виявляти та діагностувати проблеми, аналізувати продуктивність системи та прогнозувати потенційні критичні помилки.

– автоматизація управління інфраструктурою - включає в себе використання інструментів інфраструктури як коду (IaC) для автоматичного створення, налаштування та управління інфраструктурою, необхідною для роботи мікросервісів. Це забезпечує швидке масштабування, відновлення системи після збоїв та зниження витрат на управління інфраструктурою.

Автоматизація відіграє ключову роль в побудові ефективної системи обліку івент-послуг на базі мікросервісної архітектури, оскільки дозволяє значно підвищити ефективність розробки, тестування, розгортання та експлуатації системи. Вона спрямована на мінімізацію ручної праці, зниження ризику помилок, прискорення випуску оновлень та підтримку високої доступності та надійності сервісів.

Автоматизація дозволить мінімізувати такі виклики під час розробки інформаційної системи обліку івент-послуг на базі мікросервісної архітектури:

– складність управління - мікросервісні системи складаються з багатьох незалежних компонентів, такі системи вимагають координованого управління версіями, конфігураціями та залежностями;

– збільшена кількість оновлень компонентів системи - незалежність сервісів дозволяє частіше оновлювати окремі частини системи, що вимагає ефективних механізмів неперервної інтеграції та доставки;

– масштабованість та відмовостійкість - автоматизація дозволяє швидко масштабувати сервіси відповідно до потреб та забезпечувати автоматичне відновлення сервісів у разі критичних помилок.

2.2 Аналіз технологій і методів, необхідних для реалізації мікросервісної архітектури інформаційної системи обліку івент-послуг, та вирішення проблемних питань розробки сервісів

Для реалізації мікросервісної архітектури в системі обліку івент-послуг необхідно вибрати відповідні технології та інструменти, які забезпечать ефективну розробку, розгортання та управління мікросервісами. Основні аспекти, які потрібно врахувати, включають мови програмування, фреймворки, бази даних, інструменти контейнеризації та оркестрації, а також системи моніторингу та логування.

Мікросервіси можуть бути реалізовані різними мовами програмування, залежно від конкретних вимог та переваг команди розробників. Популярними варіантами є:

Java – одна з найпопулярніших мов програмування для розробки мікросервісів, завдяки своєї надійності, широкому функціоналу та великій екосистемі [18]. Використання Java дозволяє створювати високопродуктивні, масштабовані та безпечні мікросервіси, які можуть ефективно взаємодіяти в розподіленій системі.

Переваги Java для розробки інформаційної системи обліку івент-послуг побудованої на базі принципів мікросервісної архітектури:

– зрілість та стабільність - Java має довгу історію успішного використання в різноманітних областях, включаючи великі корпоративні системи, що забезпечує надійність та стабільність для мікросервісних рішень;

– широка екосистема - Java має велику кількість бібліотек, фреймворків та інструментів, які спрощують розробку мікросервісів. Це включає веб-фреймворки, бібліотеки для роботи з базами даних, системи моніторингу та логування, а також засоби для автоматизації тестування та розгортання;

– спільнота та підтримка - Java має одну з найбільших спільнот розробників, що забезпечує швидке вирішення проблем, доступ до кращих практик та велику кількість навчальних матеріалів;

– висока продуктивність та масштабованість - Java Virtual Machine (JVM) оптимізована для високої продуктивності та масштабованості, що є критично важливим для інформаційної системи обліку івент-послуг побудованої на базі мікросервісної архітектури, особливо в умовах великого навантаження;

– безпека - Java має потужні вбудовані механізми безпеки, що дозволяє розробляти захищені мікросервіси, здатні протистояти зовнішнім загрозам.

Використання Java для розробки мікросервісів має свої переваги, але й супроводжується певними викликами та обмеженнями:

– високі вимоги до об’єму обчислювальних ресурсів – Java-додатки можуть вимагати значних обчислювальних ресурсів, зокрема великої кількості оперативної пам'яті. Це може стати проблемою під час розгортання великої кількості мікросервісів, особливо в обмежених або ресурсоощадних середовищах, таких як контейнери;

– стартове навантаження – Java-додатки можуть мати відносно довгий час старту через процес ініціалізації Java Virtual Machine (JVM). Це може ускладнити швидке масштабування мікросервісів відповідно до змін у навантаженні;

– управління залежностями – в великих мікросервісних системах управління залежностями та їх версіями може стати складним завданням. Конфлікти залежностей між різними мікросервісами можуть призвести до проблем із сумісністю та стабільністю;

– вертикальне масштабування – традиційно Java-додатки оптимізовані для вертикального масштабування (додавання ресурсів до одного сервера), що може бути не таким ефективним рішенням під час розробки інформаційної системи обліку

івент-послуг побудованої на базі мікросервісної архітектури, де перевага надається горизонтальному масштабуванню (додаванню більшої кількості серверів);

– складність розробки та підтримки – Java-додатки можуть бути складнішими у розробці та підтримці порівняно з деякими іншими мовами програмування, що використовують для будування та розробки мікросервісів, через строгу типізацію та велику кількість шаблонного коду. Це може збільшити час розробки та вимагати від розробників глибших знань Java екосистеми.

Node.js є популярною платформою для розробки інформаційних систем побудованих на базі мікросервісів, особливо коли мова йде про створення легковагових, ефективних та високопродуктивних веб-сервісів.

Використання Node.js для мікросервісної архітектури має кілька ключових переваг:

– асинхронна, модель побудована на подіях - Node.js використовує неблокуючі, подієві цикли, що робить його ідеальним для обробки великої кількості одночасних з'єднань між клієнтом та сервером без значного збільшення навантаження на ресурси [19]. Це особливо важливо для мікросервісів, ефективність яких напряму залежить від високої пропускної здатності та мінімальної затримки;

– швидкість та продуктивність - V8 JavaScript Engine, який використовується в Node.js, забезпечує високу швидкість виконання JavaScript-коду. Це дозволяє мікросервісам побудованим на Node.js швидко обробляти запити та ефективно взаємодіяти з іншими сервісами та базами даних;

– одномовна розробка - використання JavaScript-коду як на клієнтській, так і на серверній стороні дозволяє розробникам легко перемикатися між клієнтськими та серверними частинами додатку, спрощуючи розробку та підтримку існуючого коду;

– велика екосистема - NPM (Node Package Manager) є одним з найбільших репозиторіїв програмного забезпечення, що надає розробникам доступ до величезної кількості пакетів та бібліотек, які можна використовувати для швидкої розробки та розширення функціональності мікросервісів;

– мікросервісні фреймворки - існує багато фреймворків, спеціально розроблених для створення мікросервісів на Node.js, таких як Express.js, Koa.js, NestJS, які надають потужні інструменти для розробки веб-сервісів, RESTful API та мікросервісної інтеграції.

Під час розробки інформаційної системи обліку івент-послуг на базі принципів мікросервісної архітектури розробник може зіткнутись з такими викликами:

– Callback Hell - хоча асинхронна природа Node.js є перевагою, неправильне управління асинхронним кодом може призвести до "callback hell", що ускладнює читання та підтримку коду [20]. Проте, це можна подолати за допомогою Promises та async/await;

– оптимізація продуктивності - для досягнення максимальної продуктивності вимагається глибоке розуміння подієвої моделі Node.js та оптимізація коду.

Також популярною мовою програмування для розробки мікросервісів є мова Python. Завдяки своїй простоті, гнучкості та широкій екосистемі. Використання Python для мікросервісної архітектури має кілька ключових переваг, але й стикається з певними викликами.

Плюсами використання Python для розробки інформаційної системи обліку івент-послуг є:

– простота та швидкість розробки - Python відомий своїм чистим та зрозумілим синтаксисом, що спрощує написання коду та знижує поріг входження для нових розробників. Це дозволяє швидко розробляти та впроваджувати нові мікросервіси;

– велика стандартна бібліотека та екосистема - Python має велику стандартну бібліотеку та широкий вибір сторонніх бібліотек та фреймворків, які покривають майже будь-які потреби розробки, від веб-розробки до роботи з даними та машинного навчання;

– підтримка асинхронного програмування - з введенням асинхронних функцій (asyncio) в Python, розробники мають можливість писати

високопродуктивний асинхронний код, що є важливим для масштабованих мікросервісних систем обліку івент-послуг;

– широке застосування - Python використовується в різних областях, від веб-розробки до наукових досліджень, що робить його універсальним інструментом для розробки мікросервісів, які можуть взаємодіяти з різними системами та даними [21].

Під час розробки інформаційної системи обліку івент-послуг на базі принципів мікросервісної архітектури розробник можу зіткнутись такими викликами:

– продуктивність - хоча Python зазвичай достатньо швидкий для більшості веб-застосунків, він може бути повільнішим за компільовані мови, такі як Java або C#. Це може стати проблемою для дуже високонавантажених систем. Однією з причин проблем з продуктивністю є глобальний інтерпретаторний блокувальник (GIL), GIL в Python обмежує виконання байт-коду до одного потоку в один момент часу, що може ускладнити використання багатопоточності для підвищення продуктивності [22];

– використання пам'яті - Python може використовувати більше пам'яті порівняно з деякими іншими мовами програмування, що може бути важливим фактором під час розгортання великої кількості мікросервісів на обмежених ресурсах.

Під час розробки інформаційної системи обліку івент-послуг на базі мікросервісної архітектури, вибір бази даних відіграє критичну роль. Важливо забезпечити, щоб обрана система управління базами даних (СУБД) відповідала специфічним потребам кожного мікросервісу, а також загальним вимогам до системи. Ось декілька ключових аспектів, які слід врахувати під час аналізу потенційних баз даних:

а) специфіка даних і операції;

1) типи даних – треба визначити чи дані будуть переважно структурованими, чи неструктурованими. Реляційні бази даних ідеально підходять для структурованих даних, тоді як NoSQL бази краще справляються з неструктурованими даними [23];

2) операції з даними – треба оцінити, які операції (читання, запис, оновлення, видалення) будуть найбільш частими, і обрати базу даних, оптимізовану для таких операцій;

б) масштабованість та продуктивність;

1) горизонтальне масштабування – системи побудовані на базі принципів мікросервісної архітектури найчастіше вимагають можливості легкого масштабування. NoSQL бази даних, як правило, краще підходять для горизонтального масштабування;

2) продуктивність – треба враховувати вимоги до швидкості відповіді на запити сервера та обробки великих обсягів даних. В деяких випадках, використання кешування або спеціалізованих баз даних для пошуку може бути корисним;

в) транзакційна послідовність;

1) ACID та BASE – реляційні бази даних забезпечують ACID гарантії (атомарність, послідовність, ізоляція, стійкість), що важливо для операцій, які вимагають серйозної транзакційної послідовності. NoSQL бази частіше пропонують BASE властивості (основна доступність, м'яка становість, врешті консистентний), що може бути достатнім для деяких мікросервісів [24];

г) інтеграція та гнучкість;

1) інтеграція з іншими сервісами – треба бути переконаним, що обрана база даних легко інтегрується з іншими компонентами інформаційної системи, наприклад, з іншими базами даних, зовнішніми сервісами та інструментами розробки;

2) адаптивність схеми – в інформаційних система на базі принципів мікросервісної архітектури вимоги до даних можуть швидко змінюватися. Бази даних з гнучкою схемою дозволяють легше адаптуватися до таких змін.

Найбільш популярними базами даних під час розробки інформаційних систем на базі принципів мікросервісної архітектури є:

– реляційні СУБД: PostgreSQL, MySQL – традиційний вибір для структурованих даних та операцій, що вимагають строгої транзакційної

послідовності, схема з прикладом структури реляційної бази даних відображено на рис 2.3;

– документо-орієнтовані NoSQL: MongoDB, Couchbase – підходять для напівструктурованих даних та швидкого розвитку мікросервісів з гнучкою схемою даних, приклад структури документо-орієнтованої NoSQL бази даних наведено на рис 2.4;

– NoSQL бази даних, які зберігають дані в форматі ключ-значення: Redis, Amazon DynamoDB – ідеально підходять для випадків, що вимагають швидкого доступу до даних за ключем, високої продуктивності та легкості горизонтального масштабування, схема з прикладом структури NoSQL бази даних, що зберігає дані в форматі ключ-значення наведено на рис 2.5;

– бази даних з широким стовпчиком: Apache Cassandra, ScyllaDB – оптимізовані для обробки великих обсягів даних з високою пропускною спроможністю, підходять для читання та запису великих наборів даних [25];

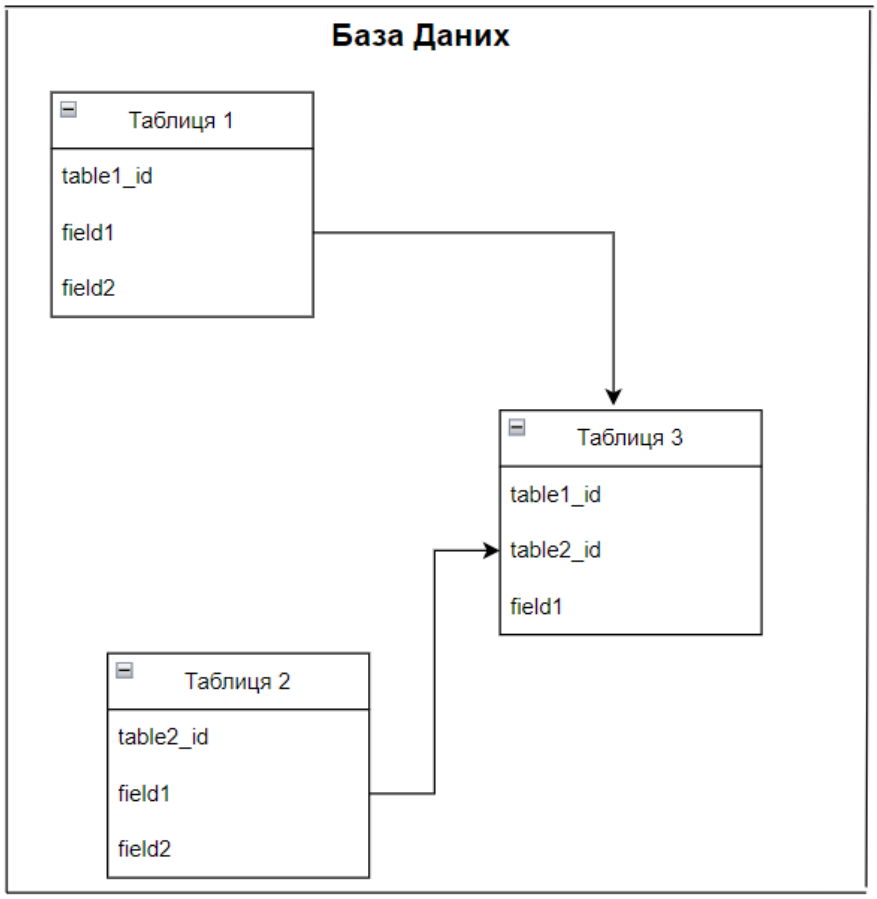

Рисунок 2.3 – Приклад структури реляційної бази даних

Ще одним з фундаментальних аспектів під час побудови інформаційної системи на базі принципів мікросервісної архетектури є комунікація між сервісами.

Розрізняють два види комунікації між сервісами: синхронна та асинхронна комунікація. Синхронна комунікація передбачає прямий виклик сервісу та очікування відповіді перед продовженням обробки. Найчастіше такий тип комунікації реалізується за допомогою HTTP REST або gRPC. Асинхронна комунікація дозволяє сервісу відправляти повідомлення без блокування в очікуванні відповіді, що підвищує продуктивність та відмовостійкість системи. Для асинхронної комунікації найчастіше використовують системи обміну повідомленнями, такі як Apache Kafka, RabbitMQ або AWS SQS, схема з різницею роботи синхронної та асинхронної комунікації наведено на рис 2.6.

Рисунок 2.4 – Приклад структури документо-орієнтованої NoSQL бази даних

Синхронна комунікація зазвичай використовується, коли:

– існує пряма залежність від відповіді – клієнтський запит вимагає негайної відповіді для продовження обробки. Наприклад, запит на отримання даних про користувача під час аутентифікації;

– необхідна міцна інтеграція між сервісами – коли сервіси міцно пов'язані та потребують частих взаємодій для виконання бізнес-операцій;

– необхідна швидка та проста реалізація – синхронні виклики, як правило, простіше реалізувати та вони є зрозумілішими для розробників, особливо в невеликих або нескладних системах;

– необхідна реалізація транзакційності – операції, які вимагають строгих транзакційних гарантій, часто реалізуються через синхронні виклики, щоб забезпечити атомарність та послідовність.

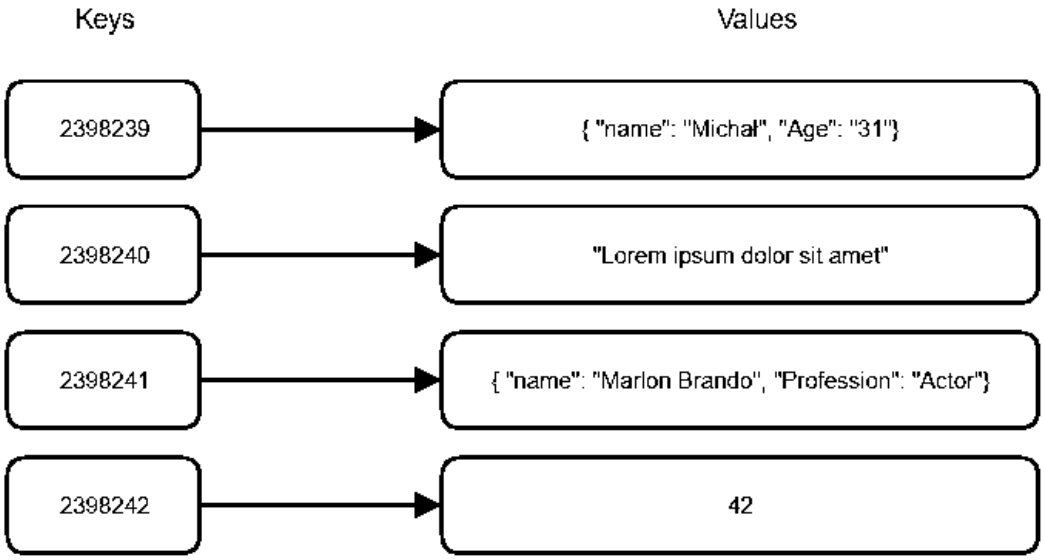

Рисунок 2.5 – Приклад структури структури NoSQL бази даних, що зберігає дані в форматі ключ-значення

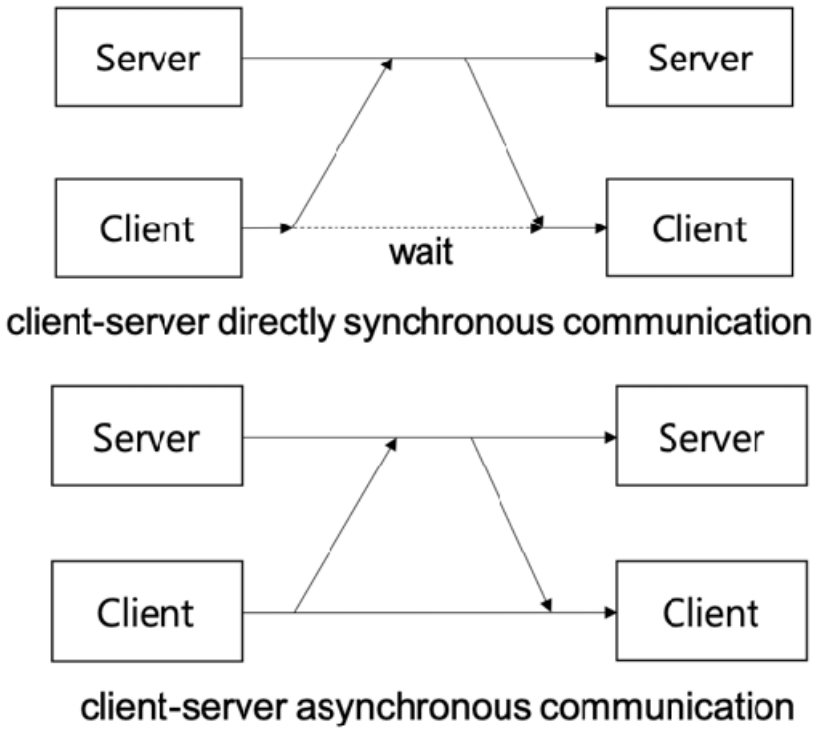

Рисунок 2.6 – Схема з різницею між синхронною та асинхронною комунікацією між сервісами

Асинхронну комунікацію використовують в випадках, коли:

– відсутня необхідність в негайній відповіді – процеси, які не вимагають миттєвої відповіді від сервісу, можуть використовувати асинхронну комунікацію, це дозволяє знизити затримки та підвищити загальну продуктивність системи;

– необхідна висока надійність та відмовостійкість системи – асинхронні системи зазвичай краще оптимізовані для роботи з можливими відмовами окремих компонентів, оскільки повідомлення можуть бути повторно оброблені або відкладені до моменту відновлення роботи сервісу;

– необхідна обробка великих обсягів даних – асинхронні системи ефективні для обробки потоків даних або великих обсягів інформації, дозволяючи розподіляти навантаження в часі;

– необхідна розосередженість компонентів та масштабованість системи – асинхронна комунікація дозволяє простіше масштабувати систему, оскільки сервіси можуть обробляти запити незалежно один від одного, зменшуючи взаємну залежність.

Для того, щоб спростити запуск додатка, на різних пристроях та середовищах, важливим компонентом буде контейнеризація. Ця технологія дозволяє упаковувати додаток разом з усіма його залежностями та конфігураціями в стандартизований блок, відомий як контейнер(рис 2.7). Кожен контейнер працює ізольовано та має власне середовище, що забезпечує консистентність у роботі додатка незалежно від того, де він запущений – на розробницькому комп'ютері, тестовому сервері або в PROD-середовищі.

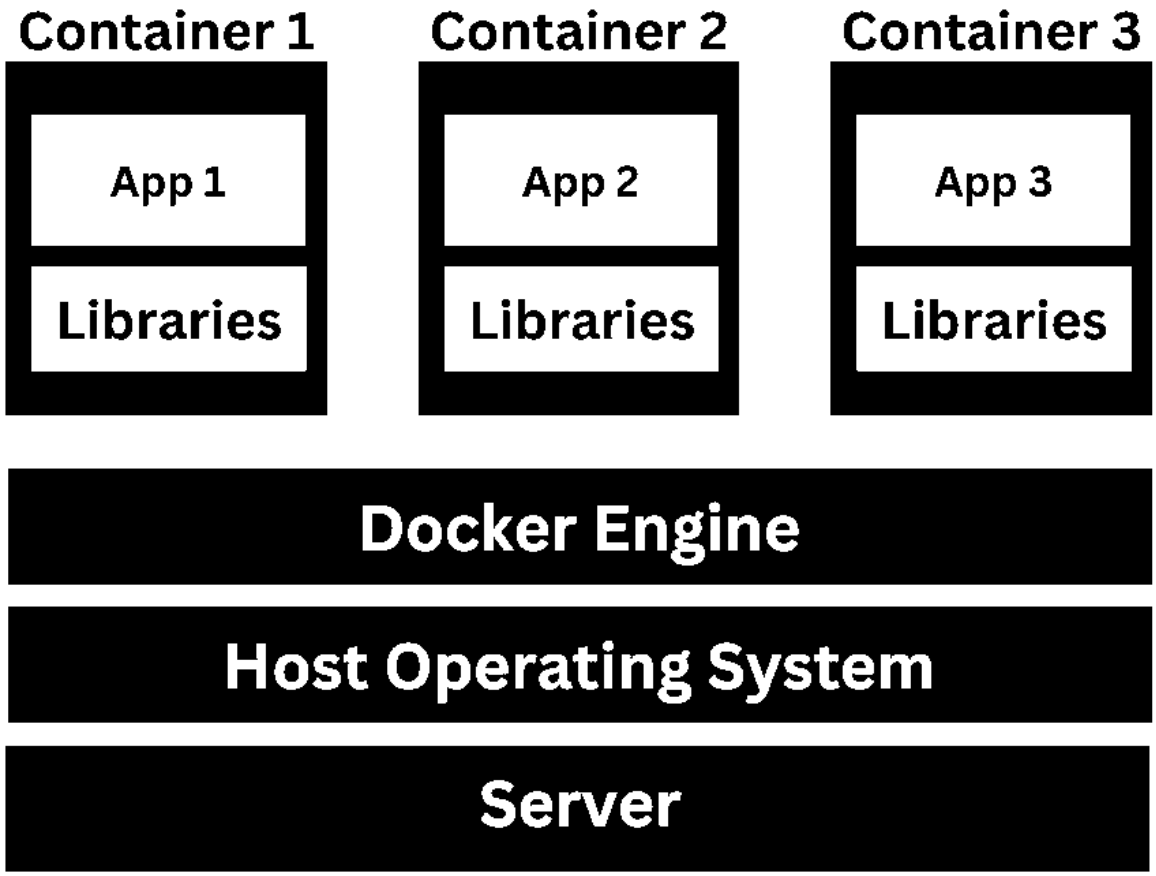

Рисунок 2.7 – Схема роботи контейнеризації в випадку використання Docker

Основними перевагами які контейнеризація надає інформаційній системі побудованій на базі принципів мікросервісної архітектури є:

– консистентність середовищ – контейнери забезпечують однакове середовище від початку, на етапах розробки, до виходу у продакшен, мінімізуючи проблеми, пов'язані з розбіжностями середовищ. Це допомагає уникнути ситуацій, коли додаток працює на одній машині, але не працює на іншій;

– ізоляція та безпека – контейнери ізолюють додатки та їх залежності один від одного, забезпечуючи додатковий рівень безпеки. Якщо один контейнер зазнає критичної помилки або стає недієздатним, інші контейнери продовжують працювати стабільно;

– легкість масштабування та розгортання – контейнери можна швидко розгортати, зупиняти та масштабувати, що робить процес управління додатками гнучким та ефективним. Це особливо корисно в інформаційних система побудованих на принципах мікросервісної архітектури, де може бути потрібно незалежно один від одного масштабувати окремі сервіси;

– ефективне використання ресурсів – контейнери потребують не так багато ресурсів, як традиційні віртуальні машини, оскільки вони ділять ядро операційної системи компьютера, який є хостом і не потребують окремої ОС для кожного контейнера. Це дозволяє ефективніше використовувати апаратні ресурси;

– підтримка практики DevOps – контейнеризація ідеально підходить для використання інструментів безперервної інтеграції та безперервної доставки (Continuous Integration/Continuous Delivery), спрощуючи процеси автоматизації тестування, розгортання та управління релізами.

Найпопулярнішими інструментами для реалізації контейнеризації є Docker та Kubernetes. Docker – це інструмент для контейнеризації, який дозволяє легко створювати, розгортати та управляти контейнерами. Docker використовує Dockerfile для автоматизації процесу створення образів контейнерів, які потім можуть бути запущені на будь-якій системі, що підтримує Docker. Kubernetes – це система оркестрації контейнерів відкритого коду, яка дозволяє автоматизувати розгортання, масштабування та управління контейнеризованими додатками. Kubernetes дозволяє управляти кластерами контейнерів, забезпечуючи високу доступність та масштабованість.

Зазвичай обидва ці інструмента використовують разом. Наприклад, для розгортання та масштабування в продакшені, Docker використовується для створення та упаковки додатків у контейнери, а Kubernetes – для їх розгортання та управління в великому масштабі, особливо в продакшен середовищах. Kubernetes дозволяє

автоматизувати розгортання, масштабування та управління контейнеризованими додатками, забезпечуючи високу доступність та ефективне використання ресурсів.

Реалізація CI/CD (Continuous Integration/Continuous Delivery) є ключовою для успішного впровадження мікросервісної архітектури, оскільки вона дозволяє автоматизувати процеси збірки, тестування та розгортання додатків. Це сприяє швидкому внесенню змін, підвищенню якості продукту та зменшенню часу виведення інформаційного продукту на ринок.

CI/CD є абревіатурою, яка об'єднує дві тісно пов'язані практики в розробці програмного забезпечення: Неперервну Інтеграцію (Continuous Integration, CI) та Неперервну Доставку (Continuous Delivery, CD). Ці практики спрямовані на автоматизацію процесів розробки, тестування та розгортання програмного забезпечення, що дозволяє підвищити швидкість виведення продукту на ринок, покращити його якість та знизити ризики, пов'язані з ручним втручанням.

Неперервна Інтеграція гарантує, що після внесення змін у код розробником, додаток буде автоматично збиратися та будуть виконуватися автоматизовані тести, що дозволить швидко виявити та виправити помилки, що дозволить покращувати якість продукту та скорочувати час на виправлення помилок.

Неперервна Доставка є наступним кроком після «CI» і передбачає автоматизацію процесу доставки зібраного коду до тестового або іншого робочого середовища. Це дозволяє командам мати готовий до розгортання (в продакшн) продукт після успішного проходження всіх тестів.

Найпопулярнішими сервісами для розробки CI/CD додатків є Jenkins, Gitlab CI/CD та CircleCI.

Jenkins – це open-source інструмент автоматизації, який широко використовується для реалізації CI/CD. Він підтримує численні плагіни, які дозволяють цьому сервісу інтегруватися з різними інструментами розробки, тестування та розгортання [26]. Його перевагами вважаються:

– велика кількість плагінів для інтеграції з іншими інструментами та сервісами;

– гнучка система конфігурації завдань;

– підтримка розподіленої збірки та тестування.

З недоліків Jenkins виділяють складність в налаштуванні та не зовсім інтуітивно-зрозмілий та зручний інтерфейс.

GitLab CI/CD – це частина екосистеми GitLab, яка забезпечує інструменти для неперервної інтеграції, доставки та розгортання. Вона тісно інтегрована з GitLab SCM (система управління кодом) та пропонує зручний веб-інтерфейс для налаштування та моніторингу процесів CI/CD [27].

Перевагами Gitlab CI/CD вважаються:

– простота в налаштуванні інструменту, завдяки використанню файлів конфігурації з розширенням «.gitlab-ci.yml»;

– вбудовані функції для моніторингу та візуалізації процесів CI/CD.

Недоліком Gitlab CI/CD вважають обмежену інтеграцію з інструментами та сервісами, що не належать до екосистеми GitLab.

CircleCI – це хмарний сервіс для неперервної інтеграції та доставки, який підтримує автоматизацію збірки, тестування та розгортання додатків. CircleCI пропонує гнучкість у налаштуванні робочих процесів та широкі можливості для інтеграції з іншими інструментами та сервісами.

Для CirleCI визначено такі переваги:

– висока швидкість збірки та тестування завдяки оптимізованій інфраструктурі та кешуванню залежностей;

– підтримка різноманітних мов програмування та фреймворків;

– можливість запуску робочих процесів у контейнерах або на віртуальних машинах.

Обмеження безкоштовного плану є недоліком цього сервісу, розширення можливостей вимагає переходу на платну підписку для великих проектів або команд.

2.3 Аналіз патернів проєктування для реалізації мікросервісів інформаційної системи обліку івент-послуг

Під час розробки інформаційної системи обліку івент-послуг розробник може зіштовхнутися з типовими проблемами, що виникають в подібних системах, для таких проблем іншими розробниками було розроблено стандартизовані рішення – патерни проектування. Схема процеса застосування патернів наведено на рис 2.8. Окрім вирішення типових проблем патерни допомагають оптимізовувати та покращувати ефективність роботи системи.

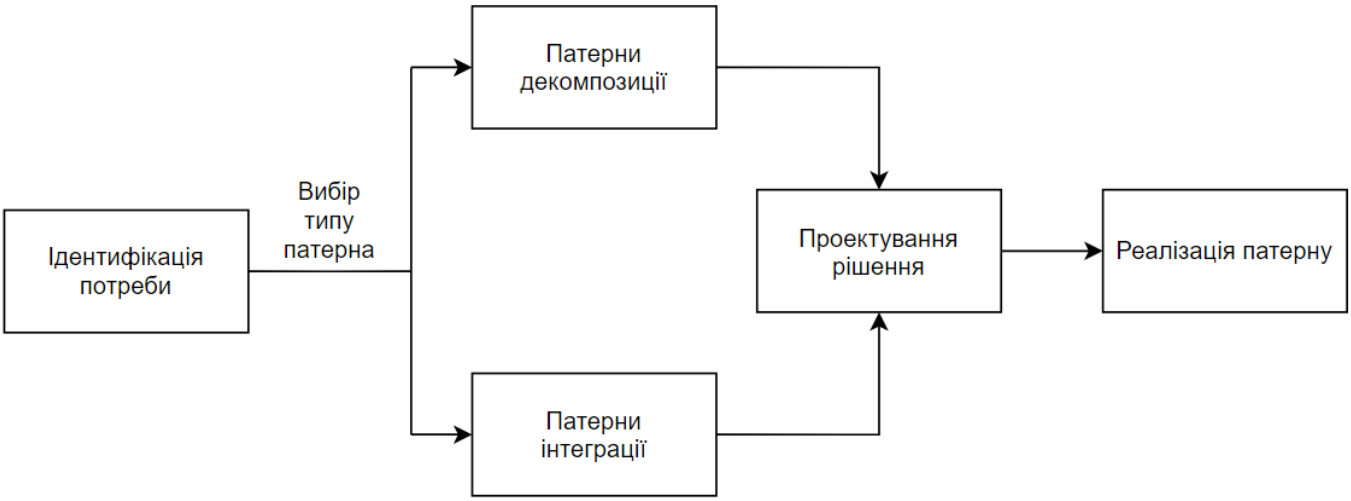

Рисунок 2.8 – Послідовність застосування патернів

Типи патернів проектування, які найбільше використовуються розробниками мікросервісів є: патерни декомпозиції, інтеграції, транзакцій, відмовостійкості та конфігурації.

Прикладами патернів декомпозиції є:

– «декомпозиція за доменними областями (Domain-Driven Design)» – розбиття системи на мікросервіси засноване на доменній моделі бізнесу, де кожен мікросервіс відповідає певній доменній області або бізнес-функції [28];

– «декомпозиція за функціональністю» – розподіл системи на сервіси відповідно до функціональних можливостей або завдань, які вони виконують.

До патернів інтеграції відносять:

– «API шлюз» – використання єдиного вхідного пункту для обробки запитів зовнішніх клієнтів до мікросервісів у системі, що спрощує маршрутизацію, агрегацію відповідей та аутентифікацію;

– «шаблони комунікації між сервісами» – визначення способу взаємодії мікросервісів, наприклад, через синхронні API виклики (REST, gRPC) або асинхронні повідомлення (використання шини подій або брокерів повідомлень).

Патерн проектування мікросервісів для оптимізації процессів, що вимагають вікористання транзакції представлений патерном Saga. Патерн Saga складається з послідовності локальних транзакцій, де кожна транзакція оновлює дані в межах одного мікросервісу. Якщо одна з транзакцій у сазі зазнає збою, сага ініціює серію компенсаційних транзакцій, які мають скасувати вплив попередніх операцій, тим самим повертаючи систему до послідовного стану.

Популярним патерном, що забеспечує відмовостійкість системи є Circuit Breaker.

Circuit Breaker (автоматичний вимикач) – це патерн проектування, який дозволяє системі припинити виконання операції, яка має велику ймовірність збою, тим самим запобігаючи повторним помилкам під час виконання та дозволяючи системі продовжувати роботу з іншими операціями. Цей патерн часто використовується для забезпечення стійкості системи під час взаємодії з зовнішніми сервісами або компонентами, які можуть бути недоступними або перевантаженими.

Працює Circuit Breaker за принципом електричного вимикача, коли кількість помилок під час виконання певної операції перевищує заданий поріг, "вимикач спрацьовує", і система тимчасово припиняє спроби виконання цієї операції, дозволяючи зовнішньому сервісу або компоненту відновити нормальну роботу. Після певного періоду "охолодження" система може спробувати відновити операцію, перевіряючи, чи зовнішній сервіс або компонент знову доступний.

3 ДОСЛІДЖЕННЯ РЕАЛІЗАЦІ ІНФОРМАЦІЙНОЇ СИСТЕМИ ОБЛІКУ ІВЕНТ-ПОСЛУГ НА БАЗІ МІКРОСЕРВІСНОЇ АРХІТЕКТУРИ

3.1 Дослідження визначеного варіанта розробки інформаційної системи на базі мікросервісної архітектури

На рис. 3.1 зображено схему зі структурою інформаційної системи обліку івент-послуг на базі принципів мікросервісної архітектури.

Система складається з наступних компонентів:

а) «Actors Service» – відповідає за управління даними про акторів, аніматорів;

б) «Events Service» – необхідний для обліку списку заходів, які проводить компанія, їх створення, оновлення та видалення;

в) «Frontend» – призначений для надання web-інтерфейсу користувачам та адміністраторам;

г) «Gateway» – виконує роль API Gateway, забезпечуючи єдиний вхідний пункт для всіх запитів до мікросервісів;

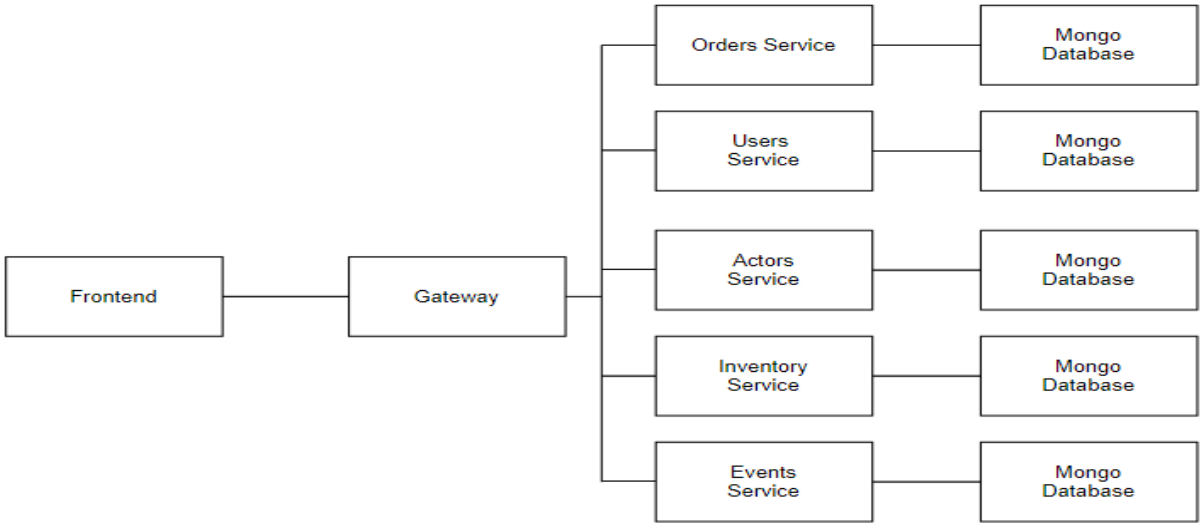

Рисунок 3.1 – Структура інформаційної системи обліку обліку івент-послуг на базі мікросервісної архітектури

д) «Inventory Service» – управління реквезитом та костюмами, необхідним для проведення заходів;

е) «Orders Service» – сервіс обліку замовлень івент-послуг;

є) «Users Service» – сервіс для зберігання інформацію про користувачів системи;

ж) «Service Registry» – сервіс, який забезпечує реєстрацію та виявлення інших мікросервісів для забезпечення динамічного скейлінгу та відмовостійкості.

Сервіс «Gateway» відіграє важливу роль у мікросервісній архітектурі, слугуючи єдиною точкою входу для всіх клієнтських запитів до системи. В інформаційній системі обліку івент-послуг архітектурний паттерн «API Gateway» реалізований за допомогою бібліотеки spring-cloud-starter-gateway.

Основні функції Gateway включають:

а) маршрутизація запитів – «Gateway» отримує всі запити від клієнтів і перенаправляє їх до відповідних мікросервісів на основі конфігурацій маршрутів. Це дозволяє централізувати логіку маршрутизації і спрощує управління трафіком між мікросервісами;

б) безпека – сервіс «Gateway» здійснює автентифікацію та авторизацію запитів, забезпечуючи захист доступу до мікросервісів, включаючи перевірку токенів безпеки, контроль доступу на основі ролей та інших механізмів безпеки;

в) моніторинг і логування – сервіс збирає та зберігає інформацію про всі запити, що проходять через нього, що дозволяє аналізувати продуктивність системи, виявляти потенційні проблеми і проводити аудит, схема принципу роботи логування системи наведено на рис 3.2;

г) «Load Balancing» – «Gateway» також може здійснювати балансування навантаження, розподіляючи запити між кількома екземплярами мікросервісів для забезпечення рівномірного навантаження і підвищення продуктивності системи.

Gateway дозволяє централізовано управляти усіма аспектами взаємодії клієнтів із системою, забезпечуючи гнучкість та масштабованість. Він також спрощує розробку і обслуговування системи, оскільки багато функцій, таких як маршрутизація, безпека і моніторинг, можуть бути реалізовані в одному місці.

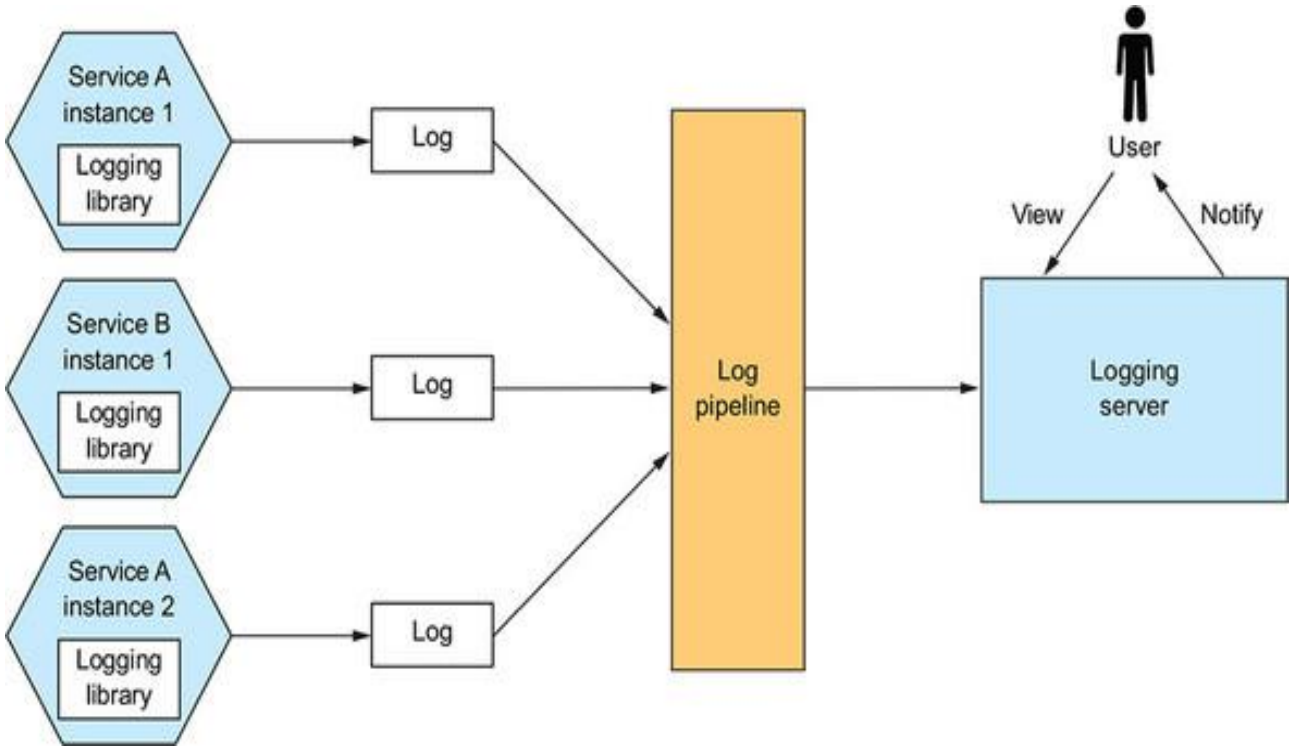

Рисунок 3.2 – Принцип роботи логування в розподіленій інформаційній системі

Принцип роботи «Gateway» сервісу:

а) прийом запитів – клієнт надсилає запит до «Gateway» сервісу замість безпосереднього звернення до мікросервісів;

б) автентифікація та авторизація – сервіс перевіряє автентичність та права доступу клієнта. Якщо запит неавторизований, він відхиляється;

в) маршрутизація – виходячи з налаштувань маршруту, «Gateway» визначає, до якого мікросервісу потрібно перенаправити запит;

г) балансування навантаження – шлюз-сервіс розподіляє запити між кількома інстанціями мікросервісів, забезпечуючи рівномірний розподіл навантаження;

д) агрегація – за необхідності, «Gateway» може об'єднати відповіді від кількох мікросервісів в одну сформовано відповідь;

е) відправка відповіді – «Gateway» пересилає оброблену відповідь клієнту, який робив запит.

«Service Registry» є реалізацією архітектурного патерну «Реєстр сервісів» та важливим компонентом мікросервісної архітектури, який забезпечує реєстрацію і

виявлення мікросервісів. Це дозволяє мікросервісам знаходити та взаємодіяти один з одним динамічно, без необхідності прямого визначення URL-адреси. В інформаційній системі обліку івент-послуг «Service Registry» реалізований за допомогою Eureka від Netflix.

«Service Registry» виконує кілька важливих функцій:

– реєстрація сервісів – кожен мікросервіс під час запуску реєструється у реєстрі, надаючи свою адресу та інші метадані. Це дозволяє реєстру зберігати актуальний список доступних сервісів;

– виявлення сервісів – клієнтські мікросервіси або «Gateway» можуть надсилати запит до реєстру для отримання адреси необхідного сервісу. Це дозволяє динамічно знаходити і підключатися до інших мікросервісів;

– моніторинг доступності – реєстр періодично перевіряє доступність зареєстрованих сервісів через механізм "heartbeat". Якщо сервіс не відповідає протягом певного часу, він видаляється з реєстру;

– підтримка високої продуктивності – реєстр може бути розгорнутий у кількох екземплярах, що забезпечує відмовостійкість і високу доступність.

В інформаційній системі обліку івент-послуг «Service Registry» реалізований за допомогою фреймоворку Eureka, який забезпечує надійну та масштабовану платформу для реєстрації і виявлення сервісів. Eureka надає такі можливості:

а) легкість інтеграції – Eureka легко інтегрується зі Spring Boot додатками через Spring Cloud Netflix, що забезпечує швидке налаштування та розгортання;

б) підтримка клієнтського кешування – клієнти можуть кешувати список доступних сервісів для зменшення навантаження на реєстр та забезпечити роботу навіть у випадку тимчасової недоступності реєстру;

в) висока продуктивність – Eureka може бути розгорнутий у кількох екземплярах з реплікацією даних між ними, що забезпечує відмовостійкість і високу доступність реєстру.

Сервіс «Frontend» – це компонент мікросервісної архітектури, який забезпечує користувацький інтерфейс для взаємодії користувачів із системою. Він відповідає за прийом і обробку запитів від користувачів, а також за відображення даних,

отриманих від інших мікросервісів. В інформаційній системі обліку івент-послуг сервіс «Frontend» реалізований з використанням сучасних веб-технологій, таких як HTML, CSS, Thymeleaf та фреймворкам Spring MVC та Spring Boot Web.

Сервіс «Frontend» виконує такі функції:

а) відображення користувацького інтерфейсу – забезпечує інтуїтивно зрозумілий і зручний інтерфейс для користувачів, дозволяючи їм взаємодіяти з системою через веб-браузер, приклад інтерфейсу зображено на рис. 3.3;

б) прийом запитів від користувачів – обробляє дії користувачів, такі як натискання кнопок, заповнення форм, навігація по сторінках, та надсилає відповідні запити до Gateway для подальшої обробки;

в) обробка відповідей від мікросервісів – отримує відповіді від Gateway, аналізує їх і відображає результати користувачам;

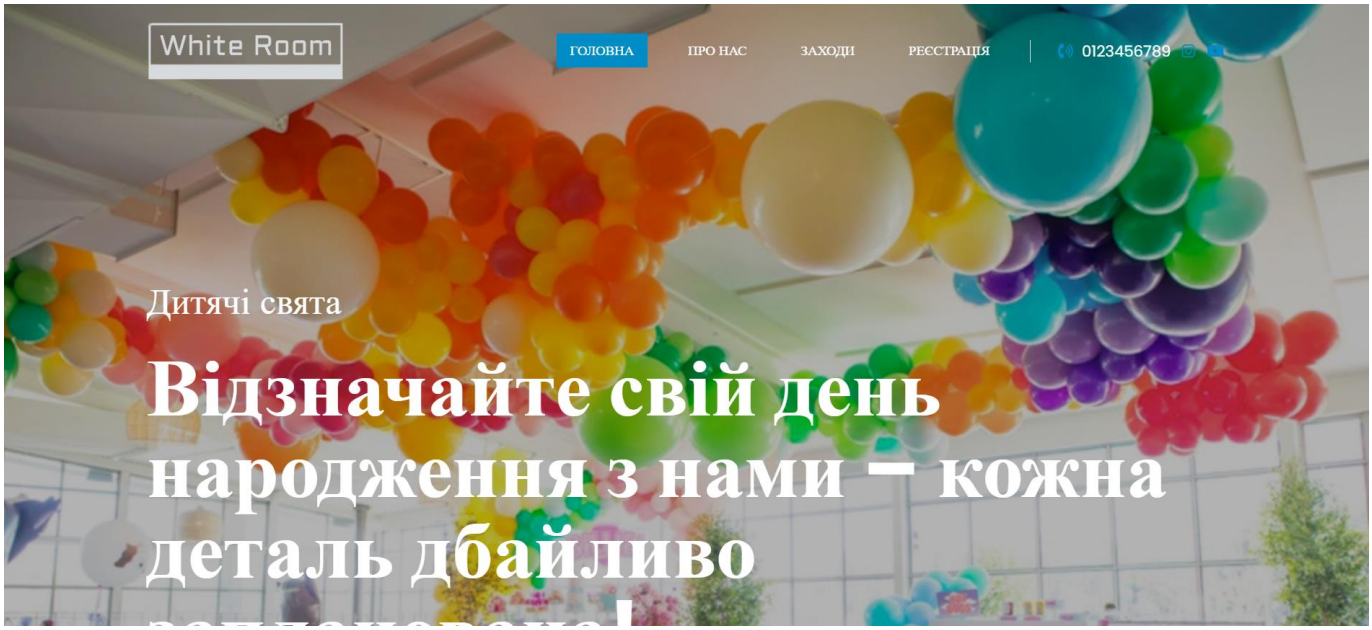

Рисунок 3.3 – Користувальницький інтерфейс сервісу «Frontend»

г) аутентифікація та авторизація – забезпечує перевірку особистості користувачів та надання доступу до ресурсів на основі їх ролей та прав, внаслідок спроби взаємодії з ресурсами які потребують авторизації, користувач буде переправлений на сторінку аутентифікації рис. 3.4;

д) валідація даних – перевіряє коректність введених користувачами даних до їх відправки на сервер, що допомагає знизити кількість помилок і покращити якість даних, приклад валідації введених користувачем даних наведено на рис.3.5;

е) сторінка для менеджерів(рис. 3.6), де можна управляти наявними ресурсами, костюмами та реквізитом, акторами, заходами тощо;

є) персональний кабінет користувача(рис. 3.7), де він може змінити особові дані(рис. 3.8), та передивитись свої замовлення(рис. 3.9).

Для реалізації сервісу «Frontend» в інформаційній системі обліку івент-послуг були використані наступні технології:

– HTML – для структурування веб-сторінок і відображення контенту;

– CSS – для стилізації і оформлення веб-сторінок, забезпечення адаптивного дизайну для різних пристроїв;

– Thymeleaf – для реалізації динамічної відображення інформації на сторінках застосунку(Model and View);

– Spring MVC та Spring Boot Web – для реалізації контроллерів, необхідних для обробки запитів від клієнтів;

– REST API – для взаємодії з «Gateway» і отримання даних від інших мікросервісів.

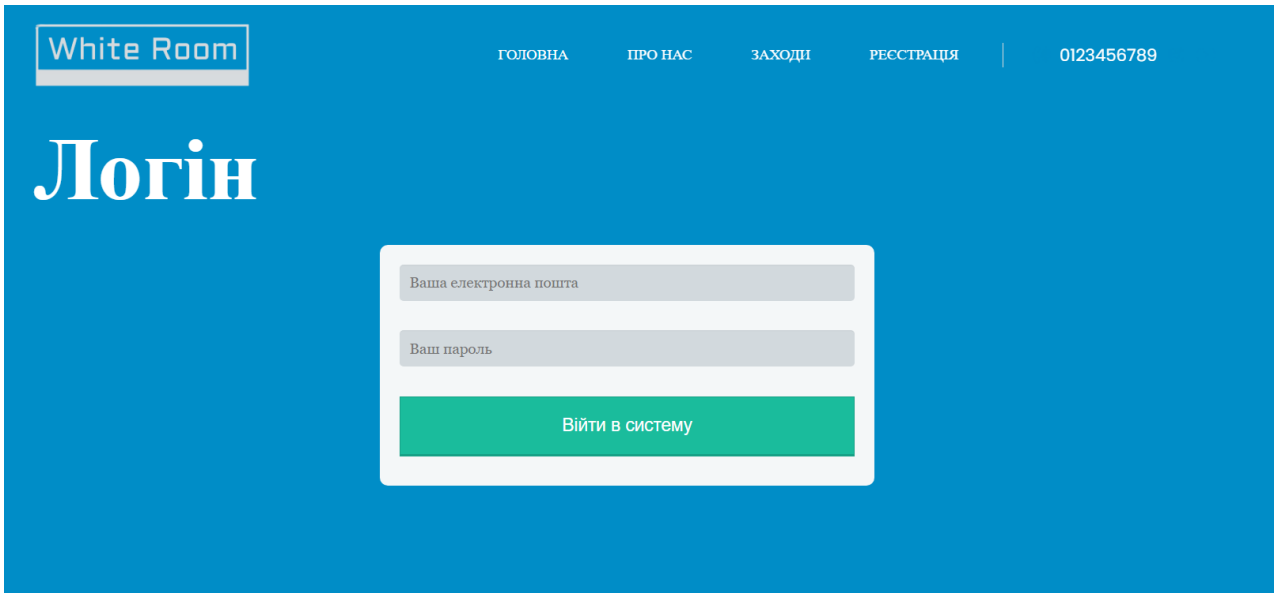

Рисунок 3.4 – Інтерфейс сторінки аутентифікації сервісу «Frontend»

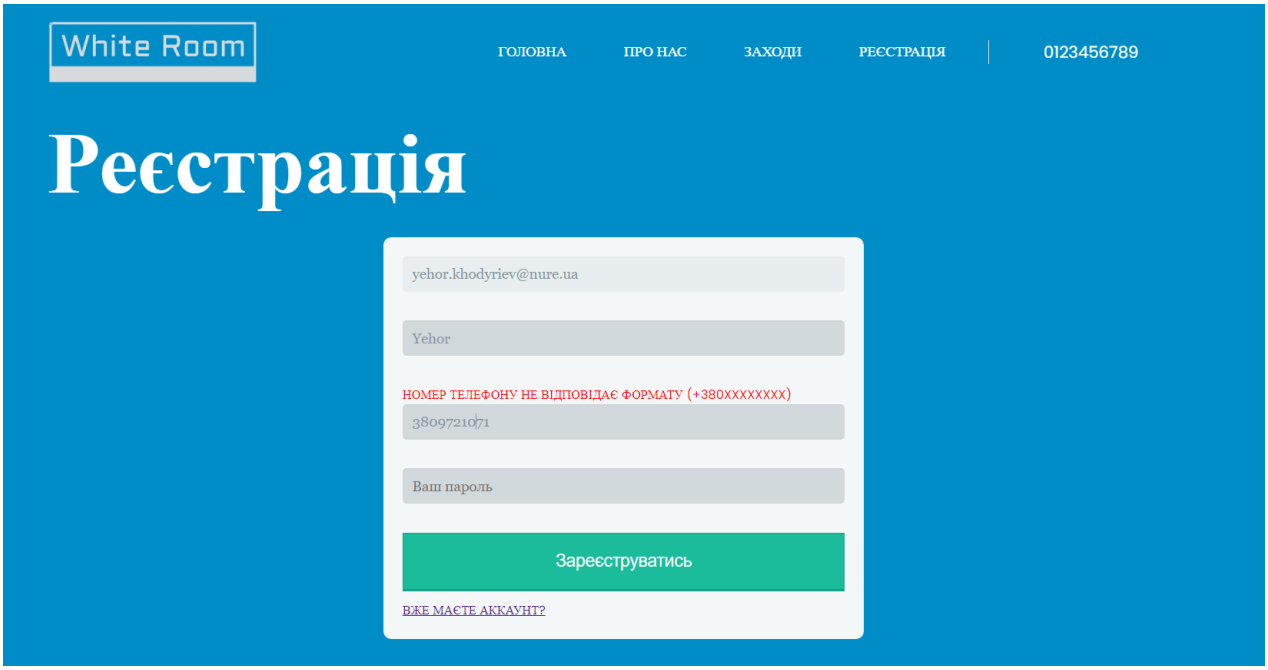

Рисунок 3.5 – Приклад валідації введених даних сервісу «Frontend»

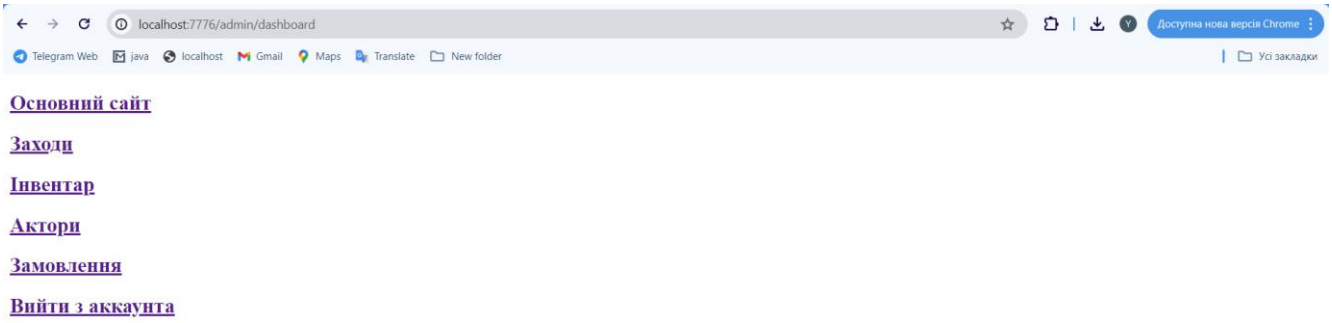

Рисунок 3.6 – Контрольна панель менеджера зі списком посилань на сторінки менеджменту ресурсів

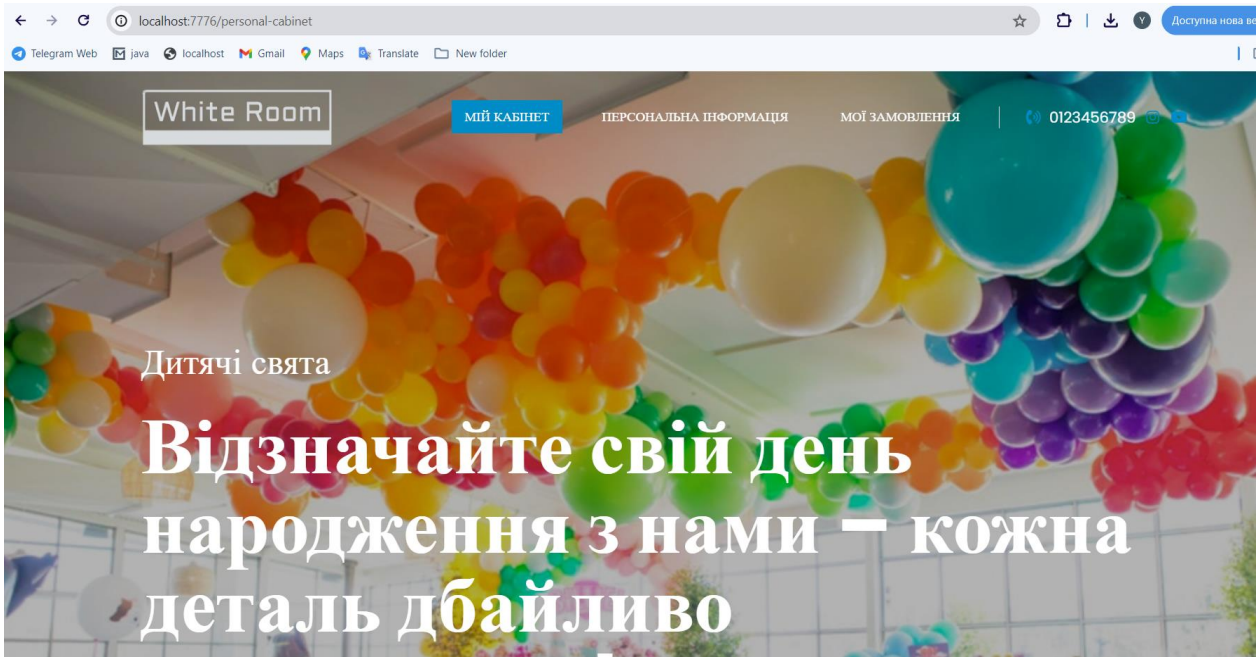

Рисунок 3.7 – Персональний кабінет користувача

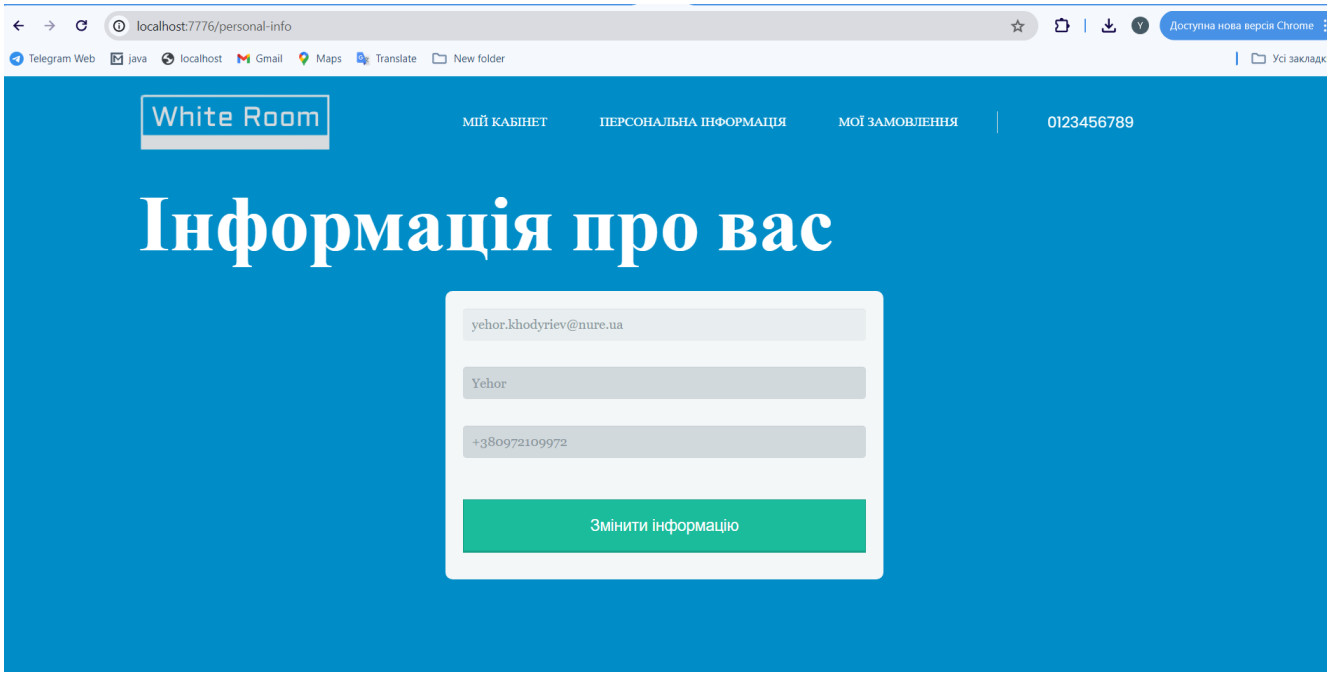

Рисунок 3.8 – Сторінка для зміни даних користувача

Рисунок 3.9 – Сторінка для перегляду замовлень користувача

«Inventory Service» мікросервіс в системі обліку івент-послуг, який відповідає за управління інвентарем, необхідним для проведення святкових заходів. Цей мікросервіс забезпечує ефективне управління ресурсами, а саме костюмами та іншим необхідним реквізитом для проведення заходів.

Основні бізнес-функції які виконує сервіс інвентарю:

а) додавання інвентарю – дозволяє додавати нові елементи костюми або реквізит до системи;

б) оновлення інвентарю – забезпечує можливість оновлення інформації про існуючі елементи інвентарю;

в) видалення інвентарю – дозволяє видаляти костюми або реквізит з системи, якщо вони більше не потрібні або вийшли з ладу;

г) перевірка доступності інвентарю – надає можливість перевірки наявності конкретних елементів інвентарю на певну дату або для конкретного заходу;

д) резервування інвентарю – дозволяє резервувати необхідний інвентар для конкретних святкових заходів, запобігаючи подвійним бронюванням і забезпечуючи доступність ресурсів;

е) облік інвентарю – надає повний список усіх костюмів або усього реквізиту, або тільки інвентар, який відповідає обраним фільтрам.

Сервіс для інвентаризації має свою окрему базу даних Mongo, в якій зберігає інформацію усю інформацію про костюми та реквізит.

В базі даних інформація про костюми представлена схемою «Costumes», яка зберігає такі дані:

а) «id» – унікальний ідентифікатор костюму;

б) «name» – назва костюму;

в) «description» – опис костюму. Це поле містить детальний опис костюму, включаючи його вигляд, стилістику та особливості;

г) «imgUrl» – URL-адреса зображення костюму. Це поле містить посилання на зображення костюму, що дозволяє відображати його в інтерфейсі користувача;

д) «color» – колір костюму. Це поле містить інформацію про основний колір костюму;

е) «condition» – стан костюму. Це поле вказує на поточний стан костюму (наприклад, новий, зношений, потребує ремонту,);

є) «size» – розмір костюму. Це поле містить інформацію про розмір костюму (наприклад, S, M, L, XL);

ж) «categories» – набір категорій, до яких належить костюм. Це поле містить набір об'єктів Category, що дозволяє класифікувати костюми за різними категоріями (наприклад, дитяче свято, корпоратив, шоу).

Інформація про реквізит в базі даних представлена схемою «Items», ака зберігає наступну інформацію про реквізит:

а) «id» – унікальний ідентифікатор реквізиту. Це поле використовується для однозначного визначення кожного реквізиту у базі даних;

б) «name» – назва предмету реквізиту.;

в) «description» – опис реквізиту. Це поле містить детальний опис реквізиту, включаючи його вигляд, функціональність та особливості;

г) «imageUrl» – URL-адреса зображення реквізиту. Це поле містить посилання на зображення реквізиту, що дозволяє відображати його в інтерфейсі користувача;

д) «type» – це поле вказує на тип реквізиту (іграшка, або інструмент, тощо);

е) «condition» – стан реквізиту. Це поле вказує на поточний стан реквізиту (наприклад, новий, використаний, пошкоджений);

є) «material» – матеріал, з якого виготовлений реквізит. Це поле містить інформацію про основний матеріал реквізиту (наприклад, дерево, метал, пластик);

ж) «categories» – набір категорій, до яких належить реквізит. Це поле містить набір об'єктів Category, що дозволяє класифікувати реквізити за різними категоріями.

«Actors Service» – сервіс обліку акторів, відповідає за управління даними про акторів. Цей мікросервіс забезпечує зберігання, оновлення, видалення та перегляд інформації про акторів, а також надає можливість керувати категоріями заходів з якими вони працюють.

Основні бізнес-функції сервісу обліку акторів:

а) додавання акторів – дозволяє додавати нових акторів до системи, включаючи таку інформацію, як ім'я, контактні дані, спеціалізація та інші важливі деталі;

б) оновлення даних акторів – забезпечує можливість оновлення інформації про існуючих акторів, включаючи зміни у контактних даних, спеціалізації, статусі тощо;

в) видалення акторів – дозволяє видаляти акторів з системи, якщо вони звільнились;

г) перегляд акторів – надає можливість користувачам переглядати деталі акторів, включаючи ім'я, контактні дані, спеціалізацію та інші важливі дані;

д) пошук і фільтрація акторів – дозволяє користувачам шукати акторів за різними критеріями, такими як ім'я, спеціалізація, статус тощо;

е) керування категоріями – забезпечує функціонал для призначення акторів на конкретні категорії заходів з якими вони працюють.

Сервіс обліку акторів має свою базу даних Mongo, зі схемою «Actors», яка зберігає наступні дані:

а) «id» – унікальний ідентифікатор актора. Це поле використовується для однозначного визначення кожного актора у базі даних;

б) «name» – ім'я актора. Це поле містить ім'я актора, що дозволяє легко його ідентифікувати;

в) «lastName» – прізвище актора. Це поле містить прізвище актора для повного ідентифікаційного запису;

г) «contactNumber» – контактний номер актора. Це поле містить телефонний номер актора, що дозволяє швидко зв'язатися з ним;

д) «email» – електронна пошта актора. Це поле містить адресу електронної пошти для контактів і повідомлень;

е) «images» – набір зображень актора. Це поле містить набір URL-адрес або шляхів до зображень актора, що дозволяє відображати його фотографії в інтерфейсі користувача;

є) «status» – поточний статус актора. Це поле вказує на поточний статус актора (наприклад, активний, неактивний, зайнятий);

ж) «categories» – набір категорій, до яких належить актор. Це поле містить набір об'єктів Category, що дозволяє класифікувати акторів за різними категоріями заходів, з якими він працює.

«Events Service» – сервіс обліку святкових заходів, забезпечує створення, оновлення, видалення та перегляд інформації про заходи, які проводить фірма «WhiteRoom», а також надає можливість керувати деталями заходу, такими як назва, опис, костюми та реквізит, тощо.

Сервіс обліку святкових заходів має такі бізнес-функції:

а) створення заходів – дозволяє додавати нові заходи до системи;

б) оновлення заходів – забезпечує можливість оновлення інформації про існуючі заходи;

в) видалення заходів – дозволяє видаляти заходи з системи, якщо вони більше не актуальні або були скасовані;

г) перегляд заходів – надає можливість користувачам переглядати деталі заходів, включаючи назву, опис, зображення, тривалість та іншу інформацію;

д) пошук і фільтрація заходів – дозволяє користувачам шукати події за різними критеріями, такими як назва, категорія, костюми, тощо.

«EventsService» має окрему базу даних Mongo, яка має схему Events, в якій зберігається наступна інформація:

а) «id» – унікальний ідентифікатор заходу. Це поле використовується для однозначного визначення кожного заходу у базі даних;

б) «name» – назва заходу. Це поле містить назву заходу, що дозволяє легко його ідентифікувати;

в) «category» – категорія заходу. Це поле містить об'єкт Category, який визначає категорію або тип заходу (наприклад, конференція, дитяче свято, концерт);

г) «imageUrl» – URL-адреса зображення заходу. Це поле містить посилання на зображення заходу, що дозволяє відображати його в інтерфейсі користувача;

д) «shortDescription» – короткий опис заходу. Це поле містить стислий опис заходу, який надає основну інформацію про захід;

е) «description» – детальний опис заходу. Це поле містить розгорнутий опис заходу, включаючи його мету, програму та інші важливі деталі;

є) «durationHours» – тривалість заходу у годинах. Це поле містить інформацію про тривалість заходу, що дозволяє планувати час проведення;

ж) «costumes» – набір костюмів, які використовуються акторами у заході. Це поле містить набір об'єктів Costume, які використовуються у межах заходу;

з) «items» – набір реквізиту, який використовують актори у заході. Це поле містить набір об'єктів Item, які використовуються для організації та проведення заходу.

«Orders Service» - сервіс обліку замовлень, цей мікросервіс забезпечує створення, оновлення, видалення та перегляд замовлень, а також надає можливість керувати деталями замовлень, такими як локація, дата проведення заходу, тощо.

Основними бізнес-функціями сервісу обліку замовлень є:

а) створення замовлень – дозволяє додавати нові замовлення до системи, включаючи такі деталі, як клієнт, захід, дата та інші важливі параметри;

б) оновлення замовлень – забезпечує можливість оновлення інформації про існуючі замовлення, включаючи зміни у датах, клієнтах, статусі тощо;

в) скасування замовлень – дозволяє видаляти замовлення з системи, якщо вони більше не актуальні;

г) перегляд замовлень – надає можливість користувачам переглядати деталі замовлень, включаючи клієнта, захід, дату, статус та іншу інформацію;

д) пошук і фільтрація замовлень – дозволяє користувачам шукати замовлення за різними критеріями, такими як дата, клієнт, статус замовлення тощо;

е) керування статусами замовлень – забезпечує функціонал для оновлення статусу замовлень (наприклад, нове, в обробці, скасоване, завершене).

Для сервиса обліку замовлень було створену окрему базу даних Mongo, яка містить схему «Orders», з наступним набором даних:

а) «id» – унікальний ідентифікатор замовлення. Це поле використовується для однозначного визначення кожного замовлення у базі даних;

б) «actors» – набір акторів, пов'язаних із замовленням. Це поле містить набір об'єктів Actor, які беруть участь у заході, пов'язаному з замовленням;

в) «event» – захід, на який зроблено замовлення. Це поле містить інформацію про захід, включаючи його назву, опис, костюми, тощо;

г) «dateTime» – дата і час замовлення. Це поле містить інформацію про дату і час, коли відбудеться виконання замовлення;

д) «status» – стан замовлення. Це поле вказує на поточний стан замовлення (наприклад, нове, в обробці, скасоване, завершене);

е) «user» – користувач, який здійснив замовлення. Це поле містить інформацію про користувача, включаючи ім'я, контактні дані тощо;

є) «address» – адреса проведення заходу. Це поле містить адресу, пов'язану з замовленням;

ж) «comment» – коментар до замовлення. Це поле містить додаткову інформацію або особливі запити, пов'язані з замовленням;

з) «price» – загальна вартість замовлення. Це поле містить інформацію про загальну суму, яку клієнт повинен сплатити за замовлення.

«Users Service» - сервіс обліку користувачів інформаційної системи обліку івент-послуг, забезпечує створення, оновлення, видалення та перегляд інформації про користувачів, а також надає можливість керувати їхніми ролями, контактною інформацією та іншими параметрами.

Основними бізнес-функціями сервісу обліку користувачів є:

а) створення користувачів – дозволяє додавати нових користувачів до системи, включаючи таку інформацію, як ім'я, контактні дані, роль та інші важливі деталі;

б) оновлення даних користувачів – забезпечує можливість оновлення інформації про існуючих користувачів, включаючи зміни у контактних даних, ролях, статусі тощо;

в) видалення користувачів – дозволяє видаляти користувачів з системи, якщо вони більше не є актуальними або не мають прав доступу;

г) перегляд користувачів – надає можливість адміністраторам переглядати деталі користувачів, включаючи ім'я, контактні дані, роль та інші важливі дані;

д) пошук і фільтрація користувачів – дозволяє адміністраторам шукати користувачів за різними критеріями, такими як ім'я, роль, статус тощо;

е) керування ролями та правами доступу – забезпечує функціонал для призначення ролей і прав доступу користувачам.

Сервіс обліку користувачів «Users Service» використовує окрему базу даних Mongo, яка зберігає схему Users, з таким переліком даних про користувача:

а) «id» – унікальний ідентифікатор користувача. Це поле використовується для однозначного визначення кожного користувача у базі даних;

б) «email» – електронна пошта користувача. Це поле містить адресу електронної пошти для контактів і повідомлень;

в) «password» – пароль користувача. Це поле містить захищений пароль для аутентифікації користувача в системі;

г) «username» – ім'я користувача. Це поле містить ім'я користувача, що дозволяє легко його ідентифікувати;

д) «lastname» – прізвище користувача. Це поле містить прізвище користувача для повного ідентифікаційного запису;

е) «phoneNumber» – контактний номер користувача. Це поле містить телефонний номер користувача, що дозволяє швидко зв'язатися з ним;

є) «role» – роль користувача. Це поле містить інформацію про роль користувача в системі (наприклад, адміністратор, менеджер, користувач).

3.2 Дослідження реалізації взаємодії між мікросервісами інформаційної системи

Інтеграція та комунікація між мікросервісами є фундаментальними аспектами мікросервісної архітектури, забезпечуючи узгоджену та ефективну роботу всіх компонентів системи. Мікросервісна архітектура базується на розподілі функціональності між незалежними сервісами, що дозволяє підвищити гнучкість, масштабованість та відмовостійкість системи. Однак для досягнення цілісності та взаємодії між цими сервісами необхідно використовувати надійні методи інтеграції та комунікації.

Для інформаційної системи обліку івент-послуг було обрано REST-архітектуру для комунікації між сервісами.

REST API дозволяє мікросервісам обмінюватися даними через HTTP-протокол за допомогою стандартних HTTP-методів:

– «GET» – для отримання даних з мікросервісу;
– «POST» – для створення нових ресурсів в мікросервісі;
– «PUT» – для оновлення існуючих ресурсів;
– «DELETE» – для видалення ресурсів.

Основні переваги REST включають простоту, легкість інтеграції та використання стандартних протоколів. Кожен мікросервіс має свій власний REST API, який визначає доступні кінцеві точки (endpoints) для взаємодії з іншими мікросервісами.

Кінцеві точки відповідають різним функціям мікросервісу, таким як створення, оновлення, видалення та отримання даних. Обмін даними між мікросервісами в інформаційній системі обліку івент-послуг відбувається у форматі JSON, це забезпечує незалежність від платформи розробки сервісу та легкість обробки даних.

Визначені методи взаємодії та ендпоінти сервісу менеджменту акторів інформаційної системи обліку івент-послуг:

– «getActors» (HTTP метод GET), з кінцевою точкою «/actors» – повертає список всіх акторів з можливістю фільтрації за допомогою параметрів запиту;

– «getActorById» (HTTP метод GET), з кінцевою точкою «/actors/{id}» – повертає інформацію про конкретного актора за його ідентифікатором;

– «createActor» (HTTP метод POST), з кінцевою точкою «/actors» – створює нового актора. Приймає об'єкт «Actor» у форматі JSON у тілі запиту;

– «updateActor» (HTTP метод PUT), з кінцевою точкою «/actors/{id}» – оновлює інформацію про існуючого актора. Приймає ідентифікатор актора та об'єкт «Actor» у форматі JSON;

– «deleteActor» (HTTP метод DELETE), з кінцевою точкою «/actors/{id}» – видаляє актора за його ідентифікатором.

Визначені кінцеві точки та методи для взаємодії з сервісом обліку костюмів та реквізиту:

– «getCostumes» (HTTP метод GET), з кінцевою точкою «/inventory/costumes» – повертає список всіх костюмів з можливістю фільтрації за допомогою запитів;

– «getCostumeById» (HTTP метод GET), з кінцевою точкою «/inventory/costumes/{id}» – повертає інформацію про конкретний костюм за його ідентифікатором;

– «createCostume» (HTTP метод POST), з кінцевою точкою «/inventory/costumes» – створює новий костюм. Приймає об'єкт «Costume» у форматі JSON у тілі запиту;

– «updateCostume» (HTTP метод PUT), з кінцевою точкою «/inventory/costumes/{id}» – оновлює інформацію про існуючий костюм. Приймає ідентифікатор костюма та об'єкт «Costume» у форматі JSON;

– «deleteCostume» (HTTP метод DELETE), з кінцевою точкою «/inventory/costumes/{id}» – видаляє костюм за його ідентифікатором;

– «getItems» (HTTP метод GET), з кінцевою точкою «/inventory/items» - повертає список всіх реквізитів з можливістю фільтрації за допомогою запитів;

– «getItemById» (HTTP метод GET), з кінцевою точкою «/inventory/items/{id}» – повертає інформацію про конкретний реквізит за його ідентифікатором;

– «createItem» (HTTP метод POST), з кінцевою точкою «/inventory/items» – створює новий реквізит. Приймає об'єкт «Item» у форматі JSON у тілі запиту;

– «updateItem» (HTTP метод PUT), з кінцевою точкою «/inventory/items/{id}» – оновлює інформацію про існуючий реквізит. Приймає ідентифікатор реквізиту та об'єкт «Item» у форматі JSON;

– «deleteItem» (HTTP метод DELETE), з кінцевою точкою «/inventory/items/{id}» – видаляє реквізит за його ідентифікатором.

Для сервісу обліку заходів було визначено наступні методи та кінцеві точки:

– «getEvents» (HTTP метод GET), з кінцевою точкою «/events» – повертає список всіх заходів з можливістю фільтрації за допомогою запитів;

– «getEventById» (HTTP метод GET), з кінцевою точкою «/events/{id}» – повертає інформацію про конкретний захід за його ідентифікатором;

– «createEvent» (HTTP метод POST), з кінцевою точкою «/events» – створює новий захід. Приймає об'єкт «Event» у форматі JSON у тілі запиту;

– «updateEvent» (HTTP метод PUT), з кінцевою точкою «/events/{id}» – оновлює інформацію про існуючий захід. Приймає ідентифікатор заходу та об'єкт «Event» у форматі JSON;

– «deleteEvent» (HTTP метод DELETE), з кінцевою точкою «/events/{id}» – видаляє захід за його ідентифікатором;

– «getCategories» (HTTP метод GET), з кінцевою точкою «/categories» – повертає список всіх категорій;

– «getCategoryById» (HTTP метод GET), з кінцевою точкою «/categories/{id}» – повертає інформацію про конкретну категорію за її ідентифікатором;

– «createCategory» (HTTP метод POST), з кінцевою точкою «/categories» – створює нову категорію. Приймає об'єкт «Category» у форматі JSON у тілі запиту;

– «updateCategory» (HTTP метод PUT), з кінцевою точкою «/categories/{id}» – оновлює інформацію про існуючу категорію. Приймає ідентифікатор категорії та об'єкт «Category» у форматі JSON;

– «deleteCategory» (HTTP метод DELETE), з кінцевою точкою «/categories/{id}» – видаляє категорію за її ідентифікатором.

Визначені методи та кінцеві точки сервісу обліку замовлень:

– «getOrders» (HTTP метод GET), з кінцевою точкою «/orders» – повертає список всіх замовлень з можливістю фільтрації за допомогою запитів;

– «getOrderById» (HTTP метод GET), з кінцевою точкою «/orders/{id}» – повертає інформацію про конкретне замовлення за його ідентифікатором;

– «createOrder» (HTTP метод POST), з кінцевою точкою «/orders» – створює нове замовлення. Приймає об'єкт «Order» у форматі JSON у тілі запиту;

– «updateOrder» (HTTP метод PUT), з кінцевою точкою «/orders/{id}» – оновлює інформацію про існуюче замовлення. Приймає ідентифікатор замовлення та об'єкт «Order» у форматі JSON;

– «deleteOrder» (HTTP метод DELETE), з кінцевою точкою «/orders/{id}» – видаляє замовлення за його ідентифікатором.

Перелік визначених методів та кінцевих точок мікросервісу обліку користувачів інформаційної системи обліку івент-послуг:

– «getUsers» (HTTP метод GET), з кінцевою точкою «/users» – повертає список всіх користувачів з можливістю фільтрації за допомогою запитів;

– «getUserById» (HTTP метод GET), з кінцевою точкою «/users/{id}» – повертає інформацію про конкретного користувача за його ідентифікатором;

– «createUser» (HTTP метод POST), з кінцевою точкою «/users» – створює нового користувача. Приймає об'єкт «User» у форматі JSON у тілі запиту;

– «updateUser» (HTTP метод PUT), з кінцевою точкою «/users/{id}» – оновлює інформацію про існуючого користувача. Приймає ідентифікатор користувача та об'єкт «User» у форматі JSON;

– «deleteUser» (HTTP метод DELETE), з кінцевою точкою «/users/{id}» – видаляє користувача за його ідентифікатором.

Для вирішення проблеми відмов під час комунікації між сервісами через недоступність одного з сервісів, через перенавантаження або збої в роботі, було

реалізовано патерн «Retry». Патерн «Повторні спроби» автоматично повторювати запити до сервісів у разі тимчасових збоїв або помилок, це корисно в ситуаціях, коли збої можуть бути тимчасовими, наприклад, через тимчасові проблеми з мережею, перевантаження сервісу або інші помилки.

Основними елементами патерну «Retry» є:

– умови для повторної спроби – визначення типів помилок або винятків, при яких варто повторювати спробу. Це можуть бути мережеві помилки, таймаути або певні HTTP статуси, наприклад, 5xx;

– максимальна кількість спроб – встановлення обмеження на кількість повторних спроб, щоб уникнути безкінечних циклів;

– інтервал між спробами – встановлення затримки між повторними спробами, яка може бути фіксованою або зростати експоненційно (exponential backoff), що дозволяє зменшити навантаження на систему і підвищити ймовірність успішного виконання операції;

– обробка успішних та невдалих спроб – визначення дій, які слід виконати у разі успіху або остаточної невдачі після всіх спроб.

Приклад реалізації патерну «Retry» для інформаційної системи обліку івент-послуг наведено в лістингу 3.1.

Лістинг 3.1 – Код реалізації патерну «Retry»

```
@Bean
public RetryTemplate retryTemplate() {
       RetryTemplate retryTemplate = new RetryTemplate();
        FixedBackOffPolicy backOffPolicy = new FixedBackOffPolicy();
        backOffPolicy.setBackOffPeriod(2000); // 2 секунди
        retryTemplate.setBackOffPolicy(backOffPolicy);
        SimpleRetryPolicy retryPolicy = new SimpleRetryPolicy();
        retryPolicy.setMaxAttempts(3);
        retryTemplate.setRetryPolicy(retryPolicy);
        return retryTemplate;
    }
 }
}
```

3.3 Дослідження засобів для розгортання та підтримки системи обліку івент-послуг

Розгортання та підтримка мікросервісної архітектури є критичними етапами забезпечення безперебійної роботи інформаційної системи обліку івент-послуг. Розгортання системи включає в себе підготовку середовища, налаштування серверів, мережевої інфраструктури та баз даних, а також забезпечення належної безпеки та доступності ресурсів.

На цьому етапі відбувається винесення кожного мікросервісу в незалежні модулі, що дозволяє забезпечити їхню ізоляцію та легкість управління залежностями. Оркестрація мікросервісів включає автоматизацію процесів розгортання, масштабування та управління, що забезпечує високу доступність та надійність системи. Підтримка системи передбачає впровадження моніторингу та логування для постійного відстеження продуктивності, швидкого виявлення та усунення несправностей, а також забезпечення безперервної інтеграції та доставки (CI/CD), що дозволяє оперативно впроваджувати нові функції та оновлення. Завдяки цим процесам розгортання та підтримка системи забезпечують її стійкість до збоїв, ефективне використання ресурсів та здатність швидко адаптуватися до змінних вимог бізнесу.

Процес розгортання мікросервісів у виробниче середовище починається з написання Dockerfile для кожного сервісу. Dockerfile визначає, як створювати образ контейнера для конкретного сервісу, включаючи всі необхідні залежності, середовища виконання та команди для запуску. Це забезпечує ізоляцію середовища для кожного мікросервісу, що дозволяє легко відтворювати та масштабувати їх, код Dockerfile для сервісу обліку костюмів і реквізиту наведено в лістингу 3.2.

Лістинг 3.2 – Код Dockerfile для «Inventory Service»

```
FROM openjdk:17-jdk
VOLUME /tmp
COPY target/inventory-0.0.1-SNAPSHOT.jar inventory-app.jar
ENTRYPOINT  ["java", "-jar", "inventory-app.jar"]
```

У рядку, де використовується оператор FROM відзначається, що образ OpenJDK 17 буде базовий для цього контейнера. Це забезпечує наявність необхідного середовища для запуску Java-додатків.

За допомогою оператора VOLUME створюється тимчасовий том у контейнері. Тома дозволяють зберігати дані за межами життєвого циклу контейнера, що може бути корисним для зберігання тимчасових файлів додатка.

Оператор COPY використовується для копіювання зібраного JAR-файлу додатка з локальної файлової системи в контейнер. У цьому прикладі ми копіюємо файл inventory-0.0.1-SNAPSHOT.jar з директорії target у файл inventory-app.jar всередині контейнера.

У рядку, де використовується оператор ENTRYPOINT вказується команда, яка буде виконана під час запуску контейнера. У цьому випадку буде запущений JAR-файл додатка за допомогою команди java -jar.

Контейнеризація додатків за допомогою Docker забезпечує кілька ключових переваг:

– ізоляція середовища – кожен контейнер має своє ізольоване середовище виконання, що дозволяє уникнути конфліктів залежностей;

– портативність – контейнери можуть бути запущені на будь-якій машині, що підтримує Docker, незалежно від її конфігурації;

– масштабованість – контейнери легко масштабуються горизонтально, дозволяючи швидко додавати нові інстанції додатків;

– швидкість розгортання – завдяки контейнеризації, процес розгортання додатків стає швидким і простим, оскільки всі залежності включені у контейнер.

Для полегшення роботи з великою кількістю сервісів, було використано Docker Compose. Ця утиліта конфігурується за допомогою файлу docker-compose.yml, який визначає механізми для розгортання всіх мікросервісів системи обліку івент-послуг. Цей файл містить налаштування для сервісів, їх залежності, порти та мережеві параметри. Він дозволяє легко керувати всією інфраструктурою як єдиним цілим [29].

Конфігурація цього файлу включає в себе такі елементи:

а) «version» – вказує версію яка використовується для сумісності з різними функціями та можливостями;

б) «services» – описує всі сервіси, які будуть розгорнуті в рамках цього файлу; Кожен сервіс має такі параметри:

1) «image» – вказує Docker-образ, який буде використовуватися для цього сервісу;

2) «ports» – визначає порти, що будуть відкриті на хості для доступу до сервісу;

3) «depends_on» – вказує на залежності цього сервісу від інших сервісів, забезпечуючи їхній запуск у правильному порядку;

4) «environment» – визначає змінні середовища, необхідні для налаштування сервісу;

5) «networks» – вказує мережу, до якої буде підключений сервіс;

в) «networks» – визначає мережу, яка буде використовуватися всіма сервісами.

Контейнеризація додатків за допомогою Docker Compose забезпечує кілька ключових переваг:

– спрощене розгортання – дозволяє одночасно розгорнути всі мікросервіси, автоматизуючи процес та зменшуючи кількість ручної роботи;

– легкість налаштування – централізоване налаштування залежностей, змінних середовища та мережевих параметрів для всіх сервісів;

– масштабованість – забезпечує легке масштабування сервісів, дозволяючи швидко додавати нові інстанції;

– ізоляція середовища – кожен сервіс працює у своєму ізольованому середовищі, що запобігає конфліктам залежностей та забезпечує стабільність роботи системи. Приклад контейнеризації інформаційної системи обліку івент агентств наведено на рис. 3.10.

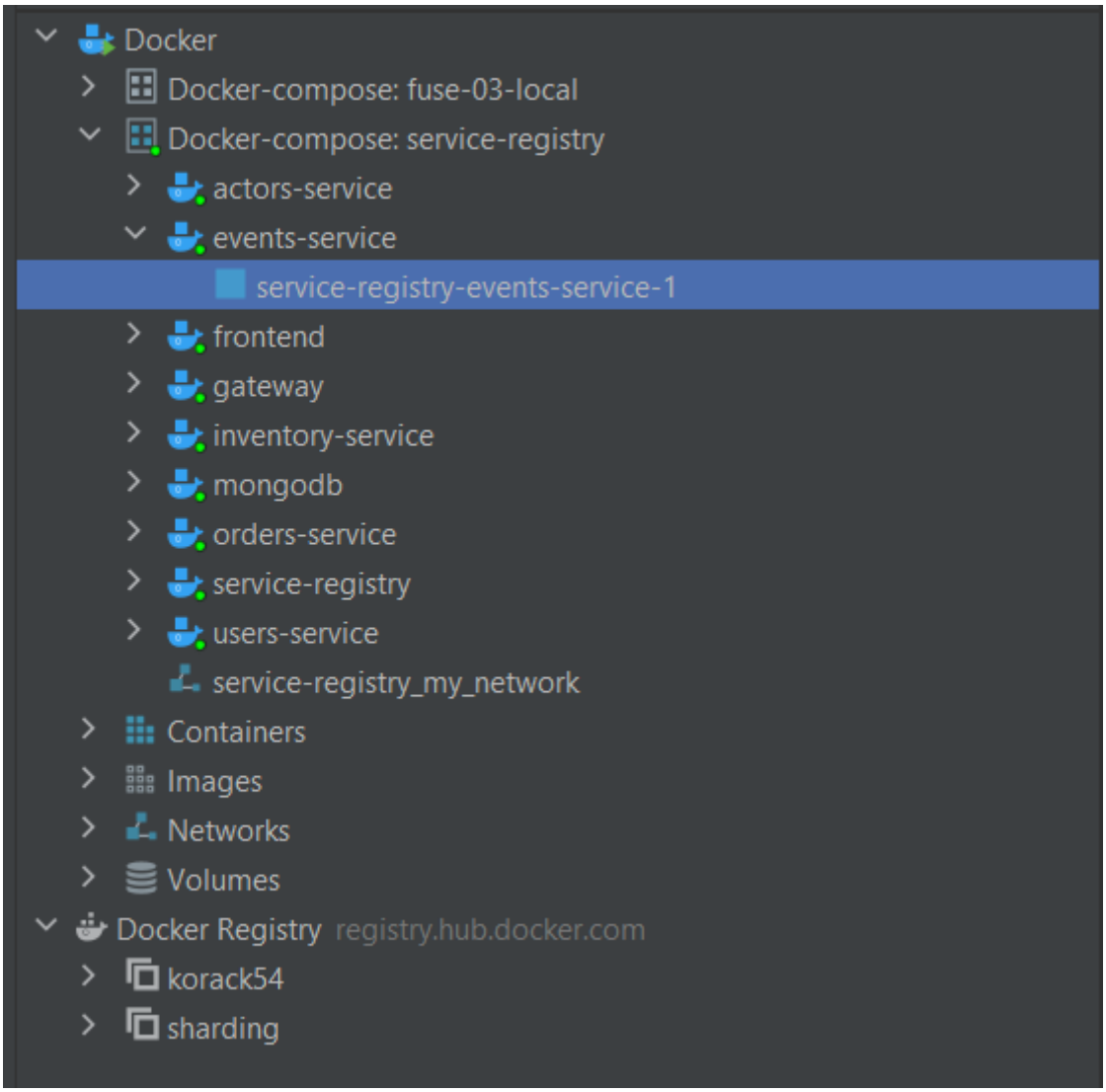

Рисунок 3.10 – Контейнеризована інформаційна система обліку івент послуг

3.4 Дослідження переваг використання мікросервісної архітектури для інформаційної системи обліку івент-послуг

Використання мікросервісної архітектури для побудови інформаційної системи обліку івент-послуг надає ряд переваг, які суттєво вплинули на ефективність, масштабованість та гнучкість системи. Основні переваги включають:

– масштабованість – завдяки використанню мікросервісів можна масштабувати окремі компоненти системи незалежно один від одного. Отже, можна збільшувати ресурси для конкретних сервісів, які потребують більшої потужності, не порушуючи продуктивність інших частин системи;

– гнучкість і модульність – розподілення функціоналу на окремі сервіси сприяє гнучкості у розробці та підтримці. Кожен сервіс може бути розроблений, розгорнутий і оновлений незалежно від інших, що полегшує внесення змін та додавання нових функцій;

– незалежне розгортання – мікросервісна архітектура дозволяє розгортати та оновлювати окремі сервіси без необхідності зупинки всієї системи. Це підвищує доступність системи і зменшує час простою;

– технологічна різноманітність – кожний мікросервіс може бути розроблений з використанням найбільш відповідних до вимог системи технологій і мов програмування, що дозволяє оптимізувати кожну частину системи під конкретні задачі;

– покращена ізоляція помилок – помилка в одному сервісі не обов’язково призводить до відмови всієї системи. Це підвищує стійкість системи до помилок та зменшує ризик загальної відмови;

– покращена продуктивність і ефективність – оптимізація окремих сервісів для конкретних навантажень може підвищити загальну продуктивність системи. Крім того, використання контейнеризації, наприклад Docker, дозволяє ефективніше використовувати ресурси.

ВИСНОВКИ

Під час проведення дослідження реалізації інформаційної системи обліку івент-послуг на базі мікросервісної архітектури було визначено конкретний метод розробки системи та такі варіанти вирішення проблемних питань:

- для побудови сервісів було використано мову програмування Java та фреймворк Spring з модулями Spring Web, для реалізації REST API, та Spring Cloud;
- для зберігання даних для кожного сервісу було визначено нереляційну СУБД Mongo;
- для вирішення проблеми окремого логування кожного сервіса було використано централізовану систему логування за допомогою стека ELK (Elasticsearch, Logstash, Kibana). Кожен мікросервіс був налаштований на відправку логів до централізованої системи за допомогою Logstash. Цей підхід забезпечив централізацію та впорядкування логів від різних мікросервісів, що значно спростило їх аналіз та моніторинг;
- для вирішення проблеми пошуку помилок у системі, де запит може проходити через декілька сервісів, було використано трасування розподілених запитів за допомогою інструмента Jaeger. Кожен запит отримує унікальний ідентифікатор (trace ID), який передається через всі сервіси, що беруть участь в обробці запиту. Це дозволяє відстежувати шлях запиту через усю систему;
- для вирішення проблеми консистентності даних, які зберігаються в окремих базах даних для кожного сервісу, було використано рішення, де кожен сервіс звертається до інших сервісів і оновлює дані безпосередньо. Коли сервіс виконує операцію, яка змінює дані, він надсилає відповідні запити до інших сервісів, щоб вони синхронно оновили свої бази даних. Це забезпечує узгодженість даних у всіх сервісах в режимі реального часу;
- для вирішення проблеми комунікації між сервісами було використано REST API. Кожен сервіс спілкується з іншими сервісами через стандартизовані

HTTP-запити, що дозволяє забезпечити гнучкість і зрозумілість взаємодії між компонентами системи;

- для вирішення проблеми обробки відмов між сервісами було використано механізми fallback та retry. У разі збою запиту сервіс спочатку повторює спробу виконати запит (retry) кілька разів, щоб дати можливість сервісу-відповідачу відновитися. Якщо всі спроби виявляються невдалими, активується механізм fallback, який забезпечує альтернативний шлях виконання операції або повернення заздалегідь визначеного значення, щоб уникнути повної відмови системи. Цей підхід дозволив забезпечити підвищену надійність та стійкість системи до збоїв;
- для вирішення проблеми, коли кожний сервіс може вимагати свій набір інструментів та технологій для розгортання в системі, було використано Docker. Кожен сервіс був контейнеризований за допомогою Docker, що дозволило ізолювати його залежності та забезпечити єдине середовище для розгортання. Це значно спростило процес управління різними технологіями та інструментами, а також забезпечило узгодженість і передбачуваність розгортання сервісів у різних середовищах;
- для вирішення проблеми, коли кожний сервіс потребує окремої аутентифікації та авторизації, використано API Gateway. API Gateway обробляє всі запити від клієнтів і здійснює централізовану аутентифікацію та авторизацію перед тим, як перенаправити запити до відповідних сервісів. Це забезпечує єдину точку входу для контролю доступу та підвищує безпеку системи, зменшуючи навантаження на окремі сервіси.

Реалізовані компоненти інформаційної системи обліку івент-послуг на базі мікросервісів готові до експлуатації та подальшого удосконалення.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Ходирєв Є.О. Дослідження методів реалізації мікросервісної архітектури для розподілу функцій проекту системи обліку івент-послуг. // Методи і засоби прийняття рішень у соціально-економічних і технічних системах. Матеріали доповідей учасників 28-го Міжнародного молодіжного форуму «Радіоелектроніка та молодь у XXI столітті». Том 6: секція 4: М-во освіти і науки України, Харків. нац. ун-т радіоелектроніки. – Харків : ХНУРЕ, 2024. – 958 с.

2. Ходирєв Є.О. Розробка компонентів інформаційної системи обліку замовлень святкових заходів : Бакалаврська робота / Харківський національний університет радіоелектроніки – 2022 – 68 с.

3. Brynjolfsson E., Mcafee A. Second machine age: work, progress, and prosperity in a time of brilliant technologies. Norton & Company Limited, W. W., 2016. 336 p.

4. Events and sustainability / M. Hughes et al. Taylor & Francis Group, 2015. 206 p.

5. Laudon J. P., Laudon K. C. Management information systems: managing the digital firm. Pearson Higher Education & Professional Group, 2017. 672 p.

6. Head First design patterns / ред.: F. E. 1965- та ін. Sebastopol, CA : O'Reilly, 2004. 638 p.

7. Chopra S. Supply chain management: strategy, planning, and operation. 4-те вид. Boston : Prentice Hall, 2010.

8. Fowler M. Refactoring: improving the design of existing code. Pearson Education, Limited, 2018.

9. Kim G. The DevOps handbook: how to create world-class agility, reliability, and security in technology organizations. 2016. 437 p.

10. Newman S. Monolith to microservices: sustaining productivity while detangling the system. O'Reilly Media, Incorporated, 2019. 150 p.

11. Site reliability engineering: how google runs production systems / N. R. Murphy та ін. O'Reilly Media, Inc., 2016. 524 p.

12. Burns B., Hightower K., Beda J. Kubernetes - up and running: dive into the future of infrastructure. O'Reilly Media, Incorporated, 2019. 261 p.

13. Richardson C. Microservices patterns: with examples in java. Manning Publications, 2018. 522 p.

14. Bass L., Weber I., Zhu L. DevOps: a software architect's perspective. Addison-Wesley Longman, Incorporated, 2015. 352 p.

15. Kleppmann M. Designing data-intensive applications: the big ideas behind reliable, scalable, and maintainable systems. O'Reilly Media, 2017. 624 p.

16. Balalaie A., Heydarnoori A., Jamshidi P. Microservices architecture enables devops: migration to a cloud-native architecture. IEEE software. 2016. Vol. 33, no. 3. P. 42–52: веб-сайт. URL: https://doi.org/10.1109/ms.2016.64 (дата звернення: 26.05.2024).

17. Masse M. REST API design rulebook. O'Reilly Media, Incorporated, 2011. 116 p.

18. Kousen K. Modern java recipes: simple solutions to difficult problems in java 8 and 9. O'Reilly Media, 2017. 322 p.

19. Wilson J. Node.js 8 the right way: practical, server-side javascript that scales. Pragmatic Bookshelf, 2018. 336 p.

20. Doglio F. Pro REST API development with node.js. Apress, 2015. 196 p.

21. Sweigart A. Automate the boring stuff with python: practical programming for total beginners. 2nd ed. No Starch Press, 2019. 592 p.

22. Fault tolerant distributed python software transactional memory / M. Popovic et al. Advances in electrical and computer engineering. 2020. Vol. 20, no. 4. P. 19–28: веб-сайт. URL: https://doi.org/10.4316/aece.2020.04003 (дата звернення: 26.05.2024).

23. Cattell R. Scalable SQL and NoSQL data stores. ACM SIGMOD Record. 2011. Vol. 39, no. 4. P. 12–27. URL: https://doi.org/10.1145/1978915.1978919 (дата звернення: 26.05.2024).

24. Pritchett D. BASE: an acid alternative. Queue. 2008. Vol. 6, no. 3. P. 48–55: веб-сайт. URL: https://doi.org/10.1145/1394127.1394128 (дата звернення: 26.05.2024).

25. Nisa B. U. A comparison between relational databases and nosql databases. International journal of trend in scientific research and development. 2018. Volume-2,

Issue-3. P. 845–848: веб-сайт. URL: https://doi.org/10.31142/ijtsrd11214 (дата звернення: 26.05.2024).

26. Smart J. F. Jenkins: continuous integration for the masses. O'Reilly Media, Incorporated, 2011. 404 p.

27. O'Grady A. GitLab Quick Start Guide: Migrate to GitLab for all your repository management solutions. Packt Publishing, 2018. 180 p.

28. Vernon V. Domain-Driven design distilled. Addison-Wesley Professional, 2016. 176 p.

29. McKendrick R. Mastering docker - fourth edition: enhance your containerization and devops skills to deliver production-ready applications. Packt Publishing, Limited, 2020.