Міністерство освіти і науки України

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «КИЄВО-МОГИЛЯНСЬКА АКАДЕМІЯ»

Кафедра мультимедійних систем факультету інформатики

Розробка веб-застосунку з використанням мікросервісної архітектури та Docker Compose

Текстова частина до курсової роботи
за спеціальністю „Інженерія програмного забезпечення”

Керівник курсової роботи
доц. Франчук О. В.

_____
(підпис)
“____” _____ 2022 р.

Виконав студент Осадчук В. І.
“____” _____ 2022 р.

Київ 2022

Зміст

Міністерство освіти і науки України

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «КИЄВО-МОГИЛЯНСЬКА АКАДЕМІЯ»

Кафедра мультимедійних систем факультету інформатики

ЗАТВЕРДЖУЮ

Зав. кафедри мережних технологій,

д.ф.-м.н., професор

_____ Г.І.Малашонок

(підпис)

“____”_____ 20_ р.

ІНДИВІДУАЛЬНЕ ЗАВДАННЯ

на курсову роботу

студенту Осадчуку Володимиру

5-го курсу факультету інформатики

ТЕМА: Розробка веб-застосунку з використанням мікросервісної архітектури та Docker Compose

Вихідні дані:

- Веб-застосунок для оголошень з прокату паперових книг, який написаний за домомогою мікросервісної архітектури та запускається в середовищі Docker.

Зміст ТЧ до курсової роботи:

1. Вступ
2. Анотація
3. Мікросервісна архітектура
4. Docker
5. Архітектура застосунку
6. Розширення застосунку
7. Порівняння з монолітною архітектурою
8. Висновки
9. Скріншоти застосунку

Дата видачі “___” _____ 202_ р.

Керівник _____ Завдання отримано _____

Календарний план виконання курсової роботи

Тема: Розробка веб-застосунку з використанням мікросервісної архітектури та Docker Compose

№ п/п	Назва етапу курсового проекту (роботи)	Термін виконання етапу	Примітка
1.	Отримання завдання на курсову роботу	листопад 2021 р.	
2.	Огляд літератури за темою роботи	листопад - грудень 2021 р.	
3.	Створення практичної частини роботи	січень - березень 2022 р.	
5.	Написання пояснювальної роботи	січень-березень 2022 р.	
6.	Створення слайдів для доповіді та написання доповіді	лютий-березень 2022 р.	
7.	Надання роботи керівнику для перевірки	квітень 2022 р.	
8.	Корегування роботи за результатами перевірки керівником	квітень 2022 р.	
9.	Остаточне оформлення пояснювальної роботи та слайдів	квітень - травень 2022 р.	
10.	Захист курсової роботи	червень 2022 р.	

Студент Осадчук В.І. _____

Керівник Франчук О. В. _____

“_____” _____ _____ р.

Вступ

Відносно недавно з’явився новий архітектурний стиль – “Мікросервісна архітектура”. Це підхід, коли додаток будується як сукупність невеликих, незалежних, самодостатніх сервісів, які спілкуються один з одним за допомогою певних механізмів.

Мікросервіси допомагають розробникам доставляти зміни швидше, безпечніше і з більш високою якістю, тобто зберігати швидкість розвитку продукту, навіть коли він досягає великих розмірів.

Мета даної роботи – створити веб-застосунок для оголошень з прокату паперових книг, написаний за домомогою мікросервісної архітектури, запропонувати варіанти його розширення та порівняти такий підхід з монолітною архітектурою.

Анотація

Роботу присвячено мікросервісній архітектурі. Було проведено роботу з сервісами в C#.NET, фронтенд фреймоворком Angular, базою даних MS SQL Server, Docker-ом та іншими бібліотеками в середовищі .NET. На основі створеного веб-застосунку, були запропоновані варіанти його розширення та проведено порівняння такого архітектурного підходу з монолітом.

Мікросервісна архітектура

Означення

Термін “Мікросервісна архітектура”, ще відомий як мікросервіси, з’явився відносно нещодавно. Цей стиль розробки отримав розповсюдження в зв’язку з розвитком DevOps та практик гнучкої розробки. На сьогоднішній день мікросервісній архітектурі приділяється багато уваги: статті, блоги, дискусії в соц. мережах та презентації на конференціях. Даний спосіб розробки має значні переваги, коли йдеться про забезпечення гнучкої розробки і доставки складних корпоративних додатків.

Архітектурний стиль мікросервісів — це підхід, коли єдиний додаток будується як сукупність невеликих, незалежних, самодостатніх сервісів, які спілкуються один з одним за допомогою таких механізмів, як HTTP/HTTPS, gRPC, AMQP. Ці сервіси побудовані навколо бізнес-потреб та розгортаються незалежно з використанням повністю автоматизованого середовища. Кожен сервіс відповідальний за конкретний бізнес-процес. Існує абсолютний мінімум централізованого управління цими сервісами. Самі по собі сервіси можуть бути написані на різних мовах і використовувати різні технології зберігання даних.

Типи комунікації між сервісами

Розрізняють три типи комунікацій в мікросервісній архітектурі:

- Синхронний – спілкування відбувається в режимі реального часу
- Асинхронний – спілкування відбувається незалежно від часу
- Гібридний – підтримує обидва типи

Синхронний

Це такий тип, при якому сервіс відправляє запит до іншого сервісу та очікує відповідь, при чому сам він блокується. Абонент може отримати відповідь за мілісекунду, а може і за кілька секунд. Незалежно від затримки програми, абонент не може перейти до наступного завдання, поки не буде отримана відповідь. Типовим прикладом синхронного обміну даними є взаємодія HTTP/HTTPS та gRPC.

Асинхронний

Це тип, у якому запит до сервісу та подальша відповідь відбуваються незалежно один від одного. Зазвичай використовують технології брокера повідомлень, такі як Kafka або RabbitMQ, щоб діяти як посередник між службами. Один сервіс публікує повідомлення, а інший – слухає. Завдяки цьому, сервіс, що публікує, не блокується, тому що не чекає на відповідь.

Гібридний

Це той, що підтримує як синхронні, так і асинхронні взаємодії. Наприклад сервіс підтримує протоколи HTTP та обмін повідомленнями.

Саме гібридний тип буде використаний у даній роботі.

Клієнт-серверна взаємодія

Пряма взаємодія

Використовується, коли різні частини сторінки клієнта надсилають запити на різні мікросервіси.

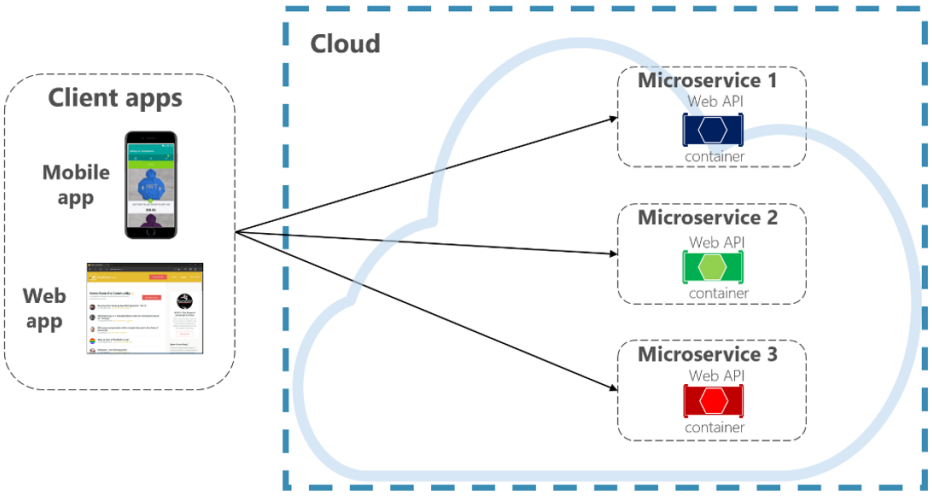


Рисунок 1 Пряма клієнт-серверна взаємодія

Слід використовувати, коли додаток не великий або дані є незалежними і не потребують додаткової агрегації.

API Gateway

Надає кінцеву точку з одним входом для групи мікросервісів, як шаблон патерну Фасад.

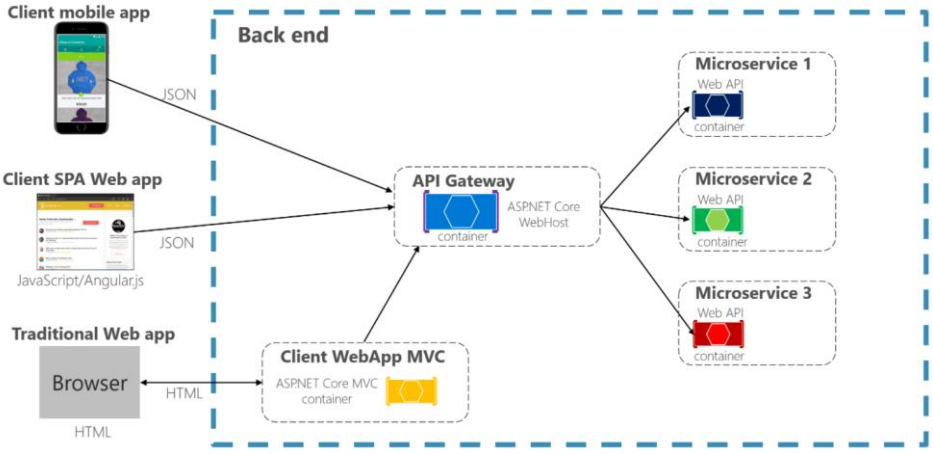


Рисунок 2 API Gateway

Він діє, як зворотний проксі-сервер і посередник між клієнтами та мікросервісами.

Може також забезпечити аутентифікацію, кеш та інші наскрізні проблеми.

Взаємодія в режимі реального часу

Спілкування в режимі реального часу можна досягти за допомогою веб-сокетів HTTP.

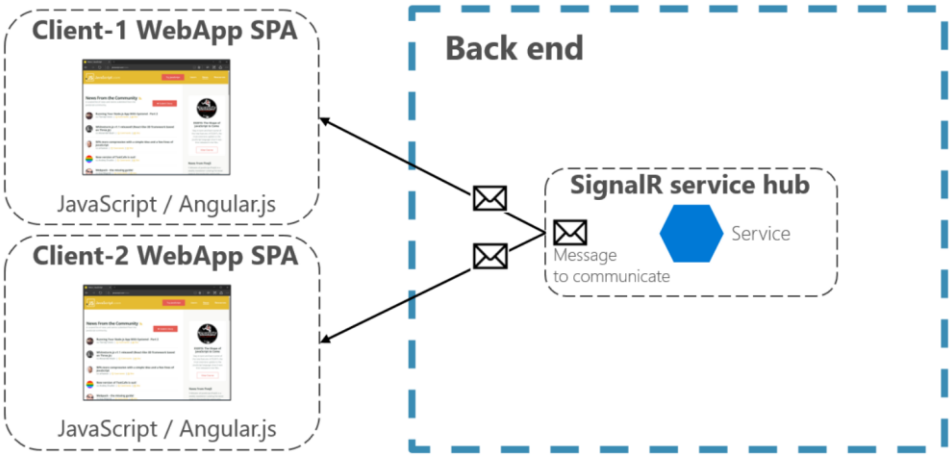


Рисунок 3 Взаємодія в режимі реального часу

Використовується, коли потрібно передати дані клієнтам безпосередньо. ASP.NET Core має бібліотеку SignalR, як технологію для зв’язку в режимі реального часу.

В даній роботі використані пряма взаємодія та взаємодія в режимі реального часу.

Узгодженість даних

Узгодженість даних – це найтяжча частина ахітектури мікросервісів, тому що в монолітних застосунках узгодженість забезпечує загальна реляційна база даних. В мікросервісах кожен сервіс має своє сховище даних.

Для вирішення даної проблеми існує два стандартних підходи.

Розподілена транзакція (Distributed transaction)

Це набір операцій над даними, які виконуються в двох або більше сховищах даних (базах даних). Зазвичай він координується між окремими вузлами, з’єднаними мережею, але також може охоплювати кілька баз даних на одному сервері.

Кінцева узгодженість (Eventual consistency)

Це гарантія того, що коли оновлення здійснюється в розподіленій базі даних, це оновлення в кінцевому підсумку буде відображено на всіх вузлах, які зберігають дані, що призведе до однакової відповіді щоразу, коли буде запит на отримання даних. Узгодженість зазвичай виконується асинхронно, тому час, необхідний, щоб отримати узгодженість, може бути невизначений.

Рекомендується уникати використання Розподіленою транзакції, де це можливо, тому в даній роботі узгодженість буде досягнута шляхом Кінцевої узгодженості.

Docker

Означення

Docker – це програмна платформа для швидкої розробки, тестування та розгортання програм. Docker упаковує програмне забезпечення в стандартизовані блоки, які називаються контейнерами. Кожен контейнер включає все необхідне роботи програми: бібліотеки, системні інструменти, код та середовище виконання. Завдяки Docker можна швидко розгортати та масштабувати програми в будь-якому середовищі.

В основі роботи Docker є стандартизований спосіб виконання коду. Docker – це операційна система контейнерів. Контейнери створюють віртуальне уявлення серверної операційної системи, це подібно до того, як віртуальна машина створює віртуальне уявлення апаратного забезпечення сервера. Після встановлення на сервер Docker надає доступ до простих команд, необхідних для будови, запуску або зупинки контейнерів.

Контейнери Docker можна використовувати як основні компоненти для створення сучасних платформ та програм. Docker спрощує складання та запуск розподілених мікросервісних архітектур, розгортання коду за допомогою стандартизованих конвеєрів безперервної інтеграції та доставки, створення високомасштабованих систем обробки даних та повністю керованих платформ для розробників.

Docker Compose

Docker Compose – це засіб для запуску одного або кількох контейнерів як однієї служби. Кожен із контейнерів в ньому працює ізольовано, але при необхідності може взаємодіяти один з одним. За допомогою мови YAML можна легко написати файл docker-compose.yml, який містить команди для запуску контейнерів. Ще одна чудова особливість Docker Compose полягає в тому, що користувачі можуть активувати всі служби (контейнери) за допомогою однієї команди.

Переваги Docker Compose

- Розгортання на одному хості – це означає, що можна запускати все на одному апаратному забезпеченні
- Швидка та проста конфігурація - завдяки скриптам YAML
- Висока продуктивність – Docker Compose скорочує час, необхідний для виконання завдань
- Безпека – усі контейнери ізольовані один від одного, що зменшує загрозу

Існує певний ряд команд для роботи з Docker Compose. Наприклад: запуск всіх сервісів, зупинка всіх сервісів, перевірка версії, встановлення за допомогою pip.

Архітектура застосунку

Загальні відомості

Бекенд-частина застосунок написана на .NET, фронтенд – на Angular та ASP.NET Core MVC, а в якості сховища даних взято базу даних MS SQL Server. Для комунікації між сервісами використовується синхронна (HTTP), асинхронна (MassTransit + RabbitMQ) та взаємодія в режимі реального часу (SignalR). Для перевірки стану сервісів використовується бібліотека AspNetCore.HealthChecks. Збірка та запуск всіх сервісів виконується в Docker за допомогою Docker-Compose. Також використано інші .NET-бібліотеки, такі як AutoMapper, Hangfire та Refit.

Застосунок складається з дев’яти мікросервісів, сім з яких написані власноруч:

- Identity
- Publishers
- Statistics
- Notifications
- WebApp
- Admin
- Watchdog

та двоє стандартних:

- SqlServer
- RabbitMQ

Також створено бібліотеку класів Common, яку використовують більшість з написаних сервісів.

Діаграма застосунку

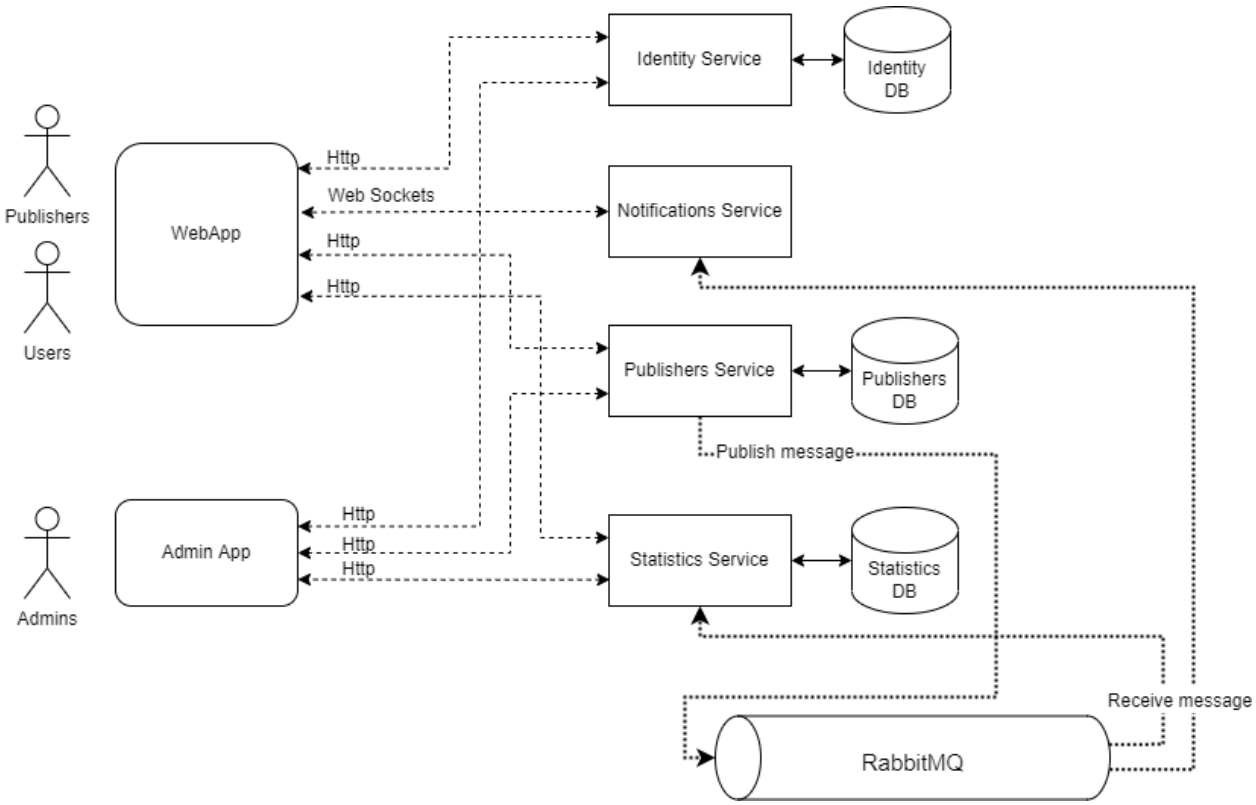


Рисунок 4 Діаграма застосунку

Мікросервіси застосунку

Identity сервіс

Сервіс призначений для реєстрації та аутентифікації користувачів, як звичайних, так і адмінів. Він використовується у таких сервісах, як WebApp та Admin.

Аутентифікація відбувається за допомогою JWT токену. Сервіс використовує бібліотеку Microsoft.AspNetCore.Identity, дані зберігаються в MS SQL Server.

Інтерфейс сервісу:

- POST */identity/register/*
  Реєстрація нового користувача
- POST /identity/login/
  Отримання токену за логіном-паролем
- PUT /identity/changepassword
  Зміна свого паролю

Publishers сервіс

Основний сервіс, який відповідає за створення, редагування, перегляд видалення публікацій, а також публікатори можуть перегляди свої профілі та редагувати їх. Даний сервіс використовується у WebApp та Admin.

Ще однією задачою сервісу є сповіщення про створення нової публікації. Для цього за допомогою RabbitMQ та бібліотеки MassTransit він відправляє повідомлення у чергу. Сервіс використовує базу даних MS SQL Server для збереження публікацій, публікаторів та відправлених повідомлень.

Також за допомогою бібліотеки Hangfire сервіс запускає Cron-джоб, який з певною періодичністю намагається запушити повідомлення, які по певним причинам не були доставлені. Дана бібліотека також використовує MS SQL Server для збереження проміжних результатів джоба.

Інтерфейс сервісу:

- GET */bookads/[query]*
  Отримати активні публікації за фільтрами
- GET */bookads/{id}*
  Отримати публікацію по Id
- POST */bookads*
  Створити нову публікацію
- PUT */bookads/{id}*
  Відредагувати публікацію
- DELETE */bookads/{id}*
  Видалити публікацію
- GET */bookads/mine*
  Отримати свої публікації (активні та не активні)
- GET */bookads/categories*
  Отримати доступні категорії для створення публікації

- PUT */bookads/{id}/changeavailability*

  Змінити статус своєї публікації
- GET */publishers/{id}*

  Отримати інформацію про публікатора
- GET */publishers/id*

  Отримати свій Id публікатора
- POST */publishers*

  Створити публікатора (викликається при реєстрації користувача)
- PUT */publishers/{id}*

  Відредагувати свої дані публікатора
- GET */publishers* – Admin only

  Отримати список всіх публікаторів (опція лише для адміністраторів)

Statistics сервіс

Сервіс, який слідкує за створеними публікаціями. Для наглядності, статистику видно і в користувацькому застосунку (WebApp) та адміністраторам (Admin).

Сервіс отримує інформацію про новостворену публікацію з черги RabbitMQ за допомогою бібліотеки MassTransit. Статистика та отримані повідомлення зберігаються в базі MS SQL Server.

Інтерфейс сервісу:

- GET */statistics*

  Отримати статистику по створеним публікаціям

Notifications сервіс

Сервіс, який відповідає за сповіщення про створені публікації. Наразі сервіс комунікує з WebApp застосунком, використовуючи комунікацію в режимі реального часу (WebSockets + бібліотека SignalR). В подальшому можна розширити, відправляючи сповіщення на електронну пошту, або push-повідомленнями на телефон.

Як і Statistics, даний сервіс отримує інформацію про новостворену публікацію з черги RabbitMQ за допомогою бібліотеки MassTransit.

Інтерфейс сервісу:

- */notifications*
  Юрл для з’єднання з сервером через HTTP WebSocket. Якщо браузер не підтримує – через Long polling

WebApp сервіс

Односторінковий застосунок, написаний на Angular 9. Відповідає за відображення інформації та взаємодію з бекенд-сервісами. Саме його використовують звичайні користувачі.

Для авторизації користувачів він використовує Identity сервіс, для публікації та перегляду оголошень – Publishers, для відображення статистики по опублікованим оголошенням – Statistics, а для отримання сповіщень по новоствореним оголошенням – Notifications.

Для запуску та роботи Angular-застосунку в Docker спочатку запускається Nginx – веб-сервер та поштовий проксі-сервер, що працює на Unix-подібних операційних системах.

Admin сервіс

ASP.NET Core сервіс, який використовується адміністраторами для управління всім застосунком. Сервіс містить як фронтенд, так і бекенд частину. У ньому використовується MVC (Model-View-Controller) архітектура.

Admin не має власного сховища даних, для отримання та управлінням інформацією сервіс використовує Identity, Publishers та Statistics сервіси. За допомогою бібліотеки Refit, яку використовує цей сервіс, досить зручно описати REST-API зовнішних сервісів для роботи з ними.

Watchdog сервіс

Сервіс, який моніторить роботу інших сірвісів. Кожен сервіс має GET юрл */health*, який повертає статус роботи своїх компонентів.

За допомогою бібліотеки AspNetCore.HealthChecks відбувається перевірка роботи MS SQL SERVER, RabbitMQ, та самого сервісу. А за допомогою AspNetCore.HealthChecks.UI сервіс кожні 10 сек перевіряє роботоздатність сервісів та у зручному вигляді відображає ці дані на сторінці.

Інтерфейс сервісу:

- GET */healthchecks*
  Повертає HTML сторінку з інформацією про статус кожного сервісу застосунку

Розширення застосунку

На даний момент у застосунку використовується пряма взаємодія між клієнтом та сервером, тобто в WebApp та Admin сервісах записані адреси усіх сервісів, що вони використовують. У разі створення нового сервісу, потрібно дописувати нові адреси в клієнтські застосунки. Було б значно зручніше мати єдину точку для клієнтів, що буде перенапрявляти запит на потрібну кінцеву адресу. Для цього потрібно додати новий сервіс API Gateway.

Наразі сервіс Statistics збирає інформацію лише по новоствореним оголошенням. Ця інформаціє не є дуже корисною, варто додати моніторинг переглядів оголошень та за якими запитами найчастіше шукають книги.

Сервіс Notifications сповіщає про всі нові оголошення через WebSocket-и, але можуть бути і інші способи сповіщень. Варто додати можливість отримувати на електронну пошту, або push-повідомленнями на телефон. Також можна додати можливість, щоб користувачі обирали, за якими запитами публікації їх цікавлять та відправляти лише їх.

Всі сервіси запускаються в Docker Compose. Було б непогано перенести все в Kubernetes – платформу з відкритим вихідним кодом для управління контейнеризованими робочими навантаженнями та супутніми службами.

Порівняння з монолітною архітектурою

Мікросервісна архітектура містить набір невеликих, незалежних та автономних модулів, які надають різні послуги. Завдяки цьому кожен сервіс може бути написаний іншою мовою та різними розробниками незалежно один від одного. Це дуже велика перевага, але її не було відчутно, тому що роботу було виконано самотужки.

Ще однією перевагою є легка зрозумілість застосунку, адже по сервісах видно з яких частин він складається та які функції він виконує.

Легше тестувати окремий сервіс, адже він зазвичай невеликий, а отже швидше запускається та легше знайти помилку за наявності. Але продебажити зв’язок сервісів локально досить тяжко, а перевірити одночасно більше трьох сервісів майже неможливо. З монолітом таких проблем немає, дебажити його значно краще.

Одразу помітно, що в мікросервісах велика увага приділяється DevOps, потрібно використовувати та налаштовувати Docker для роботи з сервісами. Для нових сервісів потрібно створювати нову базу даних в якості сховища, за необхідності. Також потрібно налаштовувати чергу повідомлень для асинхронного обміну повідомлень. Це все займає чимало часу, а в моноліті цього робити не потрібно.

Висновки

Мету роботи було досягнуто: створено веб-застосунок для оголошень з прокату паперових книг, який написаний за домомогою мікросервісної архітектури. Основною мовою для розробки сервісів взято C#.NET, для фронтенду – Angular 9 та ASP.NET Core MVC.

Запропоновано варіанти розширення застосунку для подільшої розробки в майбутньому.

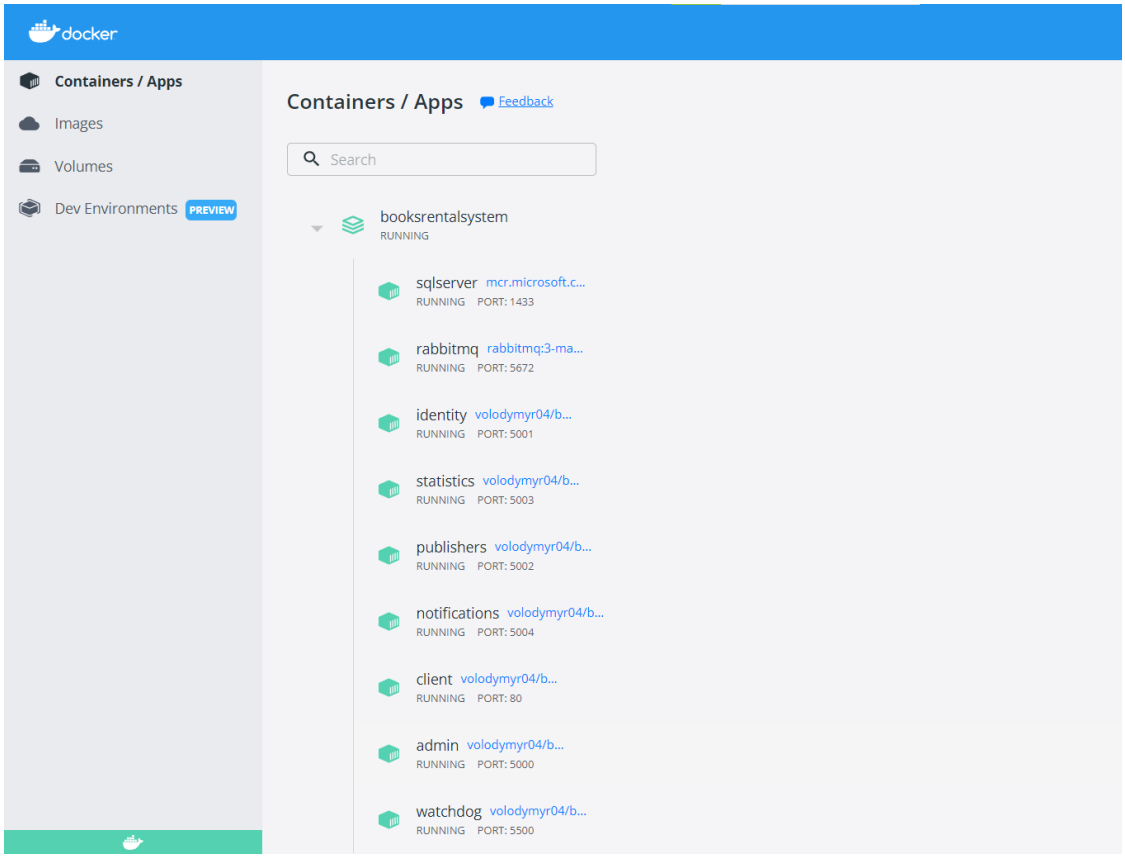
Також було порівняно такий архітектурний підхід з монолітом. Висновок – оскільки даний застосунок не є дуже обширним, його легше та значно швидше було б написати з використанням монолітної архітектури. Мікросервісна архітектура краще підходить для великих додатків, які витримують велике навантаження користувачів.

Список джерел

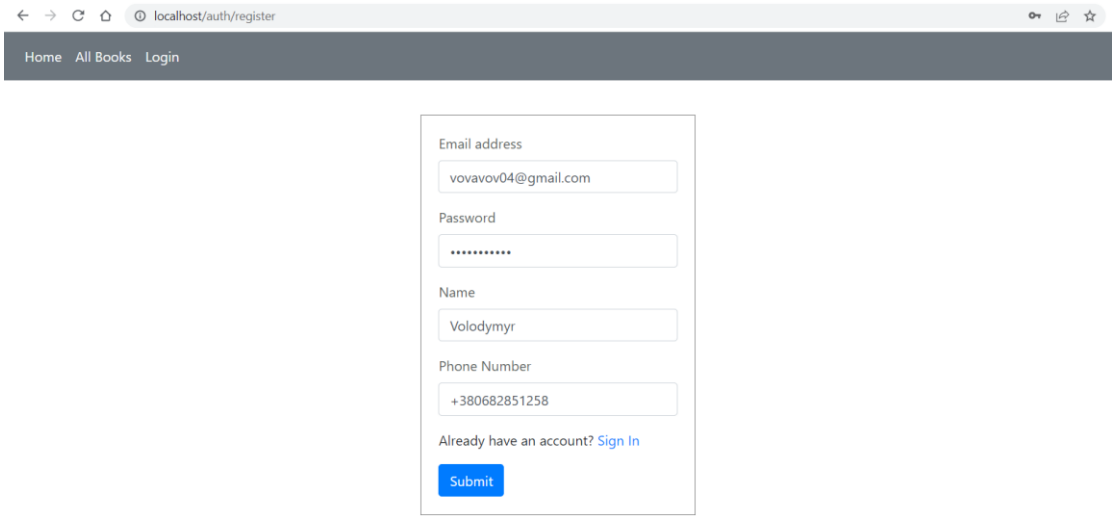
1. https://docs.microsoft.com/en-us/azure/architecture/guide/architecture-styles/microservices – мікросервісна архітектура (Microsoft).
2. https://cloud.google.com/learn/what-is-microservices-architecture – мікросервісна архітектура (Google).
3. https://aws.amazon.com/microservices/ - мікросервісна архітектура (Amazon).
4. https://docs.microsoft.com/en-us/aspnet/core/introduction-to-aspnet-core?view=aspnetcore-5.0 – документація ASP.NET Core.
5. https://docs.microsoft.com/en-us/aspnet/core/host-and-deploy/docker/building-net-docker-images?view=aspnetcore-5.0 – документація по роботі з Docker в ASP.NET Core.
6. https://docs.docker.com/compose – документація по Docker Compose.
7. https://masstransit-project.com/quick-starts/rabbitmq.html – MassTransit та RabbitMQ.
8. https://docs.microsoft.com/en-us/aspnet/core/signalr/introduction?view=aspnetcore-5.0 – документація по SignalR.
9. https://angular.io/start – документація по Angular.

Скріншоти застосунку

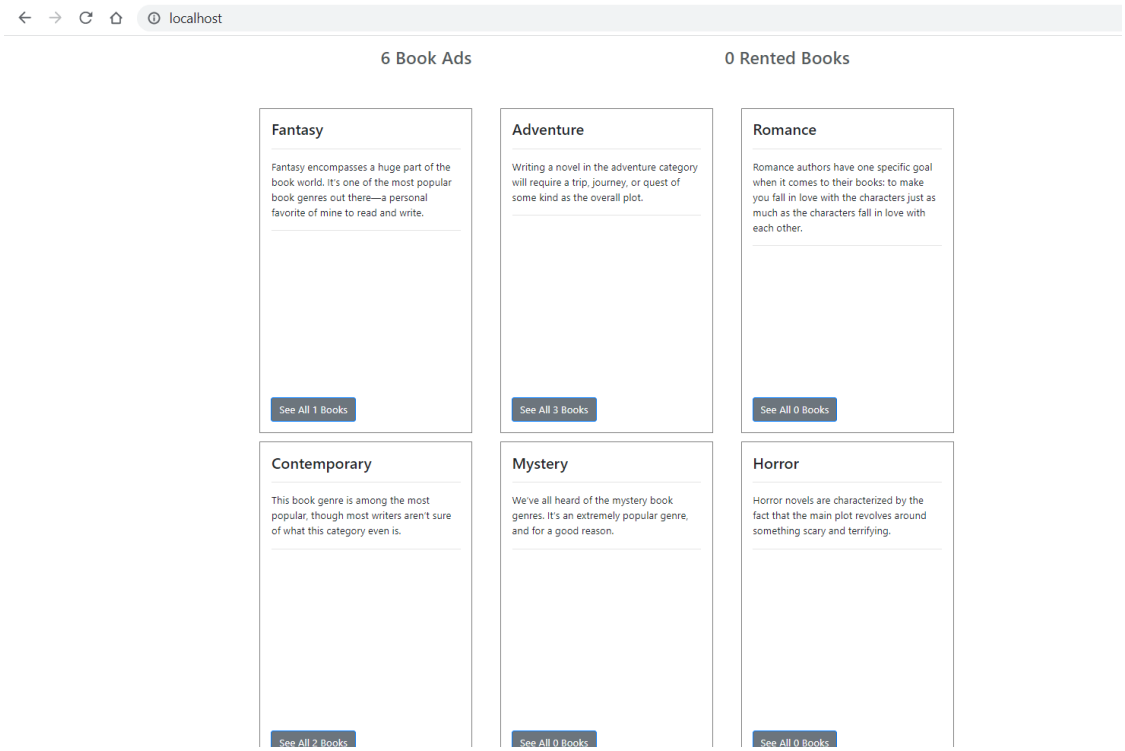
Запущені сервіси в застосунку Docker Desktop



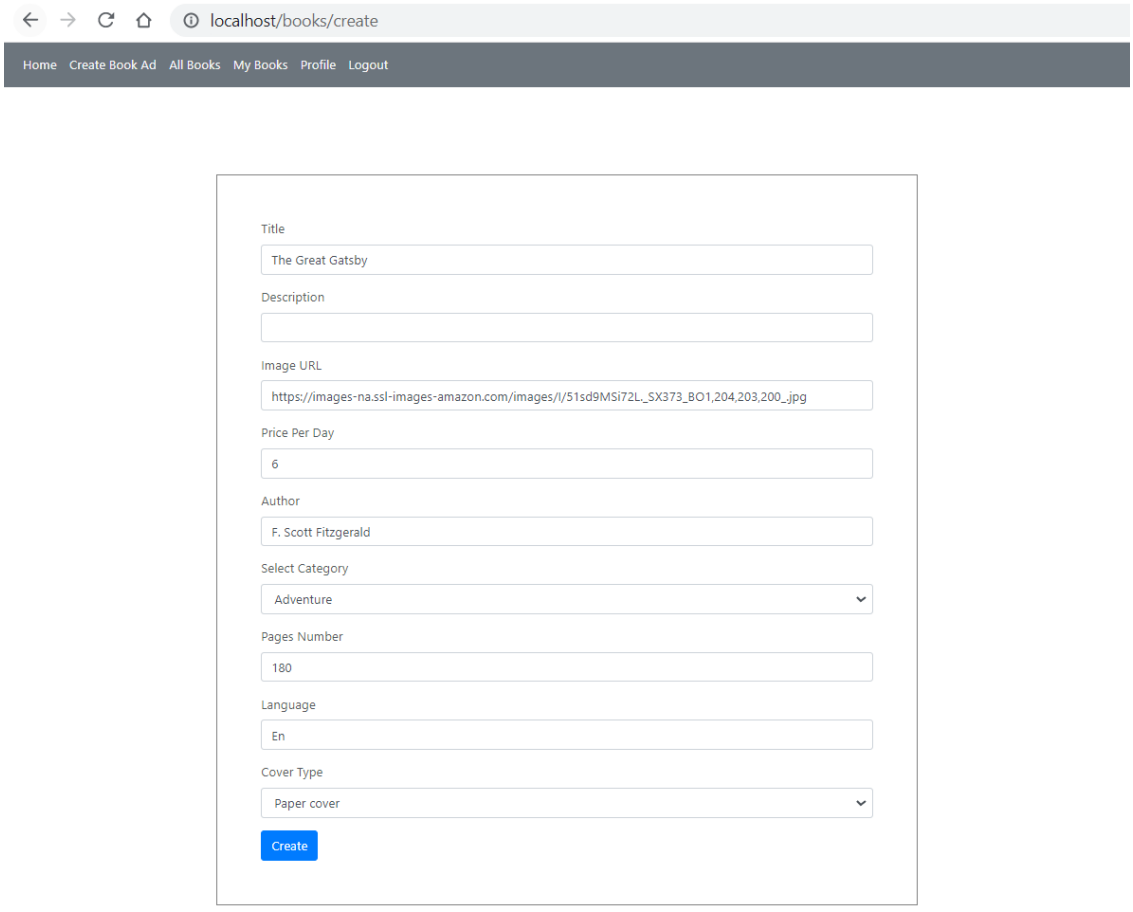
Сторінка реєстрації користувача



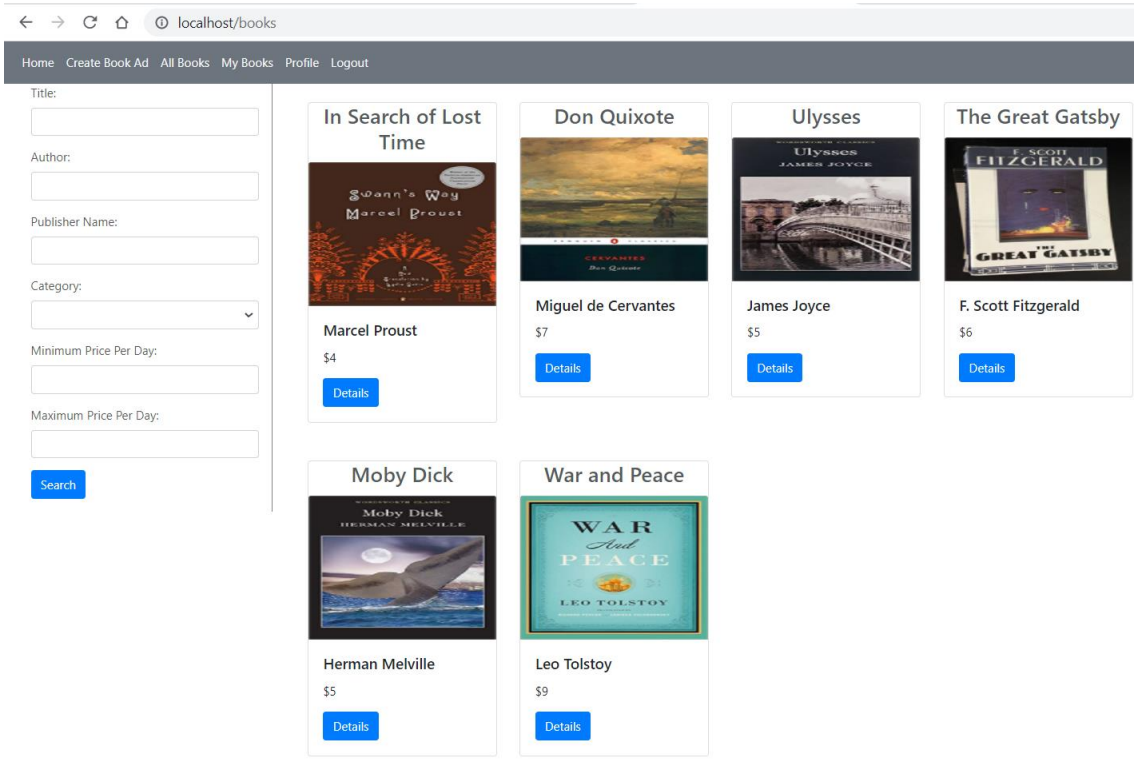
Головна сторінка



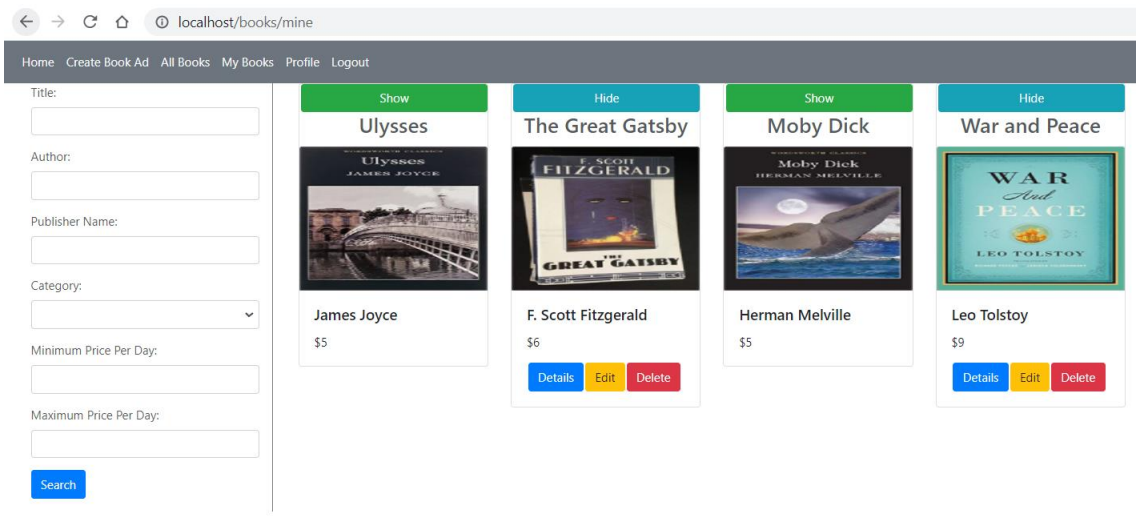
Форма створення оголошення



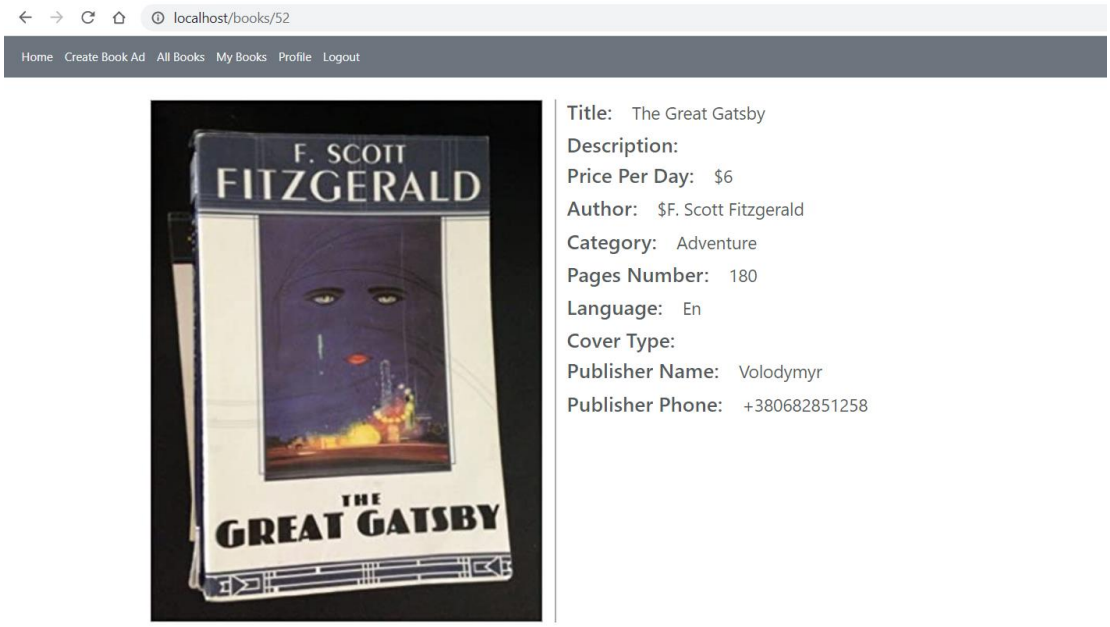
Перегляд та фільтрація активних оголошень



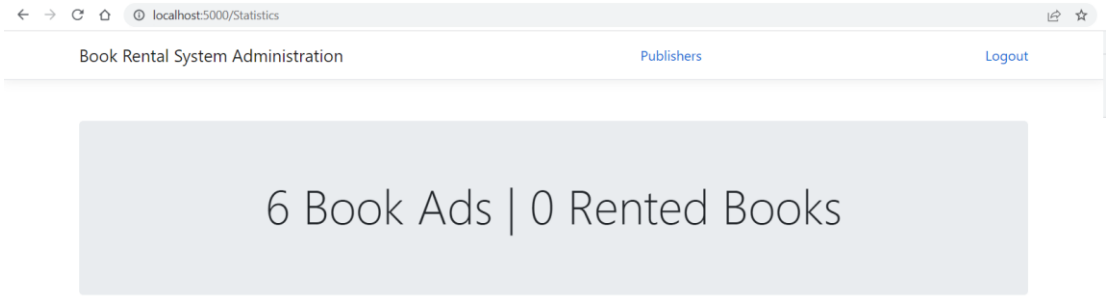
Перегляд створених оголошень



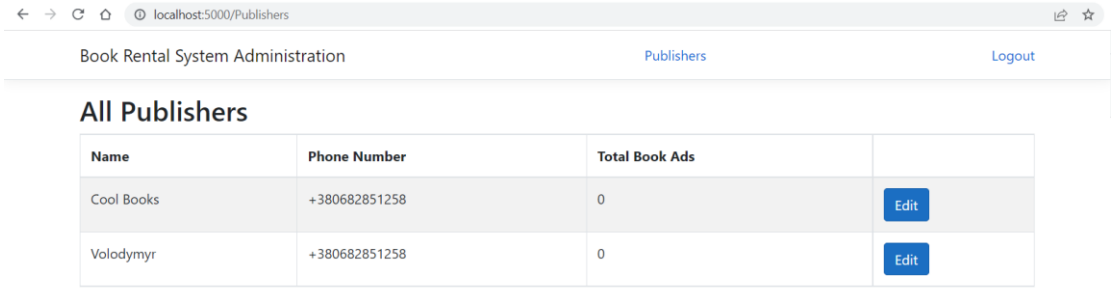
Інформація про конкретне оголошення



Головна сторінка для адміністраторів



Управління користувачами для адміністраторів



Перегляд стану сервісів