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

Факультет інформатики та обчислювальної техніки

Кафедра інформаційних систем та технологій

До захисту допущено:

Завідувач кафедри

_____ Олександр РОЛІК

«___»_____20__ р.

Дипломний проєкт

на здобуття ступеня бакалавра
за освітньо-професійною програмою «Інформаційне забезпечення робототехнічних систем»
спеціальності 126 «Інформаційні системи та технології»
на тему: «Система розкладу роботи співробітників барбершопу та замовлень послуг»

Виконав (-ла):
студент (-ка) IV курсу, групи ІК-92
Шелєханов Артем Андрійович _____

Керівник:
Ст. викладач, к.т.н.
Орленко Сергій Петрович _____

Рецензент:
Доцент каф. ОТ, к.т.н.,
Порєв Віктор Миколайович _____

Засвідчую, що у цьому дипломному проєкті немає запозичень з праць інших авторів без відповідних посилань.

Студент (-ка) _____

Київ – 2023 року

Національний технічний університет України

«Київський політехнічний інститут імені Ігоря Сікорського»

Факультет інформатики та обчислювальної техніки

Кафедра інформаційних систем та технологій

Рівень вищої освіти – перший (бакалаврський)

Спеціальність – 126 «Інформаційні системи та технології»

Освітньо-професійна програма «Інформаційне забезпечення робототехнічних систем»

ЗАТВЕРДЖУЮ

Завідувач кафедри

_____ Олександр РОЛІК

«___»_____20__ р.

ЗАВДАННЯ

на дипломний проєкт студенту

Шелєханову Артему Андрійовичу

1. Тема проєкту «Система розкладу роботи співробітників барбершопу та замовлень послуг», керівник проєкту Орленко Сергій Петрович, Ст. викладач, к.т.н., затверджені наказом по університету від «31» травня 2023 р. № 2101-с

2. Термін подання студентом проєкту: «12» червня 2023 р.

3. Вихідні дані до проєкту: можливість реєструватися у систему, можливість переглядати замовлення, можливість робити замовлення, можливість бачити розклад роботи, можливість створювати відгук.

4. Зміст пояснювальної записки: огляд предметної області та аналіз існуючих рішень реалізації, вибір технологій, проектування архітектури та структури проекта, вибір сховища даних для збереження інформації, обгрунтування вибору інструментів реалізації, розробка мікросервісів, безпека авторизації системи, розподілена система трасування.

5. Перелік графічного матеріалу (із зазначенням обов’язкових креслеників, плакатів, презентацій тощо): діаграма мікросервісу Catalog, діаграма мікросервісу Booking, діаграма мікросервісу Feedback, діаграма мікросервісу Schedule, діаграма мікросервісу User, діаграма прецедентів

6. Дата видачі завдання 15.02.2023 р.

Календарний план

№ з/п	Назва етапів виконання дипломного проєкту	Термін виконання етапів проєкту	Примітка
1	Ознайомлення з завданням	17.04.23 – 23.04.23	
2	Аналіз існуючих рішень та предметної області	24.04.23 – 30.04.23	
3	Проектування архітектури	01.05.23 – 12.05.23	
4	Розробка системи	13.05.23 – 04.06.23	
5	Оформлення документації	05.06.23 – 12.06.23	
6	Норм. контроль		
7	Перевірка на співпадіння		
8	Захист		

Студент Артем ШЕЛЄХАНОВ

Керівник Сергій ОРЛЕНКО

АНОТАЦІЯ

Шелєханов А. А. Система розкладу роботи співробітників барбершопу та замовлень послуг. КПІ ім. Ігоря Сікорського. Київ, 2023.

Проект містить 61 сторінки тексту, 30 рисунки, 15 посилань на літературні джерела, 6 конструкторські документи.

Ключові слова: мікросервісна архітектура, мікросервіси, система розкладу роботи, співробітники барбершопу, система замовлень послуг.

Об’єктом розробки є система для розкладу роботи співробітників та замовлень послуг.

Мета роботи: метою цієї роботи є розробка та впровадження системи розкладу роботи співробітників барбершопу та замовлень послуг, з використанням мови програмування Java, Spring framework, Spring Cloud та технології мікросервісної архітектури. Основним завданням проекту є поліпшення управління розкладом роботи та замовлень, забезпечення ефективного використання ресурсів та підвищення якості обслуговування в барбершопі.

У першому розділі було проведено аналіз рішень існуючих, їх переваги та недоліки. У другому розділі було проведено вибір технологій для розробки системи. У третьому розділі було проведено проектування архітектури та структури проекта, порівняли мікросервісну та монолітну архітектуру. У четвертому розділі було обрано базу даних для сховища інформації У п’ятому розділі було проведено обгрунтування вибору інструментів реалізації У шостому розділі було проведено розробку мікросервісів У сьомому розділі було розглянуто протокол комунікації між мікросервісами У восьмому розділі було реалізовано бузпеку авторизації У дев’ятому розділі було обрано та реалізовано розподілену систему трасування У десятому розділі було обрано кросс-браузерну сумісність для розробки вебзастосунку.

ANNOTATION

Sheliekhanov A. A. Schedule system for barbershop employees and service orders. Igor Sikorsky KPI. Kyiv, 2023.

The project contains 61 pages of text, 30 figures, 15 references to literary sources, 6 design documents.

Keywords: microservices architecture, microservices, work scheduling system, barbershop employees, service ordering system

The development object is a schedule system for barbershop employees and service orders.

Work objective: The objective of this work is to develop and implement a schedule system for barbershop employees and service orders, using the Java programming language, Spring framework, Spring Cloud, and microservices architecture technology. The main task is to improve the management of work schedules and orders, ensure efficient resource utilization, and enhance the quality of service in the barbershop.

In the first section, an analysis of existing solutions was conducted, including their advantages and disadvantages. The second section involved the selection of technologies for system development. The third section focused on the architecture and project structure design, comparing microservices and monolithic architecture. In the fourth section, a database was chosen as the information repository. The fifth section justified the selection of implementation tools. The sixth section involved the development of microservices. The seventh section discussed the communication protocol between microservices. The eighth section implemented secure authorization. In the ninth section, a distributed tracing system was selected and implemented. The tenth section addressed the selection of cross-browser compatibility for web application development.

Номер рядка	Формат	Позначення	Найменування	Кільк. аркушів	Номер екзем.	Примітка
			Документація загальна			
			Знову розроблена			
		ІК92.330БАК.004 ПЗ	Пояснювальна записка	61		
		ІК92.330БАК.004 Д1	Діаграма класів мікросервісу	1		
			Catalog			
		ІК92.330БАК.004 Д2	Діаграма класів мікросервісу	1		
			Booking			
		ІК92.330БАК.004 Д3	Діаграма класів мікросервісу	1		
			Feedback			
		ІК92.330БАК.004 Д4	Діаграма класів мікросервісу	1		
			Schedule			
		ІК92.330БАК.004 Д5	Діаграма класів мікросервісу	1		
			User			
		ІК92.330БАК.004 Д6	Діаграма прецедентів	1		

Зм.	Аркуш	№ докум.	Підпис	Дата	ІК92.330БАК.004 ТП		
Розроб.		Шелєханов А. А.			Система розкладу роботи співробітників барбершопу та замовлень послуг. Відомість проєкту	Літ.	Аркуш 1 / Аркушів 1
Керівн.		Орленко С. П.					
Затв.						КПІ ім. Ігоря Сікорського Група ІК-92	

Пояснювальна записка
до дипломного проєкту
на тему: «Система розкладу роботи співробітників барбершопу та замовлень послуг»

Київ – 2023 року

ЗМІСТ

ВСТУП

Система розкладу роботи співробітників барбершопу та замовлень послуг є актуальною у сучасному світі, де барбершопи стають все популярнішими місцями для чоловіків, які прагнуть професійного та якісного обслуговування. Ефективне управління розкладом роботи співробітників та замовленнями послуг важливо для задоволення потреб клієнтів та підтримання репутації барбершопу.

Незважаючи на те, що деякі барбершопи все ще використовують традиційні методи ручного розкладу роботи та замовлень, такий підхід може призводити до помилок, конфліктів та незадоволеності клієнтів. Тому впровадження автоматизованої системи розкладу роботи та замовлень стає необхідним для покращення ефективності та якості обслуговування.

Система замовлень послуг є невід'ємною частиною системи розкладу роботи. Вона надає можливість клієнтам заздалегідь резервувати послуги та встановлювати зручний для них час. Це дозволяє уникнути непотрібних затримок та очікувань, забезпечити високий рівень задоволеності клієнтів та побудувати стійкий позитивний досвід взаємодії з барбершопом.

Метою даного проекту є створення та впровадження такої системи розкладу роботи та замовлень в барбершопі. Ця система не лише спростить процес планування робочого часу, а й забезпечить високий рівень обслуговування та зручність для клієнтів.

У плані реалізації даного проекту, ми будемо використовувати сучасні технології та інструменти, зокрема Java, PostgreSQL, Spring Framework та Spring Cloud, що дозволить побудувати масштабовану та надійну мікросервісну архітектуру. Даний підхід гарантує гнучкість системи, а також високу продуктивність та безпеку даних.

Отже, актуальність теми полягає в необхідності впровадження сучасної системи розкладу роботи та замовлень, що сприятиме покращенню

ефективності та якості обслуговування в барбершопі, задоволенню потреб клієнтів та забезпеченню конкурентоспроможності на ринку.

У подальших розділах даної роботи ми розкриємо всі аспекти системи розкладу роботи співробітників барбершопу та замовлень послуг, що дозволить читачам отримати повний уявлення про її функціональність та переваги.

1 АНАЛІЗ ІНСУЮЧИХ РІШЕНЬ

1.1 Огляд існуючих систем та програм для розкладу роботи

Важливим аспектом при огляді існуючих систем та програм, призначених для автоматизації розкладу роботи співробітників барбершопу, є її функціональні можливості, зручність використання та сумісність з потребами барбершопу.

Під час огляду виявив наступні системи та програми:

– Acuity Scheduling є популярною онлайн-системою для керування розкладом роботи та замовлень. Вона надає можливість створювати розклад роботи співробітників, приймати замовлення в режимі реального часу та надавати клієнтам зручний спосіб записатися на послуги. Програма також підтримує автоматичне нагадування клієнтам про заплановані зустрічі;

– Shedul є іншою популярною системою для керування розкладом роботи барбершопу. Вона має широкий набір функцій, включаючи створення розкладу роботи, обробку замовлень, керування клієнтськими базами та звітності. Shedul також надає можливість клієнтам самостійно записуватися на послуги через онлайн-систему;

– SimplyBook.me є універсальною системою для керування розкладом роботи та замовлень, яка може бути використана в різних галузях, включаючи барбершопи. Ця система дозволяє створювати персоналізовані розклади роботи, приймати замовлення, надавати онлайн-платежі та надсилати підтвердження та нагадування клієнтам.

Тому, загальний огляд існуючих систем та програм показав, що є різні рішення для автоматизації розкладу роботи та замовлень в барбершопах. При виборі системи необхідно враховувати потреби, функціональність та можливості барбершопу, а також забезпечити зручний доступ для співробітників та клієнтів.

1.2 Переваги та недоліки

Аналізуючи існуючі системи можна виявити ряд переваг та недоліків, які корисно врахувати при виборі оптимального рішення.

Переваги:

– Автоматизація процесів: Використання систем для розкладу роботи дозволяє автоматизувати процеси планування, прийому замовлень та організації робочого часу співробітників. Це спрощує управління барбершопом і підвищує ефективність роботи;

– Зручність для клієнтів: Багато систем мають онлайн-інтерфейс, що дозволяє клієнтам зручно записатися на послуги, переглянути доступні часові слоти та вибрати зручний для них варіант. Це полегшує процес бронювання та забезпечує задоволення клієнтів;

– Оптимізація робочого часу: Системи розкладу роботи дозволяють оптимізувати використання робочого часу співробітників, розподіляючи замовлення та плануючи робочі графіки. Це дозволяє забезпечити ефективніше використання ресурсів барбершопу та зменшити час очікування для клієнтів;

– Збереження історії та статистики: Багато систем зберігають історію замовлень, дозволяючи аналізувати дані та статистику. Це може бути корисним для аналізу популярних послуг, планування ресурсів та прийняття стратегічних рішень.

Недоліки:

– Обмежена функціональність: Деякі готові системи мають обмежений функціонал та можливості настроювання. Це може ускладнити врахування специфічних потреб барбершопу та адаптацію системи під них;

– Складність інтеграції: При виборі існуючої системи слід враховувати ймовірність необхідності інтеграції з іншими програмними рішеннями, Це може бути складним процесом і вимагати додаткових зусиль.

Висновки до розділу 1

Аналізуючи існуючі системи та програми для розкладу роботи співробітників барбершопу, можна зробити деякі висновки. Важливим аспектом при виборі системи є функціональні можливості, зручність використання та сумісність з потребами барбершопу.

З огляду на існуючі системи, такі як Acuity Scheduling, Shedul і SimplyBook.me, можна виділити наступні переваги:

– автоматизація;
– зручний інтерфейс.

Проте, також важливо врахувати деякі недоліки. Деякі готові системи можуть мати обмежений функціонал та можливості настроювання, що ускладнює врахування специфічних потреб барбершопу.

Отже, при виборі системи для розкладу роботи барбершопу необхідно зважати на її функціональність, зручність використання, можливості настроювання та інтеграції.

2 ВИБІР ТЕХНОЛОГІЙ

2.1 Мова програмування Java

Мова програмування Java є однією з найпопулярніших мов у світі програмування. Вона була розроблена компанією Sun Microsystems (зараз належить Oracle Corporation) і має широке застосування в розробці програмного забезпечення. Java є повноцінною об'єктно-орієнтованою мовою програмування. Це означає, що програми на Java будуються на основі об'єктів, які взаємодіють між собою шляхом виклику методів. Об'єктно-орієнтований підхід сприяє покращенню структури програми, полегшує розробку, підтримку та розширення коду.

Java має велику екосистему, яка включає різноманітні бібліотеки, фреймворки та інструменти для розробки програмного забезпечення. Наприклад, для системи розкладу роботи співробітників барбершопу та замовлень послуг можна використовувати фреймворки Spring та Spring Boot, які спрощують розробку, управління залежностями та інтеграції.

Також одним з основних переваг Java є її переносимість. Програми, написані на Java, можуть працювати на різних операційних системах, таких як Windows, macOS, Linux та інші, без необхідності змінювати код. Це досягається завдяки використанню віртуальної машини Java (JVM), яка інтерпретує та виконує Java-код.

2.2 Технології Spring Framework

Spring Framework є одним з найпопулярніших фреймворків розробки програмного забезпечення для мови програмування Java. Він надає комплексний набір інструментів та бібліотек, які спрощують розробку і розгортання програмних додатків.

Spring Framework використовує Inversion of Control (IoC) контейнер для управління об'єктами та їх залежностями. Це дозволяє розробникам легко створювати, конфігурувати та управляти компонентами додатків. Також підтримує Dependency Injection (DI), що дозволяє інтегрувати залежності між об'єктами без прямого залежності реалізації. Це полегшує тестування, розширення та підтримку кодової бази.

Також Spring Framework включає в собі такі компоненти, як Spring Data, цей модуль дозволяє спростити роботу з базами даних, надаючи уніфікований підхід до роботи з різними системами керування базами даних. Він включає у себе підтримку для ORM (Object-Relational Mapping), JPA (Java Persistence API) та інших технологій доступу до даних.

Модуль Spring Security забезпечує можливості аутентифікації, авторизації та керування безпекою в додатках. Він дозволяє захистити доступ до ресурсів та забезпечити безпеку веб-додатків.

Присутній такий веб-фреймворк як Spring MVC, що надає потужні засоби для розробки веб-додатків. Він базується на моделі-представленні-контролері (Model-View-Controller) і дозволяє ефективно обробляти HTTP-запити, керувати переходами між сторінками та обробляти вхідні дані.

Функціонал Spring Boot, який спрощує розгортання та налаштування додатків. Воно автоматично налаштовує багато аспектів додатка, що дозволяє розробникам швидко створювати готові до використання додатки з мінімальними зусиллями.

2.3 Технології Spring Cloud

Spring Cloud - це набір інструментів, які розширюють функціональні можливості Spring Framework для розробки розподілених систем та мікросервісів. Він надає комплексні рішення для розгортання, конфігурації, моніторингу та управління мікросервісами у розподіленому середовищі.

Функціональні можливості Spring Cloud включають:

– «Service Discovery», Spring Cloud дозволяє реалізувати механізм виявлення сервісів, що дозволяє мікросервісам взаємодіяти між собою без необхідності знати точні адреси та порти. Це забезпечує гнучкість та масштабованість системи.

– «Load Balancing», Spring Cloud надає можливість розподілення навантаження між екземплярами мікросервісів. Він автоматично виявляє доступні сервіси та виконує розподіл запитів, що покращує швидкодію та надійність системи.

– «Circuit Breaker», Цей компонент дозволяє забезпечити відмовостійкість системи шляхом обмеження впливу недоступних сервісів на інші частини системи. Він контролює запити та може перехоплювати помилки, що дозволяє забезпечити ефективне управління помилками.

– «Distributed Configuration», Spring Cloud дозволяє централізовано керувати конфігурацією мікросервісів. Він забезпечує механізми для зберігання та оновлення конфігураційних параметрів без необхідності перезавантаження додатків.

– «Distributed Tracing», Цей компонент дозволяє відстежувати та аналізувати шляхи виклику сервісів у розподіленій системі. Він дозволяє збирати дані про виклики, час виконання та інші метрики, що допомагають виявляти проблеми та оптимізувати шляхи виклику.

Висновок до розділу 2

Провівши аналіз даного розділу можна зробити деякі висновки. А саме, що мова програмування Java є популярною та широко застосовується в розробці програмного забезпечення. Вона має повну підтримку об'єктно-орієнтованого підходу, що полегшує розробку та підтримку коду.

Застосувавши мову програмування Java та фреймворки Spring Framework і Spring Cloud в системі розкладу роботи співробітників барбершопу може

допомогти спростити розробку, підтримку та управління. Вони забезпечать високу ефективність, масштабованість та безпеку системи.

Таким чином, використання мови програмування Java разом із фреймворками Spring Framework і Spring Cloud в системі розкладу роботи співробітників барбершопу є обґрунтованим вибором, оскільки це сприяє полегшенню розробки

3 ПРОЕКТУВАННЯ АРХІТЕКТУРИ ТА СТРУКТУРИ ПРОЕКТА

У комп’ютерному світі присутні наступні способи створення програмного забезпечення: мікросервіси та монолітна архітектура. У них є як хороші, так і погані сторони, і вибір правильного дуже важливий для успішної роботи вашого проекту. Тому важливо зрозуміти різницю між ними та про що слід пам’ятати, вибираючи одину із них.

3.1 Структура мікросервісної архітектури

Мікросервісна архітектура — це підхід до розробки програмного забезпечення, в якому програма розбивається на невеликі незалежні сервіси, кожен з яких виконує конкретну функцію та має свій власний набір API. Ці сервіси працюють разом, взаємодіючи один з одним через мережу.

Основна ідея мікросервісної архітектури полягає в тому, щоб розділити великі та складні системи на невеликі, легко керовані компоненти, які можуть бути розгорнуті, масштабовані та оновлювані окремо. Кожен мікросервіс відповідає за свою власну функціональність та має чітко визначений контекст.

Кожен мікросервіс функціонує незалежно від інших компонентів системи. Він може бути розроблений, тестований, розгорнутий та масштабований окремо. Це дозволяє командам розробників працювати над окремими мікросервісами без впливу на решту системи.

Мікросервіс має чітко визначену функціональність, яку він виконує. Він не повинен включати зайвий код або функціональність, що не відноситься до його області відповідальності. Це допомагає забезпечити чистоту та простоту кожного мікросервісу. Комунікація мікросервісів відбувається через мережу, використовуючи стандартизовані протоколи комунікації, такі як HTTP, REST або Messaging. Це дозволяє їм комунікувати та обмінюватися даними для виконання комплексних бізнес-процесів.

Ця технологія дозволяє мікросервісам бути незалежно розгорнуті та масштабовані. Це означає, що можна гнучко налаштовувати ресурси для кожного мікросервісу в залежності від його потреб. Також дозволяє ефективно використовувати ресурси та забезпечує гнучкість управління навантаженням.

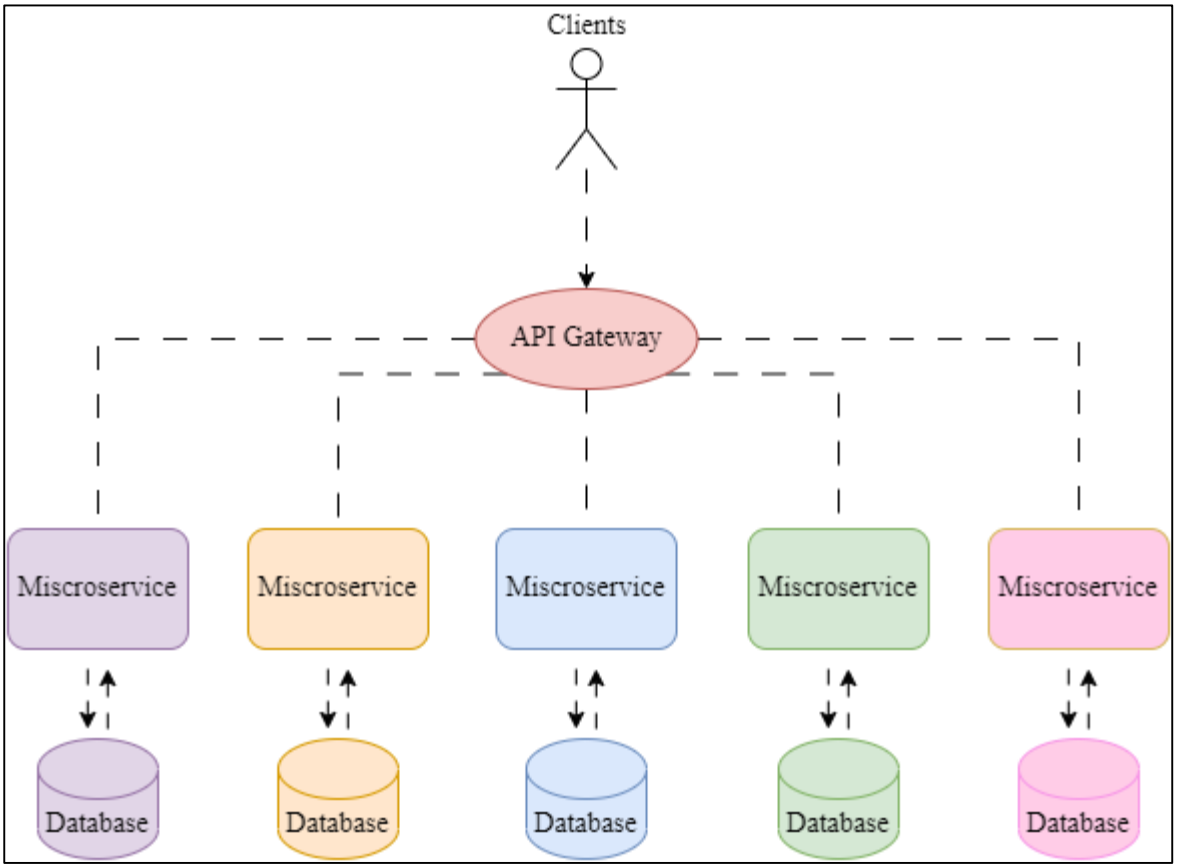

Рисунок 3.1 — Приклад будови мікросервісної архітектури

Основна ідея мікросервісної архітектури полягає в тому, щоб розділити великі та складні системи на невеликі, легко керовані компоненти, які можуть бути розгорнуті, масштабовані та оновлювані окремо.

Переваги мікросервісної архітектури:

– Гнучкість: Мікросервіси можуть бути незалежно розгорнуті, масштабовані та оновлювані. Це дозволяє розробникам працювати над окремими сервісами без впливу на решту системи;

– Легкість розгортання та оновлення: Кожен сервіс може бути розгорнутий окремо, що спрощує процес розгортання та оновлення програмного забезпечення;

– Масштабованість: Мікросервіси можуть бути масштабовані окремо, що дозволяє ефективно використовувати ресурси та забезпечує гнучкість управління навантаженням;

– Зручність для розробників: Кожен сервіс може бути розроблений незалежно від інших, що спрощує процес розробки та підтримки кодової бази.

Недоліки мікросервісної архітектури:

– Складність управління: З більшою кількістю сервісів стає складніше управляти та координувати їх взаємодію.

– Комплексність тестування: Кожен сервіс потребує окремого тестування, а також тестування взаємодії між сервісами.

– Збільшення навантаження на мережу: Взаємодія між сервісами відбувається через мережу, що може призводити до збільшення навантаження та затримок.

Можна зробити висновок, що мікросервісна архітектура є потужним інструментом для розробки складних систем, забезпечуючи гнучкість, масштабованість та швидкість розробки. Проте, перед впровадженням такої архітектури слід ретельно проаналізувати вимоги проекту та збалансувати переваги та недоліки для досягнення найкращих результатів.

3.2 Структура монолітної архітектури

Звичайний метод побудови додатків як єдиного, автономного блоку називається монолітною архітектурою. Усі елементи та функції в цій архітектурі тісно пов’язані та виконуються єдиним процесом виконання. Кодова база, база даних і одиниця розгортання для програми часто однакові. Коли потрібні зміни або оновлення, розробники співпрацюють,

використовуючи одну кодову базу. Це гарантує, що зміни впроваджуються по всьому стеку одночасно.

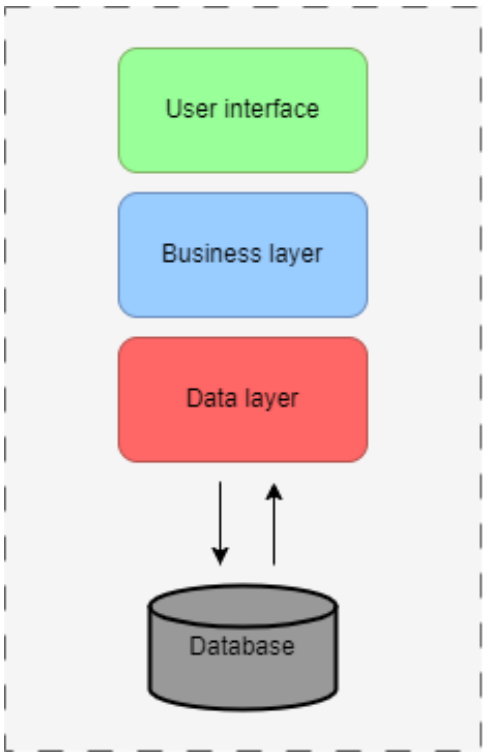

Рисунок 3.2 — Приклад монолітної архітектури

Використовуючи саме цю архітектуру, яка складається з таких частин: база даних, ступінь доступності та комунікації даними, бізнес-логіка та інтерфейсу клієнта.

Аналізуючи ці рівні можна сказати, що масштабовані програми можуть стати перешкодою при використанні монолітної архітектури через взаємопов’язаність модулів системи. Все ж таки якщо виникне потреба щось додати або оновити функціональні можливості, уся система повинна бути модифікована, включаючи збірку, тестування та впровадження, адже присутня міцна зв’язаність між модулями системи. Монолітна архітектура найкраще працює в проектах обмеженого масштабу, які не потребують частих змін кодової бази.

Висновок до розділу 3

Можна зробити висновок, що і мікросервіси, і монолітні архітектури мають свої переваги та вимоги. Монолітна архітектура забезпечує простоту та легкість розробки, тоді як архітектура мікросервісів забезпечує масштабованість, гнучкість і стійкість. Тому авжеж, кращим варіантом буде використання мікросервісної архітектури.

4 БАЗА ДАНИХ

В архітектурі мікросервісів, додатки складаються з невеликих незалежних сервісів із вибором різних баз даних, як реляційних, так і нереляційних. Популярним підходом є поліглотна стійкість (Polyglot persistence), яка передбачає вибір різних баз даних на основі конкретних потреб і характеристик кожного мікросервісу. Технологія не обмежується використанням єдиної технології бази даних.

У моєму випадку буду використовувати лише реляційну базу даних, а саме PostgreSQL.

PostgreSQL — це потужна та безкоштовна реляційна база даних з відкритим кодом, відома своєю надійністю та широким набором функцій.

На основі одного мікросервіса покажемо як підключити нашу базу даних до проекта. Тому, спочатку потрібно у системі PostgreSQL сворити відповідну базу даних. Створена база даних до одного з сервісів показана на рис. 4.1.

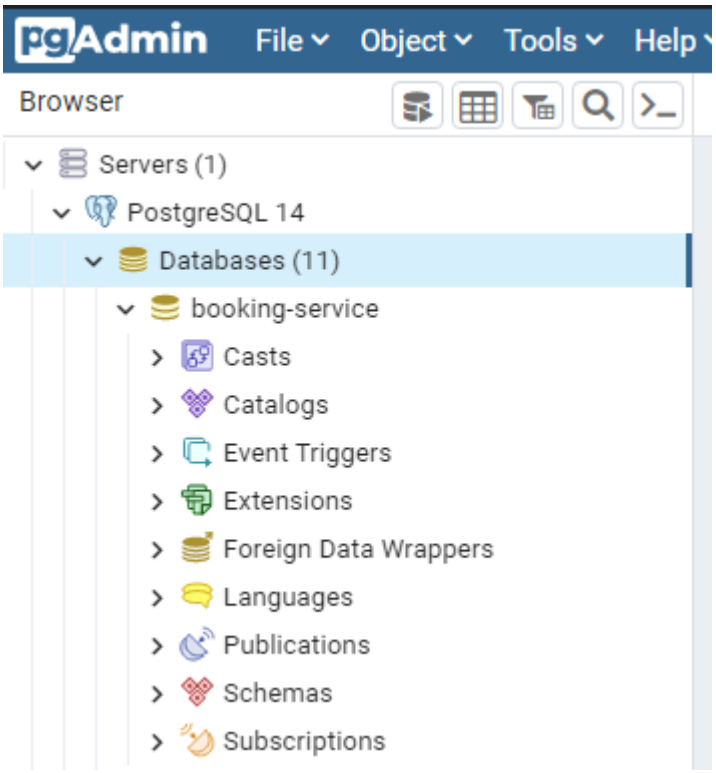

Рисунок 4.1 — Створена БД до одного з мікросервісів

Далі, щоб під'єднати нашу створену базу даних до проекта, тобто до одного мікросервіса, нам потрібно у файлі application.yml зробити відповідну конфігурацію яка зображена на рис. 4.2.

```yaml
spring:
  datasource:
    url: jdbc:postgresql://localhost:5432/booking-service
    username: postgres
    password: postgres
    driver-class-name: org.postgresql.Driver
  application:
    name: booking-service
  jpa:
    hibernate:
      ddl-auto: update
    show-sql: true
    properties:
      hibernate:
        format_sql: true
    database: postgresql
    database-platform: org.hibernate.dialect.PostgreSQLDialect
```

Рисунок 4.2 — Налаштування application.yml файлу для підключення до БД

Для того щоб перевірити чи коректно було проведено налаштування нашого файлу, потрібно зробити тестове підключення до бази даних. Тому, давайте це зробимо використовуючи таке середовище розробки як IntelliJ IDEA, рис. 4.3.

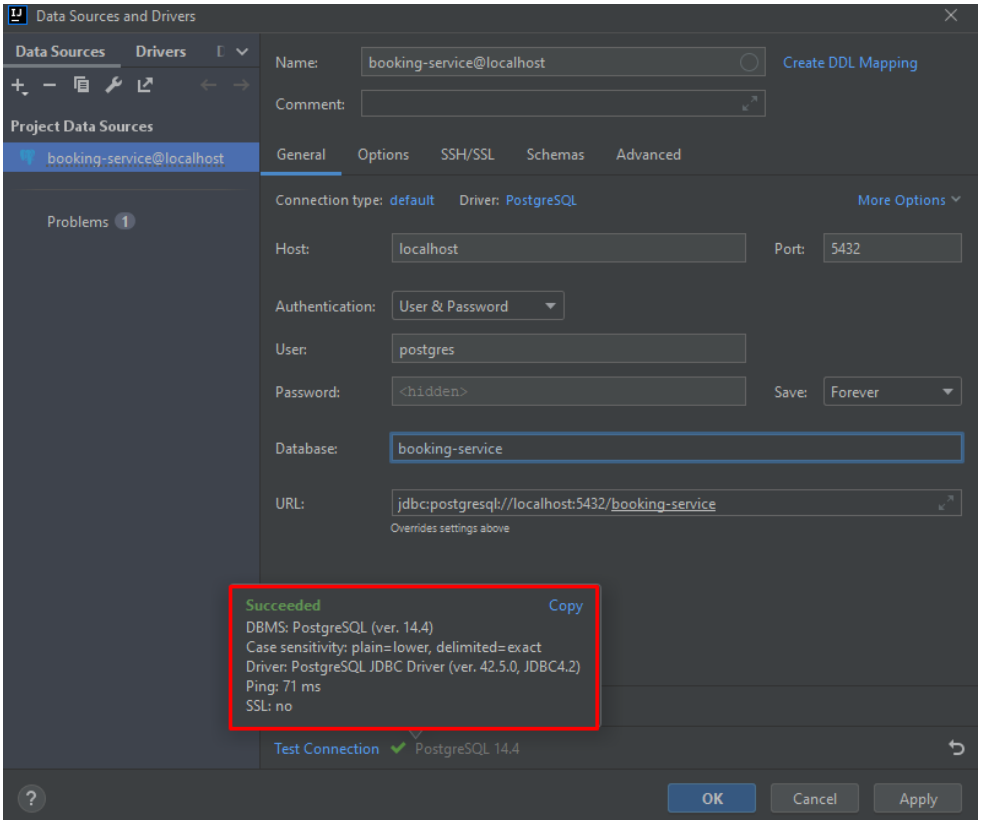

Рисунок 4.3. — Успішне підключення до БД

Як видно, наше тестове підключення пройшло успішно, тому в подальшому можемо створювати та підключати відповідні бази даних до наших мікросервісів.

Висновки до розділу 4

У цьому розділі ми розібрали як використовують базу даних у мікросервісній архітектурі. Зазначили, що система складається з невеликих незалежних сервісів, які можуть використовувати різні типи баз даних, реляційні або нереляційні. Відповідно обрали таку базу даних як PostgreSQL та провели тестовий процес підключення бази даних до одного з мікросервісів.

5 ОБГРУНТУВАННЯ ВИБОРУ ІНСТРУМЕНТІВ РЕАЛІЗАЦІЇ

У сучасному світі, де швидкість розвитку технологій постійно зростає, вибір правильних інструментів для реалізації проектів стає ключовим фактором успіху. Тому зважаючи на нашу епоху, переважно більшість людей шукають або розробляють щось ефективніше в порівнянні з іншими технологіями. У контексті розробки системи розкладу роботи співробітників барбершопу та замовлень послуг, ми звернули увагу на низку передових інструментів, які можуть забезпечити необхідні функціональності та переваги. Java можна вважати однією з популярних мов програмування, оскільки вона має активну спільноту та високу популярність серед розробників.

Саме цей, один із інструментів є найпоширенішим у світі програмування який в свій час набув резонанс серед розробників та має широкий набір ресурсів та бібліотек.

На основі мови програмування Java було розроблено Spring Framework та Spring Cloud які вважаються потужними фреймворками для розробки мікросервісних архітектур. Адже, їхня гнучкість, простота використання та розширюваність в свою чергу надають можливість ефективно розробляти та управляти розподіленими системами, саме це робить їх привабливими для багатьох розробників.

При проектуванні системи потрібно обрати його архітектуру. Порівнявши між монолітною та мікросервісною архітектурою ми прийшли до висновку, що у сучасному світі, де-факто, мікросервісна архітектура стала стандартом для розробки сучасних додатків. Ця архітектурна парадигма дозволяє розбити систему на невеликі, незалежні компоненти - мікросервіси, що працюють разом як єдине ціле.

Також, коли мова йде про веб-розробку потрібно визначитися з підходящою базою даних. Оскільки вже було розглянуто, знаєм, що мікросервісна архітектура складається з певних кількістю сервісів, які в свою чергу мають відповідну реляційну або нереляційну базу даних. Переглядаючи

на ринку надійні бази даних я зупинився на PostgreSQL. Саме ця реляційна база даних виступає як надійна система. Вона надає широкий набір функцій, включаючи підтримку транзакцій, високу надійність та ефективне управління даними у нашій системі.

Вибір саме цих технологій обгрунтований наявністю широкого спектру функціональності, високої продуктивності, підтримки співтовариством розробників та їх популярністю в сучасному IT-середовищі. Використання цих інструментів дозволить створити потужну та ефективну систему.

Висновки до розділу 5

У даному розділі було обгрунтовано вибір інструментів для реалізації проекту розкладу роботи співробітників барбершопу та замовлень послуг. Враховуючи швидкий розвиток технологій, було обрано сучасну і ефективну мову програмування Java та її широкий набір ресурсів і бібліотек. Також, для розробки мікросервісних архітектур було обрано Spring Framework та Spring Cloud. Архітектурною парадигмою, яка була обрана, є мікросервісна архітектура. Для бази даних було обрано PostgreSQL, реляційну систему.

6 РОЗРОБКА МІКРОСЕРВІСІВ

Важливо виконувати поетапне впровадження окремих мікросервісів. Краще вести розробку основних мікросервісів, які менш залежать від інших, поступово переходячи до більш складних сервісів. Використаємо технології та фреймворки, які відповідають конкретним вимогам кожного мікросервісу.

Відповідно нашій темі диплому можна розробити наступні мікросервіси:

– Schedule service: керує плануванням послуг і часовими інтервалами;

– Booking service: обслуговує бронювання клієнтів, дії адміністратора;

– Feedback service: керує відгуками клієнтів;

– User service: обробляє автентифікацію користувачів і ролі;

– Catalog service: Надає каталог послуг та перелік майстрів салону.

У кожному мікросервісі можна структурувати проект папок таким чином:

– «controller»: містить контролери, які обробляють вхідні запити та визначають кінцеві точки API;

– «dto»: містить об’єкти передачі даних (DTO), які визначають структуру даних, якими обмінюються клієнт і мікросервіс;

– «model»: містить класи сутностей, які представляють моделі даних і зіставлені з таблицями бази даних;

– «repository»: містить сховища або об’єкти доступу до даних, які обробляють операції та запити бази даних;

– «service»: містить класи послуг, які реалізують бізнес-логіку та взаємодіють зі сховищами.

Крім того, кожна мікрослужба може мати папку `resources`, яка містить файли конфігурації, такі як application.yml або application.properties.

Важливим компонентом у мікросервісній архітектурі є:

– Discovery server: обслуговує пошук та реєстрацію мікросервісів.

Також неможливо уникнути використання API Gateway, яка виступає у ролі зворотного проксі-сервера, розташованого між клієнтами та основними мікросервісами:

– API Gateway: ця служба діє як шлюз API для мікросервісів, надаючи єдину точку входу для клієнтських запитів і обробляючи маршрутизацію та балансування навантаження.

6.1 Discovery server

Discovery server — відомий як реєстр служб, є центральним компонентом, який керує реєстром усіх служб, доступних у нашій системі. Він діє як каталог, де мікросервіси можуть реєструватися та знаходити інші сервіси. Сервер виявлення забезпечує централізований динамічний спосіб виявлення та зв’язку зі службами.

Сервер виявлення забезпечує децентралізоване спілкування між мікросервісами. Службам не потрібно заздалегідь знати мережеве розташування чи IP-адреси інших служб. Натомість вони можуть покладатися на сервер виявлення, щоб надати необхідну їм інформацію, що забезпечує слабкий зв’язок і легшу масштабованість.

Для реалізації Discovery server можна використати доступні інструменти та технології як Netflix Eureka який входить до Spring Cloud модуля. Тому, щоб наш реєстр служб працював потрібно виконати наступні дії:

– Додати до додатка залежність для використання Netflix Eureka;

– Додати анотацію яка вказуватиме, що саме цей клас є головним для сервера;

– Налаштування конфігурації application.yml файла.

Давайте розберемо ці пункти більш детально. Відповідно, щоб використовувати потрібні інструменти та технології потрібно підключити наступну залежність яка показана на рис. 6.1.

```xml
<dependency>
    <groupId>org.springframework.cloud</groupId>
    <artifactId>spring-cloud-starter-netflix-eureka-server</artifactId>
</dependency>
```

Рисунок 6.1 — Залежність для підключення Eureka server

Не менш важливим етапом є призначення анотацією @EnableEurekaServer клас, який вказуватиме, що є головним для сервера, саме це зображено на рис. 6.2.

```java
@SpringBootApplication
@EnableEurekaServer
public class DiscoveryServerApplication {
    public static void main(String[] args) {
        SpringApplication.run(DiscoveryServerApplication.class, args);
    }
}
```

Рисунок 6.2 — Підключення анотації @EnableEurekaServer

Наступним кроком, буде налаштування конфігурації application.yml файла для подальшого запуска цього сервера, рис. 6.3.

```yaml
eureka:
  instance:
    hostname: localhost
  client:
    register-with-eureka: false
    fetch-registry: false
    serviceUrl:
      defaultZone: http://localhost:8761/eureka/
```

Рисунок 6.3 — Налаштування application.yml для Eureka server

Наданий фрагмент налаштування є прикладом конфігурації для мікросервісу, який використовує сервер Eureka для реєстрації та виявлення сервісу. Розберемо його більш детально:

– eureka: Це основна частина конфігурації сервера Eureka;

– instance: Цей підрозділ визначає параметри конфігурації для самого екземпляра сервера Eureka;

– hostname: «localhost» вказує ім’я хоста примірника сервера Eureka. У цьому прикладі встановлено «localhost»;

– client: Цей розділ визначає параметри конфігурації для клієнта Eureka, який відповідає за реєстрацію мікросервісів на сервері Eureka та виявлення інших сервісів;

– register-with-eureka: «false», цей параметр визначає, чи повинен мікросервіс автоматично реєструватися на сервері Eureka. У цьому прикладі встановлено значення false, що вказує на те, що мікросервіс не зареєстровано;

– fetch-registry: «false», цей параметр визначає, чи повинен мікросервіс отримувати інформацію реєстру з сервера Eureka. Встановлення значення «false» означає, що мікросервіс не буде отримувати інформацію про інші сервіси з сервера Eureka;

– serviceUrl: Цей підрозділ визначає конфігурацію URL-адреси служби для клієнта Eureka;

– defaultZone: http://localhost:8761/eureka/, Це URL-адреса REST API сервера Eureka. Мікросервіс використовуватиме цю URL-адресу для зв’язку із сервером Eureka. У цьому прикладі для URL-адреси встановлено `http://localhost:8761/eureka/`.

Підсумовуючи, наданий фрагмент конфігурації налаштовує мікросервіс для використання сервера Eureka для реєстрації та виявлення сервісу. Мікросервіс налаштовано так, щоб не реєструватися на сервері Eureka (register-with-eureka: «false») і не отримувати інформацію реєстру з сервера Eureka (fetch-registry: «false»). Для URL-адреси REST API сервера Eureka встановлено значення `http://localhost:8761/eureka/`.

Виконавши всі ці етапи отримаємо на виході Discovery Server, запустивши якого та перейшовши на адресу http://localhost:8761/eureka/ можемо побачити його працездатність і реєстрування сервісів, рис. 6.4.

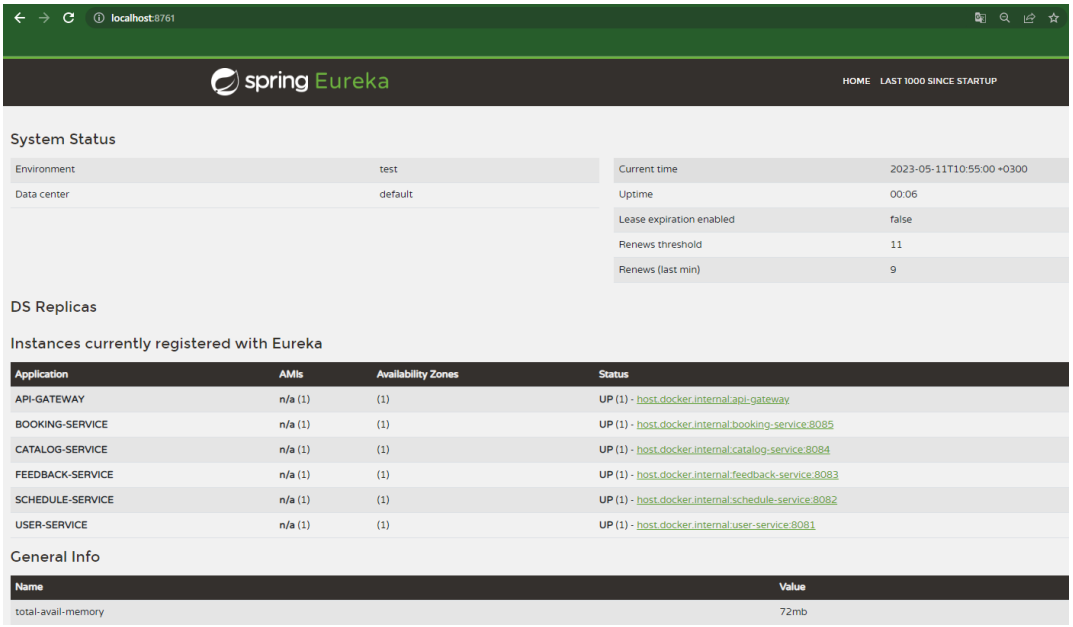

Рисунок 6.4 — Успішний запуск Eureke Server та вже зареєстровані сервіси

6.2 Schedule service

Schedule Service — це мікросервіс, створений з використанням середовища Spring, зокрема Spring Cloud, який надає інструменти та бібліотеки для розробки розподілених систем. Його основне призначення — керувати доступністю майстрів салону, відстежувати запис на прийом і надавати інформацію про розклад у реальному часі.

Сервіс розкладу відповідає архітектурі мікросервісу, використовуючи Spring Boot і Spring Cloud для бездоганної інтеграції та масштабованості. Він використовує такі технології, як Spring Data JPA для збереження бази даних.

Він інтегрується з сервером виявлення, таким як Netflix Eureka, щоб увімкнути виявлення служб і балансування навантаження. Тому, щоб реалізувати інтеграцію з сервером виявлення потрібно встановити наступну залежність яка зображена на рис. 6.5.

```xml
<dependency>
    <groupId>org.springframework.cloud</groupId>
    <artifactId>spring-cloud-starter-netflix-eureka-client</artifactId>
</dependency>
```

Рисунок 6.5 — Залежність для комунікації з Eureka у дескрипторі pom.xml

Також потрібно призначити до класу відповідну анотацію @EnableDiscoveryClient яка використовується в програмах Spring Boot, щоб увімкнути реєстрацію служб і можливості виявлення, надані основним механізмом виявлення служб, таким як Eureka, рис. 6.6.

```java
@SpringBootApplication
@EnableDiscoveryClient
public class ScheduleServiceApplication {

    public static void main(String[] args) {
        SpringApplication.run(ScheduleServiceApplication.class, args);
    }

}
```

Рисунок 6.6 — Підключення анотацію @EnableDiscoveriClient

Наступним кроком буде налаштування файлу application.yml для взаємодії з Discovery Server, що зображено на рисунку 6.7.

```yaml
eureka:
  client:
    serviceUrl:
      defaultZone: http://localhost:8761/eureka
server:
  port: 8082
```

Рисунок 6.7. — Конфігурація файлу для взаємодії з Discovery Server

6.3 Booking service

Booking service — це мікросервіс, розроблений з використанням фреймворку Spring, зокрема Spring Cloud, який пропонує набір інструментів і бібліотек для створення розподілених систем. Його головна мета — керувати процесом бронювання, від вибору послуг і майстрів до керування часовими інтервалами.

Служба бронювання розроблена за допомогою Spring Boot і Spring Cloud із використанням різних модулів і фреймворків у екосистемі Spring. Він використовує Spring Data JPA для збереження даних, що дозволяє ефективно зберігати та отримувати інформацію

Служба бронювання також інтегрується з сервером виявлення, а саме Netflix Eureka, щоб увімкнути виявлення служби та балансування навантаження. Він використовує безпечні протоколи зв’язку, такі як OAuth 2.0, щоб забезпечити конфіденційність і цілісність даних клієнта під час процесу бронювання.

6.4 Feedback service

Служба зворотного зв’язку, як частина архітектури мікросервісу, призначена для збору цінних відгуків від клієнтів і сприяння постійному вдосконаленню барбершопу.

Feedback service — це мікросервіс, створений за допомогою фреймворку Spring та інтегрована в архітектуру мікросервісу, відповідає за керування відгуками клієнтів у додатку салону краси. Це дозволяє клієнтам залишати відгуки про їхній досвід обслуговування, а також полегшує спілкування між керівництвом салону та клієнтами.

Цей сервіс має інтеграцію з сервером виявлення, таким як Netflix Eureka, налаштування якого відбувається так само як і у попередніх прикладах.

6.5 User service

User service — це мікросервіс, створений з використанням фреймворку Spring, зокрема з використанням Spring Boot і Spring Security. Його основна мета полягає в обробці всіх аспектів керування користувачами в системі барбершопу, включаючи реєстрацію користувачів, автентифікацію та авторизацію.

Служба користувача розроблена з використанням фреймворку Spring Boot, використовуючи переваги його простоти та продуктивності. Spring Security використовується для реалізації механізмів автентифікації та авторизації, забезпечуючи надійні заходи безпеки для керування користувачами.

Зроблено комунікацію із сервером виявлення, для внесенні у реєстр у Discovery server.

6.6 Catalog service

У барбершопах надання різноманітних послуг і безперебійний доступ до інформації є важливими для залучення та утримання клієнтів. Служба каталогу, ключовий компонент програми барбершопу на основі мікросервісів, відіграє вирішальну роль в управлінні та представленні послуг салону та доступних майстрів.

Catalog service — це мікросервіс, створений із використанням фреймворку Spring, зокрема з використанням Spring Boot і Spring Data JPA. Його основне призначення полягає в управлінні та пошуку сервісної та основної інформації в програмі салону краси. Надаючи вичерпний каталог послуг і доступних майстрів. Служба каталогу покращує досвід клієнтів і спрощує процес бронювання.

Цей сервіс реалізований за допомогою Spring Boot, який забезпечує легку та ефективну структуру для створення мікросервісів. Spring Data JPA використовується для безперебійної взаємодії з основною базою даних, що дозволяє легко зберігати та отримувати службову та основну інформацію.

У ньому також реалізовано комунікацію із сервером виявлення, для внесенні у реєстр у Discovery server.

6.7 API Gateway

У сучасній епохі розподілених систем і архітектури мікросервісів, ефективне управління складністю зв'язку між сервісами є надзвичайно важливим. Роль API Gateway як централізованої точки входу та ключового компонента програми на основі мікросервісів є незамінною.

API Gateway виступає як зворотний проксі-сервер, розташований між клієнтами і основними мікросервісами. Він служить єдиним вхідним пунктом для всіх запитів і виконує маршрутизацію, балансування навантаження, забезпечення безпеки та протокольного перекладу. API Gateway грає ключову роль у абстрагуванні складності базової мікросервісної архітектури, надаючи уніфікований інтерфейс для взаємодії клієнтів з системою.

Тому, щоб зробити цей зв’язок між клієнтами та основними мікросервісами потрібно виконати наступні етапи, а саме:

– Завантажити залежність у pom.xml файл для можливості використовувати технології api gateway;

– Налаштувати маршрути у конфігурації application.yml.

Розглянемо більш детально використовуючи приклад. На першому етапі потрібно підтягнути відповідну залежність для функціонування api gateway. Тому саме на рис. 6.8. це зображено.

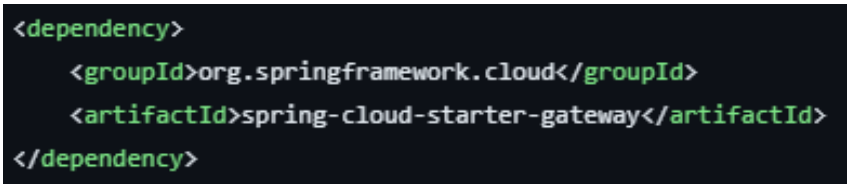

Рисунок 6.8 — Завантаження залежності api-gateway

Далі потрібно у файлі application.yml зробити конфігурацію маршрутів для API Gateway в контексті Spring Cloud Gateway, рис. 6.9. Взявши за основу усі створені мікросервіси та служби, а саме: Discovery server, Schedule service, Booking service, Feedback service, User service, Catalog service, налаштуємо шлях до кожного з них для комунікації, щоб отримувати доступ до даних.

Тобто, API Gateway виступає маршрутизатором до серверної частини, що полегшує логіку та покращує ефективність отримання потрібної інформації з серверної частини. За допомогою API Gateway забезпечується централізований доступ до різних сервісів, що дозволяє спростити інтеграцію та зменшити залежності між окремими сервісами.

Правильна настройка API Gateway є важливим етапом проектування архітектури мікросервісів і має велике значення для забезпечення успішного впровадження та роботи системи. Також ця служба може виконувати додаткові функції, такі як аутентифікація та авторизація запитів, кешування даних, логування та моніторинг. Це дозволяє контролювати доступ до сервісів, забезпечувати безпеку та слідкувати за метриками продуктивності.

```yaml
cloud:
  gateway:
    routes:
      ## Booking Service ROUTE
      - id: booking-service
        uri: lb://booking-service
        predicates:
          - Path=/api/booking
      ## Catalog Service ROUTE
      - id: catalog-service
        uri: lb://catalog-service
        predicates:
          - Path=/api/catalog
      ## Feedback Service ROUTE
      - id: feedback-service
        uri: lb://feedback-service
        predicates:
          - Path=/api/feedback
      ## Schedule Service ROUTE
      - id: schedule-service
        uri: lb://schedule-service
        predicates:
          - Path=/api/schedule
      ## User Service ROUTE
      - id: user-service
        uri: lb://user-service
        predicates:
          - Path=/api/user
      ## Discovery Server ROUTE
      - id: discovery-server
        uri: http://localhost:8761/eureka
        predicates:
          - Path=/eureka/web
        filters:
          - SetPath=/
      ## Discovery Server Static Resources ROUTE
      - id: discovery-server-static
        uri: http://localhost:8761/eureka
        predicates:
          - Path=/eureka/**
```

Рисунок 6.9 — Конфігурація маршрутів API Gateway

Це налаштування включає в собі такі основні елементи:

– cloud - це ключове слово, яке вказує на використання функціональності хмар або додатків, що базуються на хмарах;

– gateway - це вказує, що відбувається налаштовування API Gateway;

– routes - це список маршрутів, які налаштовуються в API Gateway;

– id - це ідентифікатор маршруту, який використовується для ідентифікації маршруту;

– uri - це URI (Uniform Resource Identifier), який вказує на місце призначення для пересилання запиту. В нашому випадку у деяких URI використовується такий параметр як “lb”, що означає "load-balanced" (збалансоване навантаження). Тобто це говорить про те, що запити будуть перенаправлені до сервісу, який працює у режимі збалансованого навантаження;

– predicates - це умови, які визначають, коли маршрут буде співпадати з запитом. У даному випадку маршрут буде співпадати з запитами, які мають шлях

В цілому, API Gateway відіграє ключову роль у спрощенні та керуванні зв'язком між клієнтами і мікросервісами у розподіленій системі. Виступаючи як централізована точка входу, він забезпечує єдиний інтерфейс, підвищує рівень безпеки, масштабованості та загальної продуктивності системи. Шляхом обробки маршрутизації запитів, трансляції протоколів, забезпечення безпеки та агрегації послуг, служба API Gateway надає значні переваги як для розробників, так і для клієнтів.

Висновок до розділу 6

Даний розділ містить опис різних мікросервісів, які використовуються в системі барбершопу. Кожен з цих мікросервісів розроблений з використанням Spring Framework та Java-фреймворку Spring Cloud і має свою власну функціональність. Важливим компонентом є "API Gateway", виступає як централізована точка входу і ключовий компонент у системі мікросервісів. API Gateway уніфікує взаємодію між клієнтами та мікросервісами і спрощує архітектуру системи.

Також було побудовано Discovery server за допомогою Netflix Eureka. Тому кожен мікросервіс інтегрований з сервером виявлення Netflix Eureka.

Загалом, ці мікросервіси разом з API Gateway сприяють ефективному управлінню процесом бронювання, збору відгуків клієнтів, управлінню користувачами та каталогом послуг у системі барбершопу. Вони забезпечують безпеку, масштабованість та продуктивність системи, використовуючи розподілену архітектуру мікросервісів.

7 ПРОТОКОЛ КОМУНІКАЦІЇ МІЖ МІСКРОСЕРВІСАМИ

Технології дуже стрімко розвиваються, тому коли розподілені системи та сервісно-орієнтована архітектура стали стандартом розробки програмного забезпечення, виникає потреба у забезпеченні ефективної комунікації між різними сервісами. Існує багато технологій та протоколів, що надають можливість взаємодіяти між сервісами, проте одним з найпопулярніших та широко використовуваних є REST API.

REST API (Representational State Transfer) є ще одним популярним протоколом комунікації між сервісами. Він базується на стандартах HTTP і використовує простий набір методів, таких як GET, POST, PUT і DELETE, для взаємодії з ресурсами. REST API надає простоту та легкість у використанні, дозволяючи розробникам швидко створювати та споживати API. Цей протокол відповідає багатьом принципам веб-архітектури, що робить його популярним вибором для багатьох розробників.

RESTful API використовує HTTP методи для здійснення операцій з ресурсами. API RESTful використовує вже існуючі стандарти та методології HTTP, які описані у протоколі RFC 2616, наприклад:

- GET для отримання ресурсу;
- PUT, щоб змінити стан або оновити ресурс, який може бути об’єктом, файлом або блоком;
- POST, щоб створити цей ресурс;
- DELETE, щоб видалити його.

Крім того, REST API підтримує кешування, що дозволяє зберігати копії відповідей сервера на клієнтській стороні. Це може покращити продуктивність та знизити навантаження на сервер. REST API також надає можливість передавати дані у різних форматах, таких як JSON, XML або навіть зображення.

Звісно є і інші протокол комунікації між сервісами, такі як SOAP (Simple Object Access Protocol) та GraphQL. Проте якщо аналізувати ці протоколи то можна сказати, що протокол SOAP уже застарілий і є досить складним у

використанні та має великий розмір повідомлень, що призводить до збільшення накладних витрат та складнощів у взаємодії з іншими системами. Щодо протокола GraphQL то одна з його переваг полягає в тому, що клієнт може вказати, які саме дані йому потрібні, тим самим уникнувши надлишкової передачі даних. Однак, GraphQL також має свої недоліки, включаючи складність налаштування та недостатню підтримку для деяких специфічних випадків.

Висновок до розділу 7

В цьому розділі був проведений розгляд рішення у протоколах комунікації між мікросервісами. Хоча існують інші протоколи комунікації між сервісами, такі як SOAP і GraphQL, REST API залишається більш популярним і широко використовуваним. Використовуючи прості методи HTTP, такі як GET, POST, PUT і DELETE, REST API дозволяє здійснювати операції з ресурсами і надає простоту та легкість використання для розробників. Тому REST API став незамінним інструментом у сфері розробки програмного забезпечення.

8 БЕЗПЕКА АВТОРИЗАЦІЇ

Напевно всім відомо, що являє собою безпека в інформаційному простору. Адже на даний час присутня велика кількість шахраїв, які можуть зашкодити вашій безпеці у інтернеті. Саме тому безпека персональних даних є надзвичайно важливим аспектом в сфері інформаційної безпеки та розробки програмного забезпечення і відіграє важливу роль у нашому житті.

Запобігання несанкціонованому доступу до ресурсів і забезпечення належного контролю доступу є основними проблемами для будь-якої системи чи програми. Швидкий розвиток технологій і збільшення кількості інтернет-додатків ставлять під загрозу безпеку наших даних і систем.

У контексті комп'ютерної безпеки авторизація означає аутентифікацію користувача та надання йому необхідних привілеїв і прав для доступу до системи чи ресурсу. Правильна та надійна авторизація є необхідною умовою для забезпечення безпеки та конфіденційності інформації, а також запобігання несанкціонованому використанню та зловживанням.

Отже одним із сучасних рішень для реалізації безпеки авторизації є система Keycloak

8.1 Система Keycloak

Система Keycloak є одним із сучасних рішень для захисту авторизації. Keycloak — це відкрита платформа керування ідентифікацією та авторизацією, яка надає розширені функції безпеки. Це дозволяє розробникам створювати надійні системи контролю доступу з використанням сучасних стандартів безпеки, такі як OpenID Connect, OAuth 2.0 і SAML.

Крім того, Keycloak забезпечує централізоване управління користувачами, включаючи створення, редагування та видалення облікових записів, а також підтримує синхронізацію зовнішніх джерел даних. Користувачі також можуть автентифікуватись за допомогою своїх облікових записів з

соціальних мереж, що спрощує процес реєстрації та входу. Keycloak також підтримує одноразовий вхід (Single Sign-On), що дозволяє автентифікуватись один раз і отримувати доступ до різних додатків без повторної автентифікації. Додатково, Keycloak надає можливість вести аудит та журналювання подій, пов'язаних з автентифікацією, авторизацією та доступом користувачів, що допомагає виявляти вразливості та відстежувати аномальні дії. Keycloak є гнучким інструментом, який може бути розширений та налаштований під конкретні потреби додатку.

8.2 Метод авторизації OAuth 2.0

Наразі взаємодія між різними системами та додатками стала невід'ємною частиною нашого щоденного життя, виникає необхідність в надійній та безпечній аутентифікації користувачів і забезпеченні доступу до ресурсів. Одним із ключових методів аутентифікації, який набув великої популярності в останні роки, є OAuth 2.0.

OAuth 2.0 є відкритим протоколом, який дозволяє користувачам надавати обмежений доступ до своїх ресурсів іншим додаткам, не передаючи свої облікові дані. Цей протокол був розроблений з метою забезпечення безпеки та простоти використання.

Основна перевага OAuth 2.0 порівняно з іншими протоколами аутентифікації полягає в тому, що користувач не повинен передавати свої облікові дані стороннім додаткам. Замість цього, протокол використовує токени доступу, що дозволяють додаткам отримати доступ до ресурсів від імені користувача.

Окрім цього, він є гнучким протоколом, який підтримує різні методи аутентифікації, такі як введення пароля, аутентифікація через соціальні мережі (наприклад, Google або Facebook) і використання сертифікатів або апаратних пристроїв. Це дає розробникам можливість вибирати найбільш підходящий метод аутентифікації залежно від вимог їх додатків та рівня безпеки.

Також цей протокол є широко використовуваним в різних сферах, включаючи соціальні мережі, фінансові сервіси, хмарні платформи та інші. Він дозволяє забезпечити безпеку та зручність взаємодії між додатками та системами, забезпечуючи високий рівень контролю доступу та захисту облікових даних користувачів.

Отже розуміння методу аутентифікації OAuth 2.0 є важливим для розробників та архітекторів, оскільки він надає ефективний та безпечний спосіб авторизації та керування доступом до ресурсів у розподілених середовищах. Знання про цей протокол дозволить розробникам використовувати його потужні можливості для створення безпечних та взаємодіючих додатків, що відповідають вимогам сучасних користувачів та бізнесу.

8.3 Забезпечення авторизації через Keyclock

Забезпечення авторизації через Keycloak вимагає деяких конфігураційних та інтеграційних крокі у нашій системі. Давайте послідовно розглянемо етапи інтеграції платформи Keyclock у систему:

- Інсталяція та конфігурація Keycloak;
- Інтеграція Keycloak у систему;
- Налаштування ролей та прав доступу;
- Тестування виконаних дій.

Для того щоб перейти до наступного етапу потрібно інсталювати Keycloak та налаштувати його у відповідності до вимог нашої системи. Це включає налаштування аутентифікаційних провайдерів та створення ролей та клієнтів для нашої системи. Використовуючи Docker hub за допомогою наступної команди, ввівши в термінал docker run -p 8181:8080 -e KEYCLOAK_ADMIN=admin -e KEYCLOAK_ADMIN_PASSWORD=admin quay.io/keycloak/keycloak:21.1.1 start-dev можемо інсталювати та запустити платформу Keyclock перейшовши по цьому посиланню http://localhost:8181/, рис. 8.1.

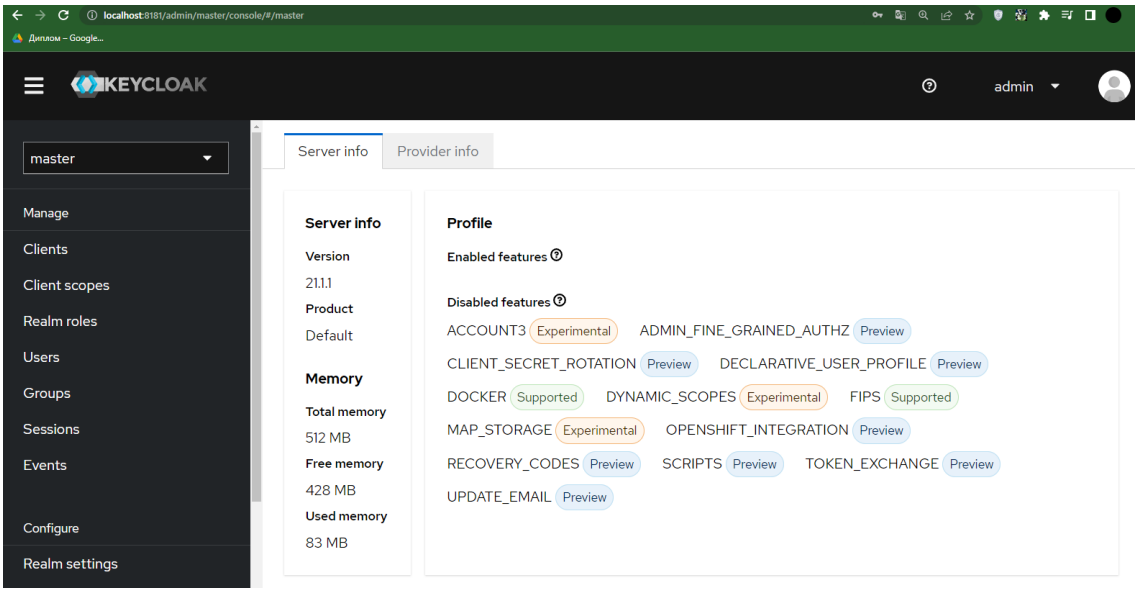

Рисунок 8.1 — Начальна сторінка платформи Keyclock

Тепер, інсталювавши систему Keyclock перейдемо до його інтеграції у нашу систему. Для інтеграції Keycloak з Java та Spring Framework потрібно вручну налаштувати Spring Security для використання Keycloak як провайдера авторизації. Це включає налаштування зв'язку з Keycloak, налаштування перехоплення авторизаційних запитів та перевірку токенів доступу.

Отже, спочатку потрібно, на платформі Keyclock створити та налаштувати власні параметри області, які керують параметрами для користувачів, програм, ролей і груп у поточній області. Створена власна область під назвою spring-boot-microservices-barber-shop-realm, зображена на рис. 8.2.

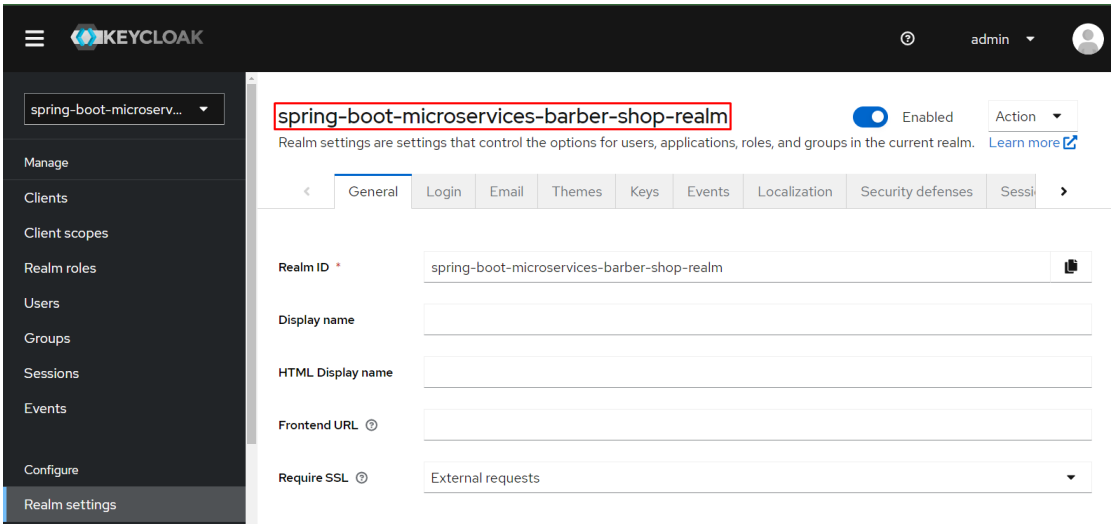

Рисунок 8.2 — Параметризована конфігураційна зона Keycloak для власної системи.

На основі вже створеної області потрібно ще зробити так званих клієнтів, які відіграють важливу роль у встановленні зв'язку між додатками та сервером Keycloak, а також у налаштуванні параметрів авторизації та отримання токенів доступу. Основна причина використання клієнтів у Keycloak полягає в тому, що вони дозволяють окремим додаткам або сервісам взаємодіяти з сервером Keycloak та отримувати авторизаційні токени для доступу до захищених ресурсів. Кожен клієнт має свій власний ідентифікатор (Client ID) та секрет (Client Secret), які використовуються для взаємодії з Keycloak API. Створений клієнт під назвою spring-cloud-client зображений на рис. 8.3.

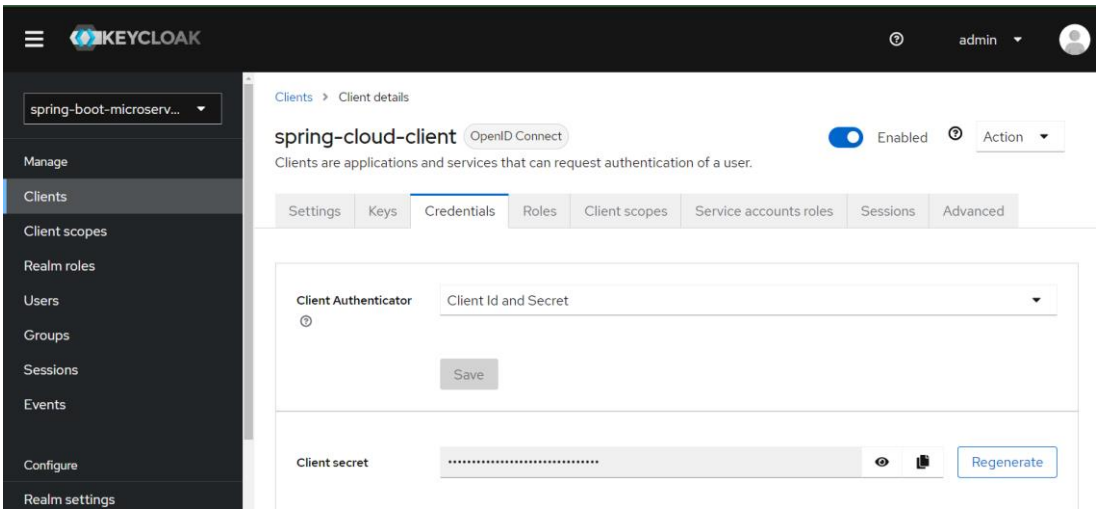

Рисунок 8.3 — Створений клієнт для зв'язку з системою

Цей клієнт також використовує OpenID протокол. Хоч OpenID і OAuth 2.0 - це два різні протоколи, які використовуються для різних цілей, але мають певну взаємозв'язок. OpenID - це протокол аутентифікації, який розширює OAuth 2.0 та додає до нього можливість аутентифікації користувача. OpenID дозволяє використовувати токени доступу OAuth 2.0 для підтвердження ідентичності користувача та отримання пов'язаних з цим даних. Це дозволяє додаткам отримувати інформацію про користувача та підтверджувати його ідентичність за допомогою одного протоколу.

Отже, OAuth 2.0 і OpenID - це два різних протоколи, проте вони часто використовуються разом для забезпечення як авторизації, так і аутентифікації в додатках. OAuth 2.0 використовується для авторизації та отримання доступу до ресурсів, тоді як OpenID використовується для аутентифікації та підтвердження ідентичності користувача.

Продовжуючи етап інтерації Keyclock, у нашу систему потрібно додати залежність яка дасть змогу в майбутньому підключитися до області та реалізувати авторизацію. Залежність зображена на рис. 8.4.

```
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-oauth2-resource-server</artifactId>
</dependency>
```

Рисунок 8.4 — Залежність для реалізації протоколу OAuth 2.0

Цей блок коду spring-boot-starter-oauth2-resource-server додає підтримку для створення ресурс-сервера в Spring Boot додатках. Надає необхідні компоненти та налаштування для перевірки та обробки токенів доступу, які використовуються для авторизації користувачів та захисту доступу до ресурсів. Отже ця бібліотека спрощує процес інтеграції аутентифікації та авторизації OAuth 2.0 в Spring Boot додатках, дозволяючи розробникам зосередитись на бізнес-логіці свого додатку замість розробки складних механізмів безпеки.

Перейшовши до файлу application.yml у API Gateway налаштуємо підключення системи до створеної області у Keyclock. Конфігурація файлу зображена на рис. 8.5.

```yaml
security:
  oauth2:
    resourceserver:
      jwt:
        issuer-uri: http://localhost:8181/realms/spring-boot-microservices-barber-shop-realm
```

Рисунок 8.5 — Конфігурація для підключення до області Keyclock

Ця конфігурація представляє налаштування для ресурс-сервера OAuth 2.0 в Spring Security конфігурації. Конфігурація вказує, що ресурс-сервер буде перевіряти та обробляти JWT (JSON Web Token) для авторизації користувачів та захисту доступу до ресурсів.

Параметр issuer-uri визначає URI (Uniform Resource Identifier) випуску (issuer) токенів доступу. В цьому випадку, вказано http://localhost:8181/realms/spring-boot-microservices-barber-shop-realm як URI випуску. Це означає, що ресурс-сервер буде перевіряти токени доступу, випущені цим URI, і буде довіряти їм для авторизації користувачів.

Тепер є мосжливість перейти до третього кроку, налаштування прав доступу. Для цього у сервісі API Gateway створимо файл SecurityConfig де налаштуємо права доступу користувачів до сервісів, рис. 8.6.

```java
@Configuration
@EnableWebFluxSecurity
public class SecurityConfig {
    @Bean
    public SecurityWebFilterChain securityWebFilterChain(ServerHttpSecurity http) {
        http.csrf() ServerHttpSecurity.CsrfSpec
                .disable() ServerHttpSecurity
                .authorizeExchange() ServerHttpSecurity.AuthorizeExchangeSpec
                .pathMatchers( ...antPatterns: "/booking/**").hasRole("ADMIN")
                .pathMatchers( ...antPatterns: "/catalog/**").authenticated()
                .pathMatchers( ...antPatterns: "/feedback/**").authenticated()
                .pathMatchers( ...antPatterns: "/schedule/**").hasAnyRole( ...roles: "ADMIN", "MASTER")
                .pathMatchers( ...antPatterns: "/user/**").authenticated()
                .anyExchange().permitAll()
                .and().oauth2ResourceServer(ServerHttpSecurity.OAuth2ResourceServerSpec::jwt);
        return http.build();
    }
}
```

Рисунок 8.6 — Налаштування прав доступу до сервісів

Даний клас SecurityConfig анотований з використанням @Configuration та @EnableWebFluxSecurity, що позначає його як конфігурацію безпеки та включає підтримку захисту для веб-застосування. У методі securityWebFilterChain використовується ServerHttpSecurity для налаштування фільтрації запитів та обробки правил безпеки.

Останнім етапом буде перевірка коректної роботи авторизації користувачів. Для цього використаємо додаток Postman та зробимо GET запит до Booking service, рис. 8.7

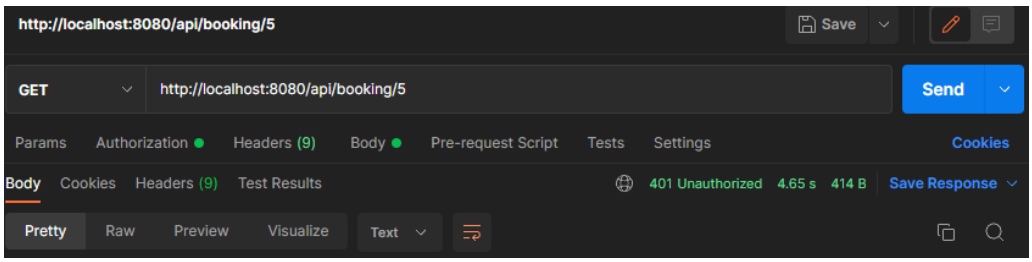

Рисунок 8.7 — Тестовий GET запит для перевірки авторизації

Бачимо, що виконаний запит говорить нам про те, що відбулося коректне інтегрування авторизації у нашу систему, адже саме HTTP статус 401 "Unauthorized" означає, що запит не має достатніх авторизаційних даних або що авторизація була відхилена. Ця помилка виникла через те, що користувач не

надав достатніх авторизаційних даних для доступу до захищених ресурсів. В даному випадку це відсутність токена.

Зробимо такий самий запит, проте вже використаємо токен для успішної авторизації, рис. 8.8.

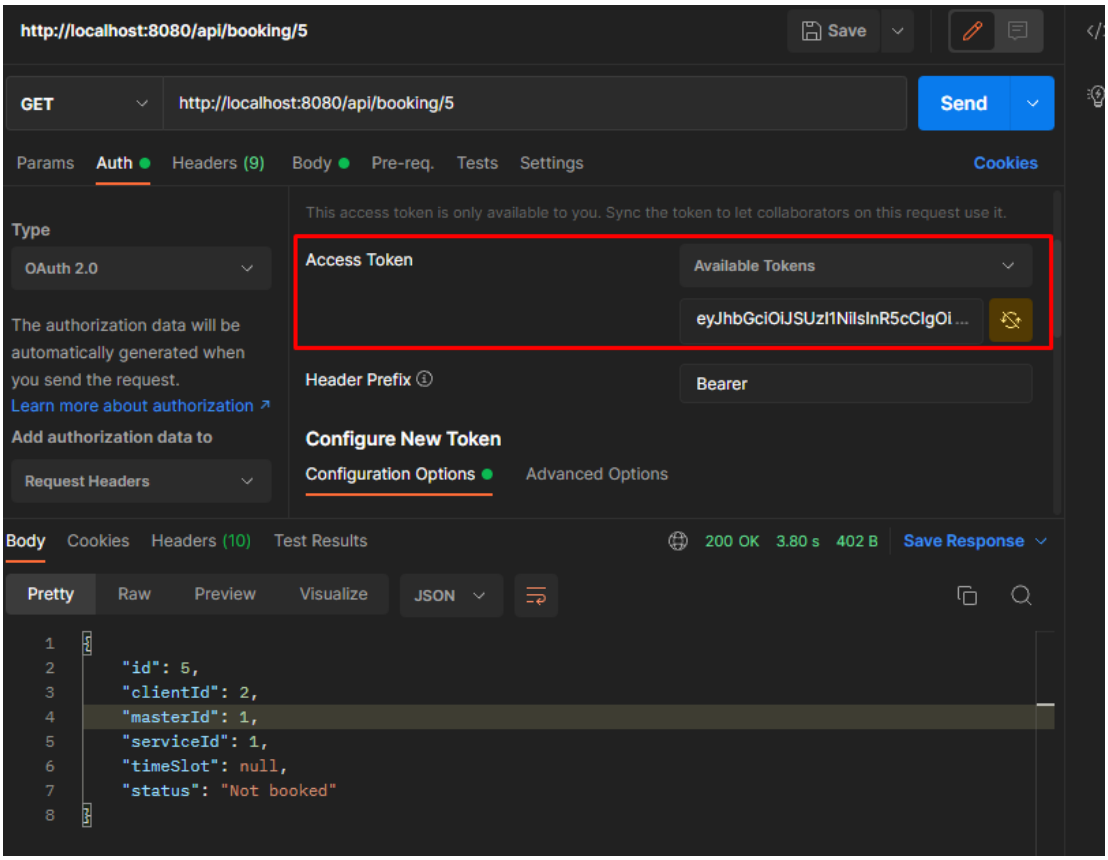

Рисунок 8.8. — Успішна авторизація та виконання GET запиту

Як бачимо, використавши згенерований токен надав можливість зробити успішну авторизацію до запиту.

Висновок до розділу 8

Висновком даного розділу є те, що безпека інформаційного простору є надзвичайно важливим аспектом в сфері інформаційної безпеки та розробки програмного забезпечення. Тому, ми не лише ознайомилися з платформою Keyclock, а на прикладі показали як успішно інтегрувати її до нашої системи.

Підключивши систему Keyclock в нас з’явилася змога налаштувати доступ користувачі до відповідних даних цим самим гарантувати безпеку користувачам та власній системі.

9 РОЗПОДІЛЕНА СИСТЕМА ТРАСУВАННЯ

Наразі, розробка розподілених систем стає все більш поширеною і складною задачею. З введенням мікросервісної архітектури, де система складається з набору незалежних сервісів, виникає потреба у зручному способі трасування запитів та аналізу пропускної здатності системи. Відстеження шляху запитів у розподілених середовищах являє собою важливим інструментом для виявлення проблем та просте усунення їх.

Розподілене відстеження — це сучасна діагностична техніка, яка дозволяє розробникам відстежувати запити через розподілені служби, програми та бази даних. Це дозволяє групам розробників аналізувати, контролювати та усувати неполадки розподілених програм.

Відстеження починається із запиту користувача в точці входу програми. Коли запит проходить через систему, він генерує траси, які відстежують усі дії обробки, які виконуються компонентами системи за запитом.

Кожен трек має унікальний ідентифікатор і охоплює певний діапазон. Ця область представляє операцію або окрему одиницю роботи, яка викликається як частина виконання запиту користувача. Слоти містять таку інформацію, як унікальні ідентифікатори, імена, метадані та позначки часу, які корисні для аналізу та налагодження.

Відстежуючи наскрізний потік запитів, команди програмного забезпечення можуть порівнювати ефективні трасування з трасуваннями винятків. Бачачи відмінності в часі, структурі та поведінці, розробники економлять гроші та можуть швидше усувати та вирішувати системні проблеми.

Отже на ринку є багато розподілених інструментів трасування, я вирішив використовувати таку систему як Zipkin.

9.1 Система трасування Zipkin

Zipkin — це розподілена система трасування з відкритим вихідним кодом, яка допомагає розробникам відстежувати та усувати неполадки програм в архітектурі мікросервісів. Він надає уявлення про затримку та залежності між різними компонентами розподіленої системи, дозволяючи виявляти вузькі місця продуктивності та діагностувати проблеми.

Насправді Zipkin має дуже багато простого та ефективного функціонала. У ньому зроблена зручна та проста візуалізація і контекст трасування. Zipkin відображає трасування у вигляді графіка, де показується послідовність викликів сервісів та час, необхідний для кожної операції. Ця графічна візуалізація допомагає розуміти трафік та затримки в системі. Кожному запиту Zipkin присвоює унікальний ідентифікатор трасування та розподіляє його між сервісами за допомогою спеціальних заголовків. Цей контекст трасування дозволяє Zipkin об'єднувати окремі етапи (окремі операції) для створення повного трасування.

Також, інструмент Zipkin забезпечує повноцінне трасування в розподіленій системі за допомогою наступних компонентів. По-перше, для збору даних про трасування додатки повинні бути інструментовані бібліотеками або фреймворками трасування, і в цьому випадку Zipkin надає клієнтські бібліотеки для популярних мов програмування. Це спрощує процес інтеграції з різними службами. По-друге, для зберігання даних трасування Zipkin підтримує різні системи зберігання, такі як Elasticsearch, MySQL, Cassandra тощо, що дозволяє зберігати дані трасування для аналізу та виконання детального вивчення причин. Нарешті, Zipkin може інтегруватися з системами моніторингу, такими як Prometheus, Grafana та Micrometer, що дозволяє виконувати кореляцію даних трасування з іншими метриками, зібраними системою моніторингу. Всі ці компоненти забезпечують ефективне збирання, зберігання та аналіз даних трасування у розподіленій системі.

В цілому Zipkin відіграє важливу роль у розумінні поведінки та продуктивності розподілених систем, надаючи розуміння взаємодії між службами. Допомагає виявляти неполадки, змінювати та підтримувати надійність складних архітектур мікросервісів.

9.2 Застосування системи трасування

Для того щоб впровадити Zipkin у нашу систему розкладу роботи співробітників барбершопу та замовлень послуг необхідно виконати такі дії як:

- Додавання залежності;
- Налаштування серверу Zipkin;
- Перегляд та аналіз даних.

Тому, дивлячись на перший пункт, нам потрібно додати необхідні бібліотеки, тобто залежності у код кожного сервісу, щоб була можливість використовувати цю систему для подальшої взаємодії з ним. Для того щоб це зробити потрібно інтегрувати залежності Maven (dependency) заснованого на Spring Boot, з метою забезпечення моніторингу та трасування додатку, саме це зображено на рис. 9.1.

```
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-actuator</artifactId>
</dependency>
<dependency>
    <groupId>io.micrometer</groupId>
    <artifactId>micrometer-tracing-bridge-brave</artifactId>
</dependency>
<dependency>
    <groupId>io.zipkin.reporter2</groupId>
    <artifactId>zipkin-reporter-brave</artifactId>
</dependency>
```

Рисунок 9.1 — залежності для використання Zipkin трасування

Залежність org.springframework.boot:spring-boot-starter-actuator — ця залежність додає модуль Actuator до вашого проекту Spring Boot. Actuator надає різноманітну інформацію про виконання додатку, включаючи здоров'я додатку, метрики, інформацію про потоки та інше. Це дозволяє здійснювати моніторинг та адміністрування додатку.

Наступна залежність io.micrometer:micrometer-tracing-bridge-brave — вона додає мікрометр, що є бібліотекою метрик для Java додатків, та містить мости для інтеграції з різними інструментами трасування. У конкретному випадку, ця залежність використовує місто-місто-брав для інтеграції з інструментом трасування Brave.

Кінцева залежність io.zipkin.reporter2:zipkin-reporter-brave — залежність яка додає Zipkin Reporter для Brave. Вона забезпечує зв'язок між Brave та Zipkin, що дозволяє передавати дані трасування до Zipkin сервера.

Наступним кроком буде налаштування серверу Zipkin. Потрібно встановити та налаштувати Zipkin сервер для зберігання та аналізу даних трасування. Конфігурація для підключення системи до Zipkin сервера для збору та трасування даних про виконання розподілених служб зображена на рис. 9.2.

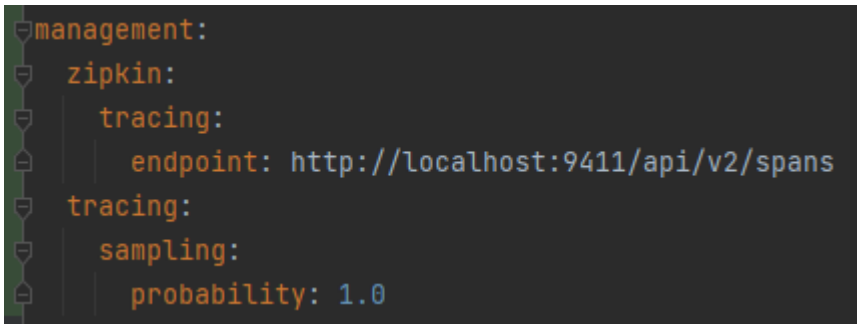

Рисунок 9.2 — Налаштування application.yml файлу для сервера Zipkin

Розглянемо цей блок коду. Розділ management використовується для налаштування різних аспектів управління вашим додатком. Підрозділ конфігурації zipkin визначає налаштування для підключення до Zipkin сервера. Цей підрозділ tracing визначає налаштування трасування даних. Конфігурація endpoint: http://localhost:9411/api/v2/spans - це параметр конфігурації, який вказує URL-адресу кінцевої точки на Zipkin сервері, де додаток буде надсилати

дані про трасування. Це також підрозділ конфігурації sampling, який визначає налаштування для вибіркового відстеження. У даному випадку, probability: 1.0 означає, що кожен запит буде відстежуватись і надсилатись до Zipkin сервера.

Отже, ця конфігурація дозволяє додатку збирати дані про трасування та надсилати їх до Zipkin сервера.

Кінцевим етапом буде перегляд коректності підключення серверу Zipkin до нашої системи задля впевненості, що виникла змога переглядати та аналізувати дані. Саме використовуючи веб-інтерфейс Zipkin, є можливість переглядати та аналізувати дані трасування запитів. Це дозволить виявляти проблеми та вдосконалювати продуктивність системи. Для того щоб побачити якийсь результат спочатку зробимо GET запит через Postman. Запит зображений на рис. 9.3.

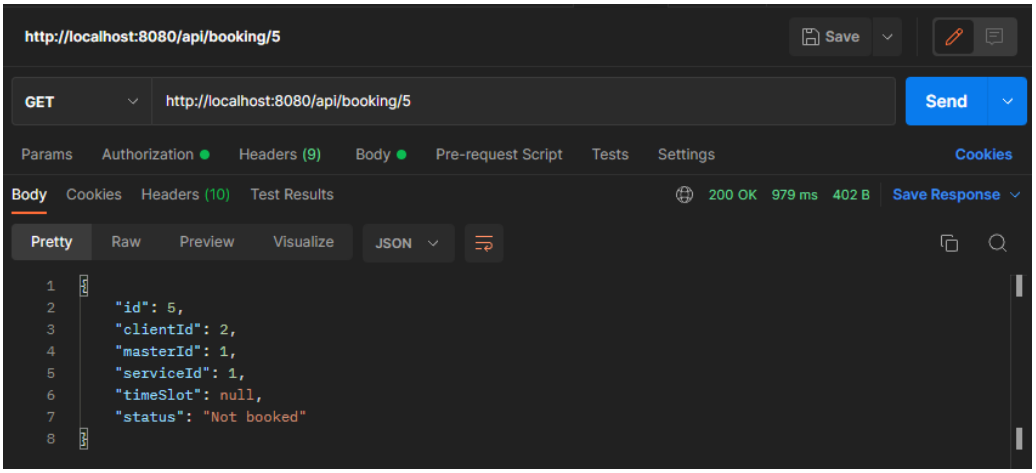

Рисунок 9.3 — Успішний GET запит до мікросервіса "Booking service"

На цьому рисунку було зроблено GET запит по протоколу HTTP. Посилання http://localhost:8080/api/booking/5 вказує на те, що у нас є API Gateway який переправляє маршрут до Booking service і звертаємося до об’єкта під id = 5. Бачимо що наш запит пройшов успішно, підтвердженням цього є HTTP статус «200 OK» та отримання JSON формат даних.

Після цього перейдемо до сервера Zipkin за наступним посиланням http://localhost:9411/. На наступному рисунку зображено веб-інтерфейс Zipkin,

через який маємо можливість переглядати та аналізувати дані трасування запитів, рис. 9.4.

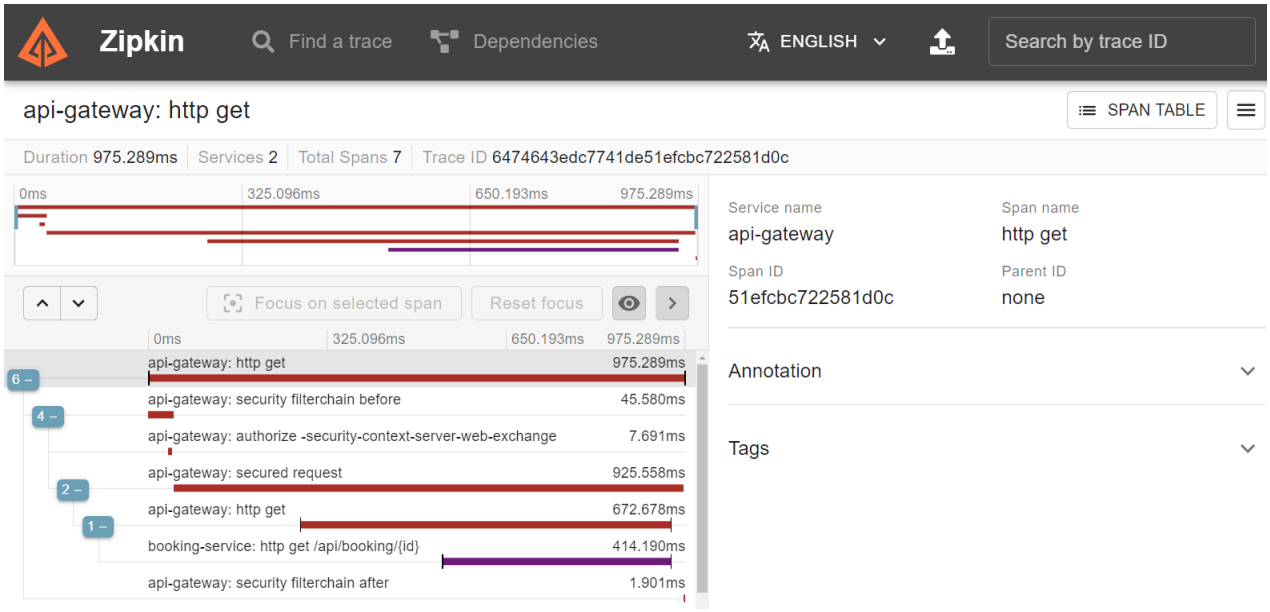

Рисунок 9.4 — Вебінтерфейс Zipkin по GET запиту

Давайте проаналізуємо що зображено на даному рисунку. Видно веб-інтерфейс Zipkin у якому графічне представлення трасування є у вигляді спрямованого графа. Граф відображає послідовність викликів сервісів та час, який вимагає кожна операція. Ця візуалізація допомагає зрозуміти шлях запиту, включаючи взаємодію між API Gateway та сервісом Booking, а також відображає затримки в системі. Відповідно, відображено як одна злита послідовність викликів сервісів, де кожен виклик представляє собою окремий, так званий проміжок (span). У даному випадку присутні 7 інтервалів. Саме ці окремі проміжки мають унікальний ідентифікатор трасування, який пов'язаний з API Gateway та сервісом Booking. Завдяки цим ідентифікаторам легко можна виявити, де саме виникає проблема.

Використовуючи мікросервісну архітектуру у системі, дуже складно зрозуміти звідки саме пішла та чи інша проблема, під час виконання запиту. Тому, підключивши Zipkin, виникає можливість використовувати так звану технологію «розподілення трасування». Це означає, що кожен запит має свій

унікальний код ідентифікації. Тому, якщо обравши для прикладу запит GET, який проходить через API Gateway до мікросервіса Booking, буде отримано унікальний «traceId». Проте, кожен проміжок «Trace» розділений на інтервали, тобто етапи які місять собі потрібну інформацію яка була надана їм під час виконання запиту. Саме ці інтервали, «Span», дають можливість виявити на якому етапі виникла помилка у такій архітектура як мікросервісна. Простим прикладом, як працює розподілення трасування показано на наступному рис. 9.5.

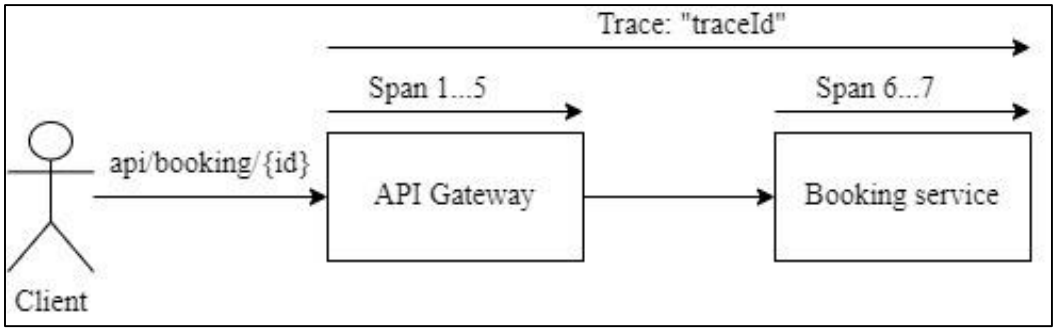

Рисунок 9.5 — Схематична візуалізація процесу розподіленого трасування

Також можна відкрити панель «Annotation» тобто анотації які пов'язані з кожною операцією або подією, що відбувається під час трасування запиту. Ці анотації надають додаткову інформацію про різні аспекти виконання операцій, Вони допомагають визначити час виконання кожного кроку процесу та знайти можливі проблеми або затримки, що виникають під час взаємодії з різними сервісами або компонентами системи. Панель «Annotation» можна побачити на рис. 9.6.

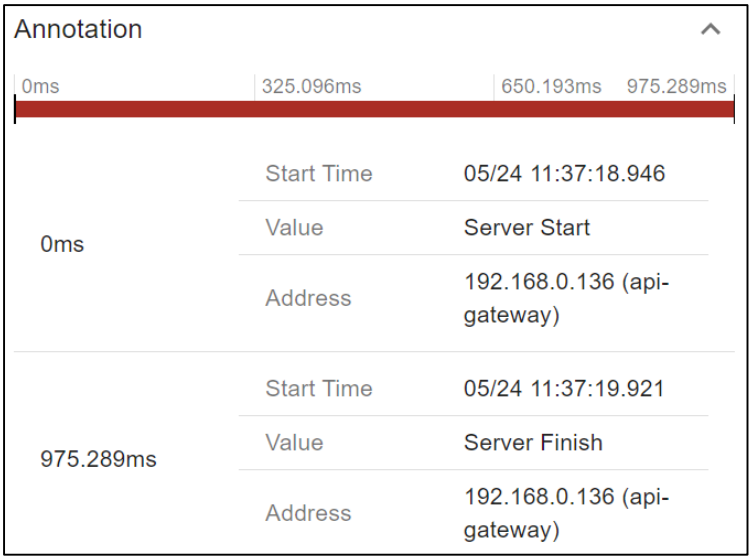

Рисунок 9.6 — Zipkin анотація до нашого GET запиту

Рисунок показує таку інформацію як Timestamp (Відмітка часу), тобто часова мітка, яка вказує точний час виконання певної події або операції. Також присутнє значення Address (Хост), який відображає інформацію про хост, на якому відбувається виконання операції, така як IP-адреса або ім'я хоста. Ну і ще одним параметром являється Value (Значення), воно вказує на конкретне значення анотації, яке може бути пов'язане з певною дією або станом операції.

Ще однією можливістю у інтерфейсі Zipkin можна переглянути «Tags», тобто теги які надають додаткову контекстну інформацію про операції, що відбуваються, рис. 9.7.

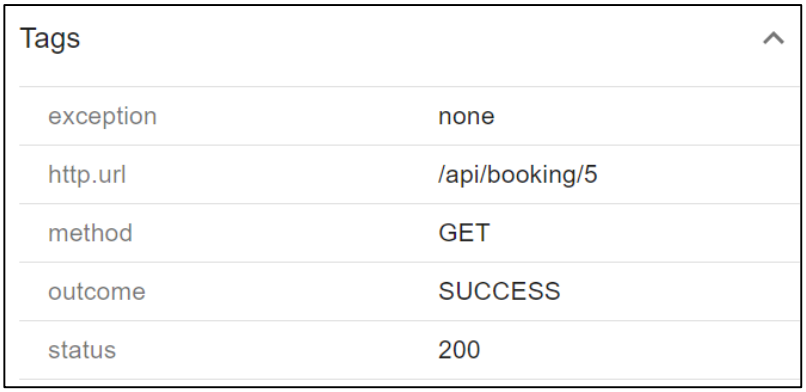

Рисунок 9.7 — Теги при GET запиті

Розберемо що саме означають ці теги з параметрами, які показані на даному рисунку. Даний тег exception вказує, чи сталася помилка або виняток під час виконання операції. У даному випадку значення "none" означає, що жодного винятку не відбулося. Цей тег http.url містить URL-адресу запиту. У даному випадку значення "/api/booking/5" вказує на шлях запиту до ресурсу з ідентифікатором "5". Наступний тег, це method, він показує метод, який був використаний у запиті. У даному випадку значення "GET" означає, що запит був виконаний методом GET (отримання даних). Тег outcome вказує на результат виконання операції. У даному випадку значення "SUCCESS" означає, що операція була успішно виконана. Кінцевий тег status який вказує на HTTP-статус відповіді на запит. У даному випадку значення "200" означає, що запит був успішно оброблений і повернув статус "200 OK".

Загалом Zipkin відіграє важливу роль у розумінні поведінки та продуктивності розподілених систем, надаючи розуміння взаємодії між службами. Це допомагає у вирішенні проблем, покращення та підтримці надійності складних архітектур мікросервісів.

Висновок до розділу 9

В цьому розділі було розглянуто такий термін як розподілене трасування. Зробили висновок, що у мікросервісній архітектурі важко відслідковувати звідки саме пішла та чи інша помилка. Тому на допомогу приходить така розподілена система трасування як Zipkin. Показали як підключити саме цю систему до нашого проекту та навели і описали рисунки в яких видно коректні результати роботи Zipkin. Також, схематично візуалізували процес роботи розподіленого трасування, який має загальний проміжок, і в свою чергу поділяється на деякі інтервали які місить власну інформацію здійсненого запиту. Саме ця інформація дає можливість виявити проблему у системі.

10 КРОСС-БРАУЗЕРНА СУМІСНІСТЬ

Кросс-браузерна сумісність є важливим аспектом розробки веб-додатків. Завдяки різним браузерам і версіям браузерів, веб-розробникам потрібно враховувати різні реалізації HTML, CSS та JavaScript для забезпечення належного відображення та функціональності додатків на різних платформа.

Усе розвивається, тому фронтенд технології не виключення. Адже фронтенд технології відіграють важливу роль у розробці сучасних веб-додатків. Вони надають розробникам створювати ефективні та інтуїтивно зрозумілі інтерфейси для користувача.

Основою, без чого не можна обійтися при розробці фронтенда це — HTML (HyperText Markup Language), CSS (Cascading Style Sheets) та JavaScript мови. HTML використовується для створення структури та контенту веб-сторінок, CSS відповідає за візуалізацію та вигляд сторінок, а JavaScript додає інтерактивність та функціональність до веб-додатків.

В свою чергу мова програмування JavaScript має достатньо фреймворків для фронтенд розробки, такі як React, Angular, Vue.js., проте у системі будемо використовувати саме React.

Фреймворк React є одним з найпопулярніших фреймворків для розробки користувацького інтерфейсу (UI) у веб-додатках. React використовує компонентний підхід у процесі розробки інтерфейсу. Він дає можливість розбити складний інтерфейс на менші, самодостатні компоненти, які легко управляти та повторно використовувати. Це спрощує структуру коду, розширення та підтримку додатків. Крім того він використовує віртуальний DOM, який дозволяє оптимізувати процес оновлення інтерфейсу. Цей підхід розрахований на ефективну роботу зі змінами, що відбуваються на стороні користувача, та автоматично оновлює лише необхідні елементи інтерфейсу, забезпечуючи більшу продуктивність.

Щоб кросс-браузерна сумісність коректно працювала, потрібно надати доступ фронтенду звертатися до ресурсів на інших доменах, тобто до серверної частини. У браузері присутня така політика як CORS (Cross-Origin Resource Sharing). Це так звана політика безпеки браузера, яка обмежує доступ до ресурсів на інших сторонах. Щоб вирішити цю проблему потрібно налаштувати CORS у сервері для обробки запитів з інших джерел. В сервісі API Gateway створимо конфігурацію для налаштування CORS. Це налаштування можна побачити на рис 10.1.

```java
@Configuration
public class CorsConfig {

    private static final String METHODS = "GET, POST, PUT, DELETE, OPTIONS";
    private static final String ORIGINS = "*";
    private static final String HEADERS = "Content-Type, Authorization";

    @Bean
    public CorsWebFilter corsWebFilter() {
        CorsConfiguration corsConfig = new CorsConfiguration();
        corsConfig.setAllowedHeaders(Arrays.asList(HEADERS.split( regex: ", ")));
        corsConfig.setAllowedMethods(Arrays.asList(METHODS.split( regex: ", ")));
        corsConfig.addAllowedOriginPattern(ORIGINS);
        corsConfig.setAllowCredentials(true);

        UrlBasedCorsConfigurationSource source = new UrlBasedCorsConfigurationSource();
        source.registerCorsConfiguration( path: "/**", corsConfig);

        return new CorsWebFilter(source);
    }
}
```

Рисунок 10.1 — Налаштування політики CORS у серверній частині

Висновок до розділу 10

У даному розділі розглядається кросс-браузерна сумісність як важливий аспект у розробці фронтенда системи. У даній системі для розробки фронтенду було обрано фреймворк React який є одним з популярних у розробці веб-додатків. Зробили необхідні налаштування для дозволу фронтенда отримувати доступ до ресурсів на інших доменах, в нашому випадку це серверної частини системи

ВИСНОВОК

В даному дипломному проєкті було проведено дослідження та розробку системи «Розклад роботи співробітників барбершопу та замовлень послуг». На основі аналізу існуючих рішень були вибрані технології, такі як мова програмування Java, технології Spring Framework та Spring Cloud, які використовувалися у розробці проекту. Було розглянуто архітектурні та структурні особливості проекту, включаючи мікросервісну та монолітну архітектури. Проте, саме завдяки використанню мікросервісного підходу при проектуванні, наша серверна частина системи стала надійною, безпечною та зручно підтримуваною.

Також було досліджено базу даних і обґрунтовано вибір інструментів для реалізації проекту. Згодом були розроблені окремі мікросервіси, включаючи Discovery server, Schedule service, Booking service, Feedback service, User service та Catalog service, а також API Gateway для керування комунікацією між сервісами.

Питання безпеки авторизації були вирішені за допомогою системи Keycloak та методу авторизації OAuth 2.0. Було забезпечено авторизацію через Keycloak для забезпечення безпеки системи.

Для трасування розподіленої системи була використана система трасування Zipkin, що дозволяє відстежувати та аналізувати комунікацію між мікросервісами.

Також була розглянута кросс-браузерна сумісність, що є важливим аспектом розробки веб-додатків. Було наголошено на використанні фреймворку React для розробки користувацького інтерфейсу, який дозволяє створювати ефективні та інтуїтивно зрозумілі інтерфейси для користувача.

Отже, результатом проведення досліджень і розробки є створена розподілена система, яка використовує сучасні технології і забезпечує безпеку, масштабованість та ефективність у роботі.

ПЕРЕЛІК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. The Best Booking System for Your Needs / Amie Parnaby / 2021 [Електронний ресурс]: Доступно: https://news.simplybook.me/the-best-booking-system-for-your-needs/
2. Why is Java so popular for developers and programmers? / [Електронний ресурс]: Доступно: https://www.frgconsulting.com/insights/why-is-java-so-popular-developers
3. Spring Framework Documentation. [Електронний ресурс]: Доступно: https://docs.spring.io/spring-framework/docs/current/reference/
4. Spring Cloud Documentation. [Електронний ресурс]: Доступно: https://spring.io/projects/spring-cloud#overview
5. "Microservices: Flexible Software Architecture." / Eberhard Wolff / Addison-Wesley Professional, 2016.
6. What Is PostgreSQL? / 2023 [Електронний ресурс]: Доступно: https://kinsta.com/knowledgebase/what-is-postgresql/
7. Spring Cloud API Gateway With An Example / Akshit Kumar / 2022 [Електронний ресурс]: Доступно: https://blog.knoldus.com/spring-cloud-api-gateway/
8. Introduction to Spring Cloud Netflix – Eureka / baelbung / 2022 [Електронний ресурс]: Доступно: https://www.baeldung.com/spring-cloud-netflix-eureka
9. "Розробка Discovery сервера для мікросервісної архітектури." / Жигунов, Д.В., & Козуб, І.В. / Вісник Черкаського державного технологічного університету. Серія: Технічні науки, № 1(79), 2021.
10. RESTful Web APIs: Services for a Changing World / Richardson, L., & Amundsen, M. / O'Reilly Media. / 2018

11. Keycloak Documentation. [Електронний ресурс]: Доступно: https://www.keycloak.org/documentation
12. Spring Cloud Gateway Keycloak OAuth2 OIDC Integration / Amrut Prabhu / 2021 [Електронний ресурс]: Доступно: https://refactorfirst.com/spring-cloud-gateway-keycloak-oauth2-openid-connect
13. Zipkin Documentation. [Електронний ресурс]: Доступно: https://zipkin.io/pages/quickstart
14. Distributed Tracing with Spring Cloud Sleuth and Spring Cloud Zipkin / Josh Long / 2016 [Електронний ресурс]: Доступно: https://spring.io/blog/2016/02/15/distributed-tracing-with-spring-cloud-sleuth-and-spring-cloud-zipkin
15. Complete Guide on Front End Web Development with React / Prateek Singh / 2023 [Електронний ресурс]: Доступно: https://www.knowledgehut.com/blog/web-development/front-end-web-development-with-react