الجمهورية العربية السورية
وزارة التعليم العالي والبحث العلمي
جامعة دمشق

البرمجة الموزعة

نظام تعليم إلكتروني باستخدام معمارية الخدمات المصغرة (Microservices)

الوظيفة الثانية

أسماء طلاب:

- **سمر احمد علوش**
- **صفا بسام خليفة**
- **فاطمة وهيب شاكر**
- **دعاء رفيق البغدادي**
- **حمزة سلمان درويش**

جدول المحتويات

المحتويات

1. الملخص التنفيذي(Abstract)

يهدف هذا المشروع إلى تصميم وتنفيذ نظام تعليم إلكتروني متكامل وقابل للتوسع، بالاعتماد على معمارية الخدمات المصغرة .(Microservices) تم تقسيم النظام إلى مجموعة من الخدمات المستقلة والقابلة للنشر بشكل منفصل، مما يعزز مرونة التطوير والصيانة ويسهل عملية التوسع الأفقي.

يستعرض هذا التقرير مراحل بناء النظام الثلاث، بدءًا من إنشاء الخدمات الأساسية (خدمة المستخدمين وخدمة الدورات)، مرورًا بتأسيس البنية التحتية اللازمة التي تشمل خادم اكتشاف الخدمات (Eureka) وبوابة الدخول الموحدة(API Gateway) ، وانتهاءً بالمرحلة المتقدمة التي ركزت على تفعيل التواصل الديناميكي بين الخدمات باستخدام(OpenFeign) ، وتطبيق آليات الحماية من الفشل مثل قاطع الدائرة (Circuit Breaker) باستخدام(Resilience4j) ، وأخيرًا التحقق من آلية توزيع الحمل.

تم التحقق من كل مرحلة من مراحل المشروع من خلال اختبارات عملية باستخدام أداة Postman، ويقدم هذا التقرير شواهد مرئية توثق نجاح تنفيذ المتطلبات الوظيفية وغير الوظيفية، مما يثبت بناء نظام موزع فعال وموثوق.

---

2. المقدمة

2.1. نظرة عامة على المشروع

يتمحور المشروع حول تطوير منصة تعليم إلكتروني تهدف إلى إدارة المستخدمين (مسؤولين، مدربين، متعلمين)، ونشر الدورات التعليمية، وإدارة عمليات الاشتراك والتقييم. يتطلب النظام قدرة المسؤول على إدارة المدربين، وقدرة المدربين على إنشاء وإدارة الدورات، مع وجود آلية موافقة مركزية من المسؤول. كما يجب أن يتمكن المتعلمون من التسجيل والاشتراك في الدورات واجتياز الاختبارات.

2.2. المشكلة والحل المقترح: لماذا معمارية الخدمات المصغرة؟

إن بناء مثل هذا النظام باستخدام المعمارية التقليدية المتجانسة (Monolithic Architecture) قد يؤدي إلى تحديات كبيرة في المستقبل، مثل صعوبة التطوير، وبطء عملية النشر، وتعقيد الصيانة، وتأثير أي خطأ على النظام بأكمله.

لذلك، تم اعتماد **معمارية الخدمات المصغرة (Microservices)** كحل استراتيجي لهذه التحديات. تقوم هذه المعمارية على تفكيك التطبيق إلى مجموعة من الخدمات الصغيرة والمستقلة، حيث تكون كل خدمة مسؤولة عن جزء محدد من وظائف النظام، ولها قاعدة بياناتها الخاصة، ويمكن تطويرها ونشرها وتوسيعها بشكل مستقل عن الخدمات الأخرى. هذا النهج يوفر مرونة عالية، ويعزز موثوقية النظام، ويسهل على الفرق العمل على أجزاء مختلفة من النظام في وقت واحد.

2.3. أهداف التقرير وهيكليته

يهدف هذا التقرير إلى تقديم توثيق فني شامل ومفصل لجميع مراحل تصميم وتنفيذ المشروع، مع التركيز على:

- شرح القرارات الهندسية التي تم اتخاذها والأساس المنطقي وراءها.
- عرض تفاصيل التنفيذ التقني لكل مكون من مكونات النظام.
- تقديم أدلة عملية وشواهد مرئية (Screenshots) تثبت عمل النظام كما هو متوقع.
- مناقشة التحديات التي تمت مواجهتها وكيفية التغلب عليها.

---

3. تصميم المعمارية والمبادئ الهندسية

3.1. مخطط المعمارية المقترحة

تتكون بنية النظام من عدة مكونات رئيسية تتفاعل مع بعضها البعض لتحقيق الأهداف الوظيفية. يمكن تصور تدفق الطلبات على النحو التالي:

1. **العميل :(Client)** يرسل جميع طلباته إلى نقطة دخول واحدة فقط، وهي بوابة واجهة برمجة التطبيقات.(API Gateway)
2. **بوابة :API Gateway** تستقبل الطلب وتتحقق منه، ثم تقوم بتوجيهه إلى الخدمة الداخلية المناسبة بناءً على مسار الطلب.
3. **خادم :Eureka** تعمل البوابة والخدمات على استشارته بشكل مستمر لمعرفة العناوين الديناميكية للخدمات الأخرى.
4. **الخدمات الداخلية :(Microservices)** تتواصل مع بعضها عند الحاجة ,.e.g) Course-Service يستدعي (User-Service لإكمال الطلب، ثم تعيد الاستجابة إلى البوابة، التي بدورها تعيدها إلى العميل.

3.2. وصف المكونات الأساسية

- **User-Service:** خدمة مسؤولة عن كل ما يتعلق بالمستخدمين: الإنشاء، التعديل، الحذف، المصادقة، وتحديد الصلاحيات.(Admin, Instructor, Learner)
- **Course-Service:** خدمة مسؤولة عن إدارة الدورات: الإنشاء، عرض الدورات، إدارة المحتوى، وآلية الموافقة على الدورات.
- **Eureka Server (Service Discovery):** يعمل كـ "دليل هاتف" للنظام. كل خدمة عند بدئها تسجل نفسها وعنوانها فيه، مما يسمح للخدمات الأخرى بالعثور عليها بالاسم بدلاً من عنوان IP ثابت.
- **API Gateway:** الواجهة الأمامية للنظام. يعمل كبروكسي عكسي (Reverse Proxy) لجميع الطلبات، ويوفر نقطة مركزية لتطبيق الأمن، وتجميع الاستجابات، وتوزيع الحمل.

3.3. المبادئ الهندسية المتبعة

- **خدمة واحدة، مسؤولية واحدة (Single Responsibility Principle)**: كل خدمة مصغرة لها مسؤولية واضحة ومحددة.
- **قاعدة بيانات لكل خدمة (Database Per Service)**: لضمان الاستقلالية التامة، كل خدمة تمتلك قاعدة بياناتها الخاصة والمعزولة، مما يمنع التبعيات المباشرة على مستوى البيانات.
- **التواصل عبر APIs**: تتواصل الخدمات مع بعضها البعض حصريًا عبر واجهات برمجة التطبيقات (APIs) المحددة جيدًا، وليس عبر الوصول المباشر لقواعد البيانات.
- **اللامركزية (Decentralization)**: لامركزية في إدارة البيانات والحوكمة، مما يمنح كل خدمة حرية اختيار التكنولوجيا الأنسب لها) على الرغم من أننا استخدمنا Spring Bootللتوحيد في هذا المشروع.(

---

4. مراحل التنفيذ التفصيلية

تم تقسيم عملية بناء النظام إلى ثلاث مراحل منطقية لضمان التدرج والتحقق المستمر.

4.1. المرحلة الأولى: بناء الخدمات الأساسية (Core Services)

في هذه المرحلة، تم التركيز على بناء منطق العمل الأساسي لكل خدمة بشكل منفصل ومعزول.

- **خدمة المستخدمين(User-Service):**
  - **الكيانات (Entities)**: تم إنشاء `User.java`لتخزين معلومات المستخدمين (ID, name, email, password) و `Role.java`وهو من نوع `Enum`لتحديد الصلاحيات بشكل ثابت وآمن (`ADMIN, INSTRUCTOR, STUDENT`).
  - **المستودع (Repository)**: تم استخدام `Spring Data JPA`لإنشاء واجهة `UserRepository`التي توفر عمليات CRUD القياسية دون الحاجة لكتابة استعلامات.SQL
  - **المتحكم (Controller)**: تم إنشاء `UserController`الذي يوفر نقاط نهاية (Endpoints) من نوع RESTful API للتعامل مع المستخدمين، مثل `POST /api/users`لإنشاء مستخدم جديد و `GET /api/users`لجلب قائمة المستخدمين.
- **خدمة الدورات:(Course-Service)**
  - **الكيانات (Entities)**: تم إنشاء `Course.java`لتخزين بيانات الدورة (ID, title, description, price)مع حقل مهم وهو `instructorId`الذي سيمثل لاحقًا الرابط المنطقي مع -User

Service. كما تم إنشاء CourseStatus.java من نوع Enum لتتبع حالة الدورة (PENDING_APPROVAL, APPROVED, REJECTED).

- **المستودع (Repository)**: تم إنشاء CourseRepository بنفس الطريقة باستخدام Spring Data JPA.
- **المتحكم (Controller)**: تم إنشاء CourseController بنقاط نهاية لإدارة الدورات مثل POST /api/courses لإنشاء دورة جديدة و PUT /api/courses/{id}/approve للموافقة عليها.

التحقق في هذه المرحلة: تم اختبار كل خدمة على حدة باستخدام Postman من خلال إرسال طلبات مباشرة إلى منافذها (e.g., localhost:8081, localhost:8082) للتأكد من أن منطق العمل الأساسي صحيح.

4.2. المرحلة الثانية: تأسيس البنية التحتية (Infrastructure)

بعد بناء الخدمات، حان الوقت لإنشاء البنية التحتية التي ستمكنها من العمل كنظام موزع.

- **خادم اكتشاف الخدمات (Eureka Server):**
  - **التنفيذ**: تم إنشاء مشروع Spring Boot مستقل (eureka-server).
  - **الإعدادات**: تم إضافة اعتمادية spring-cloud-starter-netflix-eureka-server وتفعيل الخادم باستخدام @EnableEurekaServer. تم تعديل ملف application.properties لمنع الخادم من تسجيل نفسه كعميل، وتم تشغيله على المنفذ القياسي 8761.
  - **الربط**: تم تعديل ملفات الإعدادات في User-Service و Course-Service لإضافة اعتمادية eureka-client وتحديد عنوان خادم Eureka، مما مكنهما من تسجيل نفسيهما تلقائيًا عند بدء التشغيل.

- **بوابة واجهة برمجة التطبيقات (API Gateway):**
  - **التنفيذ**: تم إنشاء مشروع Spring Boot مستقل (api-gateway) باستخدام Spring Cloud Gateway.
  - **الإعدادات**: تم تعريف "قواعد التوجيه (Routing Rules)" في ملف application.yml. هذه القواعد تحدد كيفية توجيه الطلبات الواردة إلى الخدمات الداخلية بناءً على مسار الطلب. على سبيل المثال:

```
# A snippet from api-gateway's application.yml
spring:
  cloud:
    gateway:
      routes:
        - id: user-service
          uri: lb://user-service # 'lb' stands for Load Balancer
          predicates:
            - Path=/api/users/**
        - id: course-service
          uri: lb://course-service
          predicates:
            - Path=/api/courses/**
```

البروتوكول `lb://` يأمر البوابة بالبحث عن عنوان الخدمة في Eureka بدلاً من استخدام عنوان ثابت، وهذا هو جوهر العنونة الديناميكية.

التحقق في هذه المرحلة :تم تشغيل جميع المكونات بالترتيب (Eureka > Services > Gateway)والتأكد من تسجيل الخدمات في لوحة تحكمEureka ، ثم تم توجيه الطلبات عبر البوابة (`localhost:8080`) والتأكد من وصولها إلى الخدمات الصحيحة.

---

5. المرحلة الثالثة: الربط، التواصل، والحماية من الفشل (التنفيذ المتقدم)

هذه هي المرحلة التي يتم فيها تحويل المكونات المنفصلة إلى نظام حي ومترابط.

5.1. تفعيل التواصل الديناميكي بين الخدمات باستخدام OpenFeign

كان من الضروري أن تتمكن `Course-Service`من استدعاء `User-Service`للتحقق من وجود المدرب قبل إنشاء دورة جديدة. تم اختيار `OpenFeign`لهذه المهمة لأنه يبسط عملية استدعاء خدمات REST ويجعلها تبدو وكأنها استدعاء لدالة Java محلية.

خطوات التنفيذ في course-service:

1. **إضافة الاعتمادية**: تم إضافة spring-cloud-starter-openfeign إلى ملف pom.xml.
2. **تفعيل الميزة**: تم إضافة التعليق @EnableFeignClients إلى كلاس التطبيق الرئيسي.
3. **إنشاء واجهة العميل**: تم إنشاء واجهة UserClient التي تُعرّف شكل الطلب الذي سيتم إرساله إلى user-service.

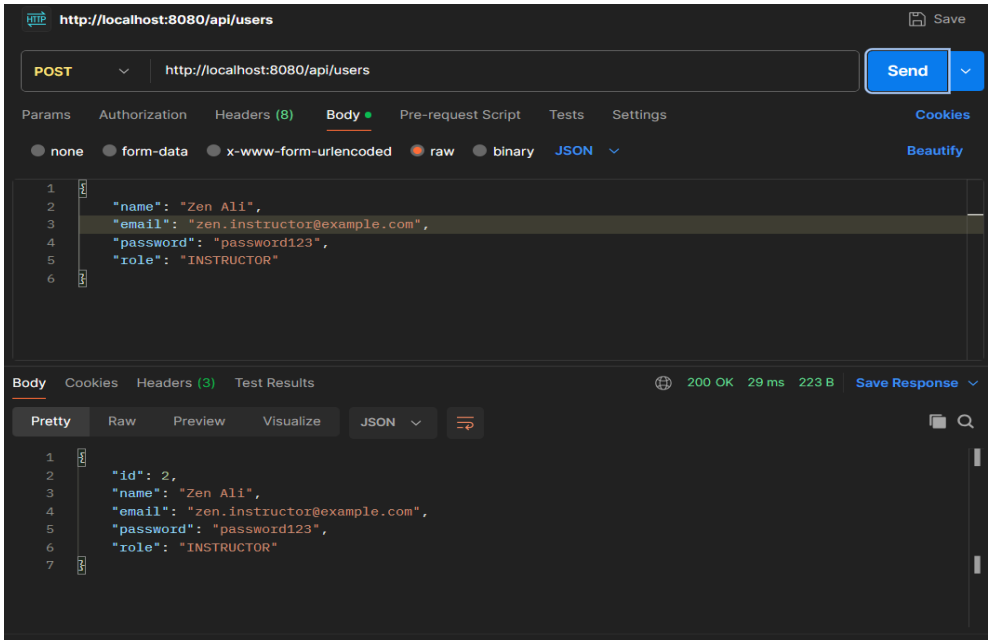
```
// UserClient.java in course-service
@FeignClient(name = "user-service")
public interface UserClient {
    @GetMapping("/api/users/{id}")
    UserDTO  findUserById(@PathVariable("id")
Long id);
}
```

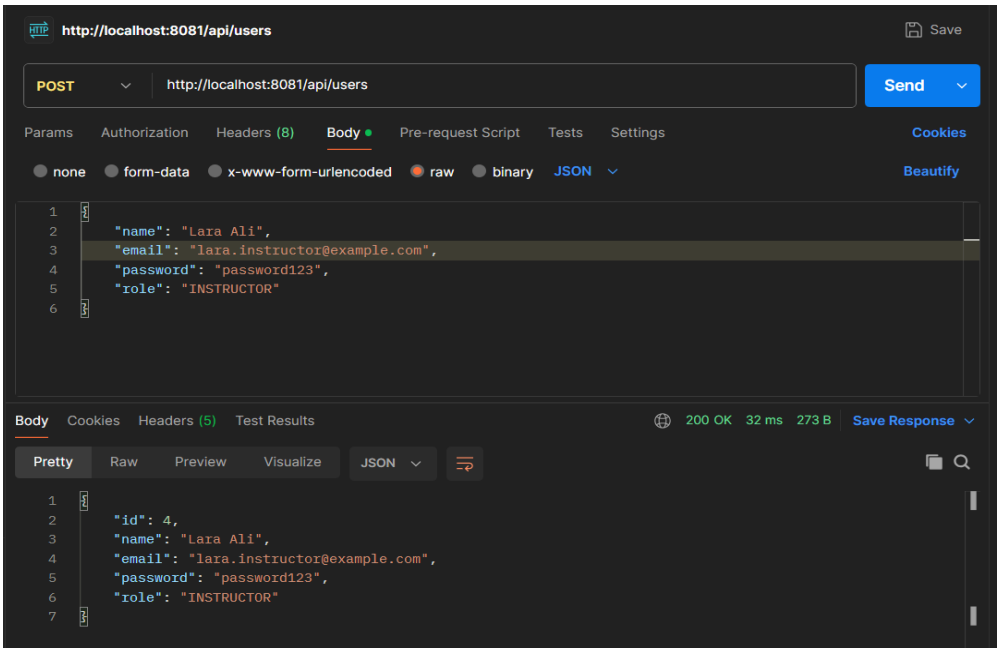
يقوم Feign تلقائيًا باستخدام اسم الخدمة user-service للبحث عن عنوانها في Eureka وإرسال طلب GET إلى المسار المحدد.

5.2 التحقق العملي من التواصل الداخلي (مع شواهد مرئية)

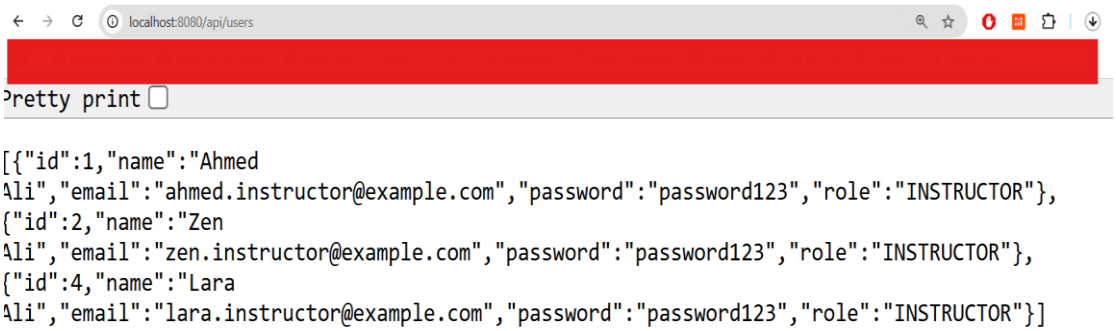
تم إجراء سلسلة من الاختبارات المتكاملة عبر البوابة (http://localhost:8080) لإثبات نجاح عملية التواصل.

1. **الخطوة الأولى: إنشاء المدربين**: تم إنشاء عدة مستخدمين بصلاحية INSTRUCTOR عبر إرسال طلب POST إلى /api/users.

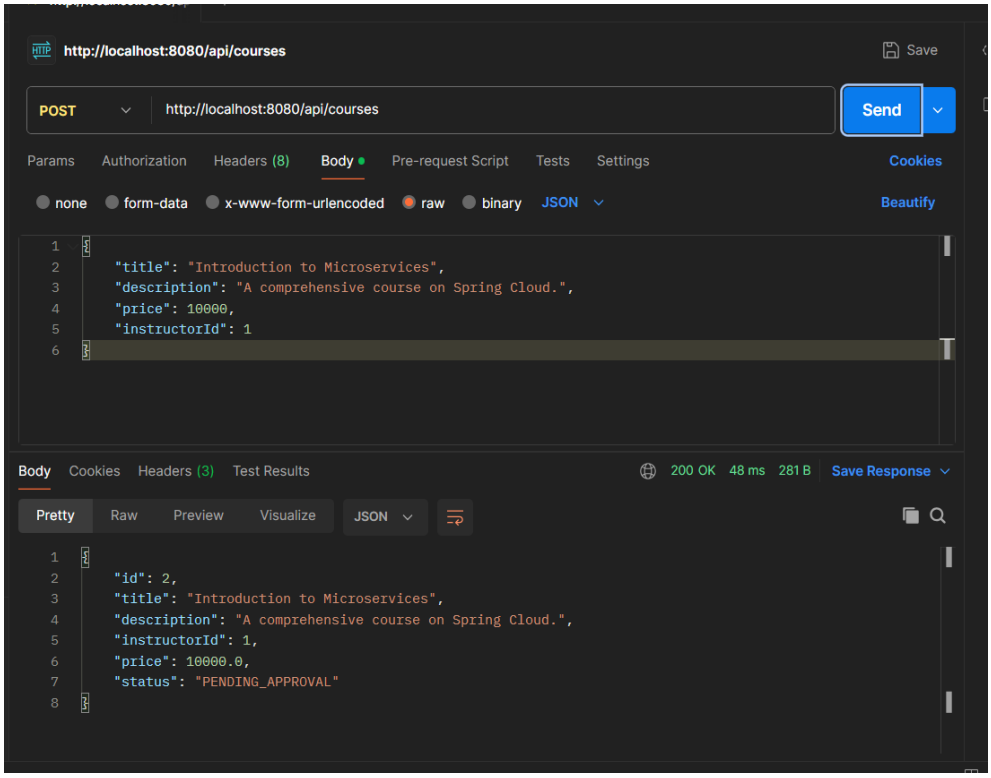




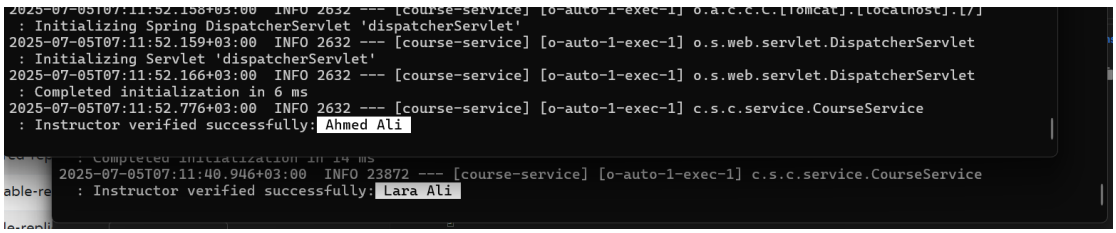
2. **الخطوة الثانية: التحقق من قائمة المدربين**: تم إرسال طلب `GET` إلى `/api/users` للتأكد من أن جميع المدربين قد تم إنشاؤهم بنجاح.



3. **الخطوة الثالثة: إنشاء دورة بمدرب صالح (السيناريو الناجح)** :تم إرسال طلب POSTإلى /api/courses/لإنشاء دورة جديدة مع instructorIdيعود لأحد المدربين الذين تم إنشاؤهم(e.g., ID=1) .



4. **الدليل القاطع (التحقق من سجلات التشغيل)** :للبرهنة على أن التواصل الداخلي قد حدث بالفعل، تُظهر سجلات التشغيل (Logs) الخاصة بـ course-serviceرسالة تؤكد أنه تم التحقق من المدرب بنجاح وجلب اسمه من user-serviceقبل إكمال عملية إنشاء الدورة .



هذه السجلات هي الدليل الأقوى على أن Feign Clientيعمل كما هو متوقع، حيث قام باستدعاء user-serviceخلف الكواليس.

5.3. تطبيق آلية الحماية من الفشل (Circuit Breaker) باستخدامResilience4j

لضمان عدم تأثر النظام بأكمله في حال تعطل user-service، تم تطبيق نمط "قاطع الدائرة."

1. **التنفيذ**: تم إضافة اعتمادية spring-cloud-starter-circuitbreaker-resilience4j إلى course-service.
2. **إنشاء السلوك الاحتياطي(Fallback)**: تم إنشاء كلاس UserClientFallback يوفر تنفيذًا بديلاً في حالة فشل الاتصال.

```
@Component
public class UserClientFallback implements
UserClient {
    @Override
    public UserDTO findUserById(Long id) {
        // هذا هو السلوك الذي يتم تنفيذه عند
فشل الاتصال
        throw new RuntimeException("User
service is currently unavailable. Please try
again later.");
    }
}
```

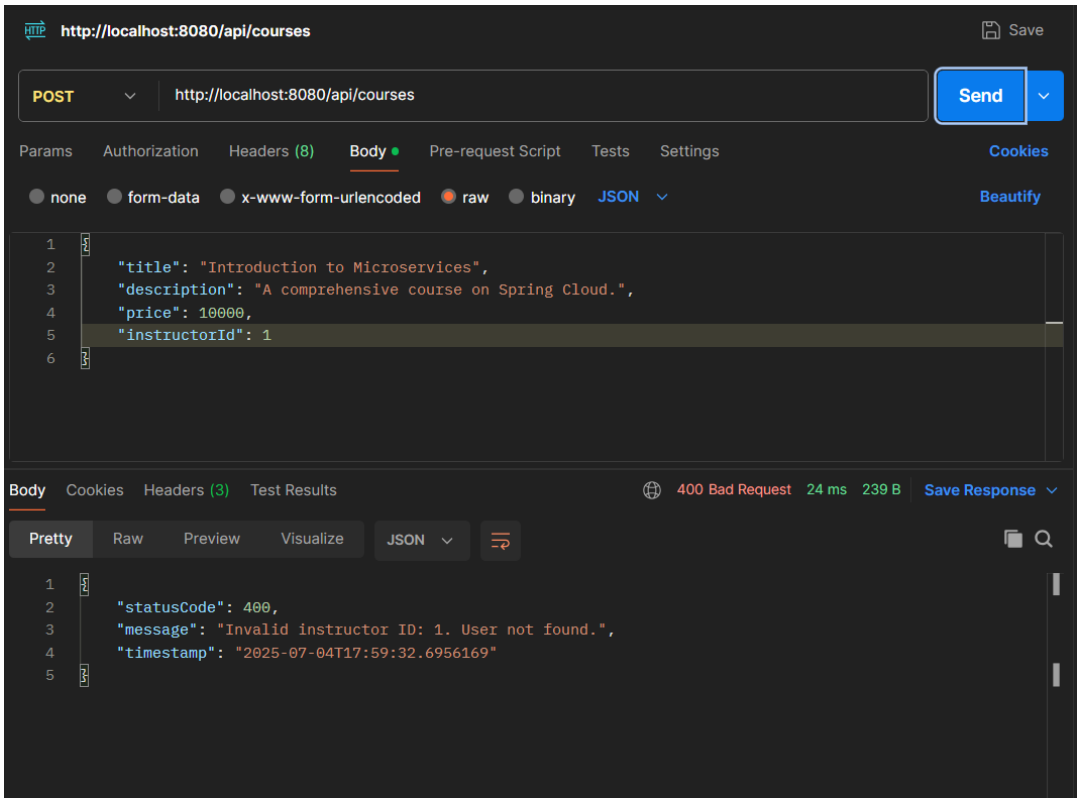
3. **الربط**: تم ربط هذا الكلاس بالعميل عبر @FeignClient(name = "user-service", fallback = UserClientFallback.class).

التحقق: عند إيقاف user-service بشكل متعمد ومحاولة إنشاء دورة، يفشل الطلب فورًا مع الرسالة المحددة في Fallback، بدلاً من الانتظار الطويل، مما يثبت أن آلية الحماية تعمل بفعالية.

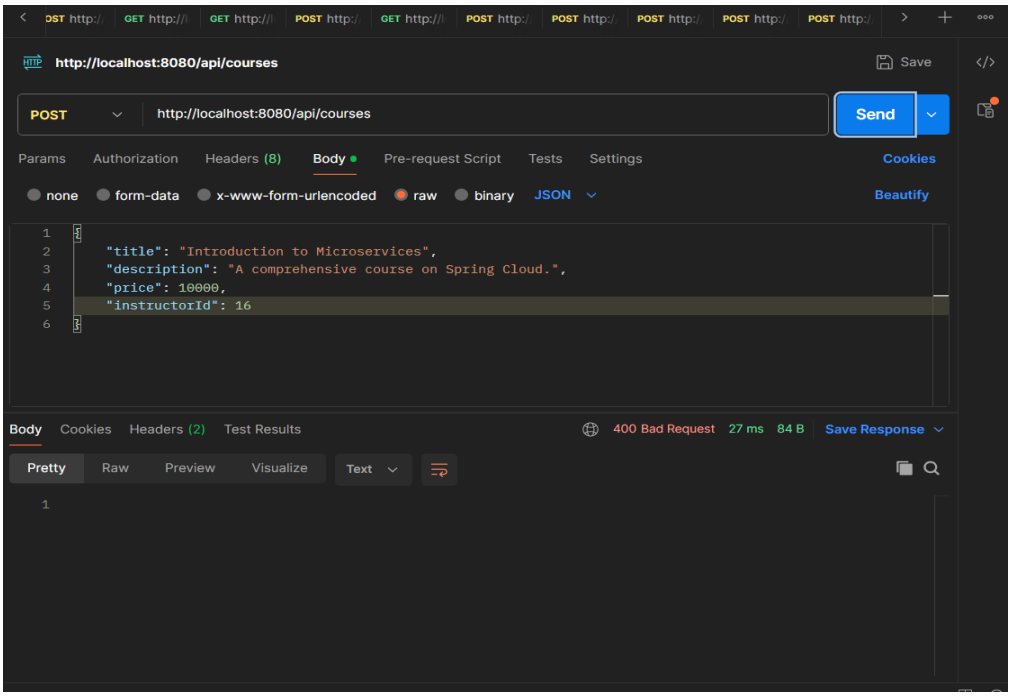
5.4. تصميم استجابات الأخطاء المخصصة (Custom Error Handling)

من المهم أن تكون رسائل الأخطاء التي يعيدها النظام واضحة ومفيدة. تم تحسين آلية التعامل مع الأخطاء في `course-service` لتعيد استجابة JSON منظمة في حالة إرسال `instructorId` غير صالح.

- **السيناريو الفاشل (ID غير موجود):** عند محاولة إنشاء دورة باستخدام `instructorId` غير موجود في قاعدة البيانات، يعيد النظام استجابة `400 Bad Request` مع رسالة واضحة تشرح سبب الخطأ .



- هذا يمثل تحسينًا كبيرًا مقارنة برسائل الخطأ الغامضة ، ويظهر نضجًا في تصميم واجهات برمجة التطبيقات.



6. قابلية التوسع وتوزيع الحمل (Scalability & Load Balancing)

أحد أهم مزايا معمارية الخدمات المصغرة هو سهولة التوسع الأفقي(Horizontal Scaling) ، أي تشغيل عدة نسخ من نفس الخدمة للتعامل مع زيادة الضغط. تم التحقق من هذه الميزة عمليًا.

6.1 المفهوم النظري للتطبيق

عند تشغيل نسختين من `course-service`، ستقوم كل نسخة بتسجيل نفسها في خادم Eureka. عندما تتلقى بوابة `API Gateway` طلبًا موجهًا إلى `course-service`، فإنها تسأل Eureka عن العناوين المتاحة. يقوم `Spring Cloud Load Balancer` المدمج مع البوابة و (Feign تلقائيًا باختيار إحدى النسخ المتاحة لتوجيه الطلب إليها، عادةً بآلية الترتيب الدوري(Round-robin) .

6.2. خطوات التحقق العملي

1. **بناء المشروع** :تم بناء ملف JAR القابل للتنفيذ لخدمة course-service.
2. **تشغيل نسخ متعددة** :تم تشغيل نسختين من الخدمة من خلال موجه الأوامر (Command Prompt)، مع تحديد منفذ مختلف لكل نسخة باستخدام server.port--.

```
# CMD Window 1
java -jar target/course-service-0.0.1-SNAPSHOT.jar --server.port=9090

# CMD Window 2
java -jar target/course-service-0.0.1-SNAPSHOT.jar --server.port=9091
```

3. **التحقق**:
   - **لوحة تحكم Eureka**: أظهرت اللوحة وجود نسختين (instances) مسجلتين لخدمة COURSE-SERVICE.
   - **سجلات التشغيل** :عند إرسال طلبات متتالية، لوحظ أن سجلات التشغيل في نافذتي الأوامر تظهر استقبال الطلبات بشكل متناوب، مما يثبت أن توزيع الحمل يعمل بنجاح دون الحاجة إلى أي إعدادات معقدة.

---

7. الخاتمة والآفاق المستقبلية

7.1. ملخص الإنجازات

لقد نجح هذا المشروع في تحقيق أهدافه من خلال بناء نظام تعليم إلكتروني قائم على معمارية الخدمات المصغرة. تم بنجاح:

- بناء خدمات أساسية مستقلة.
- تأسيس بنية تحتية قوية للاكتشاف والتوجيه الديناميكي.
- تفعيل التواصل الآمن والمحمي من الفشل بين الخدمات.
- إثبات قابلية النظام للتوسع الأفقي من خلال توزيع الحمل.

7.2. الدروس المستفادة والتحديات

- **أهمية التخطيط المعماري** :وضوح المسؤوليات بين الخدمات منذ البداية يمنع الكثير من التعقيدات لاحقًا.
- **التعامل مع الشبكة** :تتطلب الخدمات الموزعة التعامل مع حتمية فشل الشبكة، مما يجعل آليات مثل قاطع الدائرة ليست رفاهية بل ضرورة.
- **الاختبار المتكامل** :اختبار كل خدمة على حدة لا يكفي. الاختبار الشامل عبر البوابة هو الطريقة الوحيدة لضمان عمل النظام كوحدة متكاملة.

7.3. التوصيات والأعمال المستقبلية

لتحويل هذا المشروع إلى نظام جاهز للإنتاج، يمكن العمل على المحاور التالية:

1. **الأمن المتقدم** :**(Advanced Security)** تطبيق آلية مصادقة وتفويض مركزية على مستوى البوابة باستخدام `Spring Security`وبروتوكول `OAuth2`مع `JWT`.
2. **التواصل غير المتزامن** :**(Asynchronous Communication)** استخدام أنظمة الرسائل مثل `RabbitMQ`أو `Kafka`للعمليات التي لا تتطلب استجابة فورية (مثل إرسال إيميل عند الموافقة على دورة)، مما يزيد من مرونة النظام.
3. **المراقبة والتحليل المركزي** :**(Centralized Logging & Monitoring)** دمج حزمة ELK (Elasticsearch, Logstash, Kibana) لتجميع وتحليل سجلات التشغيل من جميع الخدمات في مكان واحد.
4. **النشر المؤتمت** :**(CI/CD)** بناء خطوط أنابيب للنشر والتكامل المستمر باستخدام أدوات مثل Jenkins أو GitLab CI لأتمتة عمليات البناء والاختبار والنشر.

---

8. الملاحق

8.1. ملحق (أ): الأدوات والتقنيات المستخدمة

- **لغة البرمجة**Java 17 :
- **إطار العمل**Spring Boot 3.x :
- **مكتبات**:**Spring Cloud**
  - Spring Cloud Gateway
  - Spring Cloud Netflix Eureka
  - Spring Cloud OpenFeign
  - Spring Cloud Circuit Breaker (Resilience4j)
  - Spring Cloud Load Balancer
- **قاعدة البيانات**) H2 In-Memory Database :لأغراض التطوير(
- **أداة البناء**Apache Maven :
- **أداة الاختبار**Postman :
- **بيئة التطوير**Visual Studio Code :

8.2. ملحق (ب): أمثلة من ملفات الإعدادات الهامة

مثال من application.properties لخدمة course-service:

```
# اسم الخدمة الفريد الذي سيسجل في Eureka
spring.application.name=course-service
# المنفذ الذي ستعمل عليه الخدمة
server.port=8082

# عنوان خادم Eureka
eureka.client.service-
url.defaultZone=http://localhost:8761/eureka/
```

مثال من application.yml لبوابة API Gateway:

```
server:
  port: 8080

eureka:
  client:
    service-url:
      defaultZone: http://localhost:8761/eureka/

spring:
  application:
    name: api-gateway
  cloud:
    gateway:
      discovery:
        locator:
          enabled: true # تفعيل التوجيه الديناميكي القائم على الاكتشاف
      routes:
        - id: user-service
          uri: lb://user-service
          predicates:
            - Path=/api/users/**
        - id: course-service
          uri: lb://course-service
          predicates:
            - Path=/api/courses/**
```