الجمهورية العربية السورية

جامعة دمشق

كلية الهندسة المعلوماتية

قسم هندسة البرمجيات ونظم المعلومات

مشروع النظم والتطبيقات الموزعة

آية صوّان

بيان خضير

بلقيس الصياصنة

شذى عصيده

عبير عصيده

بإشراف المهندسة:

آية المعطي

السؤال الأول:

نظام إدارة الموظفين والدردشة باستخدام RMI

مقدمة:

يهدف هذا المشروع إلى إنشاء نظام إدارة الموظفين الذي يمكن المدير من تسجيل الموظفين، وإدارة سجلاتهم، والتفاعل معهم عبر الدردشة. تم تطوير النظام باستخدام تقنيات Java وRMI (Remote Method Invocation) للتواصل بين الخادم والعميل.

مكونات المشروع

١. RmiServer

المهام الرئيسية:

- تشغيل خادم RMI.
- تسجيل الموظفين.
- بدء جلسة دردشة.

الشرح التفصيلي:

يتم تشغيل خادم RMI باستخدام الكود التالي:

```
registry = LocateRegistry.createRegistry( port: 1080);
Manager manager = new ManagerImp();
registry.bind( name: "HRService", manager);
```

يقوم الخادم بإنشاء سجل RMI وربط خدمة `HRService` بكائن `ManagerImp`.

خيارات القائمة:

١. تسجيل الموظف:

- يطلب النظام من المستخدم إدخال الاسم وكلمة المرور.

- يتم تسجيل الموظف عبر استدعاء طريقة `register` من الكائن `manager`.

٢. بدء الدردشة:

- يتم فتح منفذ جديد لـ ServerSocket لبدء جلسة الدردشة.

- ينتظر الخادم اتصال العميل ويبدأ جلسة الدردشة باستخدام BufferedReader وPrintWriter.

٣. الخروج:

- ينهي البرنامج.

٢. RmiClient

المهام الرئيسية:

- التواصل مع خادم RMI.

- عرض قائمة للمدير لإدارة الموظفين والتفاعل معهم.

الشرح التفصيلي:

يتصل العميل بالخادم عبر الكود التالي:

```
Registry registry = LocateRegistry.getRegistry(serverIpAddress,  port: 1080);
Manager manager = (Manager) registry.lookup( name: "HRService");
```

يتم الحصول على الكائن البعيد `Manager` لتنفيذ الأوامر على الخادم.

خيارات القائمة:

١. قائمة الموظفين المسجلين:

- يعرض النظام قائمة الموظفين المسجلين عبر استدعاء طريقة `getEmployee` من الكائن `manager`.

٢. التقاط لقطة شاشة:

- يطلب النظام اسم الموظف وينفذ أمر التقاط لقطة شاشة عبر استدعاء طريقة `screenshot` على كائن الموظف.

٣. التقاط صورة:

- يطلب النظام اسم الموظف وينفذ أمر التقاط صورة عبر استدعاء طريقة `photo` على كائن الموظف.

٤. بدء الدردشة:

- يبدأ جلسة دردشة مع الخادم عبر الاتصال بمنفذ ServerSocket على الخادم.

٥. الخروج:

- ينهي البرنامج.

بنية النظام

مكونات الخادم:

- Registry: سجل RMI لإنشاء الربط وتسجيل الخدمات.

- ManagerImp: الكائن الذي ينفذ واجهة `Manager` ويوفر خدمات تسجيل الموظفين وإدارتهم.

مكونات العميل:

- Registry: سجل RMI للبحث عن الخدمة البعيدة.

- Manager: الواجهة التي توفر خدمات إدارة الموظفين والدردشة.

نموذج الدردشة:

- ServerSocket: يستخدم لبدء جلسة الدردشة على الخادم.

- Socket: يستخدم لبدء جلسة الدردشة على العميل.

- BufferedReader وPrintWriter: يستخدمان لقراءة وإرسال الرسائل بين العميل والخادم.

وقد تم إنشاء نظام RMI وتوزيع الـ Interfaces بهذا الشكل:

3. Employee

واجهة `Employee`هي واجهة بعيدة (Remote Interface) توفر الوظائف التي يمكن تنفيذها على كائنات الموظفين عبر .RMI تشمل هذه الوظائف:

- `getName`: للحصول على اسم الموظف.
- `getPassword`: للحصول على كلمة مرور الموظف.
- `screenshot`: لالتقاط لقطة شاشة.
- `photo`: لالتقاط صورة باستخدام الكاميرا.
- `Chat`: لبدء جلسة دردشة.
- `printScreen`:لطباعة لقطة الشاشة.

4. EmployeeImp

كلاس `EmployeeImp` هو تنفيذ للواجهة `Employee`، ويوفر الوظائف التالية:

١. **الخصائص:**
   - `name` and `password`: لتخزين اسم وكلمة مرور الموظف.
٢. **الدوال:**
   - `getName` and `getPassword`: للحصول على اسم وكلمة مرور الموظف.
   - `screenshot`: لالتقاط وحفظ لقطة شاشة للشاشة الحالية على الجهاز. يتم حفظ اللقطة في مجلد محدد مسبقًا.
   - `photo`: لالتقاط صورة باستخدام الكاميرا وحفظها في مجلد محدد مسبقًا.
   - `Chat`: لبدء جلسة دردشة مع العميل. يتم تشغيل خادم دردشة والاستماع للاتصالات من العملاء.
   - `printScreen`: لطباعة لقطة الشاشة (لم يتم تنفيذها بالكامل).

- **screenshot**: تلتقط هذه الدالة لقطة شاشة كاملة للجهاز وتقوم بحفظها كصورة PNG في مجلد محدد. يتم تحديد اسم الملف باستخدام طابع زمني لضمان التفرد.

```java
@Override
public void screenshot() throws RemoteException {
    try {
        Dimension screenRect = Toolkit.getDefaultToolkit().getScreenSize();
        Rectangle captureRect = new Rectangle( x: 0, y: 0, screenRect.width, screenRect.height);
        BufferedImage capture = new BufferedImage(captureRect.width, captureRect.height, BufferedImage.TYPE_INT_RGB);
        Robot robot = new Robot();
        capture = robot.createScreenCapture(captureRect);
        SimpleDateFormat formatter = new SimpleDateFormat( pattern: "yyyyMMdd_HHmmss");
        String fileName = "screenshot_" + formatter.format(new java.util.Date()) + ".png";

        File screenshotFile = new File( pathname: "D:\\photos\\" + fileName);

        File screenshotDir = new File( pathname: "D:/photos");
        if (!screenshotDir.exists()) {
            screenshotDir.mkdirs();
        }
        ImageIO.write(capture, formatName: "PNG", screenshotFile);

        System.out.println("Screenshot captured and saved as " + screenshotFile.getPath());
    } catch (IOException e) {
        System.err.println("Error saving screenshot: " + e.getMessage());
        e.printStackTrace();
    } catch (AWTException e) {
        System.err.println("Error capturing screenshot: " + e.getMessage());
        e.printStackTrace();
    }

}
```

- **photo**: تستخدم مكتبة OpenCV لفتح الكاميرا الافتراضية والتقاط صورة وحفظها كملف JPG في مجلد محدد. يتم أيضًا تحديد اسم الملف باستخدام طابع زمني لضمان التفرد.

```java
@Override
public void photo() throws RemoteException {
    System.loadLibrary(Core.NATIVE_LIBRARY_NAME);
    VideoCapture capture = new VideoCapture( index: 0);
    if (!capture.isOpened()) {
        System.out.println("Error: Camera not opened!");
        return;
    }
    Mat frame = new Mat();

    if (capture.read(frame)) {
        SimpleDateFormat formatter = new SimpleDateFormat( pattern: "yyyyMMdd_HHmmss");
        String fileName = "captured_image_" + formatter.format(new java.util.Date()) + ".jpg";
        File photoFile = new File( pathname: "D:/photos/" + fileName);

        Imgcodecs.imwrite(photoFile.getAbsolutePath(), frame);
        System.out.println("Image captured successfully and saved as " + photoFile.getPath());
    } else {
        System.out.println("Error: Failed to capture image!");
    }

    capture.release();
}
```

- **Chat**: تقوم بإنشاء خادم دردشة باستخدام `ServerSocket` للاستماع على منفذ محدد. يتم قبول الاتصالات من العملاء ويبدأ التفاعل عبر الرسائل النصية باستخدام `BufferedReader` و`PrintWriter`.

```java
@Override
public void Chat(int PORT) throws RemoteException {
    try {
        ServerSocket serverSocket = new ServerSocket(PORT);
        System.out.println("Server is running...");
        System.out.println("Server started. Waiting for clients...");
        Socket clientSocket = serverSocket.accept();
        System.out.println("Client connected.");

        BufferedReader in = new BufferedReader(new InputStreamReader(clientSocket.getInputStream()));
        PrintWriter out = new PrintWriter(clientSocket.getOutputStream(), autoFlush: true);
        boolean continueChat = true;
        while (continueChat) {
            String receivedMessage = in.readLine();
            System.out.println("Received message from client: " + receivedMessage);
            System.out.println("If you want to send a message, enter your message. If you want to close the chat, enter 'exit'.");
            BufferedReader userInput = new BufferedReader(new InputStreamReader(System.in));
            String reply = userInput.readLine();
            if (reply.equalsIgnoreCase( anotherString: "exit")) {
                clientSocket.close();
                serverSocket.close();
                continueChat = false;
            } else {
                out.println(reply);
            }
        }
    } catch (IOException e) {
        System.err.println("Error during chat: " + e.getMessage());
        e.printStackTrace();
    }
```

- **retrieveCurrentIpAddress**: تسترجع عنوان IP الحالي للجهاز باستخدام `InetAddress`.

```java
public static String retrieveCurrentIpAddress() {
    InetAddress localHost = null;
    try {
        localHost = InetAddress.getLocalHost();
    } catch (UnknownHostException e) {
        e.printStackTrace();
    }
    assert localHost != null;
    return localHost.getHostAddress();
}
```

يوفر هذا المشروع نظامًا فعالًا لإدارة الموظفين والتفاعل معهم عبر الدردشة. باستخدام تقنيات Java وRMI، يمكن للمدير تسجيل الموظفين، إدارة سجلاتهم، والتفاعل معهم بشكل سهل وفعال. يعد هذا المشروع نموذجًا قويًا لتطبيقات الأعمال باستخدام Java.

السؤال الثاني:

- **بنية مشروع C2C (من مستهلك إلى مستهلك)**

المشروع يتكون من عدة مكونات:

المجلد Entity: يحتوي على تعاريف الكائنات (الموديلات) التي تستخدم في المشروع. يوجد Customer وProduct1 والتي تمثل المستهلكين والمنتجات.

المجلد Repository: يحتوي على واجهات الاستخدام (CustomerRepo وProduct1Repo) التي تعرف العمليات الممكنة على الكائنات مثل البحث والحذف والتحديث.

المجلد Service: يحتوي على خدمات الأعمال (CustomerService وProduct1Service) التي تنفذ المنطق التجاري للعمليات المرتبطة بالمستهلكين والمنتجات.

المجلد Controller:

(CustomerController وProduct1Controller) يحتوي على مكونات التحكم في الواجهة، وهم المسؤولون عن استقبال الطلبات الواردة وتوجيهها إلى الخدمات المناسبة لتنفيذ العمليات المطلوبة.

بشكل عام يحتوي هذا النظام على عدة وظائف منها، نظام تسجيل المستهلكين وتسجيل الدخول للمستهلكين (registercustomer وloginCustomer)، وإضافة وشراء المنتجات (addProduct1 و buyProduct1)، وتحديث وحذف المستهلكين والمنتجات (updateCustomer و deleteCustomer و updateProduct1 و deleteProduct1)، واسترداد المنتجات المرتبطة بعميل معين (getProduct1sByCustomerId) وغيرها من العمليات.

- **بنية مشروع B2B (من شركة إلى شركة)**

المشروع يتكون من عدة مكونات:

المجلد Entity: يحتوي على تعاريف الكائنات (الموديلات) التي تستخدم في المشروع. يوجد Company وProduct والتي تمثل الشركات والمنتجات.

المجلد Repository: يحتوي على واجهات الاستخدام (CompanyRepo و ProductRepo) التي تعرف العمليات الممكنة على الكائنات مثل البحث والحذف والتحديث.

المجلد Service: يحتوي على خدمات الأعمال (CompanyService و ProductService) التي تنفذ المنطق التجاري للعمليات المرتبطة بالشركات والمنتجات.

المجلد Controller:

(CompanyController وProductController) يحتوي على مكونات التحكم في الواجهة، وهم المسؤولون عن استقبال الطلبات الواردة وتوجيهها إلى الخدمات المناسبة لتنفيذ العمليات المطلوبة.

بشكل عام يحتوي هذا النظام على عدة وظائف منها، نظام تسجيل الشركات وتسجيل الدخول للشركات (registercompany و loginCompany)، وإضافة وشراء المنتجات (addProduct و buyProduct)، وتحديث وحذف الشركات والمنتجات (updateCompany و deleteCompany و updateProduct و deleteProduct)، واسترداد المنتجات المرتبطة بشركة معينة (getProductsByCompanyId) وغيرها من العمليات.

- قمنا بتحقيق خدمة discovery service عن طريق :

Eureka Serverهو خادم خدمة التسجيل والاكتشاف المستخدم في تطبيقات الميكروسيرفس. يعمل Eureka Server كمركز لتسجيل واكتشاف الخدمات المختلفة في بيئة التطوير لدي.

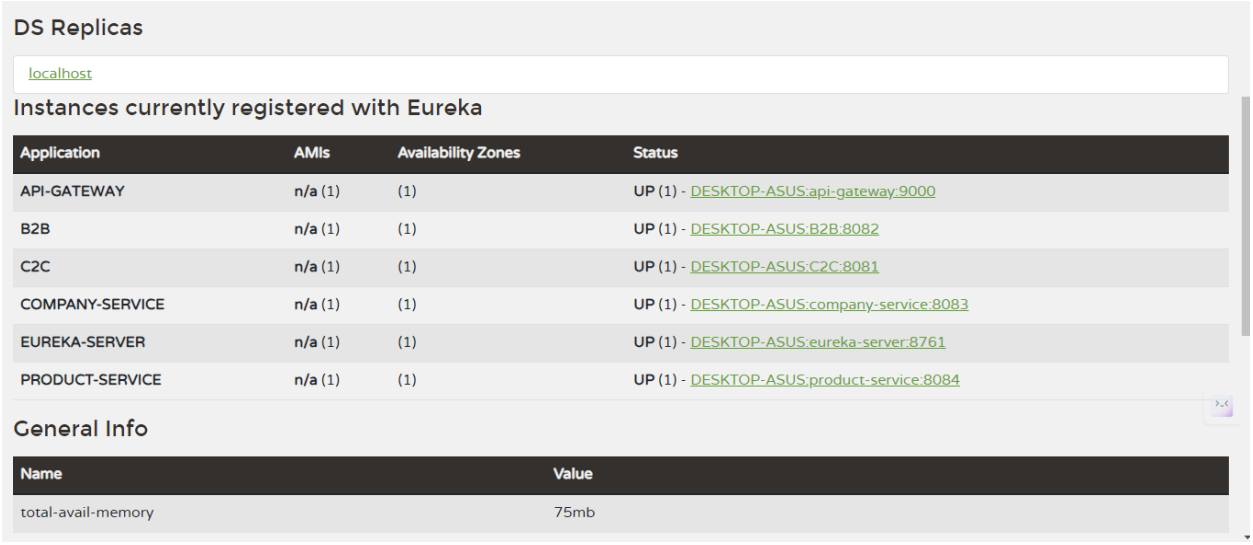
حيث اولا يتم تهيئة السيرفر وتشغيله على المنفذ ٨٧٦١.

وعن طريقه نلاحظ تسجيل الخدمات المختلفة مثل C2C، B2B، PRODUCT-SERVICE، COMPANY-SERVICE، API-GATEWAY في Eureka Server.

و من خلاله نستطيع ملاحظة انه تم **عنونة الخدمات بشكل ديناميكي** :

```
2024-05-27 17:38:18.875  INFO 5740 --- [       Thread-9]
e.s.EurekaServerInitializerConfiguration : Started Eureka Server
```

```
2024-05-27 17:38:21.412  INFO 5740 --- [nio-8761-exec-5]
c.n.e.registry.AbstractInstanceRegistry  : Registered instance PRODUCT-
SERVICE/DESKTOP-ASUS:product-service:8084 with status UP (replication=false)

2024-05-27 17:38:21.919  INFO 5740 --- [nio-8761-exec-6]
c.n.e.registry.AbstractInstanceRegistry  : Registered instance PRODUCT-
SERVICE/DESKTOP-ASUS:product-service:8084 with status UP (replication=true)

2024-05-27 17:38:48.591  INFO 5740 --- [nio-8761-exec-5]
c.n.e.registry.AbstractInstanceRegistry  : Registered instance COMPANY-
SERVICE/DESKTOP-ASUS:company-service:8083 with status UP (replication=false)

2024-05-27 17:38:49.100  INFO 5740 --- [nio-8761-exec-8]
c.n.e.registry.AbstractInstanceRegistry  : Registered instance COMPANY-
SERVICE/DESKTOP-ASUS:company-service:8083 with status UP (replication=true)

2024-05-27 17:38:59.405  INFO 5740 --- [nio-8761-exec-6]
c.n.e.registry.AbstractInstanceRegistry  : Registered instance API-
GATEWAY/DESKTOP-ASUS:api-gateway:9000 with status UP (replication=false)

2024-05-27 17:38:59.913  INFO 5740 --- [nio-8761-exec-7]
c.n.e.registry.AbstractInstanceRegistry  : Registered instance API-
GATEWAY/DESKTOP-ASUS:api-gateway:9000 with status UP (replication=true)

2024-05-27 17:39:18.866  INFO 5740 --- [a-EvictionTimer]
c.n.e.registry.AbstractInstanceRegistry  : Running the evict task with
compensationTime 0ms
```



إجمالاً، يمكننا استنتاج أن Eureka Server هو المسؤول عن تسجيل الخدمات وإدارتها، وتوفير واجهة للتطبيقات الأخرى لاكتشاف الخدمات المتاحة في البيئة.

- تعمل Gateway كواجهة أمامية وتقوم بالعديد من المهام المهمة مثل التوجيه (Routing) والتحويل (Proxying) والتصفية (Filtering) والتحكم في الحماية (Security Control).
يتم استخدام Spring Cloud Gateway كخدمة Gateway للتوجيه والتحكم في حركة المرور بين العملاء والخدمات المختلفة.

  `server.port=9000`: تعيين المنفذ الذي سيستخدمه خدمة Gateway للاستماع والتفاعل مع العملاء.

  `eureka.client.service-url.defaultZone=http://localhost:8761/eureka/`: تكوين خدمة Gateway للتواصل مع Eureka Server لاكتشاف والوصول إلى الخدمات المسجلة.

- قمنا بتحقيق خدمة WebService باستخدام RESTful

١. حيث قمنا بتحديد الموارد (Resources) التي نرغب في توفيرها كجزء من خدماتنا B2B، C2C . مثل المستهلكين، الشركات ، المنتجات، الطلبات، وما إلى ذلك.

٢. تحديد العمليات (Operations): قمنا بتحديد العمليات التي يمكن تنفيذها على الموارد المحددة. مثل الاستعلام (GET)، الإنشاء (POST)، التحديث (PUT)، والحذف (DELETE).

٣. تحديد نمط التوجيه (Routing): قمنا بتحديد كيفية توجيه الطلبات إلى الموارد المناسبة.

٤. تنفيذ الخدمة: قمنا بتنفيذ الخدمة باستخدام Spring Boot. ومن ثم تعريف نقاط النهاية (Endpoints) لكل موارد وعملية وتنفيذ طرق المعالجة المناسبة لكل عملية.

٥. استخدام الطرق الصحيحة للتواصل: مثل GET لاستعلام الموارد (شركة / مستهلك /منتج)، POST لإنشاء موارد جديدة، PUT لتحديث الموارد، وDELETE لحذف الموارد.

- ثم قمنا **باستدعاء خدمات النظام عن طريق البوابة gateway** كـ الآتي :

```
spring.cloud.gateway.routes[0].id=company-service
spring.cloud.gateway.routes[0].uri=lb://company-service
spring.cloud.gateway.routes[0].predicates[0].=Path=/company/**

spring.cloud.gateway.routes[1].id=C2C
spring.cloud.gateway.routes[1].uri=lb://C2C
spring.cloud.gateway.routes[1].predicates[0].=Path=/customer/**


spring.cloud.gateway.routes[2].id=C2C
spring.cloud.gateway.routes[2].uri=lb://C2C
spring.cloud.gateway.routes[2].predicates[0].=Path=/product1s/**

spring.cloud.gateway.routes[3].id=B2B
spring.cloud.gateway.routes[3].uri=lb://B2B
spring.cloud.gateway.routes[3].predicates[0].=Path=/company/**

spring.cloud.gateway.routes[4].id=B2B
spring.cloud.gateway.routes[4].uri=lb://B2B
spring.cloud.gateway.routes[4].predicates[0].=Path=/products/**
```

إعدادات `spring.cloud.gateway.routes`: تحدد تصنيفات التوجيه لخدمة Gateway. في المثال المعطى، هناك عدة تصنيفات محددة.

- `spring.cloud.gateway.routes[٠]`:

يحدد توجيه المسار "**/company/" إلى خدمة "company-service".

- `spring.cloud.gateway.routes[١]`:

يحدد توجيه المسار "**/customer/" إلى خدمة "C2C".

- `spring.cloud.gateway.routes[٢]`:

يحدد توجيه المسار "**/product1s/" إلى خدمة "C2C".

- `spring.cloud.gateway.routes[٣]`:

يحدد توجيه المسار "**/company/" إلى خدمة "B2B".

- `spring.cloud.gateway.routes[٤]`:

يحدد توجيه المسار "**/products/" إلى خدمة "B2B".

يتم استخدام `lb://` قبل اسم الخدمة للإشارة إلى أنه يجب تحقيق توازن التحميل لهذه الخدمات.

بهذه الإعدادات، يقوم خدمة Gateway بتوجيه الطلبات وفقًا للمسارات المحددة إلى الخدمات المناسبة.

- قمنا بتحقيق **التواصل بين الخدمات واستخدام العنونة بشكل ديناميكي** عن طريق:

١. استخدام خدمة التسجيل واكتشاف (Service Discovery) لتسجيل الخدمات النشطة واكتشافها بواسطة البوابة والخدمات الأخرى. تقوم كل خدمة بتسجيل نفسها عند تشغيلها وتحديث حالتها بشكل دوري.

٢. تكوين البوابة: قمنا بتكوين البوابة لاستخدام خدمة التسجيل واكتشاف للحصول على عناوين الخدمات النشطة ديناميكيًا.

٣. استخدام العنونة الديناميكية في البوابة: قمنا بتعيين العناوين المسترجعة من خدمة التسجيل واكتشاف في تكوينات البوابة بشكل ديناميكي.

٤. استخدام التوجيه الديناميكي: عندما يتم استدعاء خدمة من خلال البوابة، يمكننا توجيه الطلبات إلى عنوان الخدمة المسترجع من خدمة التسجيل واكتشاف. حيث نستطيع تحديث التوجيه بشكل دوري للتأكد من أنه يعكس العناوين الحالية للخدمات.

- **إدارة الفشل والتأخر في التواصل بين الخدمات (Hystrix).**

```
public static final String Company_SERVICE = "company-service";

@GetMapping(value="/{companyId}")
@CircuitBreaker(name = Company_SERVICE, fallbackMethod =
"companyServiceFallBack")
public Company getCompany(@PathVariable("companyId") String
companyId)
{
    Company companyOne = new Company(companyId, "Company Name " +
companyId,
```

```
            "xxxxx" + companyId ,"Company Password " + companyId,
"Company Location " + companyId);

    Products products = restTemplate.getForObject(
            "http://product-service/product/1", Products.class);
    companyOne.setProducts(products);



    return companyOne;
}

public Company companyServiceFallBack(Exception userException)
{
    return new Company("1", "Company One", "xyz"
    , "0000000" , "not Found");
}
```

الكود السابق يستخدم إدارة الفشل والتأخر في التواصل بين الخدمات باستخدام مكتبة Hystrix. هنا توضح الخطوات التي تم اتباعها:

١. تم استخدام التعليق `@CircuitBreaker` من مكتبة Hystrix لتعليم النظام على فتح إدارة الفشل عند حدوث أخطاء في الاتصال بالخدمة المستهدفة. تم تعيين اسم الدارة ليكون "Company_SERVICE" وتم تحديد الطريقة التي ستعمل كنقطة الدخول للدارة المكسورة وهي `companyServiceFallBack`.

٢. الطريقة `getCompany` هي نقطة الدخول الأساسية للخدمة. عند استدعاء هذه الطريقة، سيتم فحص حالة الخدمة المستهدفة. إذا كانت الخدمة تعمل بشكل صحيح، سيتم الحصول على معلومات الشركة والمنتجات من الخدمة المستهدفة باستخدام `restTemplate`.

٣. إذا حدث خطأ في الاتصال بالخدمة المستهدفة، ستتم استدعاء الطريقة `companyServiceFallBack` كبديل. في هذه الحالة، ستقوم الطريقة بإنشاء كائن `Company` بقيم افتراضية تشير إلى أن الشركة غير موجودة.

٤. تم تحديد تفاصيل الشركة المنتجة في كائن `Company` باستخدام القيم المميزة التي تم تمريرها إلى الدالة البنائية.

٥. تم استخدام `restTemplate` لإجراء طلب GET للحصول على منتجات الشركة من الخدمة المستهدفة. تم تمرير عنوان URL للخدمة المستهدفة ونوع البيانات المتوقعة (`Products.class`) إلى `getForObject`. إذا كان الطلب ناجحًا، ستتم تعيين المنتجات في كائن `Company`.

بهذه الطريقة، إذا حدث خطأ في الاتصال بالخدمة المستهدفة، ستتم استدعاء طريقة الاحتياطية `companyServiceFallBack` وسيتم إرجاع كائن `Company` بالقيم الافتراضية. هذا يساعد في تجنب تأثير الأخطاء في الخدمة المستهدفة على العملية الرئيسية للخدمة الحالية.

- تطبيق توزيع الحمل Load Balancing.

```
@SpringBootApplication
@EnableDiscoveryClient
public class ProductServiceApplication {

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

    @LoadBalanced
    @Bean
    public RestTemplate getRestTemplate() {
        return new RestTemplate();
    }

}
```

//

```java
@SpringBootApplication
@EnableEurekaClient
public class CompanyServiceApplication {

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


    @LoadBalanced
    @Bean
    public RestTemplate getRestTemplate() {
       return new RestTemplate();
    }


}
```

في الكود السابق، تم استخدام خاصية تحميل التوازن (Load Balancing) باستخدام تعليمة `@LoadBalanced` في كل من مشروعي خدمة الشركة وخدمة المنتج.

باستخدام تعليمة `@LoadBalanced` وتكوين RestTemplate في كل من خدمة الشركة وخدمة المنتج، يتم تمكين التحميل التوازني للطلبات الصادرة من الخدمتين. هذا يعني أنه عندما يقوم خدمة الشركة بطلب خدمة المنتج، سيتم توجيه الطلب إلى إحدى المثيلات المتاحة للخدمة المنتج باستخدام توازن الحمل.

عند تشغيل خدمة الشركة وخدمة المنتج، نستطيع توزيع الحمل بينهما عن طريق تعيين بورتات مختلفة لكل منهما. في هذه الحالة، يمكننا تشغيل خدمة الشركة على بورت ٨٠٨٧ وخدمة المنتج على بورت ٨٠٨٥.

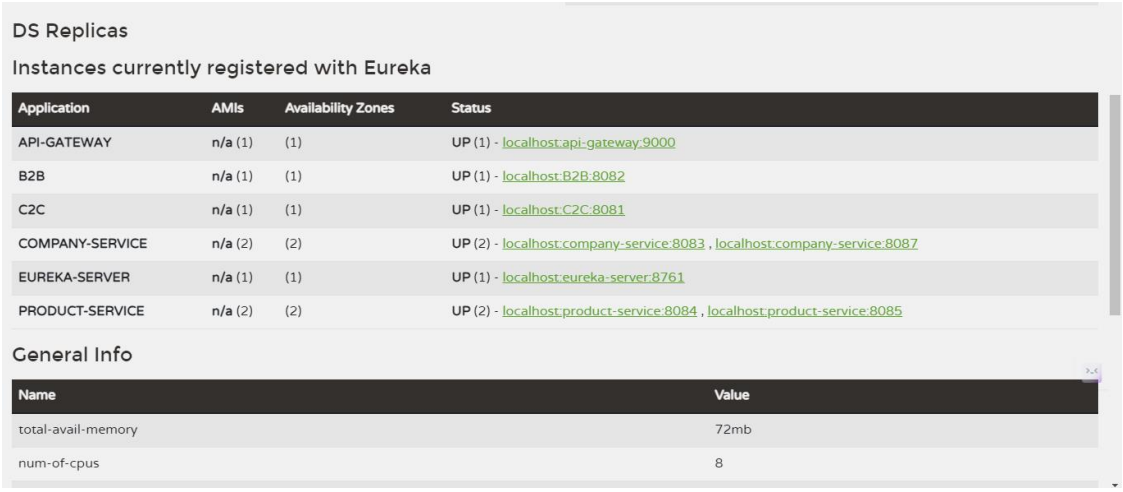
لتحقيق ذلك، قمت باستخدام الأوامر التالية لتشغيل الخدمات على بورتات مختلفة:

لتشغيل خدمة الشركة:

```
java -Dserver.port=808 ٧-jar product-service-0.0.1-SNAPSHOT.jar
```

لتشغيل الخدمة المنتج:

```
java -Dserver.port=8085 -jar product-service-0.0.1-SNAPSHOT.jar
```



DS Replicas

Instances currently registered with Eureka

Application	AMIs	Availability Zones	Status
API-GATEWAY	n/a (1)	(1)	UP (1) - localhost:api-gateway:9000
B2B	n/a (1)	(1)	UP (1) - localhost:B2B:8082
C2C	n/a (1)	(1)	UP (1) - localhost:C2C:8081
COMPANY-SERVICE	n/a (2)	(2)	UP (2) - localhost:company-service:8083 , localhost:company-service:8087
EUREKA-SERVER	n/a (1)	(1)	UP (1) - localhost:eureka-server:8761
PRODUCT-SERVICE	n/a (2)	(2)	UP (2) - localhost:product-service:8084 , localhost:product-service:8085

General Info

Name	Value
total-avail-memory	72mb
num-of-cpus	8

نلاحظ من خلال واجهة Eureka انه تم تشغيل خدمة الشركة على منفذين مختلفين وتوزيع الحمل بينهما، و بالمثل فإن خدمة المنتج أيضا تم تشغيلها على منفذين مختلفين.