1. **Utworzenie projektu mikroserwisu NotificationService w Spring Boot za pomocą Spring Initializr**

   - Ustaw parametry projektu:
     1) **Project**: *Maven Project*
     2) **Language**: *Java*
     3) **Spring Boot Version**: *3.3.6*
     4) **Group**: *com.pbs.edu*
     5) **Artifact**: *NotificationService*
     6) **Packaging**: *Jar*
     7) **Java**: *17*
   - Dodanie zależności:
     1) Spring Web
     2) Spring Cloud Gateway
     3) Java Mail Sender
     4) Spring Data JPA
     5) MySQL Driver
     6) Lombok
   - Pobierz projekt i rozpakuj plik ZIP, a następnie otwórz go w wybranym IDE (np. IntelliJ IDEA).

2. **Konfiguracja mikroserwisu do wysyłania powiadomień np. Email poprzez mailtrap.io.**

   - Przejdź do folderu *src/main/resources* i otwórz plik *application.properties*.
   - Dodaj konfiguracje SpringMail:

```
spring.mail.host=smtp.mailtrap.io
spring.mail.port=587
spring.mail.username=31aafe68f86106
spring.mail.password=230bdd9bdf3ef2
spring.mail.properties.mail.smtp.auth=true
spring.mail.properties.mail.smtp.starttls.enable=true
```

   - Dodaj konfiguracje bazy danych:

```
spring.datasource.url=jdbc:mysql://weather-db:3306/weather
spring.datasource.username=weatheruser
spring.datasource.password=p@ssw0rd
spring.jpa.hibernate.ddl-auto=update
spring.jpa.show-sql=true
spring.datasource.driver-class-name=com.mysql.cj.jdbc.Driver
spring.jpa.properties.hibernate.dialect=org.hibernate.dialect.MySQLDialect
```

3. **Implementacja NotificationService**

- Utworzenie encji *Notification*:

```
package com.pbs.edu.NotificationService.model;

import jakarta.persistence.Entity;
import jakarta.persistence.GeneratedValue;
import jakarta.persistence.GenerationType;
import jakarta.persistence.Id;
import lombok.Data;

@Entity
@Data
public class Notification {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    private String email;
    private String subject;
    private String message;
    private boolean sent;
}
```

- Utworzenie rezpozytorium *NotificationRepository*:

```
package com.pbs.edu.NotificationService.repository;

import com.pbs.edu.NotificationService.model.Notification;
import org.springframework.data.jpa.repository.JpaRepository;

public interface NotificationRepository extends
JpaRepository<Notification, Long> {

}
```

4. **Utworzenie *EmailService*.**

```
package com.pbs.edu.NotificationService.service;

import com.pbs.edu.NotificationService.model.Notification;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.mail.javamail.JavaMailSender;
import org.springframework.mail.javamail.MimeMessageHelper;
import org.springframework.stereotype.Service;

@Service
public class EmailService {

    @Autowired
    private JavaMailSender mailSender;

    public void sendEmail(Notification notification) {
        try {
            var message = mailSender.createMimeMessage();
```

```
            var helper = new MimeMessageHelper(message, true);
            helper.setTo(notification.getEmail());
            helper.setSubject(notification.getSubject());
            helper.setText(notification.getMessage(), true);
            mailSender.send(message);
        } catch (Exception e) {
            throw new RuntimeException("Failed to send email", e);
        }
    }
}
```

5. **Utworzenie *NotificationController***

```
package com.pbs.edu.NotificationService.controller;

import com.pbs.edu.NotificationService.model.Notification;
import com.pbs.edu.NotificationService.repository.NotificationRepository;
import com.pbs.edu.NotificationService.service.EmailService;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.web.bind.annotation.*;

import java.util.List;

@RestController
@RequestMapping("/notifications")
public class NotificationController {

    @Autowired
    private NotificationRepository repository;

    @Autowired
    private EmailService emailService;

    @PostMapping
    public Notification sendNotification(@RequestBody Notification
notification) {
        emailService.sendEmail(notification);
        notification.setSent(true);
        return repository.save(notification);
    }

    @GetMapping
    public List<Notification> getAllNotifications() {
        return repository.findAll();
    }
}
```

6. **Uruchom lokalnie aplikacje.**

- Zbuduj aplikację Spring Boot np. używając komendy:

```
mvn clean package -DskipTests
```

7. **Utwórz plik Dockerfile oraz zmodyfikuj plik docker-compose.yml, a następnie uruchom.**

- W folderze głównym projektu utwórz plik *Dockerfile*:

```
FROM openjdk:17-jdk-slim
WORKDIR /app
COPY target/NotificationService-0.0.1-SNAPSHOT.jar app.jar
COPY dockerize /usr/local/bin/dockerize
RUN chmod +x /usr/local/bin/dockerize
EXPOSE 8081
# Komenda startowa: czekaj na dostępność bazy danych na porcie 3306 przed
uruchomieniem aplikacji
ENTRYPOINT ["dockerize", "-wait", "tcp://weather-db:3306", "-timeout",
"60s", "java", "-jar", "app.jar"]
```

Link do dockerize: https://github.com/jwilder/dockerize/releases/download/v0.8.0/dockerize-linux-amd64-v0.8.0.tar.gz

- Zmodyfikuj plik *docker-compose.yml*:

```
services:
  weather-db:
    image: mysql:5.7
    container_name: weather-db
    restart: unless-stopped
    environment:
      MYSQL_ROOT_PASSWORD: root
      MYSQL_DATABASE: weather
      MYSQL_USER: weatheruser
      MYSQL_PASSWORD: p@ssw0rd
    ports:
      - "3306:3306"
    volumes:
      - weather-db-data:/var/lib/mysql

  weather-service:
    build:
      context: ./WeatherDataService
      dockerfile: Dockerfile
    container_name: weather-service
    environment:
      SPRING_DATASOURCE_URL: jdbc:mysql://weather-db:3306/weather
      SPRING_DATASOURCE_USERNAME: weatheruser
      SPRING_DATASOURCE_PASSWORD: p@ssw0rd
      SPRING_JPA_HIBERNATE_DDL_AUTO: update
    ports:
      - "8080:8080"
    depends_on:
      - weather-db

  notification-service:
    build:
      context: ./NotificationService
      dockerfile: Dockerfile
    container_name: notification-service
    environment:
      SPRING_DATASOURCE_URL: jdbc:mysql://weather-db:3306/weather
      SPRING_DATASOURCE_USERNAME: weatheruser
      SPRING_DATASOURCE_PASSWORD: p@ssw0rd
      SPRING_JPA_HIBERNATE_DDL_AUTO: update
      SPRING_MAIL_HOST: smtp.mailtrap.io
      SPRING_MAIL_PORT: 587
      SPRING_MAIL_USERNAME: 31aafe68f86106
      SPRING_MAIL_PASSWORD: 230bdd9bdf3ef2
```

```
    ports:
      - "8081:8080"
    depends_on:
      - weather-db

volumes:
  weather-db-data:
```

- Uruchomienie:

```
docker-compose up --build
```

8. **W celu przetestowania działania mikroserwisu w kontenerze pobierz aplikacje Postman:**

```
sudo snap install postman
```

9. **Uruchom aplikacje Postman**

Utwórz nową kolekcję o nazwie *Weather Notifications* oraz dodaj zapytanie *Send Notification* typu POST. Adres to: http://localhost:8081/notifications. W zakładce *params* dodaj nowy klucz *Content-Type* o wartości *application/json*.

Następnie w zakładce body wybierz raw i ustaw na JSON:

```
{
  "email": "pawplo003@pbs.edu.pl",
  "subject": "Rain Alert",
  "message": "Rain is expected tomorrow. Don't forget your umbrella."
}
```

10. **Kliknij Send i zweryfikuj poprawność zapytania w aplikacji Postman.**

11. **Uruchom terminal i zweryfikuj poprawność danych**

```
docker exec -it weather-db mysql -u weatheruser -p
USE weather;
SHOW TABLES;
SELECT * FROM notification;
exit
```

12. **Zweryfikuj wysłane powiadomienie na stronie mailtrap.io w zakładce Email Testing -> Inboxes -> My Inbox .**