1. **Utworzenie projektu aplikacji *webapp* w Spring Boot za pomocą Spring Initializr**
   - Ustaw parametry projektu:
     1) **Project**: *Maven Project*
     2) **Language**: *Java*
     3) **Spring Boot Version**: *3.3.7*
     4) **Group**: *com.pbs.edu*
     5) **Artifact**: *webapp*
     6) **Packaging**: *Jar*
     7) **Java**: *17*
   - Dodanie zależności:
     1) Spring Web
     2) Spring Data JPA
     3) MySQL Driver
     4) Lombok
     5) Spring Security
     6) Spring Boot DevTools
     7) Thymeleaf
     8) Prometheus
   - Pobierz projekt i rozpakuj plik ZIP, a następnie otwórz go w wybranym IDE (np. IntelliJ IDEA).

2. **Konfiguracja projektu**
   - Przejdź do folderu *src/main/resources* i otwórz plik *application.properties*.
   - Dodaj konfiguracje mikroserwisów oraz bazy danych itd.:

```
spring.application.name=webapp

server.port=8083

# Mikroserwisy
weather.service.url=http://weather-service:8080
notification.service.url=http://notification-service:8081
user.service.url=http://user-service:8082

# Prometheus
management.endpoints.web.exposure.include=*
management.endpoint.health.show-details=always

# Thymeleaf
spring.thymeleaf.prefix=classpath:/templates/
spring.thymeleaf.suffix=.html
spring.thymeleaf.mode=HTML
spring.thymeleaf.encoding=UTF-8
spring.thymeleaf.cache=false
```

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

- Utwórz w *com.pbs.edu* folder config, a w nim pliki:
    - *RestTemplateConfig.java*

```
package com.pbs.edu.webapp.config;

import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import org.springframework.web.client.RestTemplate;

@Configuration
public class RestTemplateConfig {

    @Bean
    public RestTemplate restTemplate() {
        return new RestTemplate();
    }
}
```

    - *SchedulerConfig.java*

```
package com.pbs.edu.webapp.config;

import org.springframework.context.annotation.Configuration;
import org.springframework.scheduling.annotation.EnableScheduling;

@Configuration
@EnableScheduling
public class SchedulerConfig {
}
```

3. Implementacja *webapp*

- Zmodyfikowanie modelu *WeatherData* w mikroserwisie *WeatherDataService* poprzez dodanie pola *private LocalDateTime date*;
- Zmodyfikowanie modelu *Notification* w mikroserwisie *NotificationService* poprzez dodanie pola *private LocalDateTime timestamp*;
- Utworzenie modeli *User*, *WeatherData* oraz *NotificationRequest* zgodnie z modelami z mikroserwisów utworzonych w poprzednich ćwiczeniach.

4. **Modyfikacja kodu niektórych plików mikroserwisów w celu dodania obsługi potrzebnych zapytań z *webapp*:**

- *NotificationController.java*

```
@GetMapping("/by-email")
public List<Notification> getNotificationsByEmail(@RequestParam("email")
String email) {
    return repository.findByEmail(email);
}
```

- *NotificationRepository.java*

```
public interface NotificationRepository extends
JpaRepository<Notification, Long> {
    List<Notification> findByEmail(String email);
}
```

- *UserService.java*

```
public boolean checkPass(String passwordGiven, String passwordExpected) {
    return passwordEncoder.matches(passwordGiven, passwordExpected);
}

public User getUserById(Long id) {
    return userRepository.findById(id).orElse(null);
}

public void updateUser(User user) {
    userRepository.save(user);
}
```

- *UserRepository.java*

```
public interface UserRepository extends JpaRepository<User, Long> {
    User findByEmail(String email);
}
```

- *UserController.java*

```
@GetMapping("/{id}")
public ResponseEntity<User> getUserById(@PathVariable Long id) {
    User user = userService.getUserById(id);
    if (user == null) {
        return ResponseEntity.status(HttpStatus.NOT_FOUND).build();
    }
    return ResponseEntity.ok(user);
}

@PutMapping("/{id}")
public ResponseEntity<User> updateUser(@PathVariable Long id,
@RequestBody User updatedUser) {
```

```
    User existingUser = userService.getUserById(id);
    if (existingUser == null) {
        return ResponseEntity.status(HttpStatus.NOT_FOUND).build();
    }
    existingUser.setCity(updatedUser.getCity());
    existingUser.setCountry(updatedUser.getCountry());
    existingUser.setRainNotification(updatedUser.isRainNotification());
    existingUser.setSnowNotification(updatedUser.isSnowNotification());
    existingUser.setWindNotification(updatedUser.isWindNotification());

existingUser.setHighTempNotification(updatedUser.isHighTempNotification()
);

existingUser.setLowTempNotification(updatedUser.isLowTempNotification());

    userService.updateUser(existingUser);
    return ResponseEntity.ok(existingUser);
}

@Autowired
private UserRepository userRepository;

@PostMapping("/authenticate")
public ResponseEntity<User> authenticateUser(@RequestParam String email,
@RequestParam String password) {
    User user = userRepository.findByEmail(email);
    if(user != null && userService.checkPass(password,
user.getPassword())) {
        return ResponseEntity.ok(user);
    }
    return ResponseEntity.status(HttpStatus.UNAUTHORIZED).build();
}
```

- *WeatherService.java*

```
public List<WeatherData> getWeatherHistory(String city, String country) {
    return weatherDataRepository.findByCityNameAndCountry(city, country);
}
```

- *WeatherDataRepository.java*

```
public interface WeatherDataRepository extends JpaRepository<WeatherData,
Long> {
    List<WeatherData> findByCityNameAndCountry(String city, String
country);
}
```

- *WeatherController.java*

```
@GetMapping("/weather/history")
public List<WeatherData> getWeatherHistory(
        @RequestParam(name = "city", required = false) String city,
        @RequestParam(name = "country", required = false) String country
) {
    if (city != null && country != null) {
```

```
        return weatherService.getWeatherHistory(city, country);
    } else {
        return weatherService.getWeatherHistory();
    }
```

5. **Utworzenie w aplikacji *webapp* poszczególnych serwisów do wysyłania zapytań do mikroserwisów z poprzednich laboratoriów.**

- *AuthService.java*

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

import com.pbs.edu.webapp.model.User;
import org.springframework.beans.factory.annotation.Value;
import org.springframework.http.ResponseEntity;
import org.springframework.stereotype.Service;
import org.springframework.web.client.RestTemplate;

import jakarta.servlet.http.HttpSession;

@Service
public class AuthService {

    private final RestTemplate restTemplate;
    @Value("${user.service.url}")
    private String userServiceUrl;


    public AuthService(RestTemplate restTemplate) {
        this.restTemplate = restTemplate;
    }

    public User authenticate(String email, String password, HttpSession
session) {
        String url = userServiceUrl + "/users/authenticate?email=" +
email + "&password=" + password;
        ResponseEntity<User> response = restTemplate.postForEntity(url,
null, User.class);

        if (response.getStatusCode().is2xxSuccessful()) {
            User user = response.getBody();
            session.setAttribute("user", user);
            return user;
        }
        throw new RuntimeException("Invalid credentials");
    }
}
```

- *NotificationServiceClient.java*

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

import com.pbs.edu.webapp.model.NotificationRequest;
import org.springframework.stereotype.Service;
import org.springframework.beans.factory.annotation.Value;
import org.springframework.web.client.RestTemplate;
```

```
import java.time.LocalDateTime;
import java.util.Arrays;
import java.util.List;

@Service
public class NotificationServiceClient {
    @Value("${notification.service.url}")
    private String notificationServiceUrl;

    private final RestTemplate restTemplate;

    public NotificationServiceClient(RestTemplate restTemplate) {
        this.restTemplate = restTemplate;
    }

    public void sendNotification(String email, String subject, String
message, LocalDateTime timestamp, boolean sent) {
        String url = notificationServiceUrl + "/notifications";
        NotificationRequest request = new NotificationRequest(email,
subject, message, timestamp, sent);
        restTemplate.postForObject(url, request, Void.class);
    }

    public List<NotificationRequest> getNotificationsForUser(String
email) {
        String url = notificationServiceUrl + "/notifications/by-
email?email=" + email;
        NotificationRequest[] notifications =
restTemplate.getForObject(url, NotificationRequest[].class);
        return Arrays.asList(notifications);
    }
}
```

- *UserServiceClient.java*

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

import com.pbs.edu.webapp.model.User;
import org.springframework.beans.factory.annotation.Value;
import org.springframework.stereotype.Service;
import org.springframework.web.client.RestTemplate;

import java.util.Arrays;
import java.util.List;

@Service
public class UserServiceClient {

    @Value("${user.service.url}")
    private String userServiceUrl;

    private final RestTemplate restTemplate;

    public UserServiceClient(RestTemplate restTemplate) {
        this.restTemplate = restTemplate;
    }

    public void createUser(User user) {
```

```
        String url = userServiceUrl + "/users";
        restTemplate.postForObject(url, user, Void.class);
    }

    public List<User> getAllUsers() {
        String url = userServiceUrl + "/users";
        return List.of(restTemplate.getForObject(url, User[].class));
    }

    public User getUserById(Long id) {
        String url = userServiceUrl + "/users/" + id;
        return restTemplate.getForObject(url, User.class);
    }

    public void updateUser(User user) {
        String url = userServiceUrl + "/users/" + user.getId();
        restTemplate.put(url, user);
    }
}
```

- *WeatherServiceClient.java*

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

import com.pbs.edu.webapp.model.WeatherData;
import org.springframework.beans.factory.annotation.Value;
import org.springframework.stereotype.Service;
import org.springframework.web.client.RestTemplate;

import java.util.List;

@Service
public class WeatherServiceClient {

    @Value("${weather.service.url}")
    private String weatherServiceUrl;

    private final RestTemplate restTemplate;

    public WeatherServiceClient(RestTemplate restTemplate) {
        this.restTemplate = restTemplate;
    }

    public WeatherData getCurrentWeather(String city, String country) {
        String url = weatherServiceUrl + "/weather?city=" + city +
"&countryCode=" + country;
        return restTemplate.getForObject(url, WeatherData.class);
    }

    public List<WeatherData> getHistoricalWeather(String city, String
country) {
        String url = weatherServiceUrl + "/weather/history?city=" + city
+ "&country=" + country;
        return List.of(restTemplate.getForObject(url,
WeatherData[].class));
    }
}
```

6. **Utworzenie *WeatherNotificationScheduler***

```
package com.pbs.edu.webapp.scheduler;

import com.pbs.edu.webapp.model.User;
import com.pbs.edu.webapp.model.WeatherData;
import com.pbs.edu.webapp.service.NotificationServiceClient;
import com.pbs.edu.webapp.service.UserServiceClient;
import com.pbs.edu.webapp.service.WeatherServiceClient;
import org.springframework.scheduling.annotation.Scheduled;
import org.springframework.stereotype.Component;

import java.time.LocalDateTime;
import java.util.List;

@Component
public class WeatherNotificationScheduler {

    private final UserServiceClient userService;
    private final WeatherServiceClient weatherService;
    private final NotificationServiceClient notificationService;

    public WeatherNotificationScheduler(UserServiceClient userService,
WeatherServiceClient weatherService, NotificationServiceClient
notificationService) {
        this.userService = userService;
        this.weatherService = weatherService;
        this.notificationService = notificationService;
    }

    @Scheduled(fixedRate = 30000)
    public void sendWeatherNotifications() {
        List<User> users = userService.getAllUsers();

        for (User user : users) {
            WeatherData weatherData =
weatherService.getCurrentWeather(user.getCity(), user.getCountry());

            if (shouldNotify(user, weatherData)) {
                notificationService.sendNotification(user.getEmail(),
"Weather Alert", generateMessage(weatherData), LocalDateTime.now(),
false);
            }
        }
    }

    private boolean shouldNotify(User user, WeatherData weatherData) {
        boolean notifyForRain = user.isRainNotification() &&
"Rain".equalsIgnoreCase(weatherData.getMain());
        boolean notifyForSnow = user.isSnowNotification() &&
"Snow".equalsIgnoreCase(weatherData.getMain());
        boolean notifyForWind = user.isWindNotification() &&
weatherData.getWindSpeed() > 20;
        boolean notifyForHighTemp = user.isHighTempNotification() &&
weatherData.getTemp() > 30;
        boolean notifyForLowTemp = user.isLowTempNotification() &&
weatherData.getTemp() < -5;

        return notifyForRain || notifyForSnow || notifyForWind ||
notifyForHighTemp || notifyForLowTemp;
    }
```

```
    private String generateMessage(WeatherData weatherData) {
        return "Current weather: " + weatherData.getMain() + ", " +
weatherData.getDescription();
    }
}
```

7. **Utworzenie kontrolerów obsługujących poszczególne strony**

- *HomeController.java*

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

import jakarta.servlet.http.HttpSession;
import org.springframework.stereotype.Controller;
import org.springframework.ui.Model;
import org.springframework.web.bind.annotation.GetMapping;

@Controller
public class HomeController {
    @GetMapping("/")
    public String home(HttpSession session, Model model) {
        model.addAttribute("user", session.getAttribute("user"));
        return "index";
    }
}
```

- *AuthController.java*

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

import com.pbs.edu.webapp.model.User;
import com.pbs.edu.webapp.service.AuthService;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Controller;
import org.springframework.ui.Model;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.ModelAttribute;
import org.springframework.web.bind.annotation.PostMapping;

import jakarta.servlet.http.HttpSession;

@Controller
public class AuthController {

    @Autowired
    private AuthService authService;

    @GetMapping("/login")
    public String showLoginForm(Model model) {
        model.addAttribute("user", new User());
        return "login";
    }

    @PostMapping("/login")
    public String login(@ModelAttribute("user") User user, HttpSession
session, Model model) {
```

```
        try {
            User authenticatedUser =
authService.authenticate(user.getEmail(), user.getPassword(), session);
            session.setAttribute("user", authenticatedUser);
            return "redirect:/";
        } catch (RuntimeException e) {
            model.addAttribute("error", "Invalid email or password");
            return "login";
        }
    }

    @GetMapping("/logout")
    public String logoutGet(HttpSession session) {
        session.invalidate();
        return "redirect:/login";
    }

    @PostMapping("/logout")
    public String logoutPost(HttpSession session) {
        session.invalidate();
        return "redirect:/login";
    }
}
```

- *UserController.java*

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

import com.pbs.edu.webapp.model.User;
import com.pbs.edu.webapp.service.UserServiceClient;
import jakarta.servlet.http.HttpSession;
import org.springframework.stereotype.Controller;
import org.springframework.ui.Model;
import org.springframework.web.bind.annotation.*;

@Controller
public class UserController {

    private final UserServiceClient userService;

    public UserController(UserServiceClient userService) {
        this.userService = userService;
    }

    @GetMapping("/register")
    public String showRegistrationForm(Model model) {
        model.addAttribute("user", new User());
        return "register";
    }

    @PostMapping("/register")
    public String registerUser(@ModelAttribute User user) {
        userService.createUser(user);
        return "redirect:/login";
    }

    @GetMapping("/settings")
    public String showSettingsForm(Model model, HttpSession session) {
        User authenticatedUser = (User) session.getAttribute("user");
```

```
        if (authenticatedUser == null) {
            return "redirect:/login";
        }
        User user = userService.getUserById(authenticatedUser.getId());
        model.addAttribute("user", user);
        return "settings";
    }

    @PostMapping("/settings")
    public String updateSettings(@ModelAttribute User user, HttpSession
session) {
        User authenticatedUser = (User) session.getAttribute("user");
        if (authenticatedUser == null ||
!authenticatedUser.getId().equals(user.getId())) {
            return "redirect:/login";
        }
        userService.updateUser(user);
        session.setAttribute("user",
userService.getUserById(user.getId()));
        return "redirect:/settings";
    }

}
```

8. Utworzenie plików html będących templateami oraz pliku css

- *index.html*

```
<!DOCTYPE html>
<html xmlns:th="http://www.thymeleaf.org">
<head>
    <title>Home Page</title>
    <link th:href="@{/motyw.css}" rel="stylesheet">
</head>
<body>
<header>
    <!--<h1>Welcome to Weather Application!</h1>-->
</header>
<nav>
    <ul class="nav-menu">
        <li><a href="/">Home</a></li>
        <li><a href="/weather">Current Weather</a></li>
        <li><a href="/history">Historical Weather</a></li>
        <li><a href="/settings">Settings</a></li>
        <li><a href="/notifications">Notifications</a></li>
        <li></li>
        <li></li>
        <li></li>
        <li></li>
        <li th:if="${user == null}"><a href="/register">Register</a></li>
        <li th:if="${user == null}"><a href="/login">Login</a></li>
        <li th:if="${user != null}"><p th:text="${user.email}"></p></li>
        <li th:if="${user != null}"><a href="/logout">Logout</a></li>
    </ul>
</nav>
<br>
<h1>Welcome to Weather Application!</h1>
</body>
</html>
```

- *register.html*

```
<!DOCTYPE html>
<html xmlns:th="http://www.thymeleaf.org">
<head>
    <title>Register</title>
    <link th:href="@{/motyw.css}" rel="stylesheet">
</head>
<body>
<nav>
    <ul class="nav-menu">
        <li><a href="/">Home</a></li>
        <li><a href="/weather">Current Weather</a></li>
        <li><a href="/history">Historical Weather</a></li>
        <li><a href="/settings">Settings</a></li>
        <li><a href="/notifications">Notifications</a></li>
        <li></li>
        <li></li>
        <li></li>
        <li></li>
        <li><a href="/login">Login</a></li>
    </ul>
</nav>
<br>
<h1>Register</h1>
<form action="/register" method="post">
    Email: <input type="email" name="email" required><br>
    Password: <input type="password" name="password" required><br>
    Country: <input type="text" name="country" required><br>
    City: <input type="text" name="city" required><br>
    <br>
    <button type="submit">Register</button>
</form>
</body>
</html>
```

- *login.html*

```
<!DOCTYPE html>
<html xmlns:th="http://www.thymeleaf.org">
<head>
    <title>Login</title>
    <link th:href="@{/motyw.css}" rel="stylesheet">
</head>
<body>
<nav>
    <ul class="nav-menu">
        <li><a href="/">Home</a></li>
        <li><a href="/weather">Current Weather</a></li>
        <li><a href="/history">Historical Weather</a></li>
        <li><a href="/settings">Settings</a></li>
        <li><a href="/notifications">Notifications</a></li>
        <li></li>
        <li></li>
        <li></li>
        <li></li>
```

```
        <li><a href="/register">Register</a></li>
    </ul>
</nav>
<br>
<h1>Login</h1>
<form th:action="@{/login}" method="post">
    <label for="email">Email:</label>
    <input type="email" id="email" name="email" required>
    <br>
    <label for="password">Password:</label>
    <input type="password" id="password" name="password" required>
    <br><br>
    <button type="submit">Log in</button>
    <div id="errorDiv" th:if="${error}">
        <p style="color:red;" th:text="${error}"></p>
    </div>
</form>
</body>
</html>
```

- *settings.html*

```
<!DOCTYPE html>
<html lang="en" xmlns:th="http://www.thymeleaf.org">
<head>
    <title>Settings</title>
    <link th:href="@{/motyw.css}" rel="stylesheet">
</head>
<body>
<nav>
    <ul class="nav-menu">
        <li><a href="/">Home</a></li>
        <li><a href="/weather">Current Weather</a></li>
        <li><a href="/history">Historical Weather</a></li>
        <li><a href="/settings">Settings</a></li>
        <li><a href="/notifications">Notifications</a></li>
        <li></li>
        <li></li>
        <li></li>
        <li></li>
        <li th:if="${user != null}"><p th:text="${user.email}"></p></li>
        <li th:if="${user != null}"><a href="/logout">Logout</a></li>
    </ul>
</nav>
<br>
<h1>User Settings</h1>
<form action="/settings" method="post">
    <input type="hidden" name="id" th:value="${user.id}">
    <input type="hidden" name="email" th:value="${user.email}"
readonly><br>
    City: <input type="text" name="city" th:value="${user.city}"><br>
    Country: <input type="text" name="country"
th:value="${user.country}"><br>
    Notifications: <br>
    <input type="checkbox" name="rainNotification"
th:checked="${user.rainNotification}"> Rain<br>
    <input type="checkbox" name="snowNotification"
th:checked="${user.snowNotification}"> Snow<br>
    <input type="checkbox" name="windNotification"
```

```
th:checked="${user.windNotification}"> Wind<br>
    <input type="checkbox" name="highTempNotification"
th:checked="${user.highTempNotification}"> High Temperature<br>
    <input type="checkbox" name="lowTempNotification"
th:checked="${user.lowTempNotification}"> Low Temperature<br>
    <br>
    <button type="submit">Update</button>
</form>
</body>
</html>
```

- *motyw.css*

```
body, h1, h2, h3, p, ul, li {
    margin: 0;
    padding: 0;
    box-sizing: border-box;
}

body {
    background-color: #121212;
    color: #FFFFFF;
    font-family: 'Arial', sans-serif;
    padding: 20px;
}

h1 {
    color: #FFFFFF;
    margin-bottom: 10px;
    text-align: center;
}
h2, h3 {
    color: #FFFFFF;
    margin-bottom: 10px;
}

nav {
    background-color: #1e1e1e;
    padding: 10px;
}

.nav-menu {
    list-style-type: none;
    padding: 0;
    display: flex;
    justify-content: center;
    background-color: #333;
}

.nav-menu li {
    margin: 0 15px;
}

.nav-menu p {
    padding: 10px;
    display: block;
    font-weight: bold;
    color: #00FFFF;
}
```

```
.nav-menu a {
    color: #f1f1f1;
    text-decoration: none;
    padding: 10px;
    display: block;
}

.nav-menu a:hover {
    background-color: #575757;
    color: white;
}

form {
    background-color: #1E1E1E;
    padding: 20px;
    border-radius: 8px;
    box-shadow: 0 2px 5px rgba(0, 0, 0, 0.5);
    max-width: 400px;
    margin: 20px auto;
}

label {
    display: block;
    margin-bottom: 5px;
    color: #FFFFFF;
}

input[type="text"], input[type="password"], input[type="email"], select,
button {
    width: 100%;
    padding: 6px;
    border: 1px solid #555555;
    border-radius: 4px;
    background-color: #2D2D2D;
    color: #FFFFFF;
    font-size: 16px;
    margin-bottom: 15px;
}

button {
    background-color: #87CEFA;
    color: white;
    border: none;
    padding: 12px;
    border-radius: 4px;
    cursor: pointer;
    font-size: 16px;
    text-align: center;
    transition: background-color 0.3s ease;
}

button:hover {
    background-color: #ADD8E6;
}

#errorDiv {
    display: flex;
    justify-content: center;
    align-items: center;
    height: 100%;
```

```
    width: 100%;
    margin: 0 auto;
    padding: 20px;
    box-sizing: border-box;
}
```

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

- W folderze głównym projektu utwórz plik *Dockerfile*:

```
FROM maven:3.9.9-eclipse-temurin-17 AS builder
WORKDIR /app
COPY pom.xml .
COPY src ./src
RUN mvn clean package -DskipTests
FROM openjdk:17-jdk-slim
WORKDIR /app
COPY --from=builder /app/target/*.jar app.jar
COPY dockerize /usr/local/bin/dockerize
RUN chmod +x /usr/local/bin/dockerize
EXPOSE 8083
# 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

  user-service:
    build:
      context: ./UserService
      dockerfile: Dockerfile
    container_name: user-service
    environment:
      SPRING_SECURITY_USER_NAME: admin
      SPRING_SECURITY_USER_PASSWORD: p@ssw0rd
      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:
      - "8082:8080"
    depends_on:
      - weather-db

  webapp:
    build:
      context: ./webapp
      dockerfile: Dockerfile
    container_name: webapp
    ports:
      - "8083:8083"
    environment:
      WEATHER_SERVICE_URL: http://weather-service:8080
      NOTIFICATION_SERVICE_URL: http://notification-service:8080
      USER_SERVICE_URL: http://user-service:8080
      SPRING_DATASOURCE_URL: jdbc:mysql://weather-db:3306/weather
      SPRING_DATASOURCE_USERNAME: weatheruser
      SPRING_DATASOURCE_PASSWORD: p@ssw0rd
    depends_on:
      - weather-service
      - user-service
      - notification-service

volumes:
  weather-db-data:
```

- Uruchomienie:

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

10. Uruchom przeglądarkę

Przejdź pod adres: http://localhost:8083/ i sprawdź czy aplikacja działa poprawnie.