Delay Management System

System Zarządzania Opóźnieniami Transportu Publicznego

Dokumentacja Techniczna
Sekcja R1: Architektura Backend Java

Zespół HackYeah 2025
October 5, 2025

Contents

1 Wprowadzenie do Architektury Backend

1.1 Opis Systemu

Delay Management System to aplikacja webowa oparta na architekturze Spring Boot, służąca do zarządzania opóźnieniami w transporcie publicznym. System umożliwia pasażerom zgłaszanie problemów, monitorowanie autobusów poprzez integrację z zewnętrznym API oraz zarządzanie danymi użytkowników.

1.2 Stos Technologiczny

- **Framework:** Spring Boot 3.5.6
- **Język:** Java 17
- **Baza Danych:** H2 (in-memory)
- **Bezpieczeństwo:** Spring Security 6
- **API:** RESTful Web Services
- **Template Engine:** Thymeleaf
- **Narzędzia:** Maven, Lombok

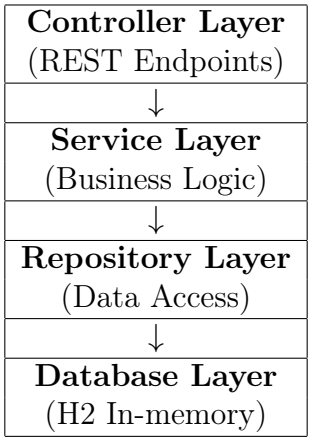
2 Struktura Pakietów i Architektura Warstwowa

2.1 Hierarchia Pakietów

System follows standard Spring Boot package structure with clear separation of concerns:

```
com.tripdelay.delaymanagement/
 config/          # Konfiguracje aplikacji
 controller/      # Warstwa prezentacji - REST API
 service/         # Logika biznesowa
 repository/      # Dostęp do danych
 model/           # Encje i modele danych
 DelayManagementApplication.java
```

2.2 Diagram Architektury Warstwowej



3 Warstwa Kontrolerów (Controller Layer)

3.1 PageController - Routing Stron HTML

```
@Controller
public class PageController {
    @GetMapping("/")      "redirect:/index.html"
    @GetMapping("/login")      "redirect:/login.html"
    @GetMapping("/main")      "redirect:/main_page.html"
    @GetMapping("/report")      "redirect:/report-page.html"
    @GetMapping("/profile")      "redirect:/profile.html"
    @GetMapping("/station")      "redirect:/bus_station.html"
    @GetMapping("/lines")      "redirect:/line-list.html"
    @GetMapping("/map")      "redirect:/mapa.html"
}
```

Listing 1: PageController.java

3.2 BusController - Zarządzanie Autobusami

```
@RestController
@RequestMapping("/api/buses")
public class BusController {
    // GET /api/buses - wszystkie autobusy
    // POST /api/buses - dodaj autobus
    // GET /api/buses/active - aktywne autobusy
    // GET /api/buses/{id} - autobus po ID
    // PUT /api/buses/{id} - aktualizuj autobus
    // DELETE /api/buses/{id} - usu  autobus
    // PUT /api/buses/{id}/location - aktualizuj lokalizacj
    // PUT /api/buses/{id}/status - aktualizuj status
    // GET /api/buses/search - wyszukaj po tablicy rejestracyjnej
}
```

Listing 2: BusController.java

3.3 ReportController - Zarządzanie Zgłoszeniami

```
@RestController
@RequestMapping("/api/reports")
public class ReportController {
    // POST /api/reports - utw rz zg oszenie
    // GET /api/reports - wszystkie zg oszenia
    // POST /api/reports/{id}/verify - weryfikuj zg oszenie
    // POST /api/reports/submit - formularz zg osze  (HTML)
}
```

Listing 3: ReportController.java

3.4 AuthController - Uwierzytelnianie

```
@RestController
@RequestMapping("/api/auth")
public class AuthController {
```

```
    // POST /api/auth/login - logowanie u ytkownika
}
```

Listing 4: AuthController.java

3.5 DelayController - Zarządzanie Opóźnieniami

```
@RestController
@RequestMapping("/api/delays")
public class DelayController {
    // POST /api/delays - utw rz  op   nienie
    // GET /api/delays - op   nienia (z opcjonalnym filtrem route)
}
```

Listing 5: DelayController.java

4 Warstwa Serwisów (Service Layer)

4.1 BusApiService - Integracja z Zewnętrznym API

```
@Service
public class BusApiService {
    private final RestTemplate restTemplate;
    private final String baseUrl; // ${external.api.base-url}

    // Metody komunikacji z zewn trznym API:
    - getAllBuses() - GET /api/Buses
    - createBus(Bus) - POST /api/Buses
    - getActiveBuses() - GET /api/Buses/active
    - getBusById(int) - GET /api/Buses/{id}
    - updateBus(int, Bus) - PUT /api/Buses/{id}
    - deleteBus(int) - DELETE /api/Buses/{id}
    - updateBusLocation(int, BusLocation) - PUT /api/Buses/{id}/location
    - updateBusStatus(int, BusStatus) - PUT /api/Buses/{id}/status
    - searchBuses(String) - GET /api/Buses/search
}
```

Listing 6: BusApiService.java

4.2 ReportService - Logika Biznesowa Zgłoszeń

```
@Service
public class ReportService {
    @Autowired private ReportRepository reportRepository;
    @Autowired private UserService userService;

    public Report createReport(Report report, String username) {
        // Przypisanie u ytkownika do zg oszenia
        // Ustawienie timestamp
        // Zapis do bazy
    }

    public List<Report> getAllReports() { ... }
    public List<Report> getVerifiedReports() { ... }
```

```
    public void verifyReport(Long reportId) { ... }
}
```

Listing 7: ReportService.java

4.3 UserService - Zarządzanie Użytkownikami

```
@Service
public class UserService {
    @Autowired private UserRepository userRepository;
    @Autowired private PasswordEncoder passwordEncoder;

    public User registerUser(User user) {
        // Kodowanie has a przed zapisem
        user.setPassword(passwordEncoder.encode(user.getPassword()));
        return userRepository.save(user);
    }

    public Optional<User> findByUsername(String username) { ... }
    public void updateReputation(User user, int points) { ... }
}
```

Listing 8: UserService.java

4.4 DelayService - Zarządzanie Opóźnieniami

```
@Service
public class DelayService {
    @Autowired private DelayRepository delayRepository;

    public Delay createDelay(Delay delay) { ... }
    public List<Delay> getDelaysByRoute(String route) { ... }
    public List<Delay> getAllDelays() { ... }
}
```

Listing 9: DelayService.java

5 Warstwa Dostępu do Danych (Repository Layer)

5.1 UserRepository - Operacje na Użytkownikach

```
public interface UserRepository extends JpaRepository<User, Long> {
    Optional<User> findByUsername(String username);
}
```

Listing 10: UserRepository.java

5.2 ReportRepository - Operacje na Zgłoszeniach

```
public interface ReportRepository extends JpaRepository<Report, Long> {
    List<Report> findByVerified(boolean verified);
    List<Report> findByLocation(String location);
```

```
}
```

Listing 11: ReportRepository.java

5.3 DelayRepository - Operacje na Opóźnieniach

```
public interface DelayRepository extends JpaRepository<Delay, Long> {
    List<Delay> findByRoute(String route);
}
```

Listing 12: DelayRepository.java

6 Modele Danych (Model Layer)

6.1 Główne Encje

6.1.1 User - Użytkownik

```
@Entity
@Table(name = "users")
public class User {
    private Long id;
    private String username;
    private String password;
    private int reputationPoints;
    // getters/setters
}
```

Listing 13: User.java - Struktura

6.1.2 Report - Zgłoszenie

```
@Entity
@Table(name = "reports")
public class Report {
    private Long id;
    @ManyToOne private User user;
    private ReportType type; // DELAY, DISRUPTION
    private String description;
    private String location;
    private LocalDateTime timestamp;
    private boolean verified;
    private int votes;

    // Pola formularza:
    private Integer lineNumber;
    private CrowdLevel crowdLevel;
    private Integer delayMinutes;
    private VehicleFailure vehicleFailure;
    private AirConditioning airConditioning;
    private Smell smell;
}
```

Listing 14: Report.java - Struktura

6.1.3 Delay - Opóźnienie

```
@Entity
@Table(name = "delays")
public class Delay {
    private Long id;
    private String route;
    private LocalDateTime scheduledTime;
    private LocalDateTime actualTime;
    private String reason;
}
```

Listing 15: Delay.java - Struktura

6.1.4 Bus - Autobus (DTO dla API zewnętrznego)

```
public class Bus {
    private int id;
    private String licensePlate;
    private String model;
    private int capacity;
    private String busNumber;
    private BusLocation location;
    private BusStatus status;

    public static class BusLocation {
        private double latitude;
        private double longitude;
        private double speed;
        private double bearing;
    }

    public enum BusStatus {
        INACTIVE(0), ACTIVE(1), MAINTENANCE(2), OUT_OF_SERVICE(3);
    }
}
```

Listing 16: Bus.java - Struktura

6.2 Enumeracje

```
public enum ReportType { DELAY, DISRUPTION }
public enum CrowdLevel { LOW, MEDIUM, HIGH, URGENT }
public enum VehicleFailure { TEMPORARY, COMPLETE }
public enum AirConditioning { FREEZING, COOL, GOOD, WARM, OVEN }
public enum Smell { FLOWERS, SLIGHT_SMELL, STINKS, VERY_STINKS }
```

Listing 17: Enumeracje systemu

7 Konfiguracja Aplikacji

7.1 SecurityConfig - Bezpieczeństwo

```java
@Configuration
@EnableWebSecurity
public class SecurityConfig {

    @Bean
    public SecurityFilterChain securityFilterChain(HttpSecurity http) {
        http.csrf().disable().cors().and()
            .authorizeHttpRequests(authz -> authz
                .requestMatchers("/api/auth/**").permitAll()
                .requestMatchers("/api/reports/**").permitAll()
                .requestMatchers("/*.html").permitAll()
                .requestMatchers("/css/**", "/js/**").permitAll()
                .anyRequest().authenticated()
            )
            .httpBasic();
    }

    @Bean public PasswordEncoder passwordEncoder() {
        return new BCryptPasswordEncoder();
    }
}
```

Listing 18: SecurityConfig.java

7.2 WebConfig - Konfiguracja CORS

```java
@Configuration
public class WebConfig implements WebMvcConfigurer {
    @Override
    public void addCorsMappings(CorsRegistry registry) {
        registry.addMapping("/api/**")
            .allowedOrigins("http://localhost:3000", "http://localhost
    :8080")
            .allowedMethods("GET", "POST", "PUT", "DELETE")
            .allowedHeaders("*")
            .allowCredentials(true);
    }
}
```

Listing 19: WebConfig.java

8 DelayManagementApplication - Klasa Główna

```java
@SpringBootApplication
public class DelayManagementApplication {

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

    @Bean
    public CommandLineRunner createAnonymousUser(UserService userService
    ) {
        return args -> {
            // Tworzenie u ytkownika anonymous je li nie istnieje
```

```
            if (userService.findByUsername("anonymous").isEmpty()) {
                User anonymous = new User();
                anonymous.setUsername("anonymous");
                anonymous.setPassword("anonymous"); // encoded
                userService.registerUser(anonymous);
            }
        };
    }
}
```

Listing 20: DelayManagementApplication.java

9 Zależności i Konfiguracja Maven

```
<dependencies>
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-security</artifactId>
    </dependency>
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-thymeleaf</artifactId>
    </dependency>
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-web</artifactId>
    </dependency>
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-data-jpa</artifactId>
    </dependency>
    <dependency>
        <groupId>com.h2database</groupId>
        <artifactId>h2</artifactId>
        <scope>runtime</scope>
    </dependency>
    <dependency>
        <groupId>org.projectlombok</groupId>
        <artifactId>lombok</artifactId>
        <optional>true</optional>
    </dependency>
</dependencies>
```

Listing 21: pom.xml - Główne zależności

10 Diagram Relacji Encji

Encja	**Relacje**	**Opis**
User	OneToMany → Report	Jeden użytkownik wiele zgłoszeń
Report	ManyToOne → User	Zgłoszenie należy do użytkownika
Delay	Brak relacji	Niezależne encje opóźnień
Bus	Brak relacji (DTO)	Model danych z API zewnętrznego

11 Wzorce Projektowe i Best Practices

11.1 Wzorce Wykorzystane w Systemie

- **MVC (Model-View-Controller)** - separacja concerns
- **Repository Pattern** - abstrakcja dostępu do danych
- **Service Layer** - enkapsulacja logiki biznesowej
- **Dependency Injection** - zarządzanie zależnościami
- **DTO Pattern** - transfer danych (Bus)

11.2 Best Practices Implementacji

- Użycie Lombok dla redukcji boilerplate code
- Wstrzykiwanie zależności przez konstruktor
- Obsługa błędów przez ResponseEntity
- Enkapsulacja logiki biznesowej w serwisach
- Użycie enum dla stałych wartości

12 Podsumowanie Architektury Backend

Architektura backendu Delay Management System opiera się na sprawdzonych wzorcach Spring Boot z wyraźnym podziałem na warstwy:

- **Warstwa prezentacji** - REST Controllers
- **Warstwa biznesowa** - Services
- **Warstwa danych** - Repositories + JPA Entities
- **Warstwa konfiguracji** - Security, CORS, Beans

System wykorzystuje nowoczesne praktyki programowania w Java i zapewnia solidną podstawę dla dalszego rozwoju funkcjonalności zarządzania transportem publicznym.