Delay Management System

System Zarządzania Opóźnieniami Transportu Publicznego

Dokumentacja Techniczna
Sekcja R2: ASP.NET Web API

Architektura Backend .NET 8.0
Zarządzanie Autobusami i Raportami

Zespół HackYeah 2025
October 5, 2025

Contents

1 Wprowadzenie do ASP.NET Web API

1.1 Opis Komponentu

ASP.NET Web API stanowi drugą część systemu Delay Management System, odpowiedzialną za zarządzanie flotą autobusów, śledzenie ich lokalizacji w czasie rzeczywistym oraz przetwarzanie raportów o stanie pojazdów. API służy jako źródło danych dla aplikacji Java Spring Boot.

1.2 Stos Technologiczny

- **Platforma:** .NET 8.0
- **Framework:** ASP.NET Core Web API
- **Baza Danych:** SQLite z Entity Framework Core 9.0.9
- **Dokumentacja:** Swagger/OpenAPI
- **Serializacja:** JSON
- **Uwierzytelnianie:** CORS z polityką AllowAll

2 Architektura Projektu

2.1 Struktura Plików

```
NikitaWebApiSolution/
 Controllers/
    BusesController.cs
 Models/
    BusModels.cs
    FileName.cs
 Program.cs
 appsettings.json
 launchSettings.json
 NikitaWebApiSolution.csproj
```

2.2 Diagram Architektury

Presentation Layer (BusesController - REST API)
↓
Business Logic Layer (Service Methods w Controller)
↓
Data Access Layer (Entity Framework Core)
↓
Database Layer (SQLite - bustracking.db)

3 Konfiguracja Aplikacji

3.1 Program.cs - Punkt Wejścia

```
var builder = WebApplication.CreateBuilder(args);

// Rejestracja serwis w
builder.Services.AddControllers();
builder.Services.AddEndpointsApiExplorer();
builder.Services.AddSwaggerGen();

// Konfiguracja CORS - pe ny dost p
builder.Services.AddCors(options =>
{
    options.AddPolicy("AllowAll", policy =>
    {
        policy.AllowAnyOrigin()
              .AllowAnyMethod()
              .AllowAnyHeader();
    });
});

// Rejestracja DbContext
builder.Services.AddDbContext<BusDbContext>();

// Nas uchiwanie na wszystkich interfejsach
builder.WebHost.UseUrls("http://*:5041");
```

Listing 1: Program.cs - Konfiguracja aplikacji

3.2 Konfiguracja Kestrel

```
{
  "Kestrel": {
    "Endpoints": {
      "Http": {
        "Url": "http://0.0.0.0:5041"
      }
    }
  }
}
```

Listing 2: appsettings.json - Konfiguracja serwera

3.3 Profil Uruchomieniowy

```
{
  "profiles": {
    "http": {
      "commandName": "Project",
      "launchBrowser": true,
      "launchUrl": "swagger",
      "applicationUrl": "http://localhost:5041",
      "environmentVariables": {
```

```
        "ASPNETCORE_ENVIRONMENT": "Development"
      }
    }
  }
}
```

Listing 3: launchSettings.json - Profile debugowania

4 Warstwa Danych - Entity Framework

4.1 DbContext - Kontekst Bazy Danych

```
public class BusDbContext : DbContext
{
    public DbSet<Bus> Buses { get; set; }
    public DbSet<BusReport> BusReports { get; set; }

    protected override void OnConfiguring(DbContextOptionsBuilder
    optionsBuilder)
    {
        optionsBuilder.UseSqlite("Data Source=bustracking.db");
    }

    protected override void OnModelCreating(ModelBuilder modelBuilder)
    {
        // Unikalny indeks dla tablic rejestracyjnych
        modelBuilder.Entity<Bus>()
            .HasIndex(b => b.LicensePlate).IsUnique();

        // Konwersja enum na int dla bazy danych
        modelBuilder.Entity<Bus>()
            .Property(b => b.Status).HasConversion<int>();
    }
}
```

Listing 4: BusDbContext.cs - Konfiguracja EF Core

4.2 Inicjalizacja Bazy Danych

```
using (var scope = app.Services.CreateScope())
{
    var context = scope.ServiceProvider.GetRequiredService<BusDbContext
    >();

    // Tworzenie bazy danych je li nie istnieje
    var created = context.Database.EnsureCreated();

    // Dodawanie danych testowych
    if (!context.BusReports.Any())
    {
        context.BusReports.AddRange(
            new BusReport { BusNumber = 101, CrowdingLevel =
    CrowdingLevel.High, ... },
            new BusReport { BusNumber = 205, CrowdingLevel =
    CrowdingLevel.Medium, ... }
```

```
        );
        context.SaveChanges();
    }
}
```

Listing 5: Inicjalizacja danych testowych

5 Modele Danych

5.1 Encja Bus - Autobus

```
public class Bus
{
    public int Id { get; set; }

    [Required]
    [StringLength(20)]
    public string LicensePlate { get; set; } = string.Empty;

    [Required]
    [StringLength(50)]
    public string Model { get; set; } = string.Empty;

    [Required]
    [Range(1, 200)]
    public int Capacity { get; set; }

    [Required]
    [StringLength(10)]
    public string BusNumber { get; set; } = string.Empty;

    public double? CurrentLatitude { get; set; }
    public double? CurrentLongitude { get; set; }
    public double? Speed { get; set; }
    public double? Bearing { get; set; }

    [Required]
    public BusStatus Status { get; set; }

    [Required]
    public DateTime LastUpdate { get; set; }
}
```

Listing 6: Bus.cs - Model autobusu

5.2 Encja BusReport - Raport Autobusu

```
public class BusReport
{
    public int Id { get; set; }

    [Required]
    public int BusNumber { get; set; }

```

```
    [Required]
    public CrowdingLevel CrowdingLevel { get; set; }

    public int? DelayMinutes { get; set; }
    public VehicleFailure? VehicleFailure { get; set; }
    public AirConditioning? AirConditioning { get; set; }
    public SmellLevel? SmellLevel { get; set; }

    [StringLength(500)]
    public string? AdditionalComments { get; set; }

    [Required]
    public DateTime ReportDate { get; set; } = DateTime.UtcNow;

    public string? UserIP { get; set; }
}
```

Listing 7: BusReport.cs - Model raportu

6 Enumeracje Systemu

6.1 Statusy Autobusów

```
public enum BusStatus
{
    Active = 0,         // Aktywny w trasie
    Inactive = 1,       // Nieaktywny
    Maintenance = 2,    // W serwisie
    OutOfService = 3    // Wycofany z u ytku
}
```

Listing 8: BusStatus.cs - Statusy pojazdów

6.2 Poziomy Wypełnienia

```
public enum CrowdingLevel
{
    Low = 0,       // Ma e zat oczenie
    Medium = 1,    //  rednie  zat oczenie
    High = 2,      // Du e zat oczenie
    Urgent = 3     // Krytyczne zat oczenie
}
```

Listing 9: CrowdingLevel.cs - Poziomy zatłoczenia

6.3 Dodatkowe Enumeracje

```
public enum VehicleFailure
{
    Temporary = 0,    // Tymczasowa awaria
    Complete = 1      // Ca kowita awaria
}

```

```
public enum AirConditioning
{
    Freezing = 0,   // Lodowato
    Cool = 1,       // Ch odno
    Good = 2,       // Dobrze
    Warm = 3,       // Ciep o
    Oven = 4        // Gor co (piekarnik)
}

public enum SmellLevel
{
    Violets = 0,        // Fio ki (bardzo przyjemny)
    LightSmell = 1,     // Lekko   mierdzi
    Smelly = 2,         // Cuchnie
    VerySmelly = 3      // Bardzo   mierdzi
}
```

Listing 10: Pozostałe enumeracje systemu

7 Kontroler BusesController

7.1 Struktura Kontrolera

```
[ApiController]
[Route("api/[controller]")]
public class BusesController : ControllerBase
{
    private readonly BusDbContext _context;

    public BusesController(BusDbContext context)
    {
        _context = context;
    }

    // Metody API...
}
```

Listing 11: BusesController.cs - Nagłówek kontrolera

7.2 Endpointy Zarządzania Autobusami

Metoda	Endpoint	Opis	HTTP
GetAllBuses	/api/buses	Pobiera wszystkie autobusy	GET
GetActiveBuses	/api/buses/active	Pobiera tylko aktywne autobusy	GET
GetBus	/api/buses/id	Pobiera autobus po ID	GET
CreateBus	/api/buses	Tworzy nowy autobus	POST
UpdateBus	/api/buses/id	Aktualizuje dane autobusu	PUT
DeleteBus	/api/buses/id	Usuwa autobus	DELETE
UpdateBusLocation	/api/buses/id/location	Aktualizuje lokalizację	PUT
UpdateBusStatus	/api/buses/id/status	Aktualizuje status autobusu	PUT
SearchBuses	/api/buses/search	Wyszukuje autobusy po tablicy	GET

7.3 Endpointy Raportów

Metoda	Endpoint	Opis	HTTP
CreateBusReport	/api/buses/report	Tworzy nowy raport	POST
GetAllBusReports	/api/buses/reports	Pobiera wszystkie raporty	GET
GetBusReportsByNumber	/api/buses/busNumber/{busNumber}/reports	Raporty dla konkretnego autobusu	GET
GetBusStatistics	/api/buses/statistics	Szczegółowe statystyki	GET
GetStatisticsSummary	/api/buses/statistics/summary	Podsumowanie statystyk	GET

8 DTO - Data Transfer Objects

8.1 Request DTOs

```
public class CreateBusRequest
{
    [Required(ErrorMessage = "License plate is required")]
    [StringLength(20, ErrorMessage = "License plate cannot exceed 20
   characters")]
    public string LicensePlate { get; set; } = string.Empty;

    [Required(ErrorMessage = "Model is required")]
    [StringLength(50, ErrorMessage = "Model cannot exceed 50 characters
   ")]
    public string Model { get; set; } = string.Empty;

    [Required(ErrorMessage = "Capacity is required")]
    [Range(1, 200, ErrorMessage = "Capacity must be between 1 and 200")]
    public int Capacity { get; set; }

    [Required(ErrorMessage = "Bus number is required")]
    [StringLength(10, ErrorMessage = "Bus number cannot exceed 10
   characters")]
    public string BusNumber { get; set; } = string.Empty;
}
```

Listing 12: CreateBusRequest.cs - Tworzenie autobusu

```
public class UpdateLocationRequest
{
    [Required(ErrorMessage = "Latitude is required")]
    [Range(-90, 90, ErrorMessage = "Latitude must be between -90 and
   90")]
    public double Latitude { get; set; }

    [Required(ErrorMessage = "Longitude is required")]
    [Range(-180, 180, ErrorMessage = "Longitude must be between -180 and
    180")]
    public double Longitude { get; set; }

    [Range(0, 200, ErrorMessage = "Speed must be between 0 and 200 km/h
   ")]
    public double? Speed { get; set; }

```

```
    [Range(0, 360, ErrorMessage = "Bearing must be between 0 and 360
    degrees")]
    public double? Bearing { get; set; }
}
```

Listing 13: UpdateLocationRequest.cs - Aktualizacja lokalizacji

8.2 Response DTOs

```
public class BusResponse
{
    public int Id { get; set; }
    public string LicensePlate { get; set; } = string.Empty;
    public string Model { get; set; } = string.Empty;
    public int Capacity { get; set; }
    public string BusNumber { get; set; } = string.Empty;
    public double? CurrentLatitude { get; set; }
    public double? CurrentLongitude { get; set; }
    public double? Speed { get; set; }
    public double? Bearing { get; set; }
    public string Status { get; set; } = string.Empty;
    public DateTime LastUpdate { get; set; }
}
```

Listing 14: BusResponse.cs - Odpowiedź API

9 Format Odpowiedzi API

9.1 Struktura Sukcesu

```
{
  "success": true,
  "data": {
    "id": 1,
    "licensePlate": "    123777    ",
    "model": "PAZ-3205",
    "capacity": 45,
    "busNumber": "101",
    "currentLatitude": 55.7558,
    "currentLongitude": 37.6173,
    "speed": null,
    "bearing": null,
    "status": "Active",
    "lastUpdate": "2024-01-15T10:30:00Z"
  }
}
```

Listing 15: Przykład pomyślnej odpowiedzi

9.2 Struktura Błędu

```
{
  "success": false,
  "message": "Bus not found",
  "errors": [
    {
      "field": "LicensePlate",
      "message": "License plate is required"
    }
  ]
}
```

Listing 16: Przykład odpowiedzi błędu

10 System Raportów i Statystyk

10.1 Tworzenie Raportów

```
[HttpPost("report")]
public async Task<IActionResult> CreateBusReport([FromBody]
    CreateBusReportRequest request)
{
    if (!ModelState.IsValid)
    {
        return BadRequest(new { success = false, errors = ... });
    }

    var userIP = HttpContext.Connection.RemoteIpAddress?.ToString();

    var report = new BusReport
    {
        BusNumber = request.BusNumber,
        CrowdingLevel = request.CrowdingLevel,
        DelayMinutes = request.DelayMinutes,
        VehicleFailure = request.VehicleFailure,
        AirConditioning = request.AirConditioning,
        SmellLevel = request.SmellLevel,
        AdditionalComments = request.AdditionalComments,
        ReportDate = DateTime.UtcNow,
        UserIP = userIP
    };

    _context.BusReports.Add(report);
    await _context.SaveChangesAsync();

    return Ok(new { success = true, message = "Bus report submitted
    successfully", data = ... });
}
```

Listing 17: CreateBusReport - Logika biznesowa

10.2 Statystyki Zaawansowane

```
public async Task<IActionResult> GetBusStatistics()
{
```

```
    var reports = await _context.BusReports.ToListAsync();

    // Statystyki wype nienia
    var crowdingStats = reports
        .GroupBy(r => r.CrowdingLevel)
        .Select(g => new {
            Level = g.Key.ToString(),
            Count = g.Count(),
            Percentage = Math.Round((double)g.Count() / reports.Count *
   100, 2)
        });

    // Statystyki op   nie
    var delayStats = new {
        AverageDelay = reports.Where(r => r.DelayMinutes.HasValue)
                              .Average(r => r.DelayMinutes) ?? 0,
        MaxDelay = reports.Where(r => r.DelayMinutes.HasValue)
                          .Max(r => r.DelayMinutes) ?? 0
    };

    // Analiza trend w
    var trends = AnalyzeProblematicTrends(reports);

    return Ok(new { success = true, data = new {
        CrowdingStatistics = crowdingStats,
        DelayStatistics = delayStats,
        ProblematicTrends = trends
    }});
}
```

Listing 18: GetBusStatistics - Algorytm statystyk

11 Algorytmy Analizy Danych

11.1 Scoring Problemów

```
private static double CalculateProblemScore(List<BusReport> reports)
{
    double score = 0;

    foreach (var report in reports)
    {
        // Wagi czynnik w problemowych
        score += (int)report.CrowdingLevel * 1.5;    // Wysokie
   wype nienie
        score += (report.DelayMinutes ?? 0) * 0.1;   // D ugie
   op  nienia
        score += report.VehicleFailure.HasValue ? 3 : 0; // Awarie
        score += report.AirConditioning == AirConditioning.Oven ? 2 : 0;
        score += (int)(report.SmellLevel ?? 0) * 1.2; // Nieprzyjemne
   zapachy
    }

    return Math.Round(score / reports.Count, 2); // Normalizacja
}
```

Listing 19: CalculateProblemScore - Algorytm oceny problemów

11.2 Analiza Trendów

```
private static object AnalyzeProblematicTrends(List<BusReport> reports)
{
    var lastWeekReports = reports.Where(r =>
        r.ReportDate >= DateTime.UtcNow.AddDays(-7)).ToList();

    var dailyStats = lastWeekReports
        .GroupBy(r => r.ReportDate.Date)
        .Select(g => new {
            Date = g.Key,
            ReportCount = g.Count(),
            AverageCrowding = Math.Round(g.Average(r => (int)r.
    CrowdingLevel), 2),
            ProblematicReports = g.Count(r => r.CrowdingLevel >=
    CrowdingLevel.High)
        })
        .OrderBy(x => x.Date)
        .ToList();

    var crowdingTrend = dailyStats.Count > 1 ?
        (dailyStats.Last().AverageCrowding - dailyStats.First().
    AverageCrowding) : 0;

    return new {
        AnalysisPeriod = "Last 7 days",
        CrowdingTrend = crowdingTrend > 0 ? "Increasing" : "Decreasing",
        WorstDay = dailyStats.OrderByDescending(x => x.
    ProblematicReports)
                             .FirstOrDefault()?.Date
    };
}
```

Listing 20: AnalyzeProblematicTrends - Analiza czasowa

12 Zależności Projektu

12.1 NikitaWebApiSolution.csproj

```
<Project Sdk="Microsoft.NET.Sdk.Web">
  <PropertyGroup>
    <TargetFramework>net8.0</TargetFramework>
    <Nullable>enable</Nullable>
    <ImplicitUsings>enable</ImplicitUsings>
  </PropertyGroup>

  <ItemGroup>
    <PackageReference Include="Microsoft.EntityFrameworkCore" Version="
    9.0.9" />
    <PackageReference Include="Microsoft.EntityFrameworkCore.Sqlite"
    Version="9.0.9" />
```

```
    <PackageReference Include="Microsoft.EntityFrameworkCore.Tools" Version="8.0.11" />
    <PackageReference Include="Swashbuckle.AspNetCore" Version="6.4.0" />
  </ItemGroup>
</Project>
```

Listing 21: Plik projektu z zależnościami

13 Bezpieczeństwo i CORS

13.1 Konfiguracja Bezpieczeństwa

```
builder.Services.AddCors(options =>
{
    options.AddPolicy("AllowAll", policy =>
    {
        policy.AllowAnyOrigin()    // Dowolna domena
              .AllowAnyMethod()    // Wszystkie metody HTTP
              .AllowAnyHeader();   // Wszystkie nag  wki
    });
});

// W pipeline
app.UseCors("AllowAll");
```

Listing 22: Polityka CORS - AllowAll

13.2 Walidacja Danych

```
[HttpPost]
public async Task<IActionResult> CreateBus([FromBody] CreateBusRequest request)
{
    if (!ModelState.IsValid)
    {
        return BadRequest(new {
            success = false,
            errors = ModelState.Values.SelectMany(v => v.Errors)
        });
    }

    // Sprawdzanie unikalno ci tablicy rejestracyjnej
    if (await _context.Buses.AnyAsync(b => b.LicensePlate == request.LicensePlate))
    {
        return BadRequest(new {
            success = false,
            message = "Bus with this license plate already exists"
        });
    }

    // Logika tworzenia...
}
```

Listing 23: Przykład walidacji w kontrolerze

14 Testowanie i Debugowanie

14.1 Swagger UI

API udostępnia interfejs Swagger pod adresem `/swagger` w trybie development, umożliwiający testowanie wszystkich endpointów.

14.2 Przykładowe Żądania HTTP

```
### Pobierz wszystkie autobusy
GET http://localhost:5041/api/buses

### Utw rz nowy autobus
POST http://localhost:5041/api/buses
Content-Type: application/json

{
  "licensePlate": "TEST123",
  "model": "Solaris Urbino 12",
  "capacity": 85,
  "busNumber": "204"
}

### Zaktualizuj lokalizacj
PUT http://localhost:5041/api/buses/1/location
Content-Type: application/json

{
  "latitude": 52.2297,
  "longitude": 21.0122,
  "speed": 45.5,
  "bearing": 90.0
}
```

Listing 24: Przykładowe żądanie do API

15 Podsumowanie Architektury ASP.NET Web API

ASP.NET Web API stanowi solidny, nowoczesny backend oparty na .NET 8.0, oferujący:

- **Pełne CRUD** dla zarządzania flotą autobusów
- **Śledzenie w czasie rzeczywistym** lokalizacji pojazdów
- **Zaawansowany system raportów** z analizą statystyczną
- **RESTful API** z standaryzowanymi odpowiedziami
- **Automatyczna dokumentacja** przez Swagger

- **Bezpieczna komunikacja** przez CORS
- **Walidacja danych** po stronie serwera

API zostało zaprojektowane z myślą o łatwej integracji z aplikacją Java Spring Boot oraz przyszłymi klientami mobilnymi i webowymi.