SNUS projekat

momci na snusu
Datum: 17. septembar 2025.

Sadržaj

1 Sažetak

Ovaj dokument služi kao dokumentacija za SNUS projekat. Sistem modeluje tri nezavisna temperaturna senzora (S1–S3), svaki sa sopstvenom bazom podataka; koordinator servis koji vrši poravnanje između senzora; i klijenta koji implementira kvorum čitanja. Serverske SOAP krajnje tačke se hostuju uz **CoreWCF**, dok klijent koristi **WCF (System.ServiceModel)**; perzistencija je **EF Core + SQLite**.

Osnova projekta

- Tri senzorska servisa (`ISensorService`) preko SOAP/HTTP (BasicHttpBinding), svaki piše merenja u sopstvenu SQLite bazu na 1–10 s (nasumično).
- Klijent primenjuje kvorum čitanja: rezultat se prihvata ako su najmanje *dva* poslednja očitavanja u okviru tolerancije t oko srednje vrednosti; u suprotnom, pokreće se poravnanje.
- Koordinator servis: periodično (svakog minuta) poravnanje; klijentska čitanja čekaju dok je poravnanje u toku; poslednji red u sve tri tabele postaje isti (prosečna vrednost poslednjih merenja).

2 Specifikacija zadatka (rezime zahteva)

Zahtevi projekta glase ukratko:

1. Napraviti WCF aplikaciju sa **3 senzora** koji na svakih 1–10 s mere temperaturu i upisuju u **odvojene baze**.
2. Napraviti klijenta koji preko WCF-a čita vrednosti i prihvata rezultat samo ako je bar **2 od 3** u intervalu ± 5 oko srednje vrednosti; inače **pokreće poravnanje**.
3. **Svakog minuta**, nezavisno od klijenta, izvršiti poravnanje tako da **poslednja vrednost** u sve tri tabele bude **ista** i jednaka proseku poslednjih merenja; **čitanja čekaju** dok poravnanje traje.
4. **Obrazložiti** kvorum repliku i CAP teoremu u kontekstu projekta.
5. Napisati **detaljnu dokumentaciju** (opis projekta i implementacije).

3 Arhitektura i raspored projekata

3.1 Struktura rešenja

- **Shared.Contracts**: WCF ugovori (`System.ServiceModel`) i DTO tipovi.
- **Sensor.Service**: EF Core entiteti/kontekst; pozadinski *sampler*; implementacija `ISensorService`.
- **Sensor.S1.Host**, **Sensor.S2.Host**, **Sensor.S3.Host**: CoreWCF hostovi koji izlažu `/sensor`.
- **Coordinator.Host**: CoreWCF host koji izlaže `/coord` i sadrži *minute scheduler*.
- **Client.CLI**: konzolni WCF klijent (System.ServiceModel) koji poziva oba servisa.

3.2 Endpointi i portovi

Host	Osnovni URL	Endpoint / Ugovor
Sensor S1	`http://localhost:5011`	`/sensor` (`ISensorService`)
Sensor S2	`http://localhost:5012`	`/sensor` (`ISensorService`)
Sensor S3	`http://localhost:5013`	`/sensor` (`ISensorService`)
Coordinator	`http://localhost:5020`	`/coord` (`ICoordinatorService`)

3.3 Tok izvršavanja

1. Svaki senzor pokreće *sampler* koji na 1–10 s upisuje merenje u sopstvenu bazu (SQLite).
2. Klijent uzima poslednje vrednosti sa S1–S3 i meri kvorum (tolerancija t). Ako su barem 2 od 3 u okviru t oko srednje vrednosti, prihvata se prosek inlajera (merenje unutar zadate tolerancije).
3. Ako nema kvoruma, klijent poziva `ReconcileAsync`. Koordinator zaključava globalno, čita poslednje vrednosti, računa prosek i na svakom senzoru poziva `AppendReconciled` — posle čega je poslednji red u sve tri tabele isti.

4 Contracti servisa i DTO tipovi

4.1 `ISensorService` (System.ServiceModel)

```
using System;
using System.ServiceModel;

namespace Shared.Contracts
{
  [ServiceContract(Name="ISensorService", Namespace="http://tempuri.org/")]
  public interface ISensorService
  {
    [OperationContract(Action="http://tempuri.org/ISensorService/Start", ReplyAction="
    *")]
    void Start();

    [OperationContract(Action="http://tempuri.org/ISensorService/Stop", ReplyAction="*
    ")]
    void Stop();

    [OperationContract(Action="http://tempuri.org/ISensorService/GetLatest",
    ReplyAction="*")]
    double GetLatest();

    [OperationContract(Action="http://tempuri.org/ISensorService/GetSnapshot",
    ReplyAction="*")]
    SensorSnapshot GetSnapshot(TimeSpan lookback);

    [OperationContract(Action="http://tempuri.org/ISensorService/AppendReconciled",
    ReplyAction="*")]
    void AppendReconciled(double value);
  }
}
```

4.2 `ICoordinatorService` (System.ServiceModel)

```
using System.ServiceModel;
using System.Threading.Tasks;

namespace Shared.Contracts
{
  [ServiceContract(Name="ICoordinatorService", Namespace="http://tempuri.org/")]
  public interface ICoordinatorService
  {
    [OperationContract(Action="http://tempuri.org/ICoordinatorService/ReconcileAsync",
     ReplyAction="*")]
    Task<ReconResult> ReconcileAsync();

    [OperationContract(Action="http://tempuri.org/ICoordinatorService/
    IsReconInProgress", ReplyAction="*")]
    bool IsReconInProgress();
  }
}
```

4.3 DTO primeri

```
using System;
using System.Runtime.Serialization;

[DataContract]
public class SensorSnapshot {
  [DataMember(Order=1)] public string SensorId { get; set; } = "";
  [DataMember(Order=2)] public DateTimeOffset From { get; set; }
  [DataMember(Order=3)] public DateTimeOffset To { get; set; }
  [DataMember(Order=4)] public double[] Values { get; set; } = Array.Empty<double>();
  public SensorSnapshot() { }
}

[DataContract]
public class ReconResult {
  [DataMember(Order=1)] public bool Success { get; set; }
  [DataMember(Order=2)] public DateTimeOffset At { get; set; }
  [DataMember(Order=3)] public double AveragedValue { get; set; }
  [DataMember(Order=4)] public string Message { get; set; } = "";
  public ReconResult() { }
  public ReconResult(bool ok, DateTimeOffset at, double avg, string msg)
  { Success = ok; At = at; AveragedValue = avg; Message = msg; }
}
```

5 Model podataka i EF Core

5.1 Šema

```
using System.ComponentModel.DataAnnotations;
using System.ComponentModel.DataAnnotations.Schema;

[Table("SensorReadings")]
public class SensorReading {
  [Key] public int Id { get; set; }
  [Required, MaxLength(8)] public string SensorId { get; set; } = "";
  public DateTimeOffset Timestamp { get; set; }
  public double TemperatureC { get; set; }
  public bool IsReconciled { get; set; }
}
```

5.2 DbContext

```
using Microsoft.EntityFrameworkCore;
using Microsoft.EntityFrameworkCore.Storage.ValueConversion;

public class SensorDbContext : DbContext {
  public SensorDbContext(DbContextOptions<SensorDbContext> options) : base(options) { }

  public DbSet<SensorReading> Readings => Set<SensorReading>();

  protected override void OnModelCreating(ModelBuilder modelBuilder) {
    var tsConverter = new DateTimeOffsetToBinaryConverter();
    modelBuilder.Entity<SensorReading>()
      .Property(r => r.Timestamp)
      .HasConversion(tsConverter);
    modelBuilder.Entity<SensorReading>()
      .HasIndex(r => r.Timestamp);
  }
}
```

Datoteke baza Svaki host čuva SQLite bazu u sopstvenom `./data/` direktorijumu (pored izvršnog fajla). Šema se kreira pri startu (`EnsureCreated()`).

6 Senzorski servisi i sampler

6.1 Pozadinski sampler (po hostu)

```
public class SamplerWorker : BackgroundService {
  private readonly ILogger<SamplerWorker> _log;
  private readonly IServiceProvider _sp;
  private readonly string _sensorId;
  private readonly ISamplingSwitch _switch;
  private readonly Random _rng = new();

  protected override async Task ExecuteAsync(CancellationToken ct) {
    using (var scope = _sp.CreateScope())
      await scope.ServiceProvider.GetRequiredService<SensorDbContext>()
            .Database.EnsureCreatedAsync(ct);
```

```
    while (!ct.IsCancellationRequested) {
      await Task.Delay(TimeSpan.FromSeconds(_rng.Next(1, 11)), ct);
      if (!_switch.Enabled) continue;

      using var scope = _sp.CreateScope();
      var db = scope.ServiceProvider.GetRequiredService<SensorDbContext>();
      var value = 20.0 + (_rng.NextDouble() - 0.5) * 0.8; // blaga varijacija
      db.Readings.Add(new SensorReading {
        SensorId = _sensorId, Timestamp = DateTimeOffset.UtcNow,
        TemperatureC = value, IsReconciled = false
      });
      await db.SaveChangesAsync(ct);
      _log.LogInformation("{SensorId} wrote {Value:F2} C", _sensorId, value);
    }
  }
}
```

6.2 Implementacija servisa

```
public class SensorService : ISensorService {
  private readonly string _sensorId;
  private readonly SensorDbContext _db;
  private readonly ISamplingSwitch _switch;

  public SensorService(string sensorId, SensorDbContext db, ISamplingSwitch
    samplingSwitch) {
    _sensorId = sensorId; _db = db; _switch = samplingSwitch;
  }

  public void Start() => _switch.Enabled = true;
  public void Stop()  => _switch.Enabled = false;

  public double GetLatest() =>
    _db.Readings.OrderByDescending(r => r.Timestamp)
                .Select(r => (double?)r.TemperatureC)
                .FirstOrDefault() ?? 20.0;

  public SensorSnapshot GetSnapshot(TimeSpan lookback) {
    var now = DateTimeOffset.UtcNow; var from = now - lookback;
    var values = _db.Readings.Where(r => r.Timestamp >= from && r.Timestamp <= now)
                             .OrderBy(r => r.Timestamp)
                             .Select(r => r.TemperatureC)
                             .ToArray();
    return new SensorSnapshot { SensorId = _sensorId, From = from, To = now, Values =
    values };
  }

  public void AppendReconciled(double value) {
    _db.Readings.Add(new SensorReading {
      SensorId = _sensorId, Timestamp = DateTimeOffset.UtcNow,
      TemperatureC = value, IsReconciled = true
    });
    _db.SaveChanges();
  }
}
```

7 Koordinator (poravnanje)

7.1 Ponašanje

- **IsReconInProgress**: izlaže globalno stanje poravnanja (klijenti čekaju dok je `true`).
- **ReconcileAsync**: zaštićeno sa `SemaphoreSlim`. Čita poslednje vrednosti S1–S3, računa prosek i upisuje ga u sve tri baze (`AppendReconciled`).
- **ReconcilerWorker**: *HostedService* koji okida poravnanje svake minute.

7.2 Koordinator

```
public class CoordinatorService : ICoordinatorService {
  private readonly ILogger<CoordinatorService> _log;
  private readonly ChannelFactory<ISensorService>[] _sensors;
  private readonly SemaphoreSlim _mutex = new(1, 1);
  private volatile bool _inProgress;

  public CoordinatorService(ILogger<CoordinatorService> log,
                            IEnumerable<ChannelFactory<ISensorService>> sensors) {
    _log = log; _sensors = sensors.ToArray();
    _log.LogInformation("Sensors registered = {Count}", _sensors.Length);
  }

  public bool IsReconInProgress() => _inProgress;

  public async Task<ReconResult> ReconcileAsync() {
    await _mutex.WaitAsync();
    _inProgress = true;
    try {
      var ch1 = _sensors[0].CreateChannel();
      var ch2 = _sensors[1].CreateChannel();
      var ch3 = _sensors[2].CreateChannel();

      var x1 = ch1.GetLatest(); var x2 = ch2.GetLatest(); var x3 = ch3.GetLatest();
      var avg = (x1 + x2 + x3) / 3.0;
      ch1.AppendReconciled(avg); ch2.AppendReconciled(avg); ch3.AppendReconciled(avg);

      ((IClientChannel)ch1).Close(); ((IClientChannel)ch2).Close(); ((IClientChannel)
    ch3).Close();
      return new ReconResult(true, DateTimeOffset.UtcNow, avg, "Reconciled to average
    of latests.");
    } catch (Exception ex) {
      _log.LogError(ex, "Reconcile failed");
      return new ReconResult(false, DateTimeOffset.UtcNow, double.NaN, $"Reconcile
    failed: {ex.Message}");
    } finally {
      _inProgress = false; _mutex.Release();
    }
  }
}
```

8 Klijent (kvorum čitanja)

8.1 Algoritam

Za poslednje vrednosti x_1, x_2, x_3 računamo $\mu = \frac{x_1+x_2+x_3}{3}$. Sa tolerancijom t brojimo inlajere $\{x_i : |x_i - \mu| \leq t\}$. Ako ≥ 2, prihvatamo prosek inlajera; inače čekamo (ako je u toku poravnanje) ili pozivamo `ReconcileAsync`, pa ponovo čitamo.

8.2 Podešavanje tolerancije (bez rekompajliranja)

```
# single-shot (podrazumevano t = 5.0)
dotnet run --project src/Client.CLI

# promena tolerancije i posmatranje u petlji (interval u ms)
dotnet run --project src/Client.CLI -- --tol 0.5 --watch --interval 3000
```

9 Testovi

Dodat je skup testova koji proveravaju: (1) da klijent **čeka** dok je poravnanje u toku (nema čitanja dok koordinator drži "in-progress" zastavicu), (2) pozitivan kvorum (*quorum hit*, bez poravnanja) i (3) *no-quorum* putanju (okida se poravnanje pa se ponovo čita).

9.1 Šta se testira

- **Čekanje tokom poravnanja**: dok je `IsReconInProgress()==true`, klijent ne zove senzore; tek po spuštanju zastavice radi jedno kružno čitanje.
- **Kvorum uspeva (bez poravnanja)**: tri bliske vrednosti i tolerancija t daju najmanje 2/3 inlajera; nema poziva `ReconcileAsync()`.
- **Nema kvoruma (sa poravnanjem)**: divergentne vrednosti i mala tolerancija; klijent poziva `ReconcileAsync()`, sačeka i ponovi čitanje.

9.2 Testni kod

```
using System;
using System.Linq;
using System.Threading;
using System.Threading.Tasks;
using FluentAssertions;
using Shared.Contracts;
using Xunit;

namespace Client.Tests
{
    // Minimalne klijentske fasade
    public interface ISensorServiceClient
    {
        double GetLatest();
        void AppendReconciled(double value);
    }

    public interface ICoordinatorServiceClient
    {
        Task<ReconResult> ReconcileAsync();
```

```
    bool IsReconInProgress();
}

// In-memory fakes
public sealed class FakeSensor : ISensorServiceClient
{
    private readonly Func<double> _get;
    public int ReadCount { get; private set; }
    public FakeSensor(Func<double> get) { _get = get; }
    public double GetLatest() { ReadCount++; return _get(); }
    public void AppendReconciled(double value) { }
}

public sealed class FakeCoordinator : ICoordinatorServiceClient
{
    private readonly TimeSpan _reconDuration;
    private int _reconcileCalls;
    public volatile bool InProgress;
    public int ReconcileCalls => _reconcileCalls;

    public FakeCoordinator(bool startInProgress = false, TimeSpan? reconDuration =
 null)
    {
        InProgress = startInProgress;
        _reconDuration = reconDuration ?? TimeSpan.FromMilliseconds(200);
    }

    public bool IsReconInProgress() => InProgress;

    public async Task<ReconResult> ReconcileAsync()
    {
        Interlocked.Increment(ref _reconcileCalls);
        InProgress = true;
        try
        {
            await Task.Delay(_reconDuration);
            return new ReconResult(true, DateTimeOffset.UtcNow, 0.0, "OK");
        }
        finally { InProgress = false; }
    }
}

public class QuorumAndReconcileTests
{
    private static int SumReads(params ISensorServiceClient[] sensors)
    {
        var total = 0;
        foreach (var s in sensors)
            if (s is FakeSensor fs) total += fs.ReadCount;
        return total;
    }

    private static async Task<(bool accepted, double value, int readsBefore, int
readsAfter, int reconCalls)>
        RunClientOnceAsync(
            ICoordinatorServiceClient coord,
            ISensorServiceClient s1,
            ISensorServiceClient s2,
```

```
            ISensorServiceClient s3,
            double tol,
            CancellationToken ct = default)
    {
        // 1) cekaj dok je poravnanje u toku
        int readsBefore = SumReads(s1, s2, s3);
        while (coord.IsReconInProgress())
            await Task.Delay(25, ct);

        // 2) procitaj tri vrednosti
        var x1 = s1.GetLatest();
        var x2 = s2.GetLatest();
        var x3 = s3.GetLatest();

        var mean = (x1 + x2 + x3) / 3.0;
        var inliers = new[] { x1, x2, x3 }.Where(v => Math.Abs(v - mean) <= tol).
ToArray();

        if (inliers.Length >= 2)
        {
            var accepted = inliers.Average();
            int readsAfterA = SumReads(s1, s2, s3);
            return (true, accepted, readsBefore, readsAfterA, (coord as
FakeCoordinator)?.ReconcileCalls ?? 0);
        }

        // 3) nema kvoruma -> poravnanje
        var _ = await coord.ReconcileAsync();

        // 4) sacekaj da spusti zastavicu
        while (coord.IsReconInProgress())
            await Task.Delay(25, ct);

        // 5) ponovo procitaj
        x1 = s1.GetLatest(); x2 = s2.GetLatest(); x3 = s3.GetLatest();
        var accepted2 = new[] { x1, x2, x3 }.Average();
        int readsAfterB = SumReads(s1, s2, s3);

        return (true, accepted2, readsBefore, readsAfterB, (coord as
FakeCoordinator)?.ReconcileCalls ?? 0);
    }

    [Fact]
    public async Task
Client_Waits_While_Reconcile_In_Progress_No_Reads_Before_Gate_Opens()
    {
        var coord = new FakeCoordinator(startInProgress: true, reconDuration:
TimeSpan.FromMilliseconds(1));
        _ = Task.Run(async () => { await Task.Delay(300); coord.InProgress = false;
 });

        var s1 = new FakeSensor(() => 20.0);
        var s2 = new FakeSensor(() => 20.1);
        var s3 = new FakeSensor(() => 20.2);

        var (accepted, _, readsBefore, readsAfter, reconCalls) =
            await RunClientOnceAsync(coord, s1, s2, s3, tol: 0.5);
```

```
            readsBefore.Should().Be(0);
            readsAfter.Should().Be(3);
            reconCalls.Should().Be(0);
            accepted.Should().BeTrue();
        }

        [Fact]
        public async Task Client_Quorum_Hit_No_Reconcile()
        {
            var coord = new FakeCoordinator(startInProgress: false);
            var s1 = new FakeSensor(() => 20.0);
            var s2 = new FakeSensor(() => 20.1);
            var s3 = new FakeSensor(() => 20.2);

            var (accepted, value, _, readsAfter, reconCalls) =
                await RunClientOnceAsync(coord, s1, s2, s3, tol: 0.5);

            accepted.Should().BeTrue();
            readsAfter.Should().Be(3);
            reconCalls.Should().Be(0);
            value.Should().BeApproximately(20.05, 0.2);
        }

        [Fact]
        public async Task Client_NoQuorum_Triggers_Reconcile_Then_ReReads()
        {
            var coord = new FakeCoordinator(startInProgress: false, reconDuration:
    TimeSpan.FromMilliseconds(150));
            var s1 = new FakeSensor(() => 20.0);
            var s2 = new FakeSensor(() => 22.0);
            var s3 = new FakeSensor(() => 24.0);

            var (accepted, _, _, readsAfter, reconCalls) =
                await RunClientOnceAsync(coord, s1, s2, s3, tol: 0.3);

            reconCalls.Should().Be(1);
            readsAfter.Should().Be(6);
            accepted.Should().BeTrue();
        }
    }
}
```

10 Build i pokretanje

10.1 Preduslovi

Instaliran .NET 8 SDK; SQLite je ugrađen (nema eksternog servera).

10.2 Komande

```
dotnet restore
dotnet build -c Debug

# senzori (tri terminala)
dotnet run --project src/Sensor.S1.Host
dotnet run --project src/Sensor.S2.Host
```

```
dotnet run --project src/Sensor.S3.Host

# koordinator
dotnet run --project src/Coordinator.Host

# klijent
dotnet run --project src/Client.CLI
```

10.3 Pokretanje testova

Testovi se nalaze u `tests/Client.Tests`. Preporučeno je da ih pokrenete nakon build-a:

```
# pokretanje celog test projekta
dotnet test tests/Client.Tests
```

11 Provera rada i inspekcija baza

11.1 GUI alat

Instalirajte *DB Browser for SQLite*, otvorite `s1.db`, `s2.db`, `s3.db` (obično pod `bin/Debug/net8.0/data/`), pogledajte tabelu `SensorReadings` i sortirajte po `Timestamp`. Posle poravnanja poslednji red treba da bude isti u sve tri baze (`IsReconciled=1`).

12 Kvorum i replika

Za tri replike usvajamo čitanje sa kvorumom $R = 2$: dovoljne su dve saglasne vrednosti (unutar tolerancije oko globalne sredine) da bi klijent prihvatio rezultat. Poravnanje je upis sa $W = 3$: prosečna poslednja vrednost se upisuje kao *poslednji* red u sve tri baze. Time obezbeđujemo da je posle poravnanja poslednja vrednost konzistentna i lako proverljiva.

13 CAP teorema (primena na projekat)

Kako postoje nezavisne usluge i mrežne pozive, particije su moguće. U trenutku poravnanja biramo **Konzistentnost** ispred **Dostupnosti** (čitamo tek kada se poravnanje završi). Van tog trenutka sistem nudi visoku dostupnost: svaki senzor može da odgovori, a klijent koristi kvorum čitanja da poveća pouzdanost rezultata. Periodično poravnanje obezbeđuje *eventualnu* konvergenciju, izgleda kao trenutna (atomska) promena poslednjeg reda u sve tri baze..