Projekat: Jednostavni WCF senzori temperature

Kompletan projekat koji sadrži:

- 3 *sensor host* konzolne aplikacije (svaki ima sopstvenu bazu - SQLite fajl)
- 1 *coordinator* (WCF klijent koji svakih 60s poravnava senzore preko WCF-a)
- 1 *klijent* koji čita vrednosti iz najmanje 2 senzora i proverava da li su u okviru ±5 od srednje vrednosti svih merenja; ako nisu, traži poravnanje

Za jednostavnost koristimo WCF (BasicHTTPBinding) i SQLite (jedan fajl po senzoru). Uputstvo ispod objašnjava kako pokrenuti.

Preduslovi

- Windows + Visual Studio (2019/2022) ili .NET Framework 4.7.2
- NuGet paketi za svaki projekat koji rade sa SQLite: `System.Data.SQLite`
- Otvorite jednu konzolnu aplikaciju (SensorHost), jednu konzolnu aplikaciju (Coordinator), jednu konzolnu aplikaciju (Client). Svi koriste isti C# kod baziran na WCF.

Struktura fajlova

- WcfSensors.sln
  - SensorHost (Console App)
  - Coordinator (Console App)
  - ClientApp (Console App)

SensorHost (Console) — šta radi

- Pokreće WCF servise na datom portu
- U pozadini: timer koji nasumično (1-10s) generiše temperaturu i upisuje u lokalnu SQLite bazu (`sensor-X.db`)
- Implementira `IsAligning` i podržava `SetLatest`
- Simple lock: `alignLock` boolean; dok traje poravnanje (SetLatest poziv), `GetLatest` čeka

SensorHost Program.cs (kompletan kod)

```
using System;
using System.IO;
using System.Data.SQLite;
```

```
using System.Collections.Generic;
using System.Linq;
using System.Runtime.Serialization;
using System.ServiceModel;
using System.ServiceModel.Web;
using System.Text;
using System.Threading;
using System.Threading.Tasks;
using System.ServiceModel.Description;

namespace WcfSensorService
{
    [ServiceBehavior(InstanceContextMode = InstanceContextMode.Single,
ConcurrencyMode = ConcurrencyMode.Multiple)]
    public class SensorService : ISensorService
    {
        readonly string _dbPath;
        readonly object _alignLock = new object();
        bool _isAligning = false;

        public SensorService(string dbPath)
        {
            _dbPath = dbPath;
            CreateDb();
        }

        private SQLiteConnection GetDatabaseConnection()
        {
            return new SQLiteConnection($"Data Source={_dbPath};Version=3;");
        }

        private void CreateDb()
        {
            if (!File.Exists(_dbPath))
            {
                SQLiteConnection.CreateFile(_dbPath);
                using (var c = GetDatabaseConnection())
                {
                    c.Open();
                    using (var cmd = c.CreateCommand())
                    {
                        cmd.CommandText = "CREATE TABLE readings(id INTEGER PRIMARY
KEY AUTOINCREMENT, ts DATETIME DEFAULT CURRENT_TIMESTAMP, value REAL);";
                        cmd.ExecuteNonQuery();
                    }
                }

            }
        }

        public double GetLatest()
        {
            //Wait if aligning
            lock (_alignLock)
            {
                while (_isAligning) Monitor.Wait(_alignLock);
            }
```

```
            using (var c = GetDatabaseConnection())
            {
                c.Open();
                using (var cmd = c.CreateCommand())
                {
                    cmd.CommandText = "SELECT value FROM readings ORDER BY id DESC
LIMIT 1";
                    var r = cmd.ExecuteScalar();
                    return r == null ? double.NaN : Convert.ToDouble(r);
                }
            }
        }

        public void SetLatest(double value)
        {
            lock (_alignLock)
            {
                _isAligning = true;
            }

            try
            {
                using (var c = GetDatabaseConnection())
                {
                    c.Open();
                    using (var cmd = c.CreateCommand())
                    {
                        cmd.CommandText = "INSERT INTO readings(value) VALUES(@v);";
                        cmd.Parameters.AddWithValue("@v", value);
                        cmd.ExecuteNonQuery();
                        Console.WriteLine($"[{_dbPath}] poravnjanje! nova
temperatura = {value}");
                    }
                }
            }
            finally
            {
                Thread.Sleep(5000);
                lock (_alignLock)
                {
                    _isAligning = false;
                    Monitor.PulseAll(_alignLock);
                }
            }
        }

        public void StartGenerating()
        {
            var rnd = new
Random((int)DateTimeOffset.UtcNow.ToUnixTimeMilliseconds());
            Task.Run(async () =>
            {
                while (true)
                {
                    await Task.Delay(rnd.Next(1000, 10000)); // 1-10 seconds
                    if (_isAligning)
                        continue;
```

```
                    double temp = Math.Round(15 + rnd.NextDouble() * 20, 2);
                    using (var c = GetDatabaseConnection())
                    {
                        c.Open();
                        using (var cmd = new SQLiteCommand("INSERT INTO
readings(value) VALUES(@v);", c))
                        {
                            cmd.Parameters.AddWithValue("@v", temp);
                            cmd.ExecuteNonQuery();
                        }
                    }
                    Console.WriteLine($"[{_dbPath}] nova temperatura = {temp}");
                }
            });
        }
    }

    public class Program
    {
        static void Main()
        {
            Console.WriteLine("Pokrećem 3 senzora u paralelnim threadovima...");

            // Run 3 service threads
            for (int i = 0; i < 3; i++)
            {
                int port = 9001 + i;
                string dbName = $"sensor-{i + 1}.db";

                int copy = i;
                new Thread(() => StartSensorInstance(port, dbName)).Start();
                Thread.Sleep(1000);
            }

            Console.WriteLine("Svi senzori aktivni! Pritisni ENTER za izlaz...");
            Console.ReadLine();
        }

        static void StartSensorInstance(int port, string dbName)
        {
            var baseAddress = new Uri($"http://localhost:{port}/SensorService");
            var service = new SensorService(dbName);
            var host = new ServiceHost(service, baseAddress);

            host.AddServiceEndpoint(typeof(ISensorService), new BasicHttpBinding(),
"");

            var smb = new ServiceMetadataBehavior { HttpGetEnabled = true };
            host.Description.Behaviors.Add(smb);

            // MEX endpoint
            host.AddServiceEndpoint(
                ServiceMetadataBehavior.MexContractName,
                MetadataExchangeBindings.CreateMexHttpBinding(),
                "mex"
            );

            host.Open();
```

```
            Console.WriteLine($"[Sensor-{port}] sluša na {baseAddress}");
            service.StartGenerating();

            // Block the thread until shutdown
            Thread.Sleep(Timeout.Infinite);
        }
    }
}
```

Coordinator (Console) — radi poravnanje svake minute

- Povezuje se na sva 3 senzora preko WCF-a
- Na svakih 60s preuzme poslednju vrednost sa svakog senzora, izračuna prosek poslednjih merenja (ovde koristimo *poslednju vrednost* svakog senzora kao zadatu), i poziva `SetLatest(avg)` na svaki sensor
- Dok sensor izvršava `SetLatest`, druge operacije `GetLatest` će čekati (prema implementaciji iz senzora)

Coordinator Program.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.ServiceModel;
using System.Threading;
using System.Threading.Tasks;

namespace ServiceCoordinator
{
    class Program
    {
        static async Task Main(string[] args)
        {
            // Kreiramo proxy-je prema servisima
            var sensor1 = new ServiceReference1.SensorServiceClient();
            var sensor2 = new ServiceReference2.SensorServiceClient();
            var sensor3 = new ServiceReference3.SensorServiceClient();

            Console.WriteLine("Koordinator pokrenut. Svakih 60s ažurira temperaturu
na prosek.\n");

            while (true)
            {
                Task<double>[] latestTempTasks = { sensor1.GetLatestAsync(),
sensor2.GetLatestAsync(), sensor3.GetLatestAsync()};
                double[] temps = await Task.WhenAll(latestTempTasks);

                double avg = temps.Average();
                Console.WriteLine($"[Koordinator] Prosek poslednjih merenja:
{avg:F2}°C");
```

```
                Console.Write("[Koordinator] Zapoceto poravnanje");
                Task[] setTempTasks = { sensor1.SetLatestAsync(avg),
sensor2.SetLatestAsync(avg), sensor3.SetLatestAsync(avg) };
                await Task.WhenAll(setTempTasks);
                Console.WriteLine("\r[Koordinator] Temperatura svih senzora
ažurirana na prosek.\n");

                Thread.Sleep(60000);
            }
        }

    }
}
```

ClientApp (Console) — kako radi

- Pita senzore za GetLatest
- Računa ukupni prosek (na osnovu prikupljenih vrednosti iz senzora sistema) i proverava za svaku od dve vrednosti da li su unutar ±5 od proseka
- Ako manje od 2 senzora imaju vrednosti u tom opsegu, poziva poravnanje (može direktno pozvati Coordinator ili zahteva SetLatest od senzora)

ClientApp Program.cs

```
using System;
using System.Runtime.CompilerServices;
using System.Threading.Tasks;

namespace ServiceClient
{
    class Program
    {
        static async Task Main(string[] args)
        {
            var client1 = new ServiceReference1.SensorServiceClient();
            var client2 = new ServiceReference2.SensorServiceClient();
            var client3 = new ServiceReference3.SensorServiceClient();

            Console.WriteLine("Klijent pokrenut. Pritisnite ENTER da vidite
najnovije informacije. Napisite exit da ugasite klijenta.");
            while (true)
            {
                var line = Console.ReadLine();
                if (line == "exit") break;

                try
                {
                    Task<double>[] getLatestTasks = { client1.GetLatestAsync(),
client2.GetLatestAsync(), client3.GetLatestAsync() };


                    Console.Write("Dobavljanje vrednosti...");
                    double[] latest = await Task.WhenAll(getLatestTasks);
                    Console.Write("\r");
```

```
                    double avg = (latest[0] + latest[1] + latest[2]) / 3.0;
                    Console.WriteLine($"Senzor 1: {latest[0]}, Senzor 2:
{latest[1]}, Senzor 3: {latest[2]}, Prosek: {avg}");

                    int inRange = 0;
                    foreach (double d in latest) if (Math.Abs(d - avg) <= 5.0)
inRange++;

                    if (inRange >= 2)
                        Console.WriteLine($"OK, bar 2 senzora su u intervalu ±5 od
proseka");
                    else
                    {
                        Console.WriteLine($"PROBLEM, manje od 2 senzora su u
intervalu od ±5 od proseka");
                        Console.Write($"Vrsi se poravnanje...");
                        Task[] tasks = { client1.SetLatestAsync(avg),
client2.SetLatestAsync(avg), client3.SetLatestAsync(avg) };
                        await Task.WhenAll(tasks);
                        Console.WriteLine($"\rPoravnanje uspesno zavrseno");
                    }
                }
                catch (Exception ex)
                {
                    Console.WriteLine("Greska: " + ex.Message);
                }
            }
        }
    }
}
```

Jednostavne SQL skripte

- `CREATE TABLE readings(id INTEGER PRIMARY KEY AUTOINCREMENT, ts DATETIME DEFAULT CURRENT_TIMESTAMP, value REAL);`
- Baze su fajlovi: `sensor-1.db`, `sensor-2.db`, `sensor-3.db`

Kvorum-bazirana replikacija — kratak pregled i kako pomaže

Kvorum replika (najčešće u distribuiranim bazama) traži da operacije čitanja i pisanja dobiju saglasnost određenog broja čvorova (npr. R čvorova za čitanje i W čvorova za pisanje) iz ukupnog broja N, tako da je obezbeđeno: `R + W > N` kako bi se garantovala konzistentnost.

Iako je naš projekat pojednostavljen (3 nezavisne baze), primenjena ideja bi bila: pisanje nove poravnate vrednosti mora biti primljeno od najmanje W senzora, dok klijent mora čitati od najmanje R senzora. Ako `R + W > 3`, garantuje se da će klijent pročitati najnoviju

poravnatu vrednost. - Primer: N=3, izaberemo W=2, R=2 → svaki zapis poravnanja mora biti upisan na najmanje 2 senzora; svaki klijent čita 2 senzora. Tako se izbegne slučaj da svi čitaju staru vrednost.

CAP teorema — povezanost sa zahtevima

CAP kaže da distribuirani sistem može imati najviše dva od sledećih: Consistency (C), Availability (A), Partition tolerance (P).

U našem projektu: - P (Partition tolerance) — u realnim mrežnim uslovima mora biti prisutna: moguće su mrežne deobe između senzora i klijenata. Dakle P je pretpostavka. - Preostali izbor je između C i A. Projekat zahteva da se tokom poravnanja čitanja čekaju što favorizuje Consistency preko Availability. Dizajn je CP sistem: tokom poravnanja klijenti ne dobijaju dostupnu vrednost, već moraju da čekaju, kako bi rezultat bio konzistentan.