Projektni zadatak - Konzistencija

Ognjen Čavić E2 161/2024

November 26, 2025

Sadržaj

1 Uvod

Potrebno je napraviti sistem u kojem 10 senzora na nasumičnim vremenskim itervalima generišu nasumične vrednosti, dok ”menadžer” vrši njihovo poravnjanje, tj. bira jednu od tih vrednosti koja će biti proglašena glavnom. Ovaj sistem se može opisati kao skup od 3 glavne komponente:

- **Simulator** - Pokreće po jednu nit za svaki senzor. Svaka nit na vremenskom intervalu između 1 i 10 sekundi generiše novu vrednost i upisuje je u sopstvenu tabelu u bazi podataka pomoću *Entity Framework*-a.
- **Menadžer**- Na svakih 60 sekundi vrši poravnanje, tako što bira čitanje senzora koje je najpovoljnije i upisuje u tabele svih senzora. Kriterijum za izbor najbolje vrednosti je najskorije očitavanje u prihvatljivom opsegu $\pm 5^oC$. Ukoliko takvo ne postoji uzima se najskorije.
- **WCF servis** - Definiše način komunikacije između menadžera i senzora.

2 Organizacija rešenja

Rešenje se sastoji od 4 manja projekta, gde svaki predstavlja jednu od prethodno navedenih celina, sa tim da dodati četvrti projekat opisuje način struktuiranja podataka za upis u bazu podataka:

- **ConsistencyService** - WCF servis
- **ConsistencyManager** - menadžer koji vrši poravnanje
- **Sensors** - pokretanje niti koje simuliraju senzore
- **SensorsDatabaseContext** - organizacija baze podataka

2.1 ConsistencyService

2.1.1 ConsistencyService/ConsistencyService.cs

Klasa **TemperatureInfo** opisuje oblik podataka koji se razmenjuju između korisnika servisa i formiraju ga 4 polja:

- **sensor_id** - Identifikacioni broj senzora, broj od 1 do 10
- **temperature** - Vrednost temperature
- **timestamp** - vremenski podatak
- **source** - izvor podatka, tj. ko ga upisuje u bazu, senzor ili menadžer tokom poravnanja

U ovom fajlu su samo još definisani interfejsi servisa. **IPublisher** definiše metod za upis u bazu, dok interfejs **ISubscriber** definiše metod za slanje upita.

2.1.2 ConsistencyService/ConsistencyService.svc.cs

Implementacija **IPublisher** i **ISubscriber** interfejsa unutar **ConsistencyService** klase. Bitno je primetiti da klasa nasleđuje oba interfejsa. Metoda **publishTemp** upisuje zadati **TemperatureInfo**, što se koristi da i menadžer i senzori upisuju u bazu, tako što **source** polje postave na odgovarajuću vrednost. Poslednje očitavanje senzora se dobija pozivom **querySensor** i prosleđivanjem odgovarajućeg rednog broja. Obe ove funkcije vrše odgovarajuću tranformaciju tipova tako da korisnici servisa mogu da pozivaju operacije razmene informacija sa bazom podataka bez poznavanja unutrašnje organizacije u istoj.

2.2 ConsistencyManager

2.2.1 ConsistencyManager/Program.cs

Sadrži dve funkcije, **Main** i **findBest**. **Main** funkcija na svakih 60 sekundi šalje upit bazi podataka za poslednji podatak iz svake tabele, potom pomoću funkcije **findBest** određuje koje od čitanja senzora je bilo najbolje. Nakon toga se izmenjuju polja **source** i **timestamp** tako da se označi da je podatak upisan kao rezultat poravnanja i vreme kada je on upisan, takav podatak se upisuje u svaku od tabela. Potrebno je napomenuti da menadžer instancira *subscribera* i *publishera*, koje koristi za upite i upise u bazu, respektivno.

2.3 Sensors

2.3.1 Sensors/Program.cs

Svrha ovog fajla je da instancira 10 objekata klase **Sensors** i pokreće 10 niti u vidu taskova.

2.3.2 Sensors/Sensor.cs

Definicija klase **Sensor**, koja sadrži 5 ali praktično 4 polja.

- **sensorID** - redni broj senzora
- **_random** - statičko polje za generator nasumičnih brojeva
- **maxTimeDelay** - maksimalni vremenski razmak do generisanja sledeće vrednosti
- **random** - generator koji je u konstrukturu postavljen na statički generator
- **publisherClient** - klijent servisa koji će pisati u bazu podataka

Polje **random** je postavljeno da bude jednako statičkom **_random** zato što ako svaka instanca senzora ima svoj generator, desiće se to da imam 10 generatora nasumičnih brojeva sa vrlo sličnim ”semenom” tj. **seed-om**. To znači da će sekvence generisanja brojeva za sve senzore biti identične zato što se implicitno koristi seed koji zavisi od vremena inicijalizacije.

2.4 SensorsDatabaseContext

2.4.1 SensorsDatabaseContext/DbContext.cs

Definicija kako su organizovani podaci unutar baze podataka. Klasa **DBInfo_Sensor** opisuje kako izgleda jedan red baze, odnosno kako se jedno očitavanje senzora predstavlja unutar baze i sadrži 4 polja:

- **id** - redni broj podatka unutar tabele, nije povezano sa rednim brojem senzora
- **temperature** - temperatura koju je senzor očitao
- **timestamp** - vreme očitavanja
- **source** - ko je upisao podatak, senzor ili menadžer

Pošto je potrebno imati odvojenu tabelu za svaki od senzora, a *Entity Framework* zahteva da svaka tabela ima posebno definisan tip, jednostavno je napravljeno dodatnih 10 klasa **DBInfo_Sensor1**, ... , **DBInfo_ Sensor10** i svaka od njih nasleđuje **DBInfo_Sensor**. **SensorsDBContext** je izveden iz **DbContext** i sadrži 10 polja, gde svako od njih predstavlja po jednu od tabela unutar baze podataka.

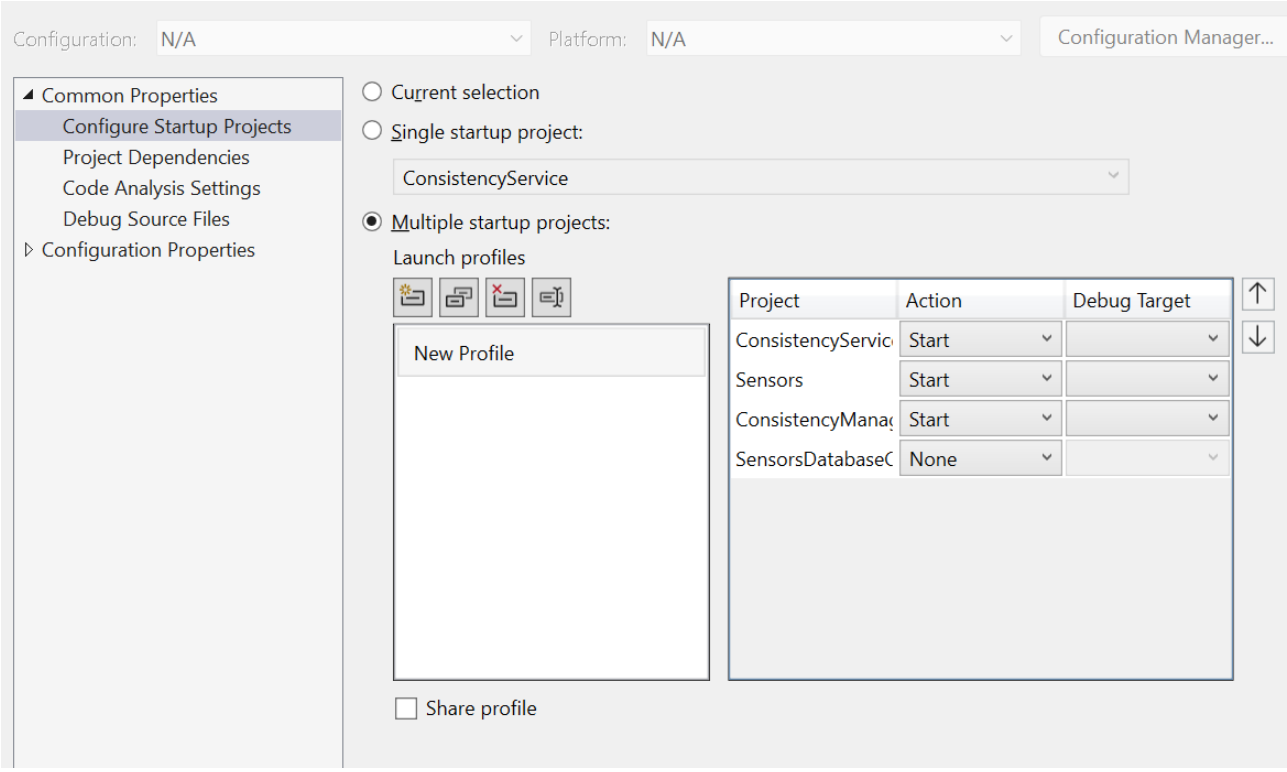
3 Rezultati

3.1 Pokretanje

Za pravilno funkcionisanje napisanog koda, potrebno je pokrenuti projekte (desnim klikom na projekat *Debug→Start new instance*) u sledećem redosledu:

1. **ConsistencyService** - WCF servis
2. **Sensors** - pokretanje niti koje simuliraju senzore
3. **ConsistencyManager** - menadžer koji vrši poravnanje

Jedna olakšica jeste to što visual studio ima opciju da se na start dugme podigne proizvoljan broj projekata u proizvoljnom redosledu. Desnim klikom na naziv celog rešenja tj. **solution-a**, i odabirom opcije properties otvara se prozor gde je potrebno sa leve strane odabrati **Configure Startup Projects** i konfigurisati ga kao na slici:



Slika 1: Podešavanje pokretanja projekta