FER3

Diplomski studij

Raspodijeljeni sustavi

Prva laboratorijska vježba — Prikupljanje i obrada senzorskih podataka

Ak. god. 2024./2025.

1 Uvod

Cilj ove laboratorijske vježbe je izgraditi centralizirani raspodijeljeni sustav temeljen na modelu klijent-poslužitelj za senzoriranje parametara okoliša kako je definirano u odjeljku 2. Često postavljena pitanja nalaze se u odjeljku 5. Tijekom izrade progamskog rješenja naučit ćete koristiti tehnologije gRPC[1] i web-usluge temeljene na prijenosu prikaza stanja resursa (REST). Ova se vježba sastoji od tri dijela:

- proučavanja primjera programskog kôda s predavanja (gRPC i web-usluge),
- oblikovanje i implementacija poslužitelja raspodijeljenog sustava koji održava informacije o senzorima i
- oblikovanje i implementacija klijenta raspodijeljenog sustava koji predstavlja senzor za parametre okoliša.

1.1 Predaja

Studenti samostalno rješavaju programski zadatak i dužni su putem sustava Moodle predati arhivu koja se sastoji od:

- izvornog kôda poslužitelja i
- izvornog kôda klijenta.

Napomena: Komponente poslužitelja i klijenta moraju biti implementirane kao dva odvojena projekta u odabranom razvojnom okruženju (npr. Eclipse, NetBeans, Visual Studio, IntelliJ Idea).

Arhiva se mora predati do roka koji će biti objavljen na stranici predmeta. Studenti koji predaju vježbu nakon navedenog roka dobit će 0 bodova iz vježbe.

Uz predaju digitalne arhive, organizirat će se **usmena prezentacija** predanog rješenja kada će studenti morati objasniti implementaciju i pokazati svoje razumijevanje primijenjenih komunikacijskih tehnologija, a kako bi se rješenje moglo ocijeniti. Termin i provedba usmenog ispita, tj. kolokviranje vježbe, bit će objavljeni na stranici predmeta. Kolokviranje je nužno kako bi predano rješenje bilo ocijenjeno.

Sve komponente sustava mogu se implementirati u bilo kojem programskom jeziku. Imajte na umu da su primjeri prikazani na predavanjima implementirani u programskom jeziku Java. Arhivu koja sadrži izvorni kôd treba nazvati **ime_prezime_jmbag**. Arhiva treba sadržavati 2 direktorija koja sadrže datoteke izvornog kôda (jedan za klijenta i jedan za poslužitelja). Dopušteno je slanje zip arhive koja sadrži izvoz projekta ako je implementiran u IDE-u (Eclipse, NetBeans itd.).

Konzultacije se održavaju **utorkom od 10:00 do 11:00**, uz prethodnu **najavu** na MS Teamsu (kanal Laboratorijske vježbe) ili e-mailom.

2 Arhitektura raspodijeljenog sustava

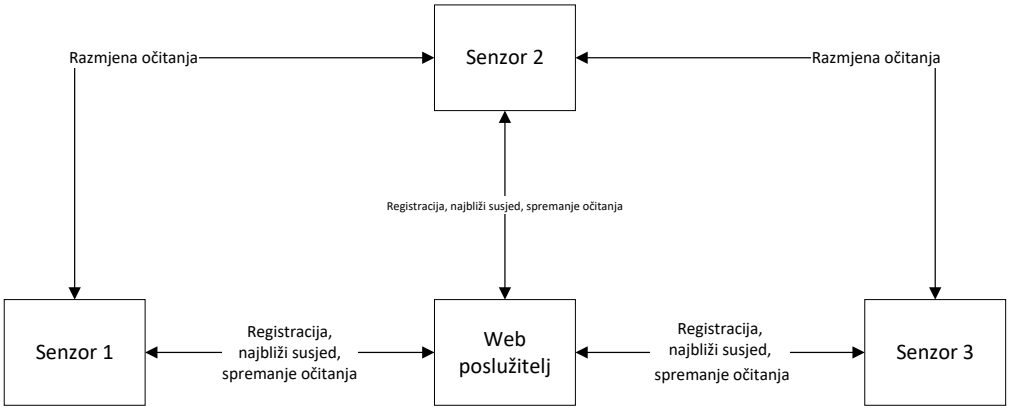
Kratak pregled funkcionalnosti sustava: Raspodijeljeni sustav koristi se za praćenje parametara okoliša u većem geografskom području u koje su postavljeni senzori (svaki senzor ima svoju geo-lokaciju) pomoću poslužitelja implementiranog kao web-usluga i skupine senzora. Arhitektura raspodijeljenog sustava prikazana je na slici 1.

Zadaća senzora je generiranje senzorskih očitanja za parametre zraka, npr. temperature, tlaka, vlage, itd. te razmjena i usporedba vlastitih očitanja s očitanjima geografski najbližeg senzora. U primjeni senzori često mogu generirati pogrešna očitanja, pa ćemo stoga korištenjem vlastitih očitanja

[1] https://grpc.io/

i očitanja s najbližeg susjeda kreirati tzv. kalibrirana očitanja. Programski se klijent raspodijeljenog sustava (u daljnjem tekstu **senzor**) sastoji se od **klijenta web-usluge**, **gRPC klijenta** i **gRPC poslužitelja**. Funkcionalnost senzora i komunikacija između senzora detaljno su objašnjeni u poglavlju 3.

Zadaća poslužitelja je upravljanje podacima o dostupnim registriranim senzorima, njihovom geografskom položaju i kalibriranim očitanjima. Svi senzori komuniciraju s poslužiteljem koji nudi web-usluge temeljene na prijenosu prikaza stanja resursa (REST) (u daljnjem tekstu **poslužitelj**) koristeći **JSON** format. Funkcionalnost poslužitelja i komunikacija između senzora i poslužitelja detaljno su objašnjene u poglavlju 4.



Slika 1: Arhitektura raspodijeljenog sustava

3 Funkcionalnost senzora

Ključne funkcije senzora uključuju mogućnost inicijalizacije i registracije na poslužitelju, generiranje vlastitih očitanja, razmjenu vlastitih očitanja sa svojim geografski najbližim susjedom putem gRPC veze, mogućnost kalibriranja vlastitih i razmijenjenih očitanja te mogućnost pohranjivanja kalibriranih očitanja na poslužitelju. Prilikom implementacije senzora, možete koristiti priloženi kôd na stranicama predmeta kao primjer za razvoj gRPC klijenta i gRPC poslužitelja. Za senzore se može koristiti bilo koji HTTP klijent (npr. Retrofit[2]). Postupak generiranja očitanja, dohvaćanje očitanja od susjeda, kalibriranje očitanja i slanje kalibriranih očitanja na poslužitelj senzor neprestano ponavlja sve dok korisnik ne zaustavi proces.

3.1 Inicijalizacija i registracija

Nakon pokretanja, senzor nasumično odabire geografsko mjesto u rasponu između $[15.87, 16]$ za zemljopisnu dužinu (eng. *longitude*) i $[45.75, 45.85]$ za zemljopisnu širinu (eng. *latitude*). Mjesto se ne mijenja tijekom vijeka trajanja senzora. Senzor se nakon toga mora registrirati na poslužitelju. Poruke razmijenjene tijekom registracije prikazane su u poglavlju 4.1.

3.2 Generiranje senzorskih očitanja

Budući da pravi senzori nisu spojeni na klijente, njihova se funkcija emulira pripremljenim očitanjima iz ulazne datoteke **readings.csv**, gdje očitanja (redci) imaju više vrijednosti (stupaca) koje predstavljaju parametre. Za generiranje očitanja iz ulazne datoteke upotrijebite jednadžbu 1 za dobivanje broja retka koji ćete koristiti za očitavanje temperature, barometarskog tlaka, relativne vlažnosti i koncentracije plinova CO i NO2 ili SO2 iz ulazne datoteke:

[2] https://square.github.io/retrofit/

$$red \leftarrow (brojAktivnihSekundi \ \% \ 100) + 1 \quad (1)$$

Varijabla *brojAktivnihSekundi* je vrijeme rada senzora (vrijeme se mjeri od trenutka pokretanja klijenta i izražava se u sekundama).

3.3 Razmjena očitanja

Nakon što senzor generira vlastito očitanje, mora zatražiti očitanje od svog geografski najbližeg susjeda putem gRPC veze. Za pronalaženje geografski najbližeg susjeda senzor koristi poslužitelja koji pohranjuje sve podatke o senzorima sustava. Izgled poruka koje senzor razmjenjuje s poslužiteljem tijekom ovog procesa definiran je u poglavlju 4.1. Nakon pronalaska najbližeg susjeda, senzor započinje proces razmjene očitanja sa susjedom tako što mu šalje zahtjev za dostavu očitanja (zahtjev definirajte proizvoljno), a susjedni senzor odgovara slanjem vlastitih očitanja (vodite računa o parametrima očitanja). Pretpostavite da se najbliži susjed senzora ne mijenja nakon pokretanja senzora. Nije potrebno tražiti novog najbližeg susjeda ako postojeći najbliži susjed više nije dostupan. Međutim, senzor mora ostati u funkciji odnosno predvidjeti mogućnost da najbliži susjed više nije dostupan. Ako senzor nema najbližeg susjeda, ne kalibrira svoja očitanja, već vlastita očitanja pohranjuje na poslužitelju.

Napomena: Senzori moraju biti u mogućnosti održavati više istovremenih komunikacijskih kanala, tj. komunicirati istodobno s više senzora kojima su potrebna njihova očitanja (tada su u ulozi gRPC poslužitelja) i otvoriti kanal za komunikaciju s maksimalno jednim susjednim senzorom (tada su u ulozi gRPC klijenta). Senzor treba ispisati odgovarajuću poruku na svom sučelju kada se gRPC veza prekine i obraditi iznimku. Tijekom procesa očitanja novih vrijednosti potrebno je ispisati relevantne informacije na sučelju senzora (odnosno koristiti **logging**).

3.4 Kalibriranje očitanja

Senzor koristi vlastita očitanja i očitanja dobivena od susjeda za **kalibraciju**, odnosno senzor potvrđuje kvalitetu vlastitih očitanja uspoređujući ih s očitanjima susjednog senzora. Kalibracija podataka vrši se izračunavanjem prosječne vrijednosti između dva očitanja (prvo je vlastito očitanje, a drugo je očitanje sa susjednog senzora). Tijekom kalibracije izračunava se prosjek za iste parametre očitanja (tj. temperatura, tlak zraka, relativna vlažnost i koncentracije plinova CO, NO2 i SO2).

Napomena: Čvorovi senzora korišteni za izradu ulazne datoteke sadrže samo dva senzora plina (CO, NO2 ili SO2), pa biste trebali biti svjesni mogućeg **odsustva nekih vrijednosti plina prilikom čitanja ulazne datoteke**. Ako neke vrijednosti nedostaju ili ako je izmjerena vrijednost postavljena na 0 (tj. došlo je do pogreške u očitanju senzora), tada upotrijebite samo vlastitu vrijednost za prosjek i obrnuto. Ako i vlastita vrijednost i vrijednost susjeda nedostaju, isključite tu vrijednost iz prosjeka (na primjer, postavite je na null).

3.5 Spremanje očitanja

Nakon što su podaci kalibrirani, senzor šalje kalibrirana očitanja poslužitelju pomoću odgovarajuće metode web-usluge. Točne poruke razmijenjene tijekom spremanja očitanja prikazane su u poglavlju 4.3.

4 Funkcionalnost poslužitelja i komunikacija sa senzorima

Senzori i poslužitelj komuniciraju putem REST sučelja i pomoću JSON formata. Iako se bilo koji jezik može koristiti za implementaciju poslužitelja, predavanja pokrivaju uporabu Spring Boota[3]. Predložak

[3] https://start.spring.io/

za poslužitelja dostupan je u materijalima kolegija. Predavanja o ovim tehnologijama dostupna su na **poveznici**.

4.1 Registracija

Senzor se registrira slanjem registracijskog zahtjeva s relevantnim podacima (njegova geo-lokacija i adresa gRPC poslužitelja) poslužitelju. Primjer podataka za registraciju prikazuje JSON 1. Poslužitelj dodjeljuje identifikator novostvorenom senzoru. Poslužitelj treba vratiti **201 Created** i URL na stvoreni resurs u zaglavlju **HTTP Location**. Ako se senzor nije uspješno registrirao (npr. server nije dostupan), treba prekinuti rad.

Napomena: obratite pažnju na vrstu HTTP zahtjeva, kao i URL na koji se zahtjev šalje.

JSON 1: Primjer zahtjeva za registraciju u JSON formatu

```
{
  "latitude": 45.75,
  "longitude": 15.87,
  "ip": "127.0.0.1",
  "port": 8080
}
```

4.2 Najbliži susjed

Nakon registracije na poslužitelju, senzor može od poslužitelja zatražiti svog geografski najbližeg susjeda. Udaljenost između dva senzora izračunava se pomoću Haversineove formule koja je definirana pseudokôdom 1 (6731 km je radijus Zemlje). Primjer odgovora na zahtjev geografski najbližeg susjeda prikazuje JSON 2.

Pseudokod 1: Haversineova formula

$R \leftarrow 6371$;
$dlon \leftarrow lon2 - lon1$;
$dlat \leftarrow lat2 - lat1$;
$a \leftarrow (\sin(dlat/2))^2 + \cos(lat1) * \cos(lat2) * (\sin(dlon/2))^2$;
$c \leftarrow 2 * atan2(sqrt(a), sqrt(1-a))$;
$d \leftarrow R * c$;

JSON 2: Primjer odgovora na zahtjev za geografski najbližeg susjeda u JSON formatu

```
  {
    "latitude": 45.76,
    "longitude": 15.88,
    "ip": "127.0.0.1",
    "port": 8081
  }
```

4.3 Spremanje očitanja pojedinog senzora

Senzor može pohraniti svoja kalibrirana očitanja na poslužitelju slanjem odgovarajućeg zahtjeva. Poslužitelj treba vratiti **201 Created** i URL na kreirani resurs u zaglavlju HTTP Location headera. Primjer podataka u zahtjevu za spremanje očitanja prikazuje JSON 3. U slučaju da ne postoji senzor sa identifikatorom za kojeg se očitanje sprema, poslužitelj treba vratiti odgovor **204 No Content**.

Napomena: senzor generira više očitanja, ali jedno očitanje pripada jednom senzoru (odnosa među entitetima je $1 : N$). Obratite pažnju na vrstu HTTP zahtjeva, kao i URL na koji se zahtjev šalje.

JSON 3: Primjer zahtjeva za spremanje očitanja u JSON formatu

```json
{
  "temperature": 32.0,
  "pressure": 1109.0,
  "humidity": 40.0,
  "co": 547.0,
  "so2": 18.0
}
```

4.4 Popis senzora

Poslužitelj treba izložiti popis registriranih senzora u JSON formatu. Primjer odgovora na zahtjev za popis senzora prikazuje JSON 4.

JSON 4: Primjer odgovora na zahtjev za popis registriranih senzora u JSON formatu

```json
[
  {
    "id": 1,
    "latitude": 45.75,
    "longitude": 15.87,
    "ip": "127.0.0.1",
    "port": 8080
  },
  {
    "id": 2,
    "latitude": 45.76,
    "longitude": 15.88,
    "ip": "127.0.0.1",
    "port": 8081
  }
]
```

4.5 Popis očitanja pojedinog senzora

Poslužitelj treba izložiti popis očitanja koja pripadaju određenom senzoru. Primjer odgovora na zahtjev za popis očitanja prikazuje JSON 5. U slučaju da ne postoji senzor sa identifikatorom za kojeg se traži popis očitanja, poslužitelj treba vratiti odgovor **204 No Content**.

JSON 5: Primjer odgovora na zahtjev za popis očitanja JSON formatu

```
[
  {
    "id": 4,
    "temperature": 27.0,
    "pressure": 749.5,
    "humidity": 46.5,
    "co": 174.5,
    "no2": 128.0,
  },
  {
    "id": 6,
    "temperature": 28.0,
    "pressure": 749.5,
    "humidity": 46.5,
    "co": 174.5,
    "no2": 125.0,
  }
]
```

5 Često postavljena pitanja

5.1 Smijem li koristiti jezik i/ili razvojno okruženje po svom odabru?

Smijete, uz opasku da će za 3. laboratorijsku vježbu biti obavezno koristi Spring Boot odnosno Javu.

5.2 Koji jezik i/ili razvojno okruženje nam preporučate?

Preporučamo korištenje:

1. Java 21
2. Gradle 8.7
3. org.springframework.boot 3.1.4
4. io.spring.dependency-management 1.0.14.RELEASE

5.3 Kako pokrenuti gRPC primjer?

1. gradle runExampleServer (pokreće SimpleUnaryRPCServer)
2. gradle runExampleClient (pokreće SimpleUnaryRPCClient)

5.4 Postoje li upute o tome kako napraviti POST podataka?

Predavanja o Springu i REST-u možete naći:

1. U sklopu on-line predavanja[4]
2. Razni oblici REST klijenata za Javu[5]

[4] https://www.youtube.com/watch?v=Wtjx-75w5CY&list=PLy0T81VDh93YLJEEe5AxydDlXxUPrPs_B
[5] https://www.youtube.com/watch?v=3zTxHA2FnOo&list=PLy0T81VDh93YLJEEe5AxydDlXxUPrPs_B&index=10

5.5 Pitanja studenata prijašnjih godina

Q1: Kada od poslužitelja tražim najbližeg susjeda, posluzitelj šalje odgovor koji se sastoji od sljedecih vrijednosti: latitude, longitude, ip, port. Problem se pojavljuje u sljedećem koraku: susjedni senzor trebao bi poslati svoje najaktualnije ocitanje (pretpostavljam da bi se to najaktualnije ocitanje ponovo trebalo dohvatiti s poslužitelja), a ja nemam identifikator susjednog senzora - ne mogu napraviti zahtjev na poslužitelj tipa GET /readings/sensor/id (dohvati ocitanja susjednog senzora).

A1: Senzor svoje najaktualije očitanje čita iz .csv datoteke, ne s poslužitelja. Što se tiče očitanja najbližeg susjeda, senzor A ce od svojeg najbližeg susjeda (recimo senzora B) dobiti očitanje putem GRPC veze, a ne Spring poslužitelja. Na slici 1 je dan okvirni tok informacija u sustavu. Komunikacija je opisana u poglavlju 3.3 u recenici: "Nakon što senzor generira vlastito ocitanje, mora zatražiti ocitanje od svog geografski najbližeg susjeda putem gRPC veze."

Q2: U zadatku 4.2 gdje tražimo najbližeg susjeda nije točno rečeno kako formirati stausni kod odgovora. Moj URL na taj endpoint je /sensors/id/neighbour. Postoje dva slučaja kad nema susjeda: ako senzor s tim ID-jem ne postoji i ako postoji senzor za kojeg se traži susjed ali ne postoji susjed. Pa me samo zanima je li u redu za oba slučaja vratit statusni kod 404 ili bih trebao kao što je u zadatku 4.5, ako baš ne postoji senzor za kojeg se traži susjed vratiti 204. Također me zanima hoće li senzor kada dobije zahtjev od drugog senzora za kalibraciju tada izgenerirati očitanje ili ga negdje čuva i samo proslijedi zadnje očitanje. A2: Možete vratiti statusni kod 204 ako ne nađe najbližeg susjeda (npr. samo je 1 senzor registriran). Statusni kod 404 znači da endpoint ne postoji, a u Vašem slučaju postoji.

Kada senzor dobije zahtjev od drugog senzora za kalibraciju očitanja, tada se kalibrirano očitanje odmah šalje na poslužitelj.

Q2: Znači li to da senzor nigdje ne čuva svoje posljednje očitanje? A2: Tako je, senzor ne čuva svoje posljednje očitanje, nego nakon nekog perioda šalje zahtjev za kalibraciju sa svojim najbližim susjedom, tj. ako nema susjeda, onda se očitanja ne kalibriraju, nego se vlastita očitanja pohranjuje na poslužitelju.

Q3: Imam problema sa gRPC odnosno pokretanjem protoc:
protoc –java_out=src/main/java/ –grpc_out=src/main/java/ src/main/proto/*.proto
Dobivam error:
protoc-gen-grpc: program not found or is not executable
–grpc_out: protoc-gen-grpc: Plugin failed with status code 1.
A3: Provjerite jeste li dobro instalirali plugine, koju verziju jave koristite, itd.). Našao sam git issue gdje korisnici komentiraju error koji i Vi dobivate. Nisam siguran hoće li Vam pomoći ili ne.
https://github.com/grpc/grpc-java/issues/9450
https://github.com/faaxm/exmpl-cmake-grpc/issues/1

Probajte pokrenuti gRPC primjer kako je napisano u tekstu labosa, poglavlje 5.3. Vidite možete li pokrenuti te primjere ili ne.

Q4: U primjeru koji je u repozitoriju (pretvorba u velika slova) šalju se samo stringovi, no u našem zadatku se očekuje da se šalje više vrijednosti (očitanja). Zanima me može li se to implementirati na način da se očitanja senzora pretvore u JSON (ili koji drugi format) i onda se pošalje kao običan string (isto kao u primjeru iz repozitorija) ili je potrebno bitno izmijeniti .proto datoteku da funkcionira s našim klasama Sensor i Reading pa da se pritom šalju klase, a ne nizovi znakova?

A4: Trebate izmijeniti .proto datoteku, ne smijete slati nizove znakova. Trebate slati "klasu" koja sadržava vrijednosti temperature, pritiska, vlage, co, no2 i so2.

Q4: Zahvaljujem, no zamolio bih ako biste mogli pojasniti kako bi to trebalo izgledati. Na 69. i 70. slajdovima prezentacije gdje se radi gRPC stavljen je primjer sa slanjem stringova, nigdje vlastitim klasama. Je li dakle potrebno samo "string" zamijeniti s nazivom drugog "message" isječka u kojem će se definirati očitanja? Govorim o: https://protobuf.dev/programmi gdje se šalje "Result" koji ima svoje parametre (tako bismo i mi trebali za Reading?). Kako bi se onda mogla sva očitanja senzora (npr. ako ih je 4) poslati unutar te jedne poruke?

A4: Napravite novu .proto datoteku (npr. reading.proto), i onda koristite message sintaksu (kao što se poslali na linku).

Q5: Pozdrav, što u slučaju kada je .zip arhiva prevelika za upload na moodle?

A5: .zip arhive nebi trebala biti veća od 10 MB. Pregledajte si projekt i izbrišite nepotrebne datoteke koje ste arhivirali zajedno sa projektom, a ne koriste se u sklopu laboratorijske vježbe.