SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

ZAVRŠNI RAD br. 1234

SUSTAV ZA PRAĆENJE STANJA POSLUŽITELJA TEMELJEN NA JEZIKU GRAPHQL

Dominik Dejanović

Zagreb, lipanj, 2024.

Ovdje dolazi tekst zadatka završnog rada na hrvatskom jeziku.

Hvala na kavi...

Sadržaj

1. Uvod

Današnja tehnologija iznimno se oslanja na poslužitelje kako bi tko? ostvarili razne funkcionalnosti koje su potrebne većini ljudi u svakodnevnom životu, od pretraživanja raznih web stranica i videa na internetu do povezanosti ogromnih sinonim za ogromni mreža računala u virtualno super-računalo, poslužitelji su neophodni u tom procesu.

Nažalost, nije dovoljno konfigurirati poslužitelja da obavlja određene zadaće i nadati se da će raditi zauvijek. Zbog raznih problema kao prirodnih katastrofa, pogrešaka u kodu, virusa i hakera, neispravnog rada komponenti, prevelikog broja zahtjeva i dr., može doći do usporenja poslužitelja, neispravnog obavljanja funkcionalnosti te čak i do potpunog prestanka rada poslužitelja. Upravo zbog tih razloga postoji puno aplikacija koje se koriste za praćenje rada poslužitelja i obavještavanje administratora u slučaju neispravnog rada. Problem koji se javalja kod većine ovakvih aplikacija je nemogućnost pregleda specifičnog vremenskog perioda u kojemu se dogodila greška, potreba za plaćanjem za naprednije funkcionalnosti aplikacije te pohranjivanje podataka samo u zadnjih nekoliko dana ili tjedana.

Cilj je ove aplikacije omogućiti praćenje jednog ili više poslužitelja kroz neograničen period vremena, što omogućuje administratorima pregledavanje podataka o radu poslužitelja tijekom specifičnog perioda vremena u kojemu je nastala greška na poslužitelju.

2. Specifikacija zahtjeva

2.1. Korisnički zahtjevi

Osnovna funkcionalnost rješenja je praćenje stanja poslužitelja te prikaz tih podataka na web-stranici.
Aplikacija za praćenje stanja poslužitelja se pokreće na poslužitelju koji se prati. Mougće je aplikaciju pokrenuti na više poslužitelja. Također je moguće konfigurirati interval slanja podataka te odabrati koji se podaci šalju pomoću JSON datoteke prije pokretanja aplikacije.
Web-stranica šalje zahtjeve na programsko sučelje, te pomoću vraćenih podataka omogućava njihov strukturiran pregled. To uključuje pregled:

- podataka o procesoru - prikazuju se kartica sa općenitim podacima o procesoru te grafovi iskorištenosti procesora i broja procesa u odabranom periodu
- podataka o privremenoj memoriji - prikazuje se kartica sa grafom na kojem su vidljivi podaci o raznim aspektima privremene memorije kao iskorištenost memorije, iskorištenost swap particije te postotak cached podataka
- podataka o trajnoj pohrani - za svaki disk se prikazuju općeniti podaci o disku kao naziv, veličina diska, proizvođač te se prikazuje graf na kojemu je vidljiva iskorištenost particija diska u odabranom periodu vremena
- obavijesti i upozorenja - klikom na "Alerts" se otvara tablica sa prikazom obavijesti i upozorenja koje je programsko sučelje generiralo prilikom upisa podataka u bazu podataka (na primjer upozorenje za visoku iskorištenost particije tvdog diska)

Svaku karticu je moguće minimizirati, nakon čega se one dinamično rasporede da stanu na zaslon (stranica je kompatibilna i za mobilne uređaje).

Također je moguće dio grafa uvećati nakon čega se šalje novi zahtjev na programsko sučelje kako bi se prikazalo više detalja za novi period. Nakon uvećanja grafa, moguće ga resetirati čime se on umanjuje na originalni period vremena te se šalje novi zahtjev na programsko sučelje za dohvat podataka za taj vremenski period.
Korisnik također može odabrati globalni period. Nakon promijene početnog ili krajnjeg datuma se svi podaci za odabrani poslužitlj osvježavaju za taj vremenski period. Prilikom početnog otvaranja web-stranice, početni datum se postavlja na tjedan dana prije trenutnog vremena, a krajnji datum je neograničen.

2.2. Funkcionalni zahtjevi

Za pohranu podataka o poslužiteljima se koristi baza podataka.
Svakom poslužitelju koji se prati se dodijeljuje jedinstveni identifikator čime se podaci poslani od više poslužitelja razlikuju. Nakon pokretanja aplikacije, šalju se podaci na centralni poslužitelj na kojem je pokrenuto programsko sučelje.
Programsko sučelje procesira zahtjeve za pisanje i čitanje podataka o poslužiteljima te komunicira s bazom podataka kako bi te podatke zapisao ili pročitao iz nje. Sučelje ima razne parametre kojima se omogućava filtriranje podataka za određeni poslužitelj i vremenski period, te parametre koji određuju način kompresije podataka (min/max/average). Na adresu programskog sučelja aplikacije za praćenje stanja poslužitelja šalju podatke koji se spremaju u bazu podataka, a web-stranica šalje zahtjeve za čitanje podataka o poslužiteljima.

3. Korištene tehnologije

U ovom projektu su korištene razne tehnologije: RDBMS, web razvojna okolina, backend tehnologije te brojne druge. U nastavku slijedi opis korištenih tehnologija.

3.1. Linux

Linux je open-source operacijski sustav baziran na Unixu koji je nastao 1991. godine. Postoje razne distribucije linuxa (Ubuntu, Mint, Arch, Fedora, CentOS i drugi) koje uključuju jezgru zajedno sa raznim softverskim paketima i modifikacijama koje čine tu distribuciju jedinstvenom. Odabran je operacijski sustav Linux jer je poznat po svojoj stabilnosti, sigurnosti i fleksibilnosti, što ga čini vrlo popularnom opcijom za poslužitelje. Njegova otvorenost omogućava korisnicima i programerima da slobodno modificiraju i dijele kod, ali unatoč tome su sve distribucije temeljene na istoj jezgri što omogućava ovom programu da ispravno radi na svim modernim Linux distribucijama. Koriste se razni linux programi za prikupljanje podataka kao:

- lscpu - podaci o procesoru
- top - podaci o trenutno pokrenutim procesima na operacijskom sustavu
- systemctl - podaci o određenom servisu kao trenutni status, lokacija, poruke te drugo
- lsblk - podaci o diskovima i particijama na računalu
- journalctl - poruke koje je određeni servis poslao

Za programiranje i testiranje programa su korištene Arch Linux i Linux Mint distribucije, ali program bi trebao raditi na svim linux distribucijama, dokle god se na njih mogu

instalirati potrebni programi za prikupljanje podataka i pokretanje aplikacija.

3.2. GraphQL

GraphQL je jezik za upite podataka, hibrid REST programskog sučelja i SQL jezika. Dizajniran je da omogući klijentima da definiraju podatake koji su im potrebni, čime se izbegava dohvaćanje previše ili premalo podataka, što su česti problemi kod tradicionalnih REST programskih sučelja.
Odabran je GraphQL umjesto REST programskog sučelja zbog raznih karakteristika GraphQL-a, neke od važniji

- *deklarativni podaci* - korisnici navode točno koji podaci im trebaju te se ne šalje ništa više od toga
- *jedan URL* - koristi se samo jedan URL za sve upite, od upita za dohvaćanje podataka do onih za slanje podataka, što znatno ubrzava razvoj programskog sučelja te olakšava njegovo održavanje
- *ugrađena validacija polja* - ovo omogućava GraphQL-u da provjeri polja koja korisnik unosi tako da se ne treba ručno programirati provjeravanje polja (na primjer GraphQL će automatski izbaciti grešku ukoliko se u brojčano polje unese znakovni niz). Također podcrtava pogreške prilikom korištenja nepoznatih parametara i polja. Primjer koda preuzet sa `https://graphql.org/learn/validation/`:

```
# INVALID: hero is not a scalar, so fields are needed
{
  hero
}
```

```
{
  "errors": [
    {
      "message": "Field \"hero\" of type \"Character
         \" must have a selection of subfields. Did
         you mean \"hero { ... }\"?",
      "locations": [
        {
          "line": 3,
          "column": 3
        }
      ]
    }
  ]
}
```

Slika 3.1. Primjer krivog GraphQL upita

- *upiti slični SQL-u* - za razliku od REST-a koji se oslanja na putanje, GraphQL koristi Query kako bi korisnik mogao pomoću određenih parametara filtirati podatke te odabrati koje podatke želi dohvatiti

- *dohvaćanje više podataka u jednom zahtjevu* - koristeći REST, korisnik bi morao za dohvaćanje raznih podataka slati puno upita, dok se u GraphQL-u može dohvatiti proizvoljan broj nepovezanih podataka u jednom zahtjevu. Primjer koda preuzet sa `https://graphql.org/learn/queries/`

```
{
  empireHero: hero(episode: EMPIRE) {
    name
  }
  jediHero: hero(episode: JEDI) {
    name
  }
}
```

```
{
  "data": {
    "empireHero": {
      "name": "Luke Skywalker"
    },
    "jediHero": {
      "name": "R2-D2"
    }
  }
}
```

Slika 3.2. Primjer GraphQL upita

- fleksibilnost razvoja programskog sučelja - stari upiti će raditi čak i ako se shema programskog sučelja promijeni, dokle god polja koja korisnik dohvaća još uvijek postoje

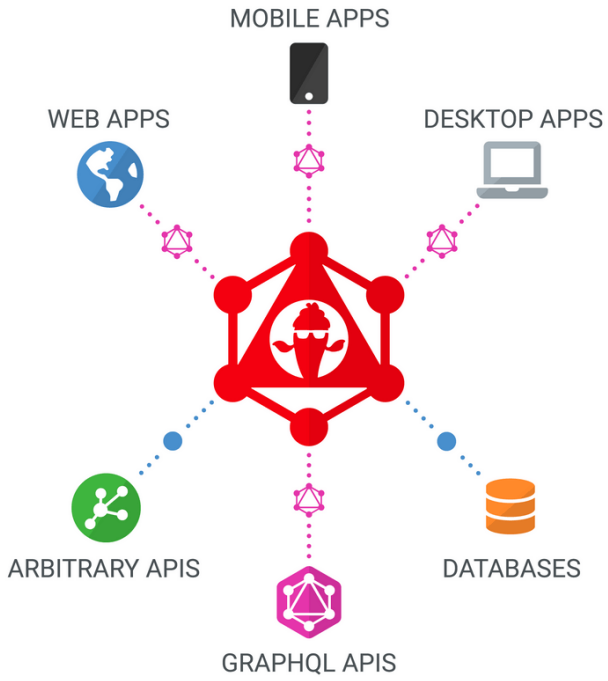
GraphQL se sastoji od četiri važna dijela:

- shema (*schema*) - glavni dio GraphQL sustava je shema koja definira vrste podataka te upite koje korisnici mogu izvršavati.
- upit (*query*) - omogućava slanje upita poslužitelju u kojem se definiraju podaci koji se vraćaju
- mutacija (*mutation*) - omogućava slanje podataka poslužitelju koji se koriste za dodavanje, izmijenu te brisanje podataka
- pretplata (*subscription*) - omogućava osvježavanje podataka u stvarnom vremenu

Sve ove karakteristike GraphQL-a omogućavaju efikasno prenošenje velike količine podataka te smanjuje kompleksnost održavanja i dokumentiranja koda, zbog čega je on odabran umjesto REST programskog sučelja.

3.3. HotChocolate

HotChocolate je C# biblioteka koja se koristi za izgradnju GraphQL poslužitelja. Ona nam omogućava korištenje svih funkcionalnosti GraphQL tehnologije pomoću .NET okoline bez da ih moramo ručno implementirati. Postoje razne druge biblioteke koje se koriste za interakciju sa GraphQL poslužiteljem, no ova datoteka je odabrana zbog opširne dokumentacije i podrške za moderne GraphQL funkcionalnosti.
Korištena verzija: 13.6.0



Slika 3.3. HotChocolate server

ADD IMAGE REFERENCE https://chillicream.com/static/cfd2ddde71f95ed876541f87c15b2a08/78
https://chillicream.com/docs/hotchocolate/v13

3.4. .NET i C#

.NET je otvorena platforma za razvoj raznih topova aplikacija (web-aplikacije, aplikacije za mobilne uređaje, video igre te drugo). Obuhvaća razne tehnologije i alate koji omogućavaju brzi razvoj aplikacija. Ključne komponente su .NET Runtime (zadužen za automatsko skupljanje smeća i upravljanje memorijom), .NET SDK (alati i biblioteke koje se koriste za razvoj .NET aplikacija, alati za debuggiranje) i .NET biblioteke (osnovne funk-

cionalnosti potrebne za razvoj aplikacija).

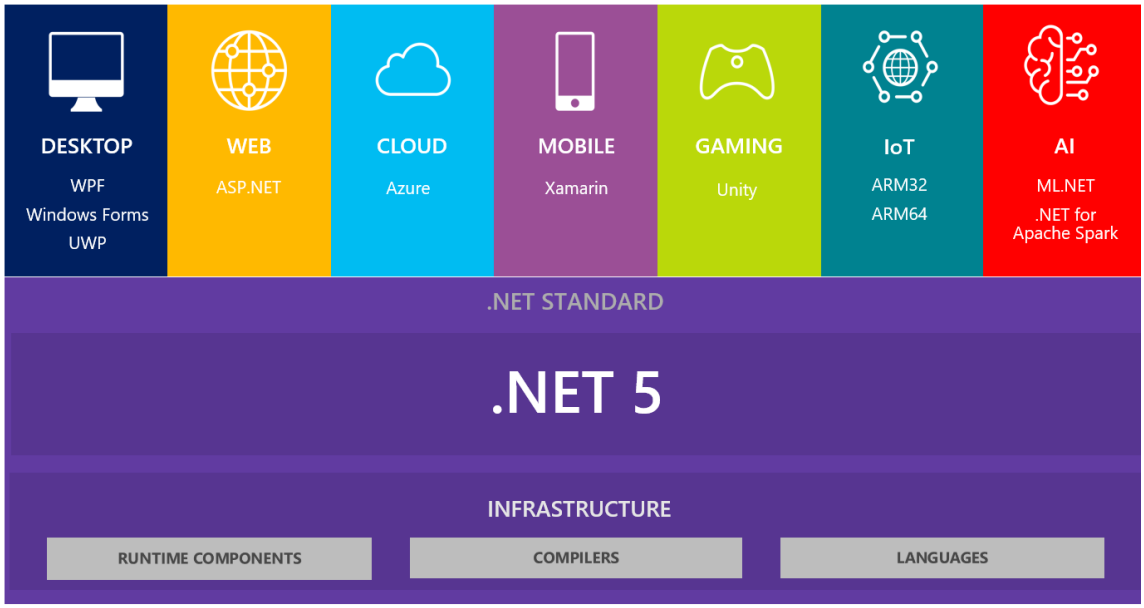
.NET okruženje podržava C#, F# i VB.NET jezike.

C# je objektno orijentiran programski jezik dizajniran 2000. godine. Sintaksa C# jezika je jednostavna te jezik omogućava izgradnju paralelnog koda što znatno ubrzava aplikacije. Često je uspoređivan s programskim jezikom Java.

Odabrane su ove tehnologije zbog jednostavnosti razvoja aplikacija u njima, te zbog toga što .NET podržava izvođenje i razvoj aplikacija na više platformi, uključujući i Windows i Linux.

Korištena .NET verzija: net8.0

Korištena C# verzija: 12.0



Slika 3.4. Alati .NET platforme

ADD IMAGE REFERENCE TO https://images.ctfassets.net/23aumh6u8s0i/2bsVrNvzMOk64yfFw

3.4.1. linq2db

linq2db je .NET biblioteka koja se koristi za pretvaranje LINQ koda u SQL kod kako bi se moglo lako pristupati bazi podataka.

Nakon instalacije paketa se pomoću određenih naredbi može spojiti na bazu podataka i iz nje generirati C# klase koje odgovaraju tablicama u bazi podataka:

```
using System;
using LinqToDB.Mapping;

[Table("Products")]
public class Product
{
  [PrimaryKey, Identity]
  public int ProductID { get; set; }

  [Column("ProductName"), NotNull]
  public string Name { get; set; }

  [Column]
  public int VendorID { get; set; }

  [Association(ThisKey = nameof(VendorID), OtherKey=
     nameof(Vendor.ID))]
  public Vendor Vendor { get; set; }

  // ... other columns ...
}
```

Slika 3.5. Primjer linq2db koda

linq2db repozitorij: `https://github.com/linq2db/linq2db`

3.4.2. Newtonsoft.Json

Newtonsoft.Json je C# biblioterka koja se koristi za serijalizaciju i deserijalizaciju JSON podataka. Newtonsoft.Json repozitorij: https://github.com/JamesNK/Newtonsoft.Json

3.5. Vue.js

Vue.js je Javascript razvojno okruženje koje se koristi za izradu korisničkog sučelja. Omogućava jednostavniji rad sa prikazom podataka i bolje strukturiranje koda od samog Javascript jezika.
Neke od prednosti Vue razvojnog okruženja:

- *progresivna arhitektura* - može se lako integrirati u postojeće projekte, te se nakon toga može postepeno proširivati
- *komponente* - koriste se komponente kako bi se aplikacija razdvojila na manje, ponovno upotrebljive dijelove
- *reaktivnost* - prikazani podaci se automatski osvježavaju kada se promijene varija-

ble na koje su podaci vezani

- *direktive* - Vue pruža direktive kao v-model, v-if, v-for te druge kako bi se jednostavno manipuliralo DOM-om

Primjer Vue koda za povećanja vrijednosti gumba nakon klika (preuzeto sa `https://vuejs.org/guide/introduction.html`):

```
<div id="app">
<button @click="count++">
Count is: {{ count }}
</button>
</div>
```

Slika 3.6. Primjer Vue koda za povećanje gumba

Korištena verzija: 3.2.13

Vue.js dokumentacija: `https://vuejs.org/guide/introduction.html`

primevue

Primevue sadržava razne Vue komponente koje se koriste za dinamički prikaz podataka na ekranu (prilikom promijene podataka se odmah mijenja i prikaz bez potrebe za dodatnim kodom). Također se koristi primeicons biblioteka koja omogućuje korištenje raznih ikona.

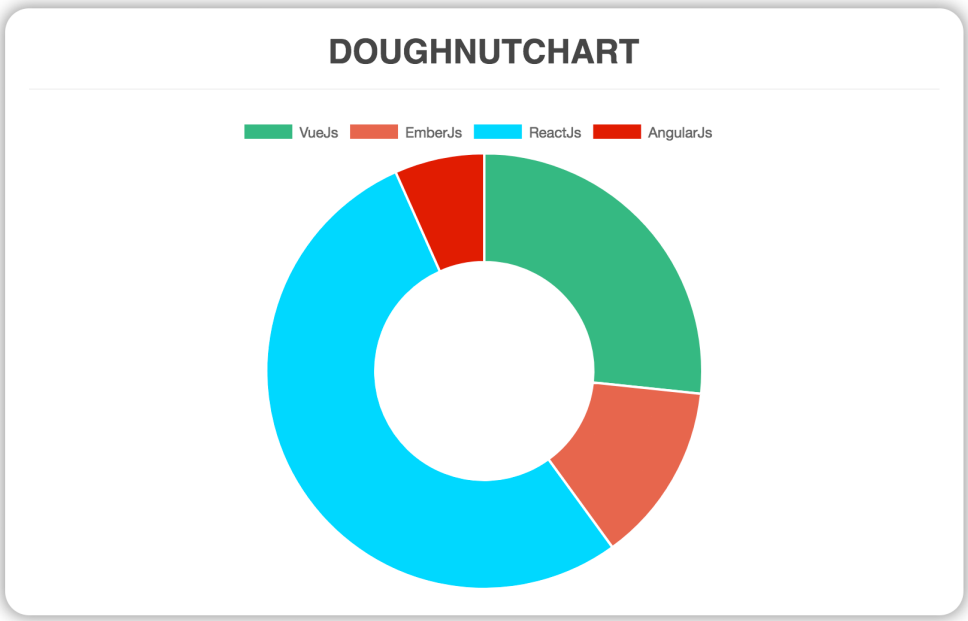
Korištena primevue verzija: 3.52.0

Korištena primeicons verzija: 7.0.0

Dokumentacija: `https://primevue.org/`

vue-chartjs

Vue-chartjs Vue je biblioteka koja se koristi za vizualizaciju podataka ovisno o predanim parametrima. Omogućuje prikaz raznih tipova grafova kao linijski, stubičasti te točkasti graf koje se može konfigurirati kroz razne parametre.



Slika 3.7. Primjer chartjs grafa

ADD IMAGE REFERENCE https://raw.githubusercontent.com/apertureless/vue-chartjs/HEAD/a

Korištena verzija: 5.2.0

Dokumentacija: `https://vue-chartjs.org`

vue-datepicker

Vue-datepicker je Vue komponenta koja se koristi za odabir datuma i vremena. Klikom na komponentu se otvara kalendar na kojem korisnik može odabrati datum i vrijeme.

Slika 3.8. Primjer datepicker komponente

Korištena verzija: 7.1.0

Dokumentacija: `https://vue3datepicker.com/`

3.6. PostgreSQL

PostgreSQL relacijski je sistem za upravljanje bazama podataka (RDBMS) koji je otvorenog koda i besplatan. Poznat je po svojoj stabilnosti i fleksibilnosti.

Važne karakteristike PostgreSQL-a:

- *otvoreni kod* - PostgreSQL kod je otvorenog koda zbog čega bilo tko može vidjeti, izmijeniti i distribuirati
- *moderni standardi* - implementira puno modernih karakteristika SQL jezika
- *podrška za JSON* - omogućava rad sa JSON podacima, uključujući njihovu pohranu
- *razni tipovi podataka* - podržava integer, varchar, boolean, array, uuid, xml te razne druge tipove podataka
- *ACID (atomicity, consistency, isolation, durability)* - osigurava pouzdane transakcije i integritet podataka

Odabran je PostgreSQL kao RDBMS sustav za upravljanje bazom podataka zato što je otvorenog koda te ima opširnu dokumentaciju i korisničku podršku.
Korištena verzija: 16.2
PostgreSQL dokumentacija: `https://www.postgresql.org/docs`

3.7. Git i Github

Git je sustav za upravljanje kodom. Podržava verzioniranje datoteka što omogućava praćenje promijena koda te suradnju sa drugim programerima. Linus Torvalds napravio ga je 2005. godine te je danas jedan od najvažnijih alata za razvoj programa. Neke od važnijih karakteristika:

- *distributiranost* - svaki korisnik ima kompletnu kopiju repozitorija na svom računalu što omogućava rad na projektu kada korisnik nema pristup internetu
- *verzioniranje datoteka* - omogućava praćenje promjena u datotekama kako se mijenjaju, što olakšava pronalazak koda koji je uzrokovao novu grešku
- *grananje* - omogućava rad na odvojenim dijelovima projekta, neovisno o main/master grani
- *rad u timu* - olakšava suradnju više programera koji svi rade na istom projektu pomoću raznih mehanizama, jedan od kojih je spriječavanje prebrisavanja koda kojeg je jedan programer objavio sa kodom kojeg drugi programer pokušava objaviti

Github je platforma za razvoj aplikacija temeljena na Git-u. Koristi se za lakše korištenje Git-a te pruža razne alate koji se mogu koristiti za organizaciju zadaća programerima.

Službena git stranica: `https://git-scm.com`

Github: `https://github.com/`

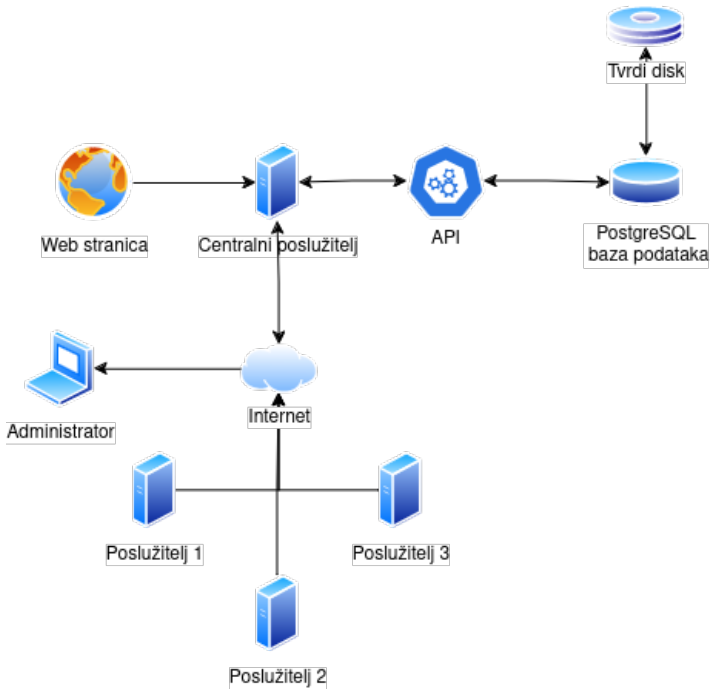
4. Opis rješenja

4.1. Softver

Napravljene su tri .NET projekta (API, Common i Monitor), web stranica te baza podataka.
API (programsko sučelje) je centralna aplikacija koja povezuje Monitor i web stranicu s bazom podataka. To je ostvareno pomoću GraphQL-a, kod kojeg se koristi Query objekt za dohvaćanje podatka za web stranicu, te Mutation objekti koji se koriste kako bi Monitor mogao pisati podatke u bazu podataka.
Monitor je aplikacija koja se pokreće na poslužitelju koji se prati. Ona u specificiranom intervalu prikuplja razne podatke o poslužitelju te ih šalje programsko sučelje koji ih zapisuje u bazu podataka.
Web stranica se zatim koristi za prikaz prikupljenih podataka o pojedinačnim serverima, te za prikaz obavijesti i upozorenja koje je programsko sučelje generiralo prilikom upisa podataka u bazu podataka.

4.2. Fizička konfiguracija

Postoji jedan centralni poslužitelj na kojem su pokrenute dvije aplikacije: Vue web stranica te programsko sučelje. Vue web stranica se dohvaća HTTP protokolom, nakon čega ona šalje zahtijeve programskom sučelju na centralnom poslužitelju radi dohvata podataka iz baze podataka.
Poslužitelji koji su konfigurirani za praćenje podataka također komuniciraju sa programskim sučeljem, ali ne za dohvaćanje nego za slanje podataka sučelju koji onda te podatke sprema u bazu podataka.

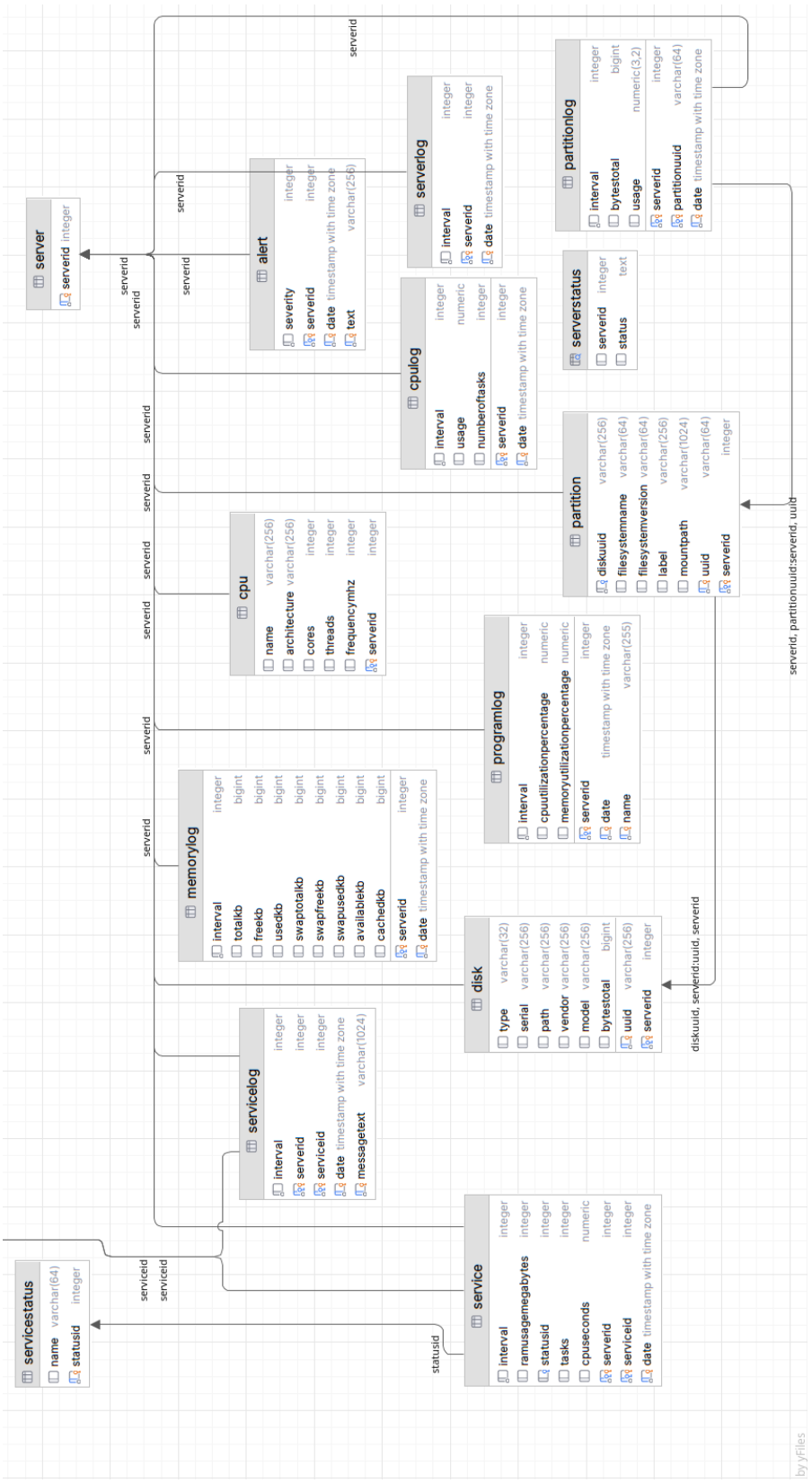


Slika 4.1. Dijagram rješenja

U nastavku slijedi detaljan opis funkcionalnosti i koda svih projekata.

5. Baza podataka

Za izradu baze podataka je korišten 3.6. Postgresql. U korijenskoj strukturi repozitorija se nalazi datoteka db.sql pomoću koje se stvara baza podataka. U nastavku je opisana struktura baze podataka.



Slika 5.1. Struktura baze podataka

5.1. Poslužitelj

Za praćenje poslužitelja se koristi tablica server koja samo pohranjuje ID poslužitelja. Ovu tablicu referencira većina drugih tablica.

Također je napravljen jedan pogled koji se koristi za dohvat statusa poslužitelja. Poslužitelj se smatra aktivnim ako je od zadnjeg slanja podataka prošlo manje od n minuta (n = interval specificiran prilikom zadnjeg slanja podataka).

```
CREATE VIEW serverStatus AS
SELECT serverid, (SELECT CASE WHEN COUNT(*) > 0 THEN '
   online' ELSE 'offline' END
                  FROM serverlog
                  WHERE
                      serverlog.serverid = server.serverid
                         AND
                      EXTRACT(EPOCH FROM timezone('utc',
                         now())-serverlog.date) < (
                         serverlog.interval * 60)) as
                         status
FROM server;
```

Slika 5.2. serverStatus pogled

5.2. Procesor

Za praćenje podataka o procesoru se koriste tablice cpu i cpulog. Tablica cpu pohranjuje općenite podatke o procesoru: ime procesora (name), arhitektura (architecture), broj jezgri (cores), broj niti (threads), frekvencija rada (frequency) te poslužitelj kojem on pripada (serverId).

cpulog tablica sprema podatke o procesoru koji se mijenjaju tijekom vremena kao iskorištenost procesora (usage) i broj procesa koje on obrađuje (numberoftasks).

5.3. Priručna memorija

Za pohranjivanje podataka o priručnoj memoriji se koristi samo tablica memorylog koja pohranjuje podatke o datumu prikupljanja podataka (date) ukupnoj memoriji (totalkb, freekb, usedkb), "swap" memoriji (swaptotalkb, swapfreekb, swapusedkb), neiskorištenoj memoriji (availablekb), "cache" memoriji (cachedkb), te ID poslužitelja kojem memorija pripada.

Za razliku od procesora, ne koristi se zasebna tablica za pohranu općenitih podataka o

priručnoj memoriji kao proizvođač, serijski broj te drugo, jer nije pronađen način da se ti podaci očitaju sa poslužitelja bez administratorskih privilegija.

5.4. Trajna pohrana

Za pohranu trajne memorije se koriste tablice:

- disk - općenite informacije o tvrdom disku: jedinstveni ID (uuid), tip (type), serijski broj (serial), putanja na kojoj se disk nalazi (path), proizvođač (vendor), model, veličina diska (bytestotal) i ID poslužitelja na kojem se disk nalazi
- partition - općenite informacije o jednoj particiji: jedinstveni ID (uuid), UUID diska kojem pripada (diskuuid), tip i verzija datotečnog sustava (filesystemname, filesystemversion), naziv particije (label), putanja na kojoj se nalazi (mountpath) te ID poslužitelja na kojem se particija nalazi
- partitionlog - podaci o particiji koji se mijenjaju tijekom vremena kao veličina particije (bytestotal) te iskorištenost (usage)

5.5. Servisi

Servisi nisu do kraja implementirani u aplikaciji, no podloga za praćenje podataka o servisima je implementirana su u bazi podataka. Za praćenje servisa se koriste tablice:

- service - prikupljeni podaci o statusu servisa kao pohrana koju koristi (ramusagemegabytes), broj procesa koje je servis stvorio (tasks), procesorska snaga koju koristi (cpuseconds) te status servisa (statusid)
- servicename - mapira jedinstveni broj u naziv servisa
- servicelog - pohranjuje poruke koje je servis poslao (messagetext) te vrijeme kada su poslane (date)
- servicestatus - mapira id statusa servisa u tekstualnu reprezentaciju

5.6. Programi

Podška za programe nije do kraja implementirana u aplikaciji, no podloga za njihovo praćenje je implementirana su u bazi podataka.
Za praćenje programa se koristi tablica programlog koji pohranjuje neke podatke kao ime programa (name) te postotak procesora (cpuutilizationpercentage) i memorije (memoryutilizationpercentage) koju program koristi.

5.7. Obavijesti

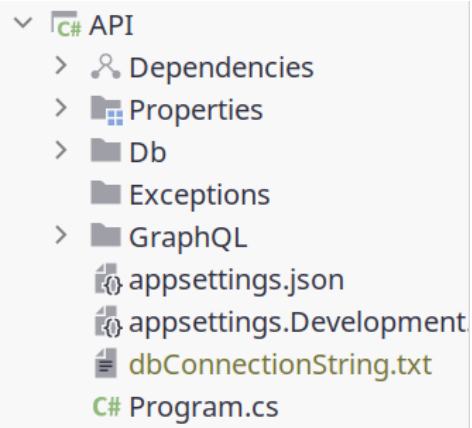
Stvorena je tablica za obavijesti koja pohranjuje kritičnost poruke (severity), ID poslužitelja za koji je relevantna (serverid) te datum (date) i tekst poruke (text).
Uz tablicu je također napravljena i metoda before_alert_insert_func() koja se poziva prije unosa podataka u tablicu. Metoda je stvorena kako bi se onemogućio unos iste obavijesti ako je ona već poslana u zadnjih sat vremena (na primjer ako se proba unijeti "Server 0 cpu load above 90%" u razmaku od 30 minuta, ta poruka će biti zapisana samo prvi put u bazu podataka, a drugi put se odbacuje uz grešku).

```
    SELECT EXTRACT(EPOCH FROM NEW.date::timestamp-date::
       timestamp)/60 FROM alert WHERE NEW.serverId =
       serverId AND NEW.text LIKE text ORDER BY date desc
       INTO mins;
    IF mins < 60 THEN
        RAISE EXCEPTION 'Same alert already raised less
           then an hour ago (% minutes ago)', mins;
    END IF;
    RETURN NEW;
```

Slika 5.3. Metoda before_alert_insert_func()

6. API

API aplikacija je implementirana koristeći 3.3. HotChocolate server paket koji implementira funkcionalnosti GraphQL poslužitelja.
Struktura projekta:



Slika 6.1. Struktura API projekta

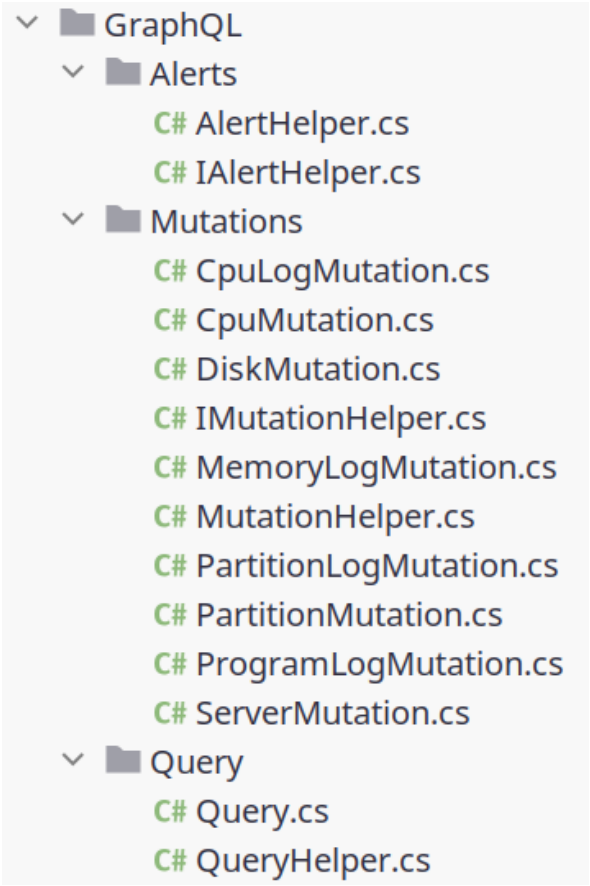
6.1. Program.cs

Program.cs je datoteka koja se pokreće prilikom pokretanja programa. Glavne funkcije koje ona izvršava su konfiguracija Dependency Injectiona te mapiranje putanja progamskog sučelja.
Dependency Injection je način strukturiranja koda kod kojeg se klase ne stvaraju direktno pomoću kodne riječi "new" nego se parametri konstruktora klase automatski predaju u nju. Odabran je ovaj način izrade koda kako bi se omogućilo lagano testiranje koda u budućnosti jer se umjesto stvarne klase koja obavlja određenu funkcionalnost može predati lažna klasa koja emulira stvarnu klasu (na primjer kao "IDb database" sučelje se umjesto klase koja obavlja funkcionalnolsti stvarne baze podataka predaje lažna klasa

koja samo provjerava da li je određena metoda pozvana tri puta; ako da onda je test ispravan a u suprotnom je neispravan).

6.2. GraphQL direktorij

GraphQL direktorij sadržava sav kod potreban za ispravan rad GraphQL poslužitelja kao Query i Mutation kodovi te kod za rad sa obavijestima.



Slika 6.2. Struktura GraphQL direktorija

6.2.1. Query.cs

Koristi se za dohvat podataka iz baze podataka. Za svaki tip podatka (CPU, memorija, tvrdi disk, ...) je napravljena metoda koja se poziva kada korisnik šalje zahtjev pomoću kojega se pokušava dohvatiti određen podatak.
Svaka metoda predstavlja tip query objekta koji se pokušava dohvatiti, a svi parametri te metode predtavljaju argumente koje korisnik specificira tijekom slanja zahtjeva.
Neki od parametara koje gotova svaka metoda ima su:

- serverId - dohvaćaju se samo podaci koje je poslao server čiji je ID jednak ovom argumentu
- startDateTime - dohvaćaju se samo podaci poslani nakon specificiranog datuma i vremena
- endDateTime - dohvaćaju se samo podaci poslani prije specificiranog datuma i vremena

Klasa se oslanja na 6.3. IDbProvider sučelje za rad sa bazom podataka, te na metodu **??** GetLogs metodu za dohvat logova iz baze podataka.
Primjer jedne takve metode:

```
public async Task<CpuOutput?> Cpu(int serverId,
    DateTimeOffset? startDateTime, DateTimeOffset?
    endDateTime, double? interval, string? method)
{
    var getEmptyRecordFunc = () => new CpuLog();
    Func<IList<CpuLogDbRecord>, CpuLog> combineLogsFunc =
        logs =>
    {
        double? usageProcessed = (double?)QueryHelper.
            CombineValues(method, logs.Select(c => c.Usage).
            ToList());
        var numberOfTasksValues = logs.Where(c => c.
            NumberOfTasks != null).Select(c => c.NumberOfTasks
            !);
        int? numberOfTasks = (int?)QueryHelper.CombineValues(
            method, numberOfTasksValues.ToList());
        return new CpuLog { Date = logs.First().Date, Usage =
            usageProcessed, NumberOfTasks = numberOfTasks };
    };

    var cpuOutput = new CpuOutput(serverId);
    await using var db = dbProvider.GetDb();
    {
        var cpu = await (from c in db.Cpus where c.ServerId
            == serverId select c).FirstOrDefaultAsync();
        if (cpu == null) return null;

        cpuOutput.Name = cpu.Name;
        cpuOutput.Architecture = cpu.Architecture;
        cpuOutput.Cores = cpu.Cores;
        cpuOutput.Threads = cpu.Threads;
        cpuOutput.Frequency = cpu.FrequencyMhz;
        var query = from l in db.CpuLogs where l.ServerId ==
            serverId select l;
        await foreach (var log in QueryHelper.GetLogs(query,
            combineLogsFunc, getEmptyRecordFunc, startDateTime
            , endDateTime, interval))
        {
            cpuOutput.Logs.Add(log);
        }
    }
```

Slika 6.3. Cpu metoda

6.2.2. QueryHelper.cs

Pomoćna klasa koju koristi 6.2.1. Query.cs klasa. Sastoji se od raznih metoda od kojih je najbitnija GetLogs metoda koja sadržava algoritam koji se koristi za spajanje više podataka u jedan. Algoritam radi tako da pomoću početnog i zadnjeg datuma izračuna interval unutar kojeg se podaci spajaju u jedan:

```
    return (int)((DateTime)endDate).Subtract((DateTime)
       startDate).TotalHours;
```

Slika 6.4. Izračun intervala

Nakon izračuna intervala se dohvačaju podaci iz tablice koji su pohranjeni nakon specificiranog početnog datuma i prije krajnjeg datuma. Nakon toga se prolazi kroz svaki podatak te se koristi 6.2.3. DatasetHelper klasa za spajanje više podataka u jedan, te se na kraju list spojenih podataka vraća.

```
  public static async IAsyncEnumerable<TLog> GetLogs<TDbLog
     , TLog>(
    IQueryable<ILog> table,
    Func<IList<TDbLog>, TLog> combineLogsFunc,
    Func<TLog> getEmptyLogFunc,
    DateTimeOffset? startDateTime,
    DateTimeOffset? endDateTime,
    double? interval) where TDbLog : ILog where TLog :
       LogBase
```

Slika 6.5. Parametri GetLogs metode

6.2.3. DatasetHelper.cs

Pomoćna klasa korištena u algoritmu spajanja više podataka u jedan podatak. Najvažnija metoda u klasi je AddLog metoda koja dodaje podatak u listu podataka koja se kasnije vraća. Broj podataka koji se spaja u jedan podatak ovisi o intervalu. Na primjer ako je interval 30, a podaci se bilježe svakih 5 minuta, spaja se 6 podataka u jedan. Ako je došlo do prekida u slanju podataka (na primjer podaci se šalju svakih 5 minuta, ali jedan podatak se pošalje nakon 7 minuta), onda se za period između ta dva podatka vraća "prazan" podatak koji signalizira prekid u slanju podataka. Algoritam radi na ovaj način kako bi se korisnike moglo obavijestiti ukoliko je došlo do prekida slanja podataka.

```
public IEnumerable<TLog?> AddLog(TDbLog log, double?
    interval)
{
  //Combining logs and adding additional logs if there
     has been some kind of a break
  //between the last log and the current one so that the
     charts goes to 0 in case of break
  if (_nextDate != null && log.Date != _nextDate)
  {
    if (_logs.Count != 0) yield return CombineLogs();
    var emptyLog = getEmptyLogFunc();
    emptyLog.Date = _nextDate.Value;
    yield return emptyLog;
    _break = true;
  }

  //Returning a log if a break happened
  if (_break)
  {
    var emptyLog = getEmptyLogFunc();
    emptyLog.Date = log.Date.AddSeconds(-1);
    yield return emptyLog;
    _break = false;
  }

  //Adding log to the list of logs
  _logs.Add(log);
  _nextDate = log.Date.AddMinutes(log.Interval);
  _intervalSum += log.Interval;
  if (_intervalSum < interval) yield break;

  //Returning the combined log
  yield return CombineLogs();
}
```

Slika 6.6. Metoda AddLog

6.2.4. Mutations direktorij

Sadržava klase i sučelja koja se koriste za dodavanje i osvježavanje podataka u bazi podataka.

Važnije klase i sučelja u direktoriju:

- IMutationHelper - sučelje koje definira funkcionalnosti koje su potrebne za umetanje, osvježavanje i brisanje pojedinačnih ili liste podataka iz baze podataka
- MutationHelper - implementacija IMutationHelper sučelja

Primjer dijela jedne mutacije:

```
[ExtendObjectType(OperationType.Mutation)]
public class CpuMutation(IMutationHelper mutationHelper)
{
    private readonly Func<IDb, CpuIdInput, IQueryable<
       CpuDbRecord>> _getCpuQuery = (db, cpu) =>
        from c in db.Cpus
        where c.ServerId == cpu.ServerId
        select c;

    private readonly Func<CpuDbRecord, CpuIdInput>
       _getCpuId = cpu => new CpuIdInput(cpu.ServerId);

    public async Task<Payload<CpuOutputBase>> AddCpu(
       CpuInput cpu)
    {
        var model = InputToDbModel(cpu);
        return await mutationHelper.AddModelAsync<
           CpuIdInput, CpuDbRecord, CpuOutputBase>(model,
           _getCpuId(model), _getCpuQuery);
    }
```

Slika 6.7. Dio CpuMutation mutacije

6.2.5. IAlertHelper.cs

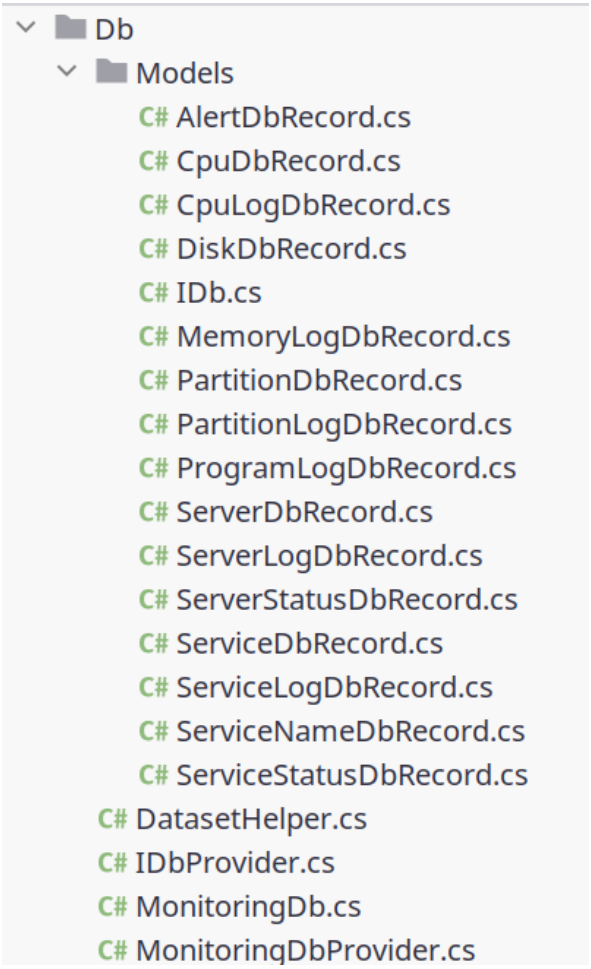
Koristi se za slanje obavijesti bazi podataka. Poruka se odbacuje ukoliko je ista poruka za isti poslužitelj već poslana prije manje od sat vremena.

```
public interface IAlertHelper
{
  Task RaiseAlert(int serverId, DateTimeOffset date,
     AlertSeverity severity, string text);
}
```

Slika 6.8. IAlertHelper.cs sučelje

6.3. Db direktorij

Db direktorij sadržava sve pomoćne klase i modele koji se koriste za interakciju sa bazom podataka. Većina klasa u direktoriju Models su automatski generirane (i djelomično ručno promijenjene) korištenjem 3.4.1. linq2db paketa pomoću naredbe "dotnet linq2db scaffold -p PostgreSQL -c "Host=localhost; Username=[username]; Password=[password];Database=[database]""



Slika 6.9. Struktura Db direktorija

Neke od važnijih klasa (te one koje nisu automatski generirane):

- Models/IDb.cs - sučelje koje sadržava popis svih tablica u bazi podataka

```
public interface IDb : IDataContext
{
  public ITable<CpuDbRecord> Cpus { get; }
  public ITable<CpuLogDbRecord> CpuLogs { get; }
  public ITable<DiskDbRecord> Disks { get; }
  public ITable<MemoryLogDbRecord> MemoryLogs { get; }
  public ITable<PartitionDbRecord> Partitions { get; }
  public ITable<PartitionLogDbRecord> PartitionLogs { get;
     }
  public ITable<ProgramLogDbRecord> ProgramLogs { get; }
  public ITable<ServiceDbRecord> Services { get; }
  public ITable<ServiceLogDbRecord> ServiceLogs { get; }
  public ITable<ServicenameDbRecord> ServiceNames { get; }
  public ITable<ServicestatusDbRecord> ServiceStatuses {
     get; }
  public ITable<AlertDbRecord> Alerts { get; }
  public ITable<ServerDbRecord> Servers { get; }
  public ITable<ServerLogDbRecord> ServerLogs { get; }
  public ITable<ServerStatusDbRecord> ServerStatus { get; }
}
```

Slika 6.10. IDb sučelje

- MonitoringDb.cs - automatski generirana implementacija IDb.cs sučelja
- MonitoringDbProvider.cs - koristi se za generiranje nove konekcije na bazu podataka (potrebno jer se koristi Dependency Injection)

6.4. Konfiguracijske datoteke

U programu se nalaze tri konfiguracijske datoteke: appsetting.json, appsettings.Development.json i dbConnectionString.txt od kojih će samo zadnja biti opisana.
dbConnectionString.txt datoteka sadržava niz znakova koji se koristi za spajanje na postojeću bazu podataka prilikom pokretanja programa:

```
    Host=localhost;Username=[username];Password=[password];
       Database=[dbName];Include Error Detail=[true for
       debugging; false for deployment]
```

Slika 6.11. Primjer datoteke dbConnectionString.txt

6.5. Pokretanje progamskog sučelja

API se pokreće iz komandne linije/terminala pomoću naredbe "dotnet run" unutar direktorija u kojem se aplikacija nalazi.

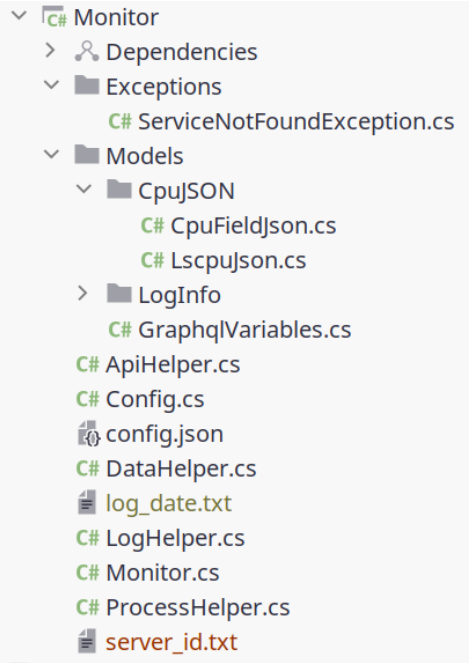
Nakon pokretanja se može pristupiti URL-u `http://localhost:3000//graphql`, prilikom čega se otvara web stranica na kojoj se može vidjeti dokumentacija programskog sučelja te se mogu izvršavati upiti.

Slika 6.12. Dohvat podataka o pohrani

Slika 6.13. Dodavanje jedne particije

7. Monitor

Aplikacija Monitor se koristi za prikupljanje podataka o poslužitelju te slanje tih podataka programskom sučelju.



Slika 7.1. Struktura Monitor projekta

Objašnjenje važnijih dijelova aplikacije:

- Monitor.cs - pokreće se prilikom pokretanja aplikacije
- Models direktorij - sadržava klase koje se koriste za serijalizaciju i deserijalizaciju podataka dohvaćenih sa terminala
- ApiHelper.cs - koristi se za slanje podataka programskom sučelju
- config.json - koristi ju korisnik kako bi konfigurirao aplikaciju
- DataHelper.cs - čita podatke sa terminala te ih deserijalizira u klase koje se nalaze

u Models direktoriju

- LogHelper.cs - pokreće proces dohvaćanja podataka svakin [n] minuta (n = definiran u config.json datoteci)
- ProcessHelper.cs - koristi se za pokretanje procesa u terminalu

7.1. Monitor.cs

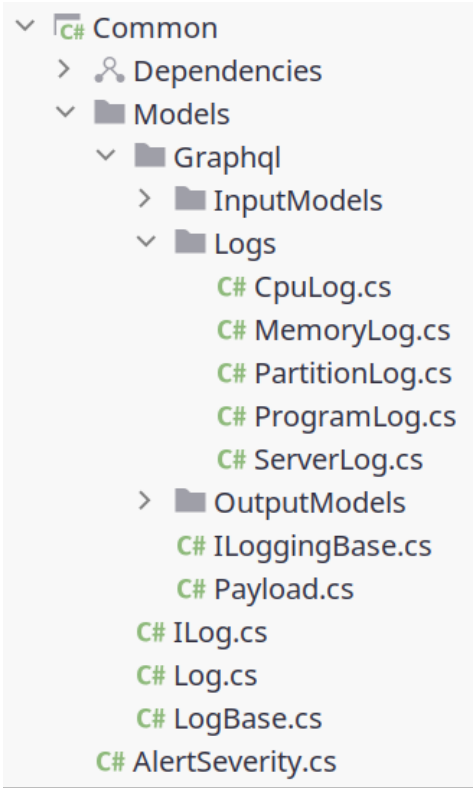
Pokreće se prilikom pokretanja aplikacija. Na početku učitava konfiguracijsku datoteku, te zatim generira nasumični broj za ID poslužitelja. Na kraju se u beskonačnoj petlji pokreće proces prikupljanja podataka te slanje tih podataka programskom sučelju.

```
private static async Task Main()
{
  var config = JsonSerializer.Deserialize<Config>(await
     File.ReadAllTextAsync("config.json"));
  if (config == null) throw new Exception("Configuration
     invalid!");

  //Creating server ID if the program is running for the
     first time
  if (File.Exists(ServerIdFilename) == false)
  {
    await File.WriteAllTextAsync(ServerIdFilename,
       DateTimeOffset.UtcNow.GetHashCode().ToString());
  }

  int serverId = int.Parse(await File.ReadAllTextAsync(
     ServerIdFilename));
  var logHelper = new LogHelper(config,
     LastLogDateFilename);
  var apiHelper = new ApiHelper(config, serverId);

  while (true)
  {
    var log = await logHelper.Log();
    await apiHelper.SendLog(log);
  }
}
```

Slika 7.2. Glavna metoda programa Monitor

8. Common

Projekt Common sadržava klase i sučelja koje koristi više projekata. Stvoren je kako bi se izbjeglo ponavljanje koda. Neke od važnijih klasa i sučelja:



Slika 8.1. Struktura Common projekta

8.1. ILog.cs

Glavno sučelje, svaki Log nasljeđuje polja definirana u njemu.

```
public interface ILog
{
  DateTimeOffset Date { get; }
  int Interval { get; }
}
```

Slika 8.2. ILog sučelje

8.2. Graphql direktorij

Graphql direktorij sadržava klase i sučelja koji se koriste za komunikaciju API i Monitor programa. Neko od važnijih komponenti direktorija:

- Payload.cs - sastoji se od Data polja koje sadržava podatke o poslužitelju koji se vraćaju korisniku te Error polja koje je prazno osim u slučaju pogreške prilikom obrade zahtjeva
- InputModels direktorij - sadržava klase koje se koriste prilikom dodavanja ili izmijene podataka. Primjer klase:

```
public class CpuInput : CpuIdInput
{
  public string? Name { get; set; }
  public string? Architecture { get; set; }
  public int? Cores { get; set; }
  public int? Threads { get; set; }
  public int? FrequencyMhz { get; set; }

  public CpuInput(int serverId) : base(serverId)
  {
  }
}
```

Slika 8.3. CpuInput klasa

- Logs direktorij - sadržava klase koje opisuju Log podatke za određen aspekt poslužitelja
- OutputModels direktorij - sadržava klase koje se koriste prilikom vraćanja API odgovora. Primjer klase:

```
public class MemoryOutput : MemoryOutputBase, ILoggingBase<
    MemoryLog>
{
  public List<MemoryLog> Logs { get; } = new();

  public MemoryOutput(int serverId) : base(serverId)
  {
  }
}
```

Slika 8.4. MemoryOutput klasa

9. Web stranica

Web stranica je napravljena pomoću 3.5. Vue razvojne okoline. Sastoji se od raznih kompoennti koje se koriste za prikaz i dohvaćanje podataka od programskog sučelja. Koristi se Vue razvojna okolina te se zbog toga koristi samo jedna HTML stranica koja se nalazi u direktoriju "public". Stranica se koristi kao kostur web-stranice u kojem se prikazuju renderirane komponente.

9.1. App.vue

Glavna Vue komponenta koja kontrolira sadržaj koji se prikazuje, stvara se prilikom pokretanja programa.
Na početku komponente se nalazi glavni izbornik koji koristi *Vue router* komponentu koja renderira određenu komponentu, ovisno o tome na kojoj putanji se korisnik nalazi. Kod za prikaz izbornika je preuzet sa službene Vue dokumentacije te je djelomično izmijenjen:

```
<Menubar :model="menu_items">
  <template #item="{item, props, hasSubmenu}">

    <!--creating a router link in case the item has a
        route-->
    <router-link v-if="item.route" v-slot="{ href,
       navigate }" :to="item.route" custom>
      <a :href="href" v-bind="props.action" @click="
         navigate">
        <span :class="item.icon" style="margin-right:
            10px;" />
        <span class="ml-2">{{ item.label }}</span>
      </a>
    </router-link>

    <!--creating a normal item in case the item has
       no route-->
    <a v-else :href="item.url" :target="item.target"
       v-bind="props.action">
      <span :class="item.icon" style="margin-right:
         10px;" />
      <span class="ml-2">{{ item.label }}</span>
      <span v-if="hasSubmenu" class="pi pi-fw pi-
         angle-down ml-2" />
    </a>

  </template>
</Menubar>
```

Slika 9.1. Kod za prikaz izbornika

Prilikom stvaranja komponente se šalje upit programskom sučelju kako bi se dohvatila i popunila lista poslužitelja, te se zatim korisnik prosljeđuje na putanju prikaza podataka prvog poslužitelja (osim ako nisu zabilježeni podaci za bilo koji poslužitelj).

9.2. Vue komponente

Vue razvojna okolina koristi komponente kako bi se dijelovi koda mogli ponovno koristiti. Sve korištene komponente se nalaze u direktoriju *components*.

Komponenta Chart

Komponenta *Chart* se koristi za prikaz linijskog grafa.

Parametri komponente:

- *name* - naslov grafa
- *chartData* - točke grafa

- *scales* - konfiguracijsko polje za x i y osi

Koristi nekoliko pomoćnih polja za ispravan rad, od kojih je najvažnije polje *options* koje omogućava pregled uvećanje grafa, postavlja x os kao vremensku od i y os kao brojčanu os, te pokreće *emitZoomChanged* događaj.
Komponenta koristi *emitZoomChanged* događaj kako bi roditelju javila da je graf uvećan, što je potrebno kako bi roditelj mogao osvježiti za taj period vremena i poslati ih ih komponenti.

Ostale komponente

Komponente *CpuInfo*, *DiskInfo* i *MemoryInfo* se koriste za prikaz podataka o procesoru i pohrani. Komponente rade na sličan način:

- komponenti se mogu predati parametri *startDate* (specificira početni datum grafa), *endDate* (završni datum grafa) i *serverId* (ID poslužitelja na kojeg se komponenta odnosi)
- nakon kreiranja komponente se učitavaju podaci pomoću *Api* klase te se ti podaci pretvaraju u podatke povoljne za prikaz grafom pomoću *ChartHelper* klase
- podaci se prikazuju pomoću *Fieldset* komponente na čijem početku se nalaze generalni podaci o komponente te ispod toga graf (ili grafovi)
- nakon primanja događaja *zoomChanged* od grafa, podaci za taj graf se ponovno učitavaju za novi period grafa

Primjer dijela jedne takve komponente:

```
<template>
  <Fieldset legend="Memory" :toggleable="true">
    <Chart
      name="Memory"
      :scales="{ x: { type: 'time' }, y: { min: 0, max: 100
          }}"
      :chart-data="this.$data.memoryChartData"
      @zoom-changed="async (limits) => {
      $data.memoryChartConfig.startDate = limits.startDate
         ?? $props.startDate
      $data.memoryChartConfig.endDate = limits.endDate ??
         $props.endDate
      await this.refreshData($data.memoryChartConfig.
         startDate, $data.memoryChartConfig.endDate)
    }"
    />
  </Fieldset>
</template>
```

Slika 9.2. Dio MemoryInfo komponente

9.3. Direktorij models

Direktorij Models sadržava razne klase koje se koriste za pohranjivanje podataka dohvaćenih s programskog sučelja.

Primjer modela:

```
export class Partition {
  uuid: string
  label: string
  filesystemName: string
  filesystemVersion: string
  mountPath: string
  logs: PartitionLog[]

  constructor(uuid: string, label: string, filesystemName:
     string, filesystemVersion: string, mountPath: string,
     logs: PartitionLog[]){
    this.uuid = uuid
    this.label = label
    this.filesystemName = filesystemName
    this.filesystemVersion = filesystemVersion
    this.mountPath = mountPath
    this.logs = logs
  }
}
```

Slika 9.3. Model particije

9.4. Klasa Api

Klasa Api je zadužena za komunikaciju s programskim sučeljem.

Primjer dohvaćanja podataka o poslužiteljima:

```
  static async getServers(){
    let queryString = `
    query {
      server {
        serverId
        online
      }
    }`

    let response = await this.executeQuery(queryString)
    if (response?.data?.server == null) return []
    let servers: Server[] = []
    response.data.server.forEach((s: any) => servers.push(s
       ))
    return servers
  }
}
```

Slika 9.4. Dohvaćanje podataka o poslužiteljima

9.5. ChartHelper.ts

Pomoćna datoteka koja se koristi za pretvorbu podataka koje programsko sučelje vraća u točke na grafu. Metode ove klase prolaze kroz *logs* parametar te za svaki podataka vraćaju podatke relevantne za graf.

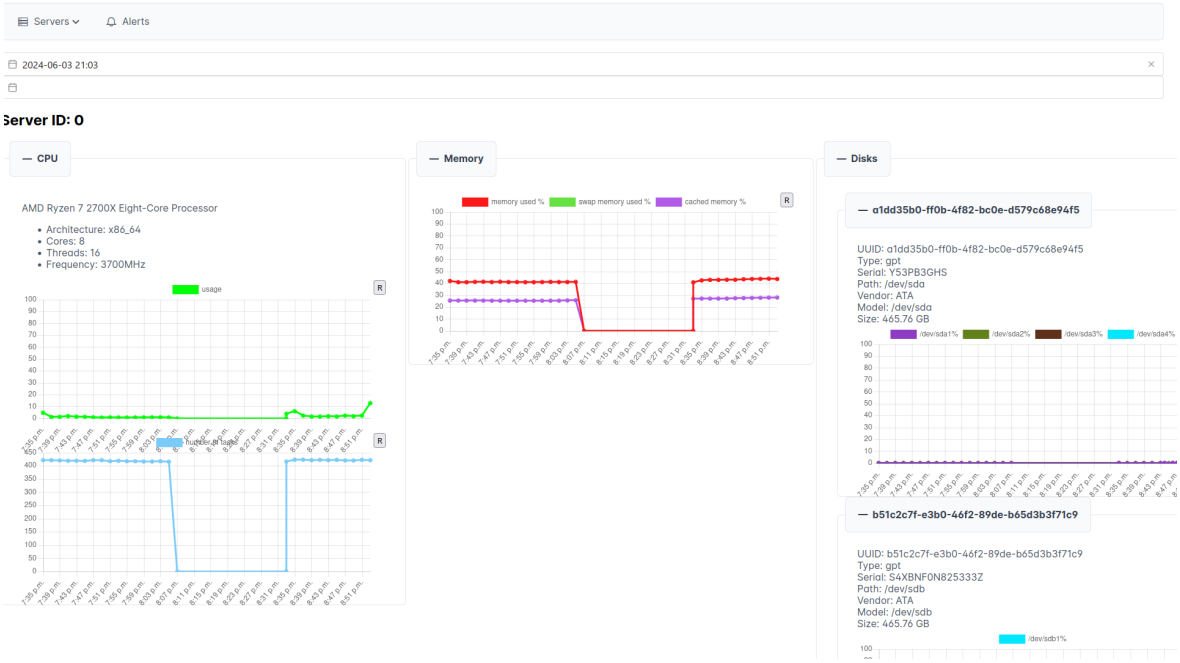
Primjer metode koja vraća točke za graf particije:

```
  static PartitionLogsToDataset(partitionLabel: string,
     color: string, logs: PartitionLog[]){
    if (logs == null || logs.length === 0) return {datasets
       : []}

    let usagePoints: object[] = []
    logs.forEach(l => {
      usagePoints.push({
        x: l.date,
        y: l.usedPercentage == null ? 0 : l.usedPercentage
           * 100
      })
    })

    return {
      showLine: true,
      label: partitionLabel + "%",
      borderColor: color,
      backgroundColor: color,
      data: usagePoints
    }
  }
}
```

Slika 9.5. Vraćanje točaka grafa za podatke particije

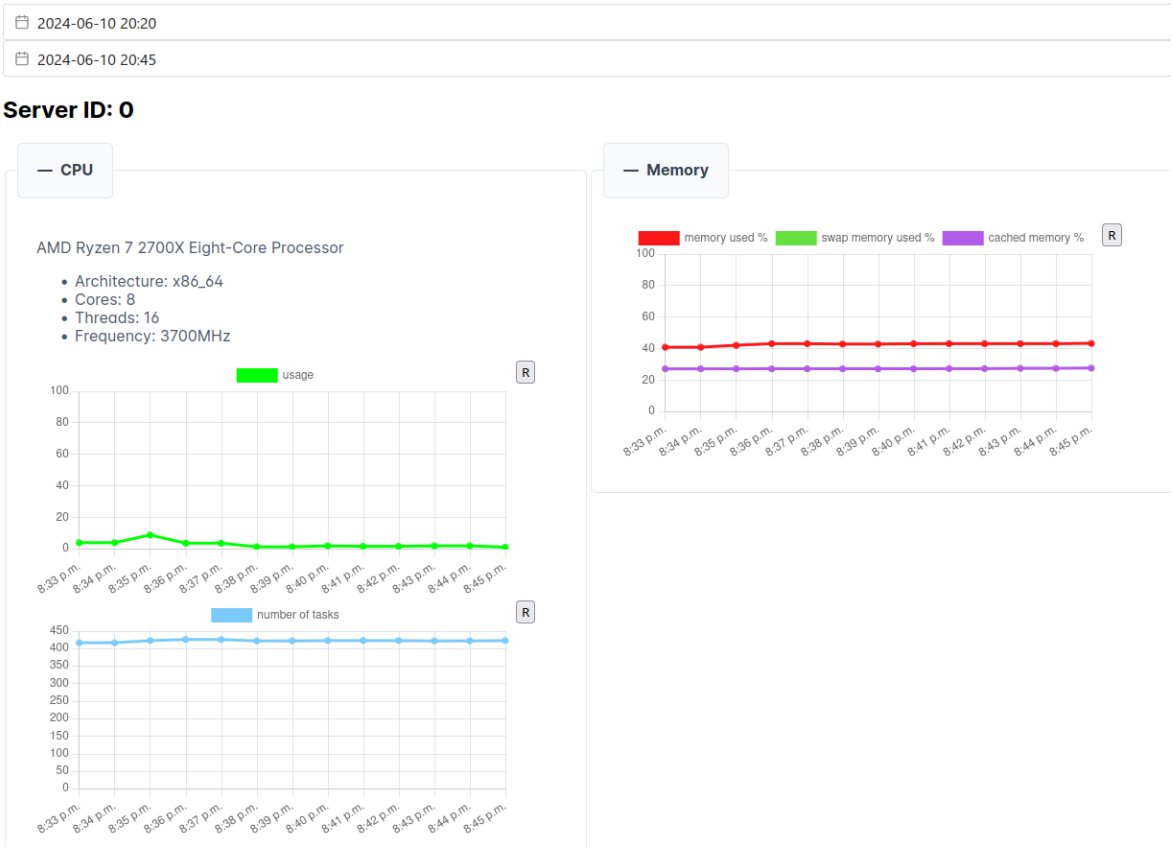
10. Korisničko sučelje i interakcija

Otvaranjem web-stranice se prikazuje glavna stranica. Na njoj se prikazuje izbornik sa listom poslužitelja i karticom za obavijesti. Ukoliko barem jedan poslužitelj postoji, prikazuju se podaci o prvom poslužitelju.



Slika 10.1. Početna stranica

Moguće je odabrati početni i završni datum, čijim mijenjanjem se osvježavaju svi podaci web stranice.



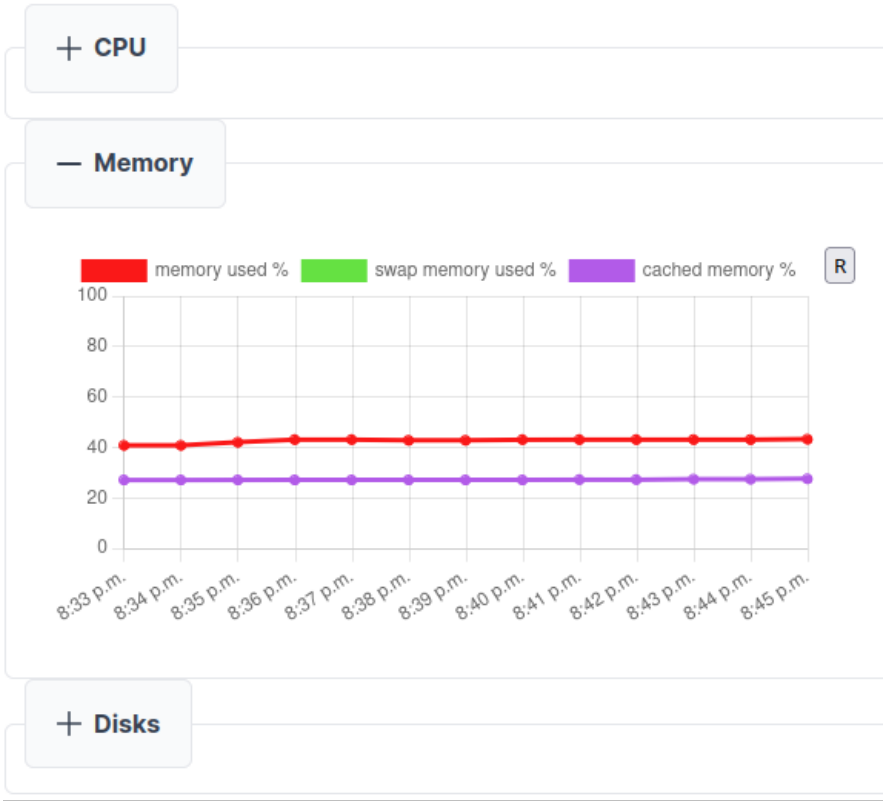
Slika 10.2. Početna stranica sa filtriranim datumom

Prilikom učitavanja podataka za graf se prikazuje poruka *Loading...*, a u slučaju da podataka nema se prikazuje *No data...*

Pojedinačne grafove je moguće uvećati čime se osvježavaju podaci. Graf je također moguće resetirati čime se osvježavaju podaci za originalni period.

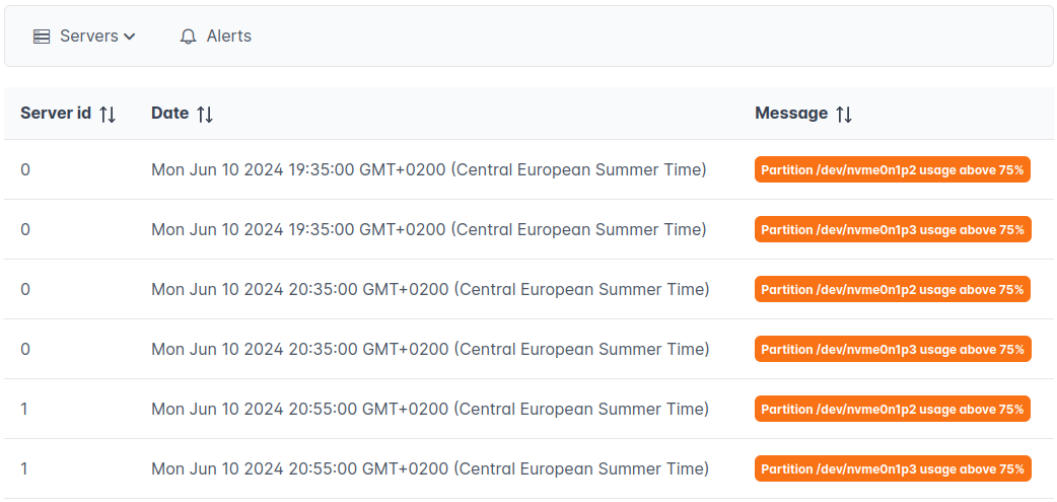
Slika 10.3. Primjer uvećanja grafa *Memory*

Kartice komponenti je moguće smanjiti čime se mijenja izgled stranice (proširuju se komponente kako bi stale na ekran).



Slika 10.4. Izgled web-stranice nakon umanjena kartica

Klikom na karticu *Alerts* se prikazuju obavijesti i upozorenja.



Slika 10.5. Prikaz obavijesti i upozorenja

Web-stranica podržava široke ekrane te mobilne uređaje.

Slika 10.6. Web-stranica na mobilnim uređajima

11. Komentari

U trenutnom obliku, napravljeno rješenje nije bolje od raznih drugih rješenja za praćenje statusa poslužitelja. Svako od tih rješenja nudi neka poboljšanja od sigurnosti, optimizacije, bolje administracije, osvježavanja podataka u stvarnom vremenu te brojne druge funkcionalnosti i poboljšanja.
Uz dovoljno vremena i truda moguće je implementirati te funkcionalnosti i brojne druge koje bi učinili ovaj alat vrlo korisnim za administraciju i praćenje poslužitelja. U ovom poglavlju će biti opisani razni aspekti aplikacije koji se mogu poboljšati ili dodati.

11.1. Sigurnost

Web sučelju i programsko sučelje nemaju implementiranu autentifikaciju što omogućava bilo kome da pristupi njima ukoliko može pristupiti web mreži i portu na kojem se aplikacije izvršavaju. Ovo predstavlja veliki sigurnosni propust, pogotovo za velike mreže poslužiteljima, o kojima bi se mogli pročitati povjerljivi podaci i dodati neispravni podaci u bazu podataka. Kako bi se ovo spriječilo potrebno je dodati određenu metodu autentifikacije kao OAuth2, čime bi se pristup omogućilo samo korisnicima sa ispravnom lozinkom ili tokenom.

11.2. Brzina rada

PostgreSQL baza podataka omogućava indekse koji nisu korišteni u ovom rješenju. Indeksi se koriste radi brzog pretraživanja podataka po određenom polju (na primjer po datumu) što može znatno ubrzati aplikaciju, pogotovo ako se nakupila velika količina podataka o više poslužitelja.
Također, prilikom dohvaćanja podataka iz programskog sučelja se uvijek učitavaju log bez obzira da li ih korisnik želi dohvatiti iako se ne vraćaju. Ovo znatno usporava apli-

kaciju jer je najviše vremena potrebno za čitanje i procesiranje velike količine logova. Potrebno je naći način da se dohvate logovi samo kada ih korisnik zapravo želi vratiti.

11.3. Administracija poslužitelja

Nakon konfiguracije i pokretanje Monitor aplikacije na poslužitelju, moguće je vidjeti podatke o top poslužitelju na web-stranici, ali nije moguće preko programskog sučelja ili web-stranice mijenjati ikakve postavke poslužitelja. Rad sa većom količinom poslužitelja bi znatno olakšala implementacija sučelja za administraciju unutar web-aplikacija preko kojeg bi se moglo zaustaviti slanje podataka od određenog poslužitelja, podesiti obavijesti i upozorenja koje poslužitelj šalje, interval u kojem se šalju podaci te razne druge postavke.
Implementacija ugrađenog *ssh terminala* bi također znatno olakšala rad sa polsužiteljima jer bi se na jednom mjesto moglo iz daljine upravljati poslužiteljima i izvršavati naredbe na njima.

11.4. Više kategorija podataka

Trenutno se samo prate podaci o procesoru te trajnoj i radnoj memoriji. Djelomično je implementirana i podrška za praćenje programa i servisa, no ona bi se trebala dovršiti. Također se mogu dodati podaci o poslužitelju (vrijeme rada, status, obavijesti te drugo), mreži (IP adrese, praćenje stanja registriranih domena, graf slanja i primanja podataka) te razni drugi podaci koji bi bili korisni administratorima poslužitelja radi otkrivanja grešaka.

11.5. Mail obavijesti

Obavijesti su trenutno vidljive samo na web-stranici. Ovo je korisno za pregled velike količine obavijesti ali je problem u tome što administratori moraju često posjećivati web-stranicu kako bi vidjeli da li je došla nova obavijest ili upozorenje. Moglo bi se osim praćenja obavijesti na web-stranici dodati opcija za slanje podatka na adresu e-pošte administratora kako bi mogao na vrijeme vidjeti obavijest i popraviti grešku ukoliko je došlo do problema sa radom nekog poslužitelja.

11.6. Osvježavanje podataka

Web-stranica omogućava pregled podataka za određeni vremenski period. Korisnik može polje za krajnji datum ostaviti prazno kako bi se dohvatili svi podaci do zadnje poslanih podataka, no nakon dohvaćanja tih podataka oni se neće osvježiti dolaskom novih podataka. GraphQL omogućava korištenje *subscription* komponente kako bi se korisnik mogao prijaviti na određeni događaj i dobiti nove podatke kada se oni pošalju. Implementacija *subscripiton* komponente bi omogućila osvježavanje podataka u stvarnom vremenu.

11.7. Podrška za Windows

.NET aplikacije se mogu izvršavati na Linux i na Windows operacijskom sustavu što omogućuje ovom rješenju da se koristi i za praćenje podataka o Windows poslužiteljima uz određene promijene. Najveći problem podržavanja više operacijskih sustava je drugačije dohvaćanje podataka o računalu (na Linux operacijskom sustavu se koriste određeni alati kao lsblk koje Windows ne podržava) zbog čega bi se trebali koristiti drugi alati i drugačije čitanje i procesiranje podataka. Jedno od potencijalnih rješenja je instalacija "Windows Subsystem for Linux" rješenja koje bi omogućilo korištenje Linux alata na Windows operacijskom sustavu, no trebalo bi se testirati ovo rješenje jer je moguće da određeni alati ne bi radili ili bi neispravno radili na Windowsu.

Sažetak

Sustav za praćenje stanja poslužitelja temeljen na jeziku GraphQL

Dominik Dejanović

Unesite sažetak na hrvatskom.

Ključne riječi: prva ključna riječ; druga ključna riječ; treća ključna riječ

Abstract

GraphQL-based server monitoring system

Dominik Dejanović

Enter the abstract in English.

Keywords: the first keyword; the second keyword; the third keyword

Privitak A: The Code