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 se iznimno oslanja na poslužitelje kako bi ostvarili razne funkcionalnosti koje su potrebne većini ljudi u svakodnevnom životu, od pretraživanje raznih web stranica i videa na internetu do povezanosti ogromnih mreža računala u virtualno super-računalo, poslužitelju su neophodni u tom procesu.

Nažalost nije dovoljno konfigurirati poslužitelj za obavljanje određene zadaće i nadati se da će raditi zauvijek. Zbog raznih problema kao prirodne katastrofe, pogrešake u kodu, virusi i hakeri, neispravan rad komponenti, prevelikog broja zahtjeva te drugih problema koji se često događaju, 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 poslužitlja i obaviještavanje administratora u slučaju neispravnog rada. Problem koji se javalja kod velike većine ovakvih aplikacija je pregled specifičnog vremenskog perioda u kojem se dogodila greška, potreba za plaćanjem za naprednije funkcionalnosti aplikacije te pohranjivanje podataka samo u zadnjih nekoliko dana ili tjedana.

Cilj ove aplikacije je omogućiti praćenje jednog ili više poslužitelja kroz neograničen period vremena, što omogućava administratorima da pregledaju podatke o radu poslužitelja za specifični period vremena tijekom kojeg je nastala greška na poslužitelju.

2. Funkcionalnosti

Aplikacije su namijenjene administratorima poslužitelja koji mogu:

- dodati poslužitelje koji se prate
- konfigurirati poslužitelj:
  - podesiti podatke koji (na primjer prate se podaci procesora, a podaci o pohrani ne)
  - podesiti putanju na kojoj se nalazi API
  - podesiti interval (u minutama) u kojem se šalju podaci na server
- pregledati podatke o pojedinom poslužitelju
  - pregledati podatke unutar određenog vremenskog perioda
  - zoomirati i osvježiti pojedine grafove
- pregledati obavijesti o poslužiteljima

3. Korištene tehnologije

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

3.1. Linux

Ovaj projekt je namijenjen za prikupljanje i prikaz podataka o poslužiteljima koji koriste Linux operacijski sustav. Razlog tome je sve veća popularnost linux u poslužitelja. 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

GrapQL je jezik korišten za dohvaćanje i slanje podataka na API. Odabran je GraphQL umjesto REST tehnologije zbog nekoliko prednosti koje GraphQL ima:

- 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
    }
    ]
  }
  ]
}
```

- 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"
    }
  }
}
```

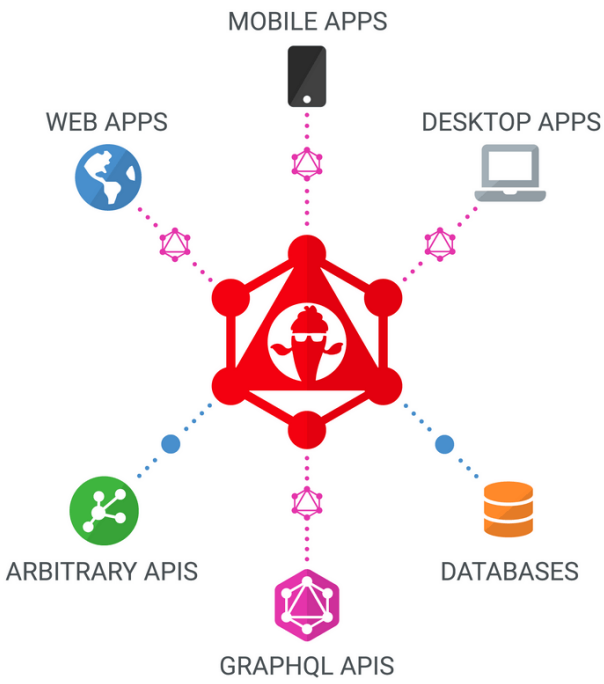
3.3. HotChocolate

HotChocolate je implementacija GraphQL poslužitelja u C# programskom jeziku. Ona nam omogućava korištenje svih funkcionalnosti GraphQL tehnologije bez da ih moramo ručno implementirati.

Korištena verzija: 13.6.0

ADD IMAGE REFERENCE https://chillicream.com/static/cfd2ddde71f95ed876541f87c15b2a08/78d4

https://chillicream.com/docs/hotchocolate/v13



Slika 3.1. HotChocolate server

3.4. .NET i C#

Za programiranje API-a i Monitor-a se koristi C# programski jezik i .NET framework.

Korištena .NET verzija: net8.0

Korištena C# verzija: 12.0

3.5. linq2db

linq2db paket 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 ...
}
```

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

3.6. Newtonsoft.Json

Koristi se za serijalizaciju i deserijalizaciju JSON podataka. Newtonsoft.Json repozitorij: https://github.com/JamesNK/Newtonsoft.Json

3.7. Vue.js

Vue.js je Javascript framework, koristi se za jednostavniji rad sa prikazom podataka i za bolje strukturiranje koda. 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>
```

Korištena verzija: 3.2.13

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

3.7.1. 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/`

3.7.2. vue-chartjs

vue-chartjs je Vue biblioteka koja se koristi za prikaz grafova ovisno o predanim parametrima.

Korištena verzija: 5.2.0

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

3.7.3. vue-datepicker

vue-datepicker je Vue komponenta koja se koristi za odabir datuma i vremena.

Korištena verzija: 7.1.0

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

3.8. PostgreSQL

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`

3.9. Git i Github

Git je korišten kao sustav za praćenje verzija koda. Također je korišten Github koji omogućava pristup kodu i uputama za instalaziju.
Službena git stranica: `https://git-scm.com`
Github: `https://github.com/`

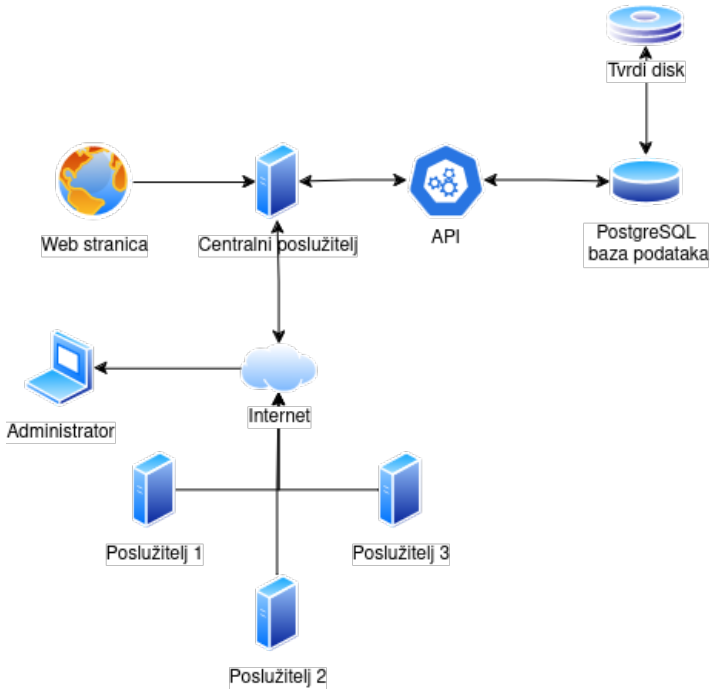
4. Opis rješenja

4.1. Softver

Napravljene su tri .NET projekta (API, Common i Monitor), web stranica te baza podataka.
API 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 API-u 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 API generirao prilikom upisa podataka u bazu podataka.

4.2. Fizička konfiguracija

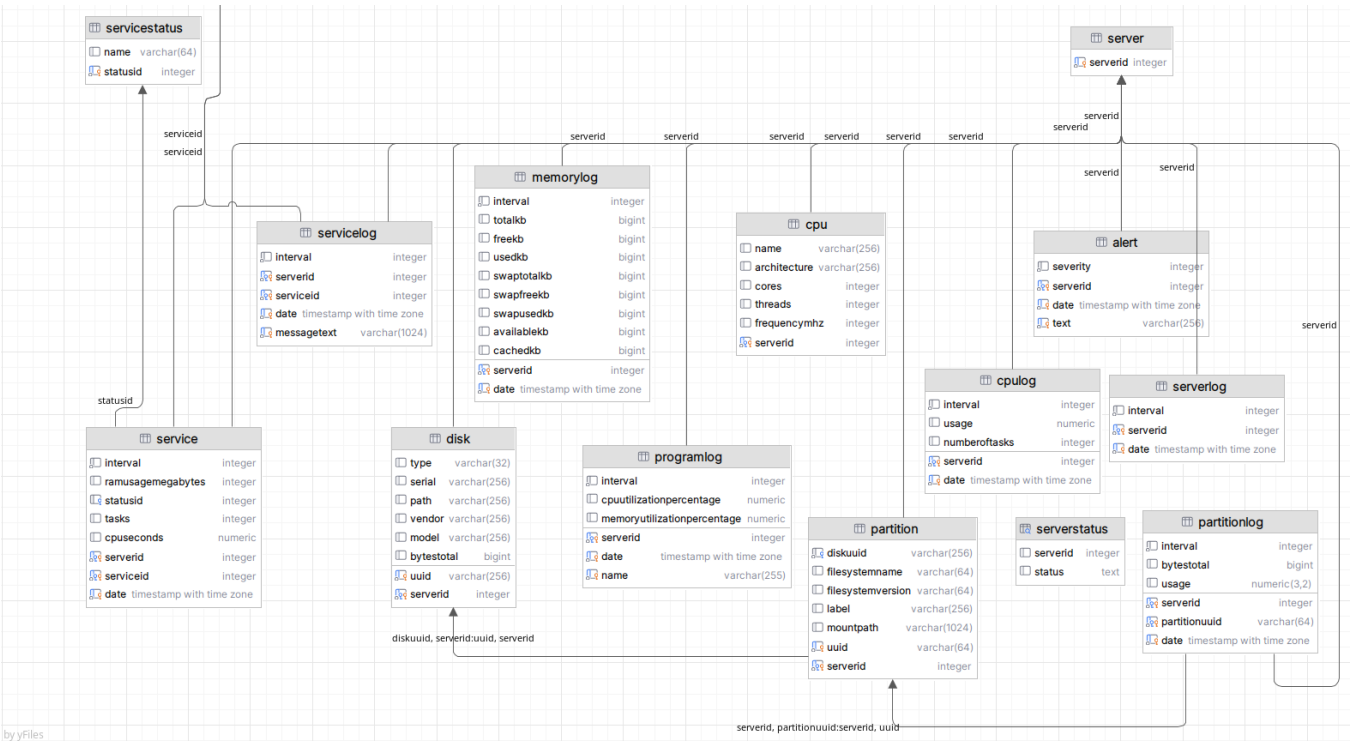
Postoji jedan centralni poslužitelj na kojem su pokrenute dvije aplikacije: Vue web stranica te API. Vue web stranica se dohvaća HTTP protokolom, nakon čega ona šalje zahtijeve API aplikaciji na centralnom poslužitelju radi dohvata podataka iz baze podataka.
Poslužitelji koji su konfigurirani za praćenje podataka također komuniciraju sa API aplikacijom, ali ne za dohvaćanje nego za slanje podataka API-u koji onda te podatke sprema u bazu podataka.
U nastavku slijedi detaljan opis funkcionalnosti i koda svih projekata.



Slika 4.1. Dijagram rješenja

5. Baza podataka

Za izradu baze podataka je korišten 3.8. 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).

serverStatus pogled

```
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;
```

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

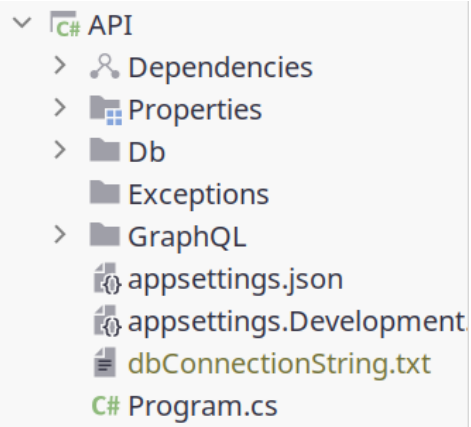
Stvorena je tablica za obavijesti koja pohranjuje kritičnost poruke (severity), ID poslužitelja za koji je relevantna (serverid) te datum (date) i tekst poruke (text).
Uz tablicu je također napravljena i metoda before_alert_insert_func() koja se poziva prije unosa podataka u tablicu. Metoda je stvorena 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).

before_alert_insert_func()

```
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;
```

6. API

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



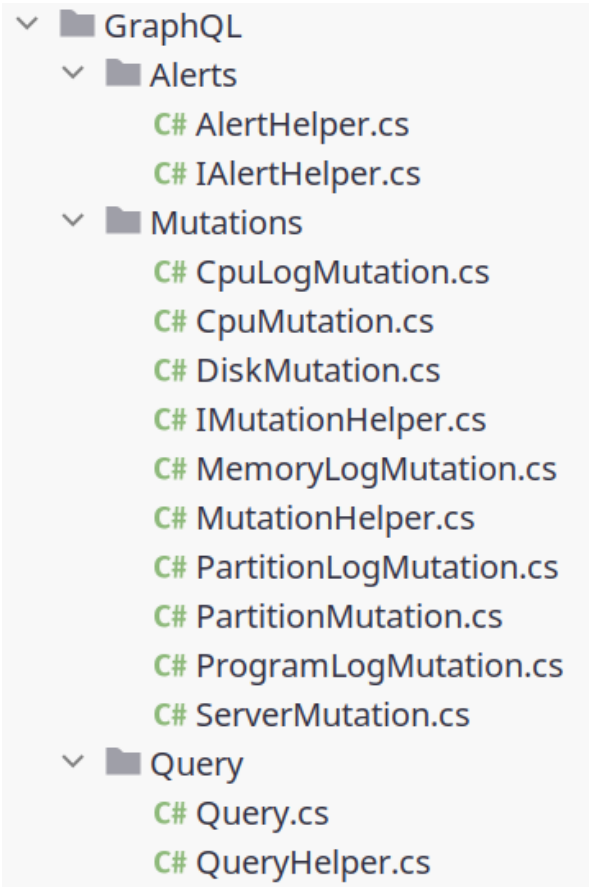
Slika 6.1. Struktura API projekta

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 API-a. 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 6.2.2. GetLogs metodu za dohvat logova iz baze podataka.
Primjer jedne takve metode:

Cpu metoda

```
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);
    }
  }
```

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:

Izračun intervala

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

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.

Parametri GetLogs metode

```
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
```

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.

AddLog metoda

```
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();
}
```

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:

CpuMutation.cs

```
[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);
    }
```

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.

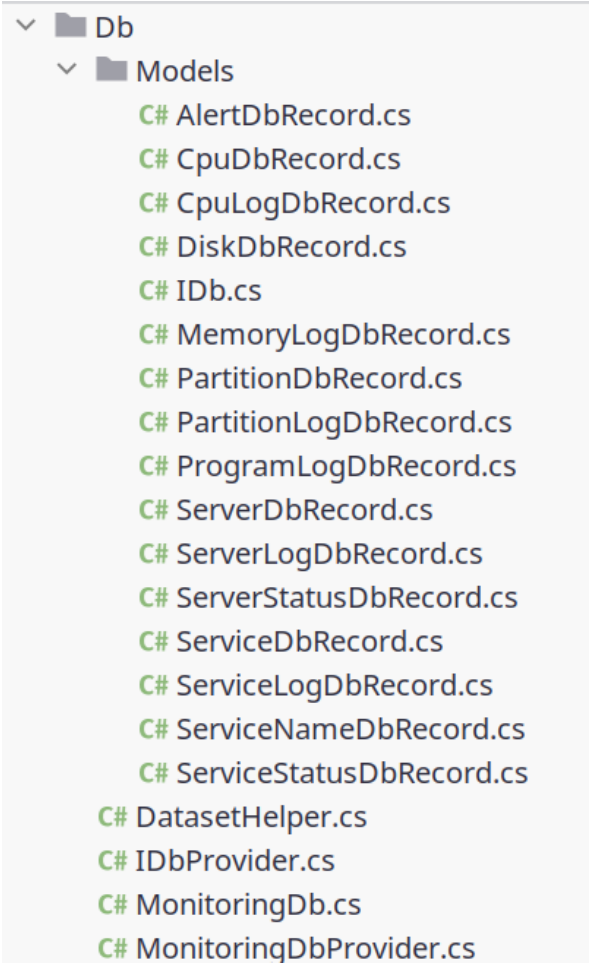
IAlertHelper.cs

```
public interface IAlertHelper
{
```

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

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 3.5. linq2db paketa pomoću naredbe "dotnet linq2db scaffold -p PostgreSQL -c "Host=localhost; Username=[username]; Password=[password];Database=[database]""



Slika 6.3. 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; }
}
```

- 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:

dbConnectionString.txt

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

6.5. Pokretanje API-a

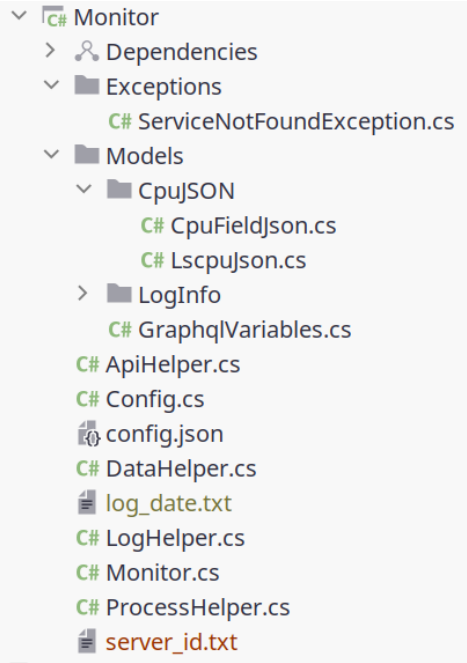
API se pokreće iz komandne linije/terminala pomoću naredbe "dotnet run" unutar direktorija u kojem se API nalazi.
Nakon pokretanja se može pristupiti URL-u `http://localhost:3000//graphql`, prilikom čega se otvara web stranica na kojoj se može vidjeti dokumentacija API-a te se mogu izvršavati upiti.

Slika 6.4. Dohvat podataka o pohrani

Slika 6.5. Dodavanje jedne particije

7. Monitor

Aplikacija Monitor se koristi za prikupljanje podataka o poslužitelju te slanje tih podataka API-u.



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 API-u
- 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

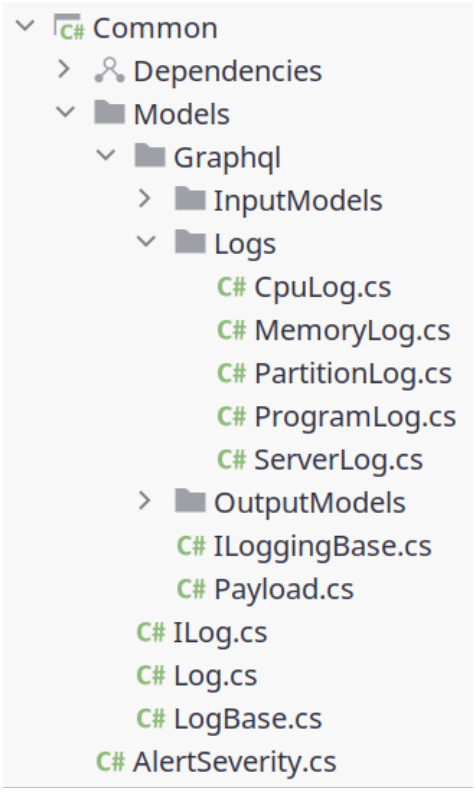
Pokreće se prilikom pokretanja aplikacija. Na početku učitava konfiguracijsku datoteku, te zatim generira nasumični broj za ID poslužitelja. Na kraju se u beskonačnoj petlji pokreće proces prikupljanja podataka te slanje tih podataka API-u.

```
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);
    }
  }
```

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; }
}
```

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:

CpuInput.cs

```
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)
  {
  }
}
```

- 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:

MemoryOutput.cs

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

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

9. Web stranica

Web stranica je napravljena pomoću 3.7. Vue frameworka.

9.1. public/index.html

Koristi se Vue framework te se zbog toga koristi samo jedna HTML stranica, u kojoj se prikazuju renderirane komponente.

9.2. App.vue

Glavna .vue datoteka koja kontrolira sadržaj koji se prikazuje, stvara se prilikom pokretanja programa.

9.3. src/components direktorij

Vue framework koristi komponente kako bi se dijelovi koda mogli ponovno koristiti. Sve komponente se nalaze u ovom direktoriju. Primjer dijela jedne takve komponente:

MemoryInfo.vue

```
    },
    "serverId": async function() {
      await this.refreshData()
    }
  }
}
</script>
```

```
<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>
```

9.4. src/models direktorij

Models direktorij sadržava klase koje se koriste za pohranjivanje podataka dohvaćenih s API-a. Primjer klase:

Partition.ts

```
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
  }
}
```

9.5. api.ts

Zadužen za dohvaćanje podataka s API-a. Primjer dohvaćanja podataka o procesoru za trenutni server:

getCpu()

```
  private static dateToString(date: Date){
    if(date === null) return "null"
    return `"${moment(date).format("yyyy-MM-DD HH:mm")}"`
  }

  static async getCpu(serverId: number, startDate: Date,
     endDate: Date) {
    let startDateString = this.dateToString(startDate)
    let endDateString = this.dateToString(endDate)
    const queryString = `
    {
      cpu(serverId: ${serverId}, startDateTime: ${
         startDateString}, endDateTime: ${endDateString}) {
        serverId,
```

```
        name,
        architecture,
        cores,
        threads,
        frequency,
        logs{
          date
          usage
          numberOfTasks
        }
      }
    }‘

    let response = await this.executeQuery(queryString)
```

9.6. ChartHelper.ts

Pomoćna datoteka koja se koristi za pretvorbu podataka koje API vraća u točke na grafu.

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