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

2. Opis problema

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. 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

```
# 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
      }
      ]
    }
    ]
  }
```

ADD REFERENCE TO https://graphql.org/learn/validation/

- 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 točno 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

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

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

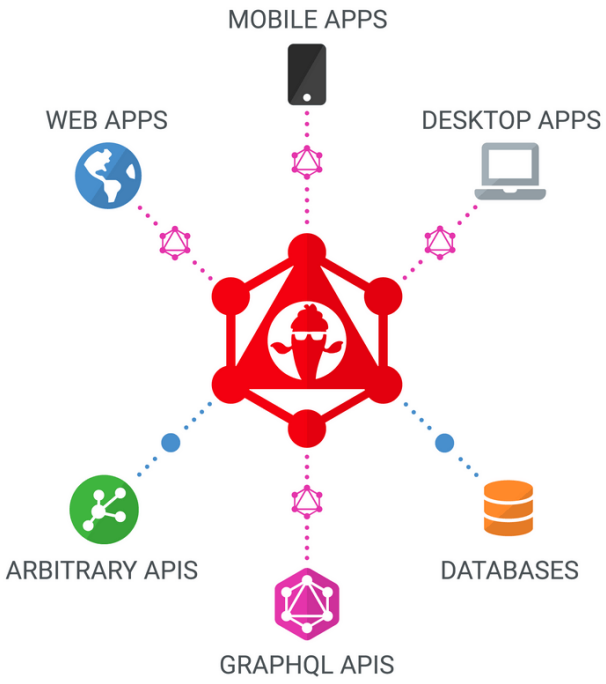
```
        "name": "R2-D2"
      }
    }
  }
```

ADD REFERENCE https://graphql.org/learn/queries/

3.2. 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



Slika 3.1. HotChocolate server

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

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

3.3. .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

3.4. 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`

3.5. Newtonsoft.Json

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

3.6. 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:

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

ADD CODE REFERENCE TO https://vuejs.org/guide/introduction.html

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

3.6.1. vue-chartjs

DOVRSI OVO

Dokumentacija: https://vue-chartjs.org

3.6.2. vue-collapsible-panel

DOVRSI OVO

Dokumentacija: https://github.com/dafcoe/vue-collapsible-panel

3.7. PostgreSQL

Odabran je PostgreSQL kao RDBMS sustav za upravljanje bazom podataka zato što je otvorenog koda te ima opširnu dokumentaciju i korisničku podršku.

PostgreSQL dokumentacija: `https://www.postgresql.org/docs`

3.8. Git

Git je korišten kao sustav za praćenje verzija koda.

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

4. Opis rješenja

Napravljene su tri .NET projekta, jedan Vue projekt te baza podataka. Slijedi detaljan opis svih aplikacija.

4.1. Baza podataka

4.2. API

API aplikacija je implementirana koristeći 3.2. HotChocolate server paket koji implementira funkcionalnosti GraphQL.
Struktura projekta: SLIKA STRUKTURE

4.2.1. 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.
ADD CODE

4.2.2. 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. Primjer jedne takve metode:

Cpu metoda

```
public async Task<CpuOutput?> Cpu(int serverId, DateTime?
    startDateTime, DateTime? endDateTime, int? 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;
    await foreach (var log in QueryHelper.GetLogs(db.
        CpuLogs, combineLogsFunc, getEmptyRecordFunc, _ =>
         "", startDateTime, endDateTime, interval))
    {
```

```
        cpuOutput.Logs.Add(log);
      }
    }

    return cpuOutput;
  }
```

4.2.3. QueryHelper.cs

Pomoćna klasa koju koristi 4.2.2. 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;
```

Parametri GetLogs metode

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

4.2.4. 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. linq2db paketa pomoću naredbe SCAFFOLD
Neke od važnijih klasa (te one koje nisu automatski generirane):

- Models/IDb.cs - sučelje koje sadržava popis svih tablica u bazi podataka ADD IMAGE
- DatasetHelper.cs - pomoćna klasa korištena u algoritmu spajanja više podataka u jedan podatak REFERENCE NA ALGORITAM
- 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)

4.2.5. 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 ADD PIC OF INTERFACE
- MutationHelper - implementacija IMutationHelper sučelja

Primjer dijela jedne mutacije:

CpuMutation.cs

```
[ExtendObjectType(OperationType.Mutation)]
public class CpuMutation(IDbProvider dbProvider,
   IMutationHelper mutationHelper)
{
    private readonly Func<CpuIdInput, IQueryable<
       CpuDbRecord>> _getCpuQuery = cpu =>
        from c in dbProvider.GetDb().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);
    }
```

4.2.6. 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]
```

4.2.7. 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 4.1. Dohvat podataka o pohrani

Slika 4.2. Dodavanje jedne particije

4.3. Monitor

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

Struktura aplikacije: ADD IMAGE

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

4.3.1. Monitor.cs

Pokreće se prilikom pokretanja aplikacija. Na početku učitava konfiguracijsku datoteku, te zatim pomoću 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,
```

```
            DateTime.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);
        }
    }
```

4.4. Common

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

4.4.1. ILog.cs

Glavno sučelje, svaki Log nasljeđuje polja definirana u njemu.

4.4.2. 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)
  {
  }
}
```

4.5. Web stranica

Web stranica je napravljena pomoću 3.6. Vue frameworka, te se koriste 3.6.1. Chart.vue i 3.6.2. vue-collapsible-panel paketi kako bi se prikazali podaci o poslužitelju.

4.5.1. public/index.html

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

4.5.2. App.vue

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

4.5.3. 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

```
<template>
    <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)
    }"
    />
```

```
</template>
```

4.5.4. 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
  }
}
```

4.5.5. api.ts

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

getCpu()

```
static async getCpu(startDate: Date, endDate: Date) {
  let startDateString = this.dateToString(startDate)
  let endDateString = this.dateToString(endDate)
  const queryString = `
  {
    cpu(serverId: 0, startDateTime: ${startDateString},
       endDateTime: ${endDateString}) {
      serverId,
      name,
      architecture,
      cores,
      threads,
      frequency,
      logs{
        date
        usage
        numberOfTasks
      }
    }
  }`

  let response = await this.executeQuery(queryString)
  console.log("cpu response: ")
  console.log(queryString)
  console.log(response)
  let cpu = response.data.cpu
  return new Cpu(cpu.serverId, cpu.name, cpu.architecture
     , cpu.cores, cpu.threads, cpu.frequency, cpu.logs)
}
```

4.5.6. 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