Univerzita Komenského v Bratislave
Fakulta matematiky, fyziky a informatiky

Spracovanie dlhodobých meraní vybraných charakteristík Internetu

Bakalárska práca

2023
Peter Trenčanský

Univerzita Komenského v Bratislave
Fakulta matematiky, fyziky a informatiky

Spracovanie dlhodobých meraní vybraných charakteristík Internetu

Bakalárska práca

Študijný program:	Informatika
Študijný odbor:	Informatika
Školiace pracovisko:	Katedra informatiky
Školiteľ:	Ing. Dušan Bernát, PhD.

Bratislava, 2023
Peter Trenčanský

Univerzita Komenského v Bratislave
Fakulta matematiky, fyziky a informatiky

ZADANIE ZÁVEREČNEJ PRÁCE

Meno a priezvisko študenta: Peter Trenčanský
Študijný program: informatika (Jednoodborové štúdium, bakalársky I. st., denná forma)
Študijný odbor: informatika
Typ záverečnej práce: bakalárska
Jazyk záverečnej práce: slovenský
Sekundárny jazyk: anglický

Názov: Spracovanie dlhodobých meraní vybraných charakteristík Internetu
Processing and visualisation of long-term measurements of selected Internet metrics

Anotácia: Analyzujte štruktúru existujúcej databázy, ktorá obsahuje výsledky dlhodobých meraní odozvy rôznych uzlov v sieti Internet (výstupu ping, traceroute a pod.). Analyzujte dostupné nástroje pre spracovanie a vizualizáciu týchto dát. Navrhnite a implementujte systém s webovým rozhraním pre prácu s nameranými dátami, ktorý používateľovi umožní predovšetkým výber, triedenie a filtrovanie (podľa adries a času), ako aj výpočet štatistík a vizualizáciu ich hodnôt, časových priebehov, prípadne geografického rozloženia. Systém musí byť modulárny v tom zmysle, aby bolo možné konkrétne štatistiky a výstupy jednoducho dopĺňať. Riešenie overte na zobrazení časového vývoja priemernej, najmenšej a najväčšej nameranej doby odpovede.

Vedúci: Ing. Dušan Bernát, PhD.
Katedra: FMFI.KI - Katedra informatiky
Vedúci katedry: prof. RNDr. Martin Škoviera, PhD.

Dátum zadania: 28.10.2022

Dátum schválenia: 31.10.2022 doc. RNDr. Dana Pardubská, CSc.
garant študijného programu

.....
študent

.....
vedúci práce

Poďakovanie: Tu môžete poďakovať školiteľovi, prípadne ďalším osobám, ktoré vám s prácou nejako pomohli, poradili, poskytli dáta a podobne.

Abstrakt

Slovenský abstrakt v rozsahu 100-500 slov, jeden odstavec. Abstrakt stručne sumarizuje výsledky práce. Mal by byť pochopiteľný pre bežného informatika. Nemal by teda využívať skratky, termíny alebo označenie zavedené v práci, okrem tých, ktoré sú všeobecne známe.

Kľúčové slová: jedno, druhé, tretie (prípadne štvrté, piate)

Abstract

Abstract in the English language (translation of the abstract in the Slovak language).

Keywords:

Obsah

Zoznam obrázkov

Zoznam tabuliek

Úvod

Text pre účely DÚ 2 sa nachádza v kapitolách 1 a 2

Cieľom tejto práce je poskytnúť študentom posledného ročníka bakalárskeho štúdia informatiky kostru práce v systéme LaTeX a ukážku užitočných príkazov, ktoré pri písaní práce môžu potrebovať. Začneme stručnou charakteristikou úvodu práce podľa smernice o záverečných prácach [8], ktorú uvádzame ako doslovný citát.

> Úvod je prvou komplexnou informáciou o práci, jej cieli, obsahu a štruktúre. Úvod sa vzťahuje na spracovanú tému konkrétne, obsahuje stručný a výstižný opis problematiky, charakterizuje stav poznania alebo praxe v oblasti, ktorá je predmetom školského diela a oboznamuje s významom, cieľmi a zámermi školského diela. Autor v úvode zdôrazňuje, prečo je práca dôležitá a prečo sa rozhodol spracovať danú tému. Úvod ako názov kapitoly sa nečísluje a jeho rozsah je spravidla 1 až 2 strany.

V nasledujúcej kapitole nájdete ukážku členenia kapitoly na menšie časti a v kapitole 4 nájdete príkazy na prácu s tabuľkami, obrázkami a matematickými výrazmi. V kapitole **??** uvádzame klasický text Lorem Ipsum a na koniec sa budeme venovať záležitostiam záveru bakalárskej práce.

Kapitola 1

Možnosti implementácie

Každý softvér si musí určiť, pre aké platformu je vyvíjaný. Samozrejme, čím väčšiu škálu platforiem pokryje, tým je dostupnejší pre väčšie množstvo používateľov. Dnes drvivá väčšina používateľov používa istú podmnožinu zo systémov Android, Linux, Windows, iOS a MacOS X. Spoločnou črtou týchto platforiem je, že medzi nimi neexistuje taká dvojica, z ktorej by jedna platforma natívne podporovala programy napísané pre druhú platformu. Vyvíjať aplikáciu päťkrát, pre každú platformu v jej natívnom prostredí, však nie je veľmi efektívne.

Namiesto toho dnes existuje niekoľko riešení umožňujúcich zdieľať časti zdrojového kódu medzi riešeniami pre rôzne systémy. Ako príklad môžme uviesť frameworky Flutter, React Native alebo MAUI pre .NET.

Ako univerzálne riešenie sa dnes používajú webové aplikácie. Takéto aplikácie majú niekoľko dôležitých výhod, ako napríklad rýchla dostupnosť, žiadna nutnosť inštalácie a jednoduché spustenie na všetkých platformách, na ktorých je dostupný webový prehliadač (napríklad Firefox, Chrome alebo Opera). Webové aplikácie zvyknú mávať dve oddelené časti - frontend, bežiaci vo webovom prehliadači klienta a backend, bežiaci na serveri. Tieto dve časti spolu komunikujú pomocou siete, najčastejšie pomocou protokolov HTTP(S) alebo WebSocket. Úlohou frontendu je poskytovať prívetivé rozhranie s ktorým môže používateľ pracovať, úlohou backendu je zase spracovávať dáta, komunikovať s databázou a poskytovať dáta frontendu. Práve pre takýto model som sa rozhodol.

1.1 Možnosti vývoja frontendového riešenia

Programovanie webových stránok sa dnes nezaobíde bez programovacieho jazyka JavaScript, ktorý používajú webové prehliadače. Alternatívne sa dajú použiť prekladače iných jazykov do WebAssembly (napríklad framework Blazor pre .NET) alebo jazyk TypeScript, ktorý je rozšírením JavaScriptu o typový systém a iné užitočné konštruk-

cie, zároveň je plne kompatibilný s JavaScriptom (je jeho nadmnožinou) a vo webových prehliadačoch sa spúšťa pomocou prekladu do JavaScriptu. Na vývoj webového frontendu je možnosť používať webové frameworky, ktoré umožňujú delenie stránok na moduly, ktoré sa dajú jednotlivo znovupoužívať a paranetrizovať na rôznych podstránkach aplikácie, čím zjednodušujú spravovanie a čitateľnosť aplikácie. V súčasnej dobe sú najpopulárnejšie frameworky React, Vue a Angular. Kým React a Vue umožňujú prácu v JavaScripte ale volitelne je možné vyvíjať aj v TypeScripte, Angular prácu v TypeScripte vyžaduje. Tieto frameworky sú podporované vo všetkých nových verziách moderných prehliadačov. Keďže všetky tieto frameworky fungujú na veľmi podobnom princípe, je veľmi ťažké povedať, ktorý by bola pre našu aplikáciu najvhodnejší. Vzhľadom na to sme sa rozhodli pre Angular, v ktorom máme naviac praktických skúseností.

1.2 Možnosti vývoja backendového riešenia

Pre vývoj backendového riešenia je vhodné vybrať taký programovací jazyk, pre ktorý existuje framework umožňujúci vývoj API, ktoré bude podporovať protokol HTTP(s). Takýchto jazykov existuje veľmi veľa, dokonca pre niektoré jazyky existuje viacero rôznych knižníc umožňujúcich výstavbu takýchto API. Ako príklad môžme uviesť jazyk Python a knižnice FastAPI alebo Flask a mnohé iné, jazyk Java a framework Spring Boot, jazyk C# a framework ASP.NET Core a mnohé iné. Všetky tieto možnosti majú dostatočné nástroje a je ťažké určiť, ktorý z nich je objektívne najvhodnejší, preto sme sa rozhodli pre jazyk C# a ASP.NET Core, s ktorým máme najviac skúseností.

1.2.1 Možnosti implementácie vo vybranom jazyku

Najprv sa musíme rozhodnúť, ktorú verziu frameworku .NET použijeme. Momentálne sú podporované dve verzie - .NET 6.0 a .NET 7.0. Kým .NET 6.0 používa verziu jazyka C# 10, .NET 7.0 používa verziu jazyka C# 11. Verzia jazyka C# 11 obsahuje oproti verzii 10 viaceré nové funkcionality, ako napríklad generické atribúty, povinné členy alebo vylepšenú prácu s konštantnými reťazcami. .NET 6.0 je však LTS verzia kým .NET 7.0 nie je, čo má za následok, že podpora .NET 6.0 časovo prekračuje podporu .NET 7.0 (máj 2024 oproti november 2024). Keďže sa viem zaobísť bez nových funkcionalít, rozhodol som sa použiť C# 10 a .NET 6.0.

Napriek tomu, že už sme si vybrali jazyk, stále máme niekoľko rôznych možností, ako poňať vývoj aplikácie, najmä čo sa týka organizácie kódu alebo knižníc. Na komunikáciu s databázou sa dá použiť buď priamo knižnica ADO.NET, ktorá umožňuje manuálne vykonávanie SQL dotazov a ich ručné mapovanie do objektov, alebo knižnice typu ORM (Object Relational Mapping), ktoré pôsobia ako wrapper pre ADO.NET a sú schopné istú časť mapovania vykonať za nás. Najpoužívanejšie ORM v C# sú

Algoritmus 1.1: Ukážka kódu pre ten istý dopyt pomocou rôznych knižníc - ADO.NET

```
1     private NpgsqlConnection CreateConnection()
2     {
3         return new NpgsqlConnection(_connectionString);
4     }
5
6     public async Task<IEnumerable<IpAddressInfo>> GetIpAddresses(
          int offset = 0, int limit = int.MaxValue)
7     {
8         await using var connection = CreateConnection();
9         await connection.OpenAsync();
10        var command = connection.CreateCommand();
11        command.CommandText = "SELECT * FROM IpAddresses LIMIT
              @limit OFFSET @offset";
12        command.Parameters.AddWithValue("@limit", limit);
13        command.Parameters.AddWithValue("@offset", offset);
14        var reader = await command.ExecuteReaderAsync();
15        var result = new List<IpAddressInfo>();
16        while (await reader.ReadAsync())
17        {
18            result.Add(new IpAddressInfo()
19            {
20                City = reader["City"] as string,
21                CountryCode = reader["CountryCode"] as string,
22                Id = Convert.ToInt32(reader["Id"]),
23                IpStringValue = reader["IpStringValue"] as string,
24            });
25        }
26
27        return result;
28    }
```

Entity Framework Core (ďalej len EFC) a Dapper. Kým Dapper mapuje len výsledky dopytov na objekty (tzv. micro-ORM), EFC je schopné mapovať kód v jazyku C# a knižnicy LINQ na SQL príkazy, čím v jednoduchších aplikáciách dokáže úplne odbúrať nutnosť písania SQL dopytov ručne.

Vzhľadom na možnú zložitosť dopytov sme sa rozhodli nepoužiť EFC, no použijeme Dapper ako skratku pre odbúranie opakujúceho sa kódu vznikajúceho pri používaní ADO.NET (otváranie a zatváranie spojenia, definície paramterov, mapovanie výsledkov na objekty).

Pri štruktúre kódu je nutné sa rozhodnúť, či budeme endpointy deliť do controllerov alebo nie (tzv. minimal API). Kým vyhodou controllerov je lepšia štrukturovateľnosť a väčšia prehľadnosť pri väčšom množstve endpointov, výhodou mimal API je mierne

Algoritmus 1.2: Ukážka kódu pre ten istý dopyt pomocou rôznych knižníc - Dapper

```
1     private NpgsqlConnection CreateConnection()
2     {
3         return new NpgsqlConnection(_connectionString);
4     }
5
6     public async Task<IEnumerable<IpAddressInfo>> GetIpAddresses(
          int offset = 0, int limit = int.MaxValue)
7     {
8         await using var connection = CreateConnection();
9         return await connection.QueryAsync<IpAddressInfo>("SELECT *
              FROM IpAddresses LIMIT @limit OFFSET @offset", new {
              limit, offset });
10    }
```

Algoritmus 1.3: Ukážka kódu pre ten istý dopyt pomocou rôznych knižníc - EFC

```
1     private ApplicationDbContext _dbContext;
2
3     public async Task<IEnumerable<IpAddressInfo>> GetIpAddresses(
          int offset = 0, int limit = int.MaxValue)
4     {
5         return await _dbContext.IpAddresses.Skip(offset).Take(limit
              ).ToListAsync();
6     }
```

menej kódu a potenciálny kratší čas písania. Keďže aplikácia môže mať väčšie množstvo endpointov, rozhodli sme sa použiť controllery.

1.2.2 Databáza

S backendovým riešením úzko súvisí aj výber databázového providera pre ukladanie spracovaných dát. V dnešnej dobe sa používajú dva základné typy databáz - SQL a NoSQL. Keďže naše dáta budú mať jasnú štruktúru, bude vhodné použiť jedného z mnohých SQL providerov, napríklad PostgreSQL, MySQL, MS SQL SQLite atď. Keďže databáza s údajmi o meraniach je PostgreSQL, rozhodli sme sa použiť tiež PostgreSQL pre zachovanie uniformity.

Kapitola 2

Požiadavky a návrh riešenia

2.1 Analýza požiadaviek

Požiadavky na aplikáciu vznikli po dohode s Ing. Dušanom Bernátom, PhD. Cieľom je vytvoriť webovú aplikáciu ktorá bude schopná vizualizovať výsledky meraní internetu v grafickej forme pomocou diagramov, grafov a máp.

2.1.1 Požiadavky na beh aplikácie

Požiadavky na beh serverovej časti:

- Systém musí byť schopný pracovať s PostgreSQL
- Systém nesmie zapisovať do databázy s meraniami
- Všetky časti systému musia bežať na Linuxovom prostredí (CentOS alebo Ubuntu)
- Systém musí byť schopný priebežne predspracovávať namerané dáta po týždňoch, aby boli vizualizácie zobrziteľné ihneď (bez dlhého čakania)
- Systém nesmie byť závislý od vonkajších servisov iných, ako databáza s meraniami (lokalizačné servisy aj mapové servery musia bežať vrámci systému)

Požiadavky na beh klientskej časti:

- Aplikácia musí bez problémov bežať na všetkých moderných prehliadačoch
- Aplikácia musí vedieť vizualizácie zobrazovať bez pridlhého načitavania

Detaily implementácie boli ponechané na moje uváženie. Detaily rozhodovania sú popísané v kapitole 1.

Serverový systém bude implementovaný na .NET 6.0. Systém bude pozostávať z niekoľkých aplikácií, z ktorých jedna bude pôsobiť ako server pre klientskú časť, jedna

bude pôsobiť ako API pre vyťahovanie spracovaných dát a zvyšné budú background workery, ktoré budú mať za úlohu priebežne spracovávať namerané dáta. Predspracované dáta sa budú ukladať do PostgreSQL 15.0 databázy. Tieto workery budú spúšťané task schedulerom jedenkrát týždenne. Systém bude bežať v niekoľkých kontaineroch v platforme Docker. Ako samostatný kontainer bude bežať databáza, API na prístup k dátam, nginx server pre klientskú aplikáciu, každý background worker a Open Street Maps tile server. Ako údaje na lokalizáciu ip adries bola použitá voľná databáza DB-IP lite [5].

Klientská časť bude implementovaná pomocou frameworku Angular 15.0. API bude volané pomocou vstavanej knižnice httpClient. Na zobrazenie presnej geografickej mapy sveta budú použité Open Street Maps pomocou knižnice Leaflet a jej wrapperu pre Angular ngx-leaflet. Na zobrazenie mapy sveta po krajinách je použitý mierne upravený verejne voľný svg template [7]. Na zobrazenie diagramov bude použitá knižnica ngx-charts.

Kapitola 3

Implementácia

Systém je implementovaný ako súbor programov bežiacich v kontaineroch. Každý program má svoju úlohu.

1. Open Street Maps tile server
2. Webový server pre frontend
3. Webový server pre backed
4. Konzolová aplikácia Seed
5. Background service IpInfoMap
6. Background service CountryPingInfo

3.1 Nástroje používané na implementáciu

Na písanie kódu sa dajú používať rôzne nástroje v závislosti od jazyka, v ktorom chceme kód písať. Na písanie backendového kódu v C# sa najčastejšie používajú nástroje Visual Studio alebo Rider. Vzhľadom na osobnú preferenciu sme sa rozhodli použiť nástroj Visual Studio 2022, ktorého Community verzia je zadarmo na stiahnutie.

Na písanie frontendového kódu v TypeScript, HTML a CSS sa najčastejšie používajú nástroje Visual Studio Code, WebStorm alebo aj rôzne iné. Vzhľadom na osobnú preferenciu sme sa rozhodoli použiť WebStorm, na ktorý máme udelenú študentskú licenciu.

3.2 Štruktúra backendového riešenia

V jazyku C# je možnosť zoskupovať viacero aplikácií do jedného riešenia. V praxi to vyzerá tak, že riešenie je znázornené ako súbor .sln a každá aplikácia alebo knižnica je

znázornená ako súbor .csproj. Aplikáciám a knižniciam sa vnútorne hovorí projekty. V našom riešení sa nachádzajú nasledovné projekty:

- IpInfoViewer.Web - projekt, v ktorom je obsiahnuté riešenie frontendovej časti aplikácie
- IpInfoViewer.Api - projekt slúžiaci ako API pre komunikáciu medzi backendom a frontendom
- IpInfoViewer.IpInfoService - projekt, ktorý slúži ako servis bežiaci na pozadí ktorý vytvorí v lokálnom dátovom sklade zoznam všetkých ip adries s ich lokalitami a krajinami
- IpInfoViewer.CountryPingInfoService - projekt, ktorý slúži ako servis bežiaci na pozadí spracovávajúci údaje v týždenných intervaloch o priemernom pingu pre danú krajinu
- IpInfoViewer.MapPointsService - projekt, ktorý slúži ako servis bežiaci na pozadí spracovávajúci údaje v týždenných intervaloch o priemernom pingu pre dané geografické okolie (zemepisnú šírku a dĺžku)
- IpInfoViewer.Libs - knižnica obsahujúca väčšinu aplikačnej logiky, tak aby bola zahrnuteľná zo všetkých ostatných projektov

3.2.1 IpInfoViewer.Api

Tento projekt je ASP.NET Core 6.0 aplikácia bežiaca na webovom serveri Kestrel. Obsahuje nasledujúce triedy:

- Program - Základná trieda každého spustiteľného programu. Jej metóda Main je vstupným bodom každého C# programu, teda po spustení programu sa automaticky začne vykonávať. Registrujú sa v nej triedy do DI kontainera (Dependency Injection), z ktorého sa potom vyberajú z iných tried vrámci behu programu. Tiež tu prebieha inicializácia potrebných nastavení programu. Od verzie .NET 6.0 je podporovaná upravená syntax tejto triedy na tzv. top-level statements. To znamená, že v súbore Program.cs je vynechaná inicializácia triedy Program a metódy Main, nakoľko bola v každom programe rovnaká. Vnútorne sú však tieto triedy stále používané, ide len o syntaktickú skratku. Používanie top-level statements je dobrovoľné, vrámci čistoty kódu ich ale využívame.
- MapPointsController - Trieda slúži ako súbor metód, ktoré sa dajú volať cez protokol HTTP z frontendovej časti aplikácie. Obsahuje metódy pre získavanie dát zoskupených podľa zemepisnej dĺžky a šírky.

Algoritmus 3.1: Ukážka kódu triedy Program bez použitia top-level statements

```
class Program
{
    public static void Main(string[] args)
    {
        System.Console.WriteLine("Hello World");
    }
}
```

Algoritmus 3.2: Ukážka kódu triedy Program s použitím top-level statements

```
System.Console.WriteLine("Hello World");
```

- CountryPingInfoController - Súbor metód volateľných cez HTTP. Obsahuje metódy pre dáta zoskupené podľa krajiny.

3.2.2 IpInfoViewer.IpInfoService

Projekt je konzolová aplikácia bežiaca ako servis na pozadí. Obsahuje nasledovné triedy:

- Program
- IpInfoServiceWorker - Treida reprezentujúca samotný servis. Trieda je zadefinovaná v triede Program ako "Hosted Service", teda pri spustení sa vykoná jej metóda ExecuteAsync. Načíta súbor s priradením medzi IP adresou a geografickými dátami a paralelne pošle každý jeho riadok na spracovanie.

3.2.3 IpInfoViewer.CountryPingInfoService

Projekt je rovnako konzolová aplikácia bežiaca ako servis na pozadí. Obsahuje nasledovné triedy:

- Program
- CountryPingInfoServiceWorker - Hosted service tohto servisu. Načita všetky známe IP adresy spracované servisom IpInfoService. Následne ich zoskupí podľa krajín prislúchajúcich daným adresám. Nakoniec ešte zistí, kedy boli dáta naposledy spracovávané a pošle na spracovanie dáta paralelne pre každý týždeň od posledného spracovania až po súčasnosť.

3.2.4 IpInfoViewer.MapPointsService

Taktiež servis na pozadí. Obsahuje nasledovné triedy:

- Program
- MapPointsServiceWorker - Hosted service tohto servisu. Funguje rovnako ako CountryPingInfoServiceWorker, ale adresy zoskupuje podľa geografických súradníc. Adresy sú rozdelené do stabilných sektorov na mape. Mapa je rozdelená do 23x60 sektorov rovnomerne podľa geografickej šírky a dĺžky.

3.2.5 IpInfoViewer.Libs

Knižnica zoskupujúca logiku všetkých C# programov. Ostatné programy ju využívajú ako referenciu. Je tomu tak preto, lebo programy navzájom zdieľajú časť logiky a bolo by nesprávne túto logiku písať do každého programu zvlášť. Tiež sme sa chceli vyhnúť referencovaniu ostatných spustiteľných programov navzájom priamo, pretože sa to ukázalo ako nepraktické. Dôvodov je niekoľko, najvážnejší s ktorým sme sa stretli je konflikt mien konfiguračných súborov. Pri C# projektoch je zaužívané používať konfiguračné súbory s menami appsettings.json a appsettings.environmentName.json. Tiež je pravidlo, že keď referencujeme iný projekt, do priečinka s výstupom kompilácie idú všetky súbory rovnako, ako keď kompilujeme projekt samotný. Keďže aj pôvodný projekt aj referencovaný projekt obshujú súbory appsettings.json, nastane konflikt a do výstupného priečinka sa nakopíruje len jeden z nich, a podľa skúseností to bol vždy ten z referencovaného projektu, čo je ten nesprávny. Tomuto sa dá vyhnúť tým, že spustiteľné projekty sa navzájom nereferencujú a namiesto toho všetky referencujú jednu knižnicu. Knižnica nemá takéto konfiguračné súbory, preto ku konfliktu nedochádza. Knižnica IpInfoViewer.Libs obsahuje nasledovné triedy:

- BaseModel - Abstraktná trieda z ktorej dedia všetky modely, teda triedy predstavujúce dáta. Obsahuje všetky parametre ktoré by mali zdieľať všetky modely v riešení. V súčasnosti je jediným takýmto parametrom Id, teda primárny kľúč v databázovej tabuľke, v našom riešení znázornené ako 64-bitové celé číslo.
- BaseWeeklyProcessedModel - Abstraktná trieda, ktorá je rozšírením triedy BaseModel o vlastnosti ValidFrom a ValidTo reprezentované časovou pečiatkou. Tieto vlastnosti sa opakovali vo všetkých časovo spracovávaných triedach.
- BaseMapModel - Abstraktná trieda ktorá je rozšírením BaseMapModel o vlastnosti IpAddressesCount a AveragePingRtT. Ide o pre nás zaujímavé vlastnosti ktoré sa vyskytovali vo viacerých modeloch. Prvá znázorňuje počet IP adries ktoré daný záznam znázorňuje, druhá ich priemerný ping v milisekundách.
- CountryPingInfo - Model predstavujúci dáta pre konkétnu krajinu v konkrétny týždeň. Rozširuje BaseMapModel o vlastnosť CountryCode, ktorá predstavuje ISO skratku konkrétnej krajiny.

- IpAddressInfo - Model predstavujúci jednu IP adresu s geografickými informáciami ako mesto, krajina a geografické súradnice.

- MapPointsController - Model predstavujúci dáta pre konkrétny geografický sektor v konkrétnom čase. Rozširuje BaseMapModel o geografickú šírku a dĺžku.

- String response - Model prestavujúci HTTP odpoveď pozostávajúcu z jediného reťazca. Takýto model je potrebný, pretože po správnosti by sa nemal vracať samotný reťazec ako telo odpovede, ale mal by byť zabalený do rodičovského objektu, pretože niektoré kižnice takéto odpovede nepodporujú.

- Week - Keďže dáta spracovávame po týždňoch, potrebovali sme si vedieť takýto týždeň znázorniť. Ako jeden týždeň sme si určili obdobie začínajúce v pondelok o polnoci a končiace v nedeľu toho istého týždňa tesne pred polnocou ako je definované v štandarde ISO 8601. Tento štandard určuje spôsob, ako vyjadriť konkrétny týždeň ako textový reťazec, a to vo formáte YYYY-'W'WW, teda napríklad deviaty týždeň roku 2023 by bol vyjadrený ako 2023-W09. Tento formát zápisu týždňa je podporovaný ako formát výstupu pre HTML input typu "week", preto je pre našu aplikáciu veľmi užitočný. V jazyku C# však neexistuje žiadna vstavaná trieda reprezentujúca takýto týždeň, jednu sme napísali podľa našich potrieb. Trieda podporuje dva rôzne konštruktory, teda dva rôzne spôsoby ako definovať týždeň. Prvý spôsob je podľa už zmieneného ISO 8601 formátu. Druhý spôsob si vezme na vstup ľubovoľný čas reprezentovaný štandardnou triedou DateTime a určí týždeň, do ktorého daný čas patrí. Na základe definície ISO 8601 každý čas patrí do práve jedného dňa a každý deň patrí do práve jedného týždňa.

- DateTimeUtilities - Trieda zoskupujúca pomocné metódy na prácu s časom. Momentálne má len jednu metódu GetWeeksFromTo, ktorá prijme dva dátumy a vráti zoznam týždňov medzi nimi.

- Host - Model predstavujúci jednu IP adresu v zdrojovej databáze.

- Ping - Model predstavujúci jedno volanie príkazu ping na konkrétnu IP adresu v zdrojovej databáze.

- IIpInfoViewerDbRepository - Rozhranie definujúce metódy, ktorými sa dajú vyťahovať dáta z lokálneho dátového skladu.

- IpInfoViewerDbRepository - Trieda slúžiaca na vyťahovanie dát z lokálneho dátového skladu. Implementuje rozhranie IIpInfoViewerDbRepository.

- IMFileDbRepository - Rozhranie definujúce metódy určené na vyťahovanie dát zo zdrojovej databázy.
- MFileDbRepository - Trieda slúžiaca na výber dát zo zdrojovej databázy. Implementuje rozhranie IMFileDbRepository.
- ICountryPingInfoFacade - Rozhranie definujúce metódy obsahujúce logické manipulácie s dátami ohľadom IP adries zoskupených podľa krajín.
- CountryPingInfoFacade - Trieda obshujúca logickú manipuláciu s dátami ohľadom IP adries zoskupených podľa krajín. Implementuje rozhranie ICountryPingInfoFacade.
- IIpAddressInfoFacade - Rozhranie definujúce metódy využívané pri priraďovaní geografických informácií k databázam.
- IpAddressInfoFacade - Implementácia priraďovania IP adries ku geografickým dátam ako krajiny a súradnice. Implementuje rozhranie IIpAddressInfoFacade.
- IMapFacade - Rozhranie definujúce metódy obsahujúce logické manipulácie s dátami ohľadom IP adries zoskupených podľa geografických súradníc.
- MapFacade - Implementácia manipulácie s dátami zoskupenými podľa geografických súradníc. Implementuje rozhranie IMapFacade.

3.3 Štruktúra frontendovej časti aplikácie

Kapitola 4

Ukážky užitočných príkazov v systéme LaTeX

V tejto kapitole si ukážeme príklady niektorých užitočných príkazov, ako napríklad správne používanie tabuliek a obrázkov, číslovanie matematických výrazov a podobne. Konkrétne príkazy použité v tejto kapitole nájdete v zdrojovom súbore `latex.tex`. Všimnite si, že pre potreby obsahu a hlavičky stránky je v zdrojovom súbore uvedený aj skrátený názov tejto kapitoly. Ďalšie užitočné príkazy nájdete aj v kapitole **??**, na ktorú sme sa na tomto mieste odvolali príkazom `\ref`.

4.1 Obrázky

Vašu prácu ilustrujte vhodnými obrázkami. Pri použití programu pdflatex je potrebné pripraviť obrázky vo formáte pdf, jpg alebo png. Vektorové obrázky (napr. eps, svg) je najvhodnejšie skonvertovať do formátu pdf, napríklad programom Inkscape.

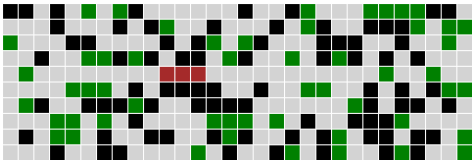
Na vkladanie obrázkov použite prostredie `figure`, ktoré obrázok umiestni na vhodné miesto, väčšinou na vrch alebo spodok stránky a tiež sa stará o automatické číslovanie obrázkov. Na každý obrázok sa treba v hlavnom texte odvolať. Napríklad ilustráciu hry Červík vidíme na obrázku 4.1. Pri odvolávaní sa na číslo obrázku používame príkaz `\ref`. Pri vložení alebo zmazaní obrázku tak nemusíme ručne všetky ostatné obrázky prečíslovať.

Podobne tabuľky vkladajte pomocou prostredia `table`, pričom samotnú tabuľku vytvoríte príkazom `tabular`. Každú tabuľku potom spomeňte aj v hlavnom texte. Napríklad v tabuľke 4.1 vidíme porovnanie časov niekoľkých fiktívnych programov.

V texte môžete tiež potrebovať dlhšie matematické výrazy, ako napríklad tento

$$\sum_{k=0}^{n} q^k = \frac{q^{n+1}-1}{q-1}. \tag{4.1}$$

Použitím prostredia `equation` bol tento výraz zarovnaný na stred na zvláštnom riadku



Obr. 4.1: Ukážka hry Červík. Červík je znázornený červenou farbou, voľné políčka sivou, jedlo zelenou a steny čiernou. Hoci tento popis obrázku je dlhší, v zdrojovom texte je aj kratšia verzia, ktorá sa zobrazí v zozname obrázkov.

Tabuľka 4.1: Doba výpočtu a operačná pamäť potrebná na spracovanie vstupu XYZ. V tomto popise môžeme vysvetliť detaily potrebné pre pochopenie údajov v tabuľke.

Meno programu	Čas (s)	Pamäť (MB)
Môj super program	25.6	120
Speedy 3.1	32.1	100
VeryOld	244.1	200

a očíslovaný. Na toto číslo sa tiež môžeme odvolať príkazom `\ref`. Napríklad rovnica (4.1) predstavuje súčet geometrickej postupnosti.

V práci tiež možno budete uvádzať úryvky kódu v niektorom programovacom jazyku. Môže vám pomôcť prostredie `lstlisting` z balíčka `listings`, v ktorom môžete nastaviť aj jazyk a kód bude krajšie sformátovaný. Ukážku nájdete ako Algoritmus 4.1.

Napokon, v texte nezabudnite citovať použitú literatúru pomocou príkazu `\cite` Napríklad ďalšie detaily o systéme LaTeX nájdete v knihe od Tobiasa Oetikera a kolektívu [6]. Pre ukážku citujeme aj článok z vedeckého časopisu [3] a článok z konferencie [2], technickú správu [4], knihu [1] a materiál z internetu [8].

Algoritmus 4.1: Algoritmus na výpočet faktoriálu v jazyku C

```
int factorial = 1;
for(int i = 1; i <= n; i++) {
    factorial *= i;
}
```

Záver

Na záver už len odporúčania k samotnej kapitole Záver v bakalárskej práci podľa smernice [8]: „V závere je potrebné v stručnosti zhrnúť dosiahnuté výsledky vo vzťahu k stanoveným cieľom. Rozsah záveru je minimálne dve strany. Záver ako kapitola sa nečísluje.“

Všimnite si správne písanie slovenských úvodzoviek okolo predchádzajúceho citátu, ktoré sme dosiahli príkazom `\uv`.

V informatických prácach niekedy býva záver kratší ako dve strany, ale stále by to mal byť rozumne dlhý text, v rozsahu aspoň jednej strany. Okrem dosiahnutých cieľov sa zvyknú rozoberať aj otvorené problémy a námety na ďalšiu prácu v oblasti.

Abstrakt, úvod a záver práce obsahujú podobné informácie. Abstrakt je kratší text, ktorý má pomôcť čitateľovi sa rozhodnúť, či vôbec prácu chce čítať. Úvod má umožniť zorientovať sa v práci skôr než ju začne čítať a záver sumarizuje najdôležitejšie veci po tom, ako prácu prečítal, môže sa teda viac zamerať na detaily a využívať pojmy zavedené v práci.

Literatúra

[1] X. Autor1 and Y. Autor2. *Názov knihy.* Vydavateľstvo, 1900.

[2] X. Autor1 and Y. Autor2. Názov článku (väčšinou z konferencie). In *Názov zborníka (väčšinou názov konferencie spolu s ročníkom)*, pages 1–100. Vydavateľstvo, 1900.

[3] X. Autor1 and Y. Autor2. Názov článku z časopisu. *Názov časopisu, ktorý článok uverejnil*, 4(3):1–100, 1900.

[4] X. Autor1 and Y. Autor2. Názov technickej správy. Technical Report TR123/1999, Inštitút vydávajúci správu, June 1999.

[5] DB-IP, 2022. Dostupné z `https://db-ip.com/db/download/ip-to-city-lite`.

[6] Tobias Oetiker, Hubert Partl, Irene Hyna, and Elisabeth Schlegl. *Nie príliš stručný úvod do systému LaTeX2e.* 2002. Preklad Ján Buša ml. a st.

[7] simplemaps.com, 2017. Dostupné z `https://simplemaps.com/resources/svg-world`.

[8] Univerzita Komenského v Bratislave. Vnútorný predpis č. 7/2018, Úplné znenie vnútorného predpisu č. 12/2013 Smernice rektora Univerzity Komenského v Bratislave o základných náležitostiach záverečných prác, rigoróznych prác a habilitačných prác, kontrole ich originality, uchovávaní a sprístupňovaní na Univerzite Komenského v Bratislave v znení dodatku č. 1 a dodatku č. 2 smernica rektora Univerzity Komenského v Bratislave o základných náležitostiach záverečných prác, rigoróznych prác a habilitačných prác, kontrole ich originality, uchovávaní a sprístupňovaní na Univerzite Komenského v Bratislave, 2013. [Citované 2020-10-19] Dostupné z `https://uniba.sk/fileadmin/ruk/legislativa/2018/Vp_2018_07.pdf`.

Príloha A: obsah elektronickej prílohy

V elektronickej prílohe priloženej k práci sa nachádza zdrojový kód programu a súbory s výsledkami experimentov. Zdrojový kód je zverejnený aj na stránke `http://mojadresa.com/`.

Ak uznáte za vhodné, môžete tu aj podrobnejšie rozpísať obsah tejto prílohy, prípadne poskytnúť návod na inštaláciu programu. Alternatívou je tieto informácie zahrnúť do samotnej prílohy, alebo ich uviesť na obidvoch miestach.

Príloha B: Používateľská príručka

V tejto prílohe uvádzame používateľskú príručku k nášmu softvéru. Tu by ďalej pokračoval text príručky. V práci nie je potrebné uvádzať používateľskú príručku, pokiaľ je používanie softvéru intuitívne alebo ak výsledkom práce nie je ucelený softvér určený pre používateľov.

V prílohách môžete uviesť aj ďalšie materiály, ktoré by mohli pôsobiť rušivo v hlavnom texte, ako napríklad rozsiahle tabuľky a podobne. Materiály, ktoré sú príliš dlhé na ich tlač, odovzdajte len v electronickej prílohe.