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

Táto bakalárska práca sa zaoberá spracovaním dlhodobých meraní vybraných charakteristík internetu. Cieľom práce je navrhnúť a implementovať systém s webovým rozhraním pre prácu s nameranými dátami. Systém by mal umožniť používateľovi výber, triedenie a filtrovanie dát podľa adries a času, ako aj výpočet štatistík a vizualizáciu ich hodnôt, časových priebehov a geografického rozloženia. Práca sa zaoberá možnosťami implementácie softvéru pre rôzne platformy a programovcie jazyky. Zaoberá sa tiež výberom databázového poskytovateľa a možnosťami vývoja frontendového a backendového riešenia. Výsledkom práce je webová aplikácia, ktorá umožňuje vizualizovať výsledky meraní internetu v grafickej forme pomocou máp. Tento systém by mal pomôcť používateľom lepšie pochopiť dlhodobé trendy a charakteristiky výtvoja rýchlosti a dostupnonsti internetu.

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

Úvod

Cieľom tejto práce je vyvoriť aplikáciu, ktorá umožní používateľovi prezerať si výsledky dlhodobých meraní internetu. Zároveň je cieľom popísať toto riešenie a proces jeho vývoja.

Na začiatku sa budeme zaoberať analýzou meraní s ktorými budeme pracovať. Tiež sa pozrieme na súvisiace projekty a existujúce riešenia.

Pokračovať budeme analýzou požiadaviek na aplikáciu a návrhom riešenia. Navrhneme používateľské scenáre, návrh rozhraní a návrh databázy.

V kapitole 3 sa budeme zaoberať možnosťami implementácie softvéru pre rôzne platformy a výberom programovacieho jazyka. Tiež sa budeme zaoberať výberom databázového poskytovateľa a možnosťami vývoja frontendového a backendového riešenia.

Nasledovať bude najdôležitejšia kapitola 4, ktorá sa zoberá samotnou implementáciou, jej štuktúrou a použitými technikami. Kapitola obsahuje podrobný opis backendového aj frontendového riešenia a opis procesu vývoja. V ďalšej kapitole si prejdeme testovaním aplikácie ručne aj automatizovanými testami.

Poslednou bude kapitola 6, ktorá vysvetľuje, ako sa aplikácia používa.

Kapitola 1

Uvedenie do problematiky a súčasný stav

1.1 Spôsob merania

Dáta sú výsledkom dlhodobého merania. Vrámci merania sa v pravidelných intervaloch opakovane volali príkazy `ping` a `tracert` na postupne rozširujúcu sa množinu IP adries. Výsledky meraní boli zapisované do PostgreSQL 8.1.9 databázy. Databáza má nasledujúcu štruktúru:

Tabuľka `h_types` - Tabuľka obshujúca zoznam všetkých poznaných typov serverov na ktorých sa vykonávajú merania:

- rank_code - celé číslo, primárny kľúč
- rank_description - reťazec - popis typu servera
- rank - reťazec - meno typu servera

Tabuľka `hosts` - Tabuľka obshujúca zoznam všetkých IP adries na ktorých sa vykonávajú merania:

- ip_addr - IPV4 adresa vo formáte cidr, primárny kľúč - konkrétna hodnota IP adresy na ktorú sa riadok vzťahuje
- rank_code - celé čislo, cudzí kľúč na tabuľku `h_types` - typ servera na IP adrese
- enter_date - časová pečiatka - dátum vloženia do zoznamu
- source - reťazec - zdroj vloženia
- comment - reťazec - komentár k IP adrese
- exclude - celé čislo - príznak, že daná adresa má byt vynechaná z merania

Tabuľka `ping` - Tabuľka obshujúca záznamy o vykonaných meraniach dostupnosti a doby odozvy:

- ip_addr - IPV4 adresa vo formáte cidr, cudzí kľúč do tabuľky `hosts` - konkrétna hodnota IP adresy na ktorú sa riadok vzťahuje
- ping_rttmin - reálne číslo - minimálna nameraná doba odozvy v danom meraní
- ping_rttmax - reálne číslo - maximálna nameraná doba odozvy v danom meraní
- ping_rttavg - reálne číslo - priemerná nameraná doba odozvy v danom meraní
- ping_rttmdev - reálne číslo - priemerná odchýlka doby odozvy v danom meraní
- ping_ploss - celé číslo - počet percent stratených packetov alebo error kôd v prípade negatívnej hodnoty
- ping_date - časová pečiatka - dátum a čas merania

Dátami v ďalších tabuľkách sa v tejto práci nebudeme zaoberať, ich štruktúru je možné vidieť v entitno-relačnom grafe 1.1. Dátový typ cidr je PostgreSQL dátový typ určený na ukladanie IP adries spolu s maskou podsiete. Pri ukladaní kontroluje, či sú za maskou len nulové bity, čím sa odlišuje od typu inet. Podporuje zoraďovanie podľa hodnoty a konverziu z reťazcovej reprezentácie [9].

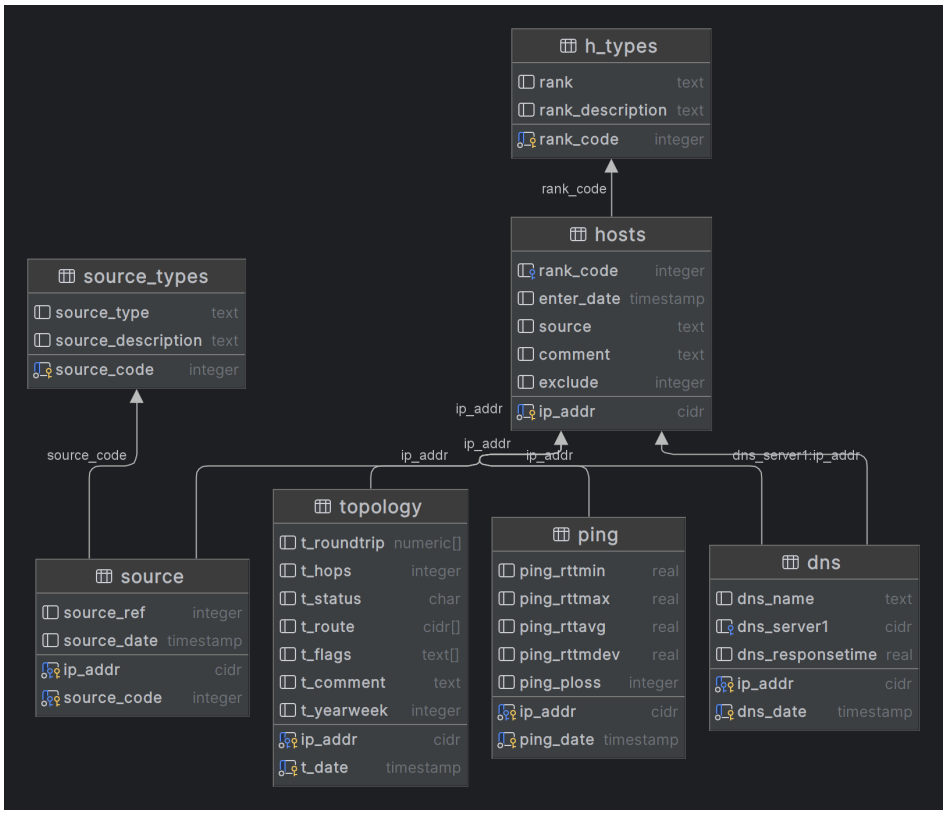
Príkaz `ping` zmeria dostupnosť a dobu odozvy servera. Príkaz pošle nastaviteľný počet paketov (predvolene 4) a sleduje, ktoré sa vrátia a za aký čas. Príkaz funguje v systéme Windows [7] aj na Linuxových systémoch [12]. Príkaz `tracert` podobne ako `ping` sleduje cestu paketov skrz uzly siete. Jeho výstupom je zoznam IP adries uzlov, ktorými paket prešiel na ceste k serveru s cieľovou IP adresou [16].

1.2 Prehľad súvisiacich projektov a existujúcich riešení

1.2.1 Projekty zaoberajúce sa dlhodobými meraniami internetu

Analýzou databázy s existujúcimi dátami sa zaoberal Dennis Vita v ročníkovom projekte. Analyzoval kvalitu meraní a určoval množinu tzv. „spoľahlivých“ adries, teda adries, ktoré často odpovedajú [19].

Dlhodobým meraním podobného charakteru a jeho vizualizácií sa venuje projekt CAIDA (Center for Applied Internet Data Analysis) na Univerzite v Kalifornii v San Diegu. V článku [3] sa zaoberajú dlhodobým meraním a vizualizáciou topológie siete.



Obr. 1.1: Entitno-relačný diagram databázy s výsledkami meraní

Podobnému meraniu a vizualizácii, ako napríklad zobrazeniu nameranej doby odozvy v mapách sa tiež venuje projekt RIPE Atlas [1].

1.2.2 Riešenia zaoberajúce sa predspracovávaním a agregovaním údajov

Dátový sklad (anglicky data warehouse) je systém, ktorý v pravidelných intervaloch zhromažďuje údaje z rôznych zdrojov, spracováva ich a vytvára medzi nimi vzťahy a tým pôsobí ako prehľadnejší zdroj užitočnejších dát ako pôvodné dáta [11]. Takéto riešenie je vhodné na analýzu veľkého množstva dát, akú budeme robiť my. Existuje niekoľko implemntácií dátových skladov, ako napríklad Snowflake, Google BigQuery alebo Amazon Redshift. My sa pokúsime o implementáciu vlastného jednoduchého dátového skladu určeného priamo namieru pre náš systém.

Kapitola 2

Požiadavky a návrh riešenia

2.1 Analýza požiadavkov

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 byť schopné 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 služieb iných, ako databáza s meraniami (lokalizačné servisy aj mapové servery musia bežať vrámci systému)
- Spracovanie dát pre jeden týždeň nesmie trvať viac ako jeden týždeň (nutné preto, aby spracovanie niekedy dobehlo)

Požiadavky na beh klientskej časti:

- Aplikácia musí fungovať na všetkých moderných prehliadačoch
- Aplikácia musí vedieť vizualizácie zobrazovať bez pridlhého načítavania

Detaily implementácie boli ponechané na moje uváženie. Detaily rozhodovania sú popísané v kapitole 3.

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 získavanie spracovaných dát a zvyšné budú služby bežiace na pozadí, ktoré budú mať za úlohu priebežne spracovávať namerané dáta. Predspracované dáta sa budú ukladať do PostgreSQL 15.0 databázy. Tieto služby budú spúšťané plánovačom úloh 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 bude použitá voľná CSV databáza DB-IP lite [5].

Klientská časť bude implementovaná pomocou systému 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 [18].

2.2 Používateľské scenáre

Keďže služby bežiace na pozadí nebudú mať žadneho používateľa a budú fungovať samy bez vstupu od používateľa, je potrebné navrhnúť len scenáre pre frontendovú aplikáciu.

2.2.1 Používateľské scenáre pre frontendovú aplikáciu

Aplikácia bude mať len jeden typ používateľa. Keďže celá webová časť je len na čítanie, nie je pootrebné obmedzovať práva pre niektorých používateľov. Z tohto dôvodu sme sa rozhodli vynechať autentifikáciu a aplikáciu sprístupniť bez mena a hesla.

Po spustení aplikácie bude vidieť domovskú obrazovku. Tá bude pozostávať z navigačného menu, mapy sveta s vyznačenými skupinami uzlov IP adries na základe ich geografickej polohy. Menu bude obsahovať dve položky, domovská obrazovka a mapa krajín. Na stránke bude prepínač týždňov, ktorým sa bude ovládať voľba týždňa, pre ktorý chceme, aby sa dáta zobrazili. Tiež sa na stránke bude nachádzať výberový zoznam na výber medzi prezeraním maximálnych, priemerných alebo minimálnych dôb odozvy. Po zmene stavu prepínačov sa zmeny v dátach prejavia okamžite pri všetkých troch selektoroch. V mape sa tiež bude nachádzať legenda podľa ktorej sa bude dať určiť hodnota odozvy podľa farby kruhu znázorňujúceho uzol a počet IP adries reprezentovaných kruhom podľa veľkosti priemeru kruhu. Pri maximálnom priblížení sa pri

kruhoch zobrazia detailné informácie o údajoch nameraných v daný týždeň.

Po kliknutí na mapu krajín v menu sa zobrazí druhá stránka. Tá bude obshovať rovnaké menu ako domovská stránka. V hlavnej časti bude mapa krajín sveta. Krajiny budú zafarbené podľa doby odozvy pre danú krajinu. Nad mapou sa bude nachádzať prepínač týždňov a rovnaké výberové zoznamy ako na domovskej stránke. V mape sa tiež bude nachádzať legenda na priradenie doby odozvy k farbe.

2.2.2 Návrh rozhrania na komunikáciu so serverovou časťou

Rozhranie by malo fungovať podľa architektonického štýlu „REST“. To určuje niekoľko pravidiel, medzi nimi napríklad oddelenosť backendu od frontendu, komunikáciu pomocou protokolu HTTP alebo nezávislosť dopytov jeden od druhého [10]. Rozhranie by malo obsahovať tieto metódy:

- `GET /CountryPingInfo/ForWeek/{week}` (parameter week - ISO 8601 reprezentácia týždňa) - Metóda, ktorá vráti dáta o dobe odozvy pre všetky odmerané krajiny pre daný týždeň.

Algoritmus 2.1: Vzorový výstup z endpointu

```
[
    {
        "id": 0,
        "week": "2022-W05",
        "ipAddressesCount": 0,
        "averagePingRtT": 0,
        "maximumPingRtT": 0,
        "minimumPingRtT": 0,
        "countryCode": "SK"
    }
]
```

- `GET /CountryPingInfo/ColoredMap/{week}` (parameter week - ISO 8601 reprezentácia týždňa) - Metóda ktorá vráti SVG obrázok s mapou sveta s krajinami zafarbenými podľa doby odozvy pre daný týždeň.

Algoritmus 2.2: Vzorový výstup z endpointu

```
<?xml version="1.0" encoding="utf-8"?>
<svg width="2000" height="857" fill="#ECECEC" stroke="
    black" stroke-width=".2"...
    <path id="AF" class=" AF" d="M1383,261.6l1.5,1.8
        -2.9,.8 -2.4,1.1 -5.9,.8 -5.3,1.3 ...
    <path class="Angola AO " d="M1121.2,572 1121.8,574
        1121.1,577.1 1122,580.1 1121.1, ...
```

```
5            <path class="Angola AO " d="M1055.3,539
                 1053.8,534.2 1056.1,531.4 1057.8,530.3 ...
6            ...
```

- `GET /CountryPingInfo/LastProcessedDate` - Metóda, ktorá vráti posledný týždeň, pre ktorý sú spracované dáta o dobe odozvy pre krajiny.

Algoritmus 2.3: Vzorový výstup z endpointu

```
1    {
2        "response": "string"
3    }
```

- `GET /MapPoints/ForWeek/{week}` (parameter week - ISO 8601 reprezentácia týždňa) - Metóda, ktorá vráti dáta o dobe odozvy pre všetky odmerané body na mape podľa polohy pre daný týždeň.

Algoritmus 2.4: Vzorový výstup z endpointu

```
1    [
2        {
3            "id": 0,
4            "week": "2022-W05",
5            "ipAddressesCount": 0,
6            "averagePingRtT": 0,
7            "maximumPingRtT": 0,
8            "minimumPingRtT": 0,
9            "latitude": 0,
10           "longitude": 0
11       }
12   ]
```

- `GET /MapPoints/LastProcessedDate` - Metóda, ktorá vráti posledný týždeň, pre ktorý sú spracované dáta o dobe odozvy pre geografické lokality. Vzorový výstup je zhodný s ukážkou 2.3.
- `GET /MapPoints/MapPointsMapLegend` - Metóda, ktorá vráti svg obrázok legendy vykreslený podľa zadaných parametrov. Parametre by mali obsahovať definíciu piatich dvojíc veľkosť-počet, ktoré budú znázorňovať veľkosť kruhu a počet adries znázorňujúci túto veľkosť.

Algoritmus 2.5: Vzorový výstup z endpointu

```
1    <?xml version="1.0" encoding="utf-8"?>
2    <svg viewBox="0 0 500 300" width="500" height="300"
         xmlns="http://www.w3.org/2000/svg">
```

```
            <rect x="1.622" y="5.63" ...
            <ellipse ...
            ...
            <text ...
            ...
            <rect ...
            <ellipse ...
            ...
            <text ...
        </svg>
```

2.2.3 Návrh lokálnej databázy

Databázový pokytovateľ bude PostgreSQL vo verzií 15. Databáza bude bežať vo vyhradenom kontajneri. Databáza bude pozostávať z troch tabuliek.

Tabuľka `ipadresses` - Pomocná tabuľka pri napĺňaní zvyšných tabuliek. Bude obsahovať zoznam známych IP adries s informáciou o ich geografickej lokalite. Bude naplnená ako prvá, skôr ako zvyšné 2 tabuľky. Bude pozostávať z nasledovných stĺpcov:

- id - celé číslo - primárny kľúč
- ipvalue - IPV4 adresa vo formáte cidr - konkrétna hodnota IP adresy na ktorú sa riadok vzťahuje
- countrycode - reťazec o dĺžke maximálne 2 znakov - ISO 3166-1 Alpha 2 kód krajiny, v ktorej sa nachádza IP adresa
- city - reťazec o dĺžke maximálne 80 znakov - názov mesta, v ktorom sa nachádza IP adresami
- latitude - reálne číslo - zemepisná šírka IP adresy
- longitude - reálne číslo - zemepisná dĺžka IP adresy

Tabuľka `countrypinginfo` bude obsahovať informácie o nameraných dobách odozvy zoskupených podľa krajín. Bude pozostávať z nasledovných stĺpcov:

- id - celé číslo - primárny kľúč
- countrycode - reťazec o dĺžke maximálne 2 znakov - ISO 3166-1 Alpha 2 kód krajiny, o ktorej riadok hovorí
- week - reťazec o dĺžke maximálne 8 znakov - ISO 8601 reprezentácia týždňa, pre ktorý je riadok platný

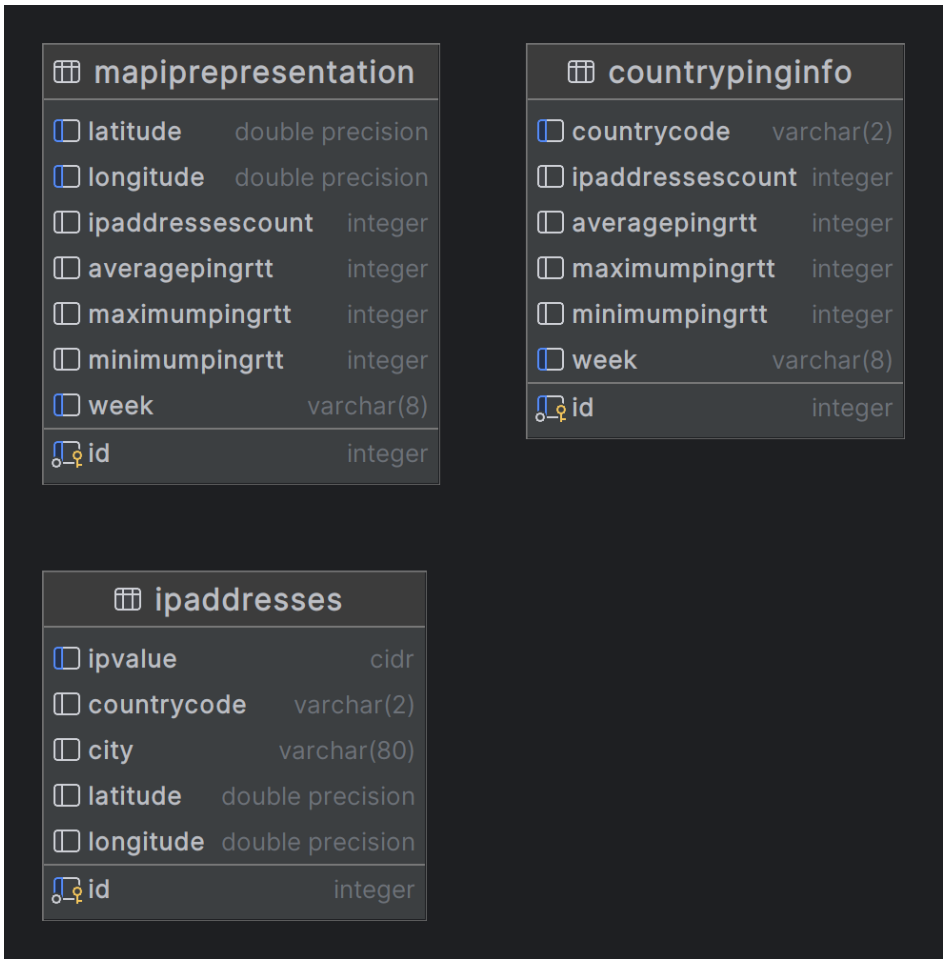
- averagepingrtt - celé číslo - zaokrúhlena priemerná hodnota odozvy pre danú krajinu v danom týždni
- maximumpingrtt - celé číslo - maximálna doba odozvy pre danú krajinu v danom týždni
- minimumpingrtt - celé číslo - minimálna doba odozvy pre danú krajinu v danom týždni

Tabuľka `mapiprepresentation` bude obsahovať informácie o nameraných dobách odozvy zoskupených podľa geografickej lokality. Bude pozostávať z nasledovných stĺpcov:

- id - celé číslo - primárny kľúč
- latitude - reálne číslo - priemerná zemepisná šírka všetkých IP adries zoskupených v danom riadku v danom týždni
- longitude - reálne číslo - priemerná zemepisná dĺžka všetkých IP adries zoskupených v danom riadku v danom týždni
- week - reťazec o dĺžke maximálne 8 znakov - ISO 8601 reprezentácia týždňa, pre ktorý je riadok platný
- averagepingrtt - celé číslo - zaokrúhlena priemerná hodnota odozvy pre danú krajinu v danom týždni
- maximumpingrtt - celé číslo - maximálna doba odozvy pre danú krajinu v danom týždni
- minimumpingrtt - celé číslo - minimálna doba odozvy pre danú krajinu v danom týždni

Tabuľky medzi sebou nebudú nijako prepojené, pretože by to neprinieslo žiadnu výhodu vzhľadom na charakter meraných dát. Entitno-relačný diagram je viditeľný na obrázku 2.1.

Databáza bude obsahovať aj indexy. Indexy sú štruktúra, ktorá umnožňuje zrýchliť vyhľadávanie v databáze na úkor spomalenia zápisu [2]. Čas nás v násom systéme zaujíma hlavne z hľadiska používania webovej aplikácie. Tá z databázy výlučne číta. Zápis budeme naopak robiť len raz do týždňa a na čase pri ňom príliš nezáleží (jediné obmedzenie je, že spracovanie týždňa musí trvať menej ako týždeň, čo je veľmi štedrý čas). Vzhľadom na to je takáto optimalizácia v našom systéme veľmi vítaná. Keďže systém môže bežať dlhý čas, spracované údaje sa môžu stať priveľké a vyhľadávanie v nich bez indexácie by bolo veľmi pomalé. Zároveň jednoznačné indexy vedia zaručiť



Obr. 2.1: Entitno-relačný digram k lokálnej databáze. Na obrázku je vidno, že tabuľky nie sú nijako previazané

.

väčšiu konzistentnosť dát, keďže vedia zaručiť, že kombinácie niektorých údajov sa môžu v tabuľke vyskytnúť maximálne raz [15].

Okrem primárneho kľúča v každej tabuľke ich bude 5. Pôjde o index zaručujúci jedinečnosť IP adresy v tabuľke `ipadresses`, index zaručujúci jedinečnosť kombinácie zemepisnej šírky, dĺžky a týždňa v tabuľke `mapiprepresentation`, index na optimalizáciu vyhľadávania podľa týždňa v tabuľke `mapiprepresentation`, index zaručujúci jedinečnosť kombinácie krajiny a týždňa v tabuľke `countrypinginfo` a index na optimalizáciu rýchlosti vyhľadávania podľa týždňa v tabuľke `countrypinginfo`.

Kapitola 3

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 prostredia 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.

3.1 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 prostredie 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 prostredia, 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 prostredia 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 prostredia sú podporované vo všetkých nových verziách moderných prehliadačov. Keďže všetky tieto prostredia 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.

3.2 Možnosti vývoja backendového riešenia

Pre vývoj backendového riešenia je vhodné vybrať taký programovací jazyk, pre ktorý existuje prostredie 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 prostredie Spring Boot, jazyk C# a prostredie 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.

3.2.1 Možnosti implementácie vo vybranom jazyku

Najprv sa musíme rozhodnúť, ktorú verziu prostredia .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 novembru 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 3.1: 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 knižnici 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 kontrolerov alebo nie (tzv. minimal API). Kým výhodou kontrolerov je lepšia štrukturovateľnosť a väčšia prehľadnosť pri väčšom množstve endpointov, výhodou mimal API je mierne

Algoritmus 3.2: 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 3.3: 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 kontrolery.

3.2.2 Databáza

S backendovým riešením úzko súvisí aj výber databázového poskytovateľa 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 4

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

4.1 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.

4.2 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

4.2.1 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.

Algoritmus 4.1: Ukážka kódu triedy Program bez použitia top-level statements

```
1          class Program
2          {
```

```
3             public static void Main(string[] args)
4             {
5                 System.Console.WriteLine("Hello World");
6             }
7         }
```

Algoritmus 4.2: Ukážka kódu triedy Program s použitím top-level statements

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

- 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.

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

4.2.2 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.

4.2.3 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.

4.2.4 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.

4.2.5 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.

- GeographicUtilities - Trieda zoskupujúca pomocné metódy na prácu s geografickými dátami. Momentálne obsahuje jeden konštantný slovník CountryCodeToNameDictionary, ktorý slúži na prevod ISO 3166-1 Alpha 2 kódu na ISO názov krajiny.

- 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.
- IMapPointsFacade - Rozhranie definujúce metódy obsahujúce logické manipulácie s dátami ohľadom IP adries zoskupených podľa geografických súradníc.
- MapPointsFacade - Implementácia manipulácie s dátami zoskupenými podľa geografických súradníc. Implementuje rozhranie IMapFacade.

4.3 Štruktúra frontendovej časti aplikácie

Frontendová časť aplikácie spočíva z jedinej aplikácie písanej pomocou frameworku Angular 15. Projekt bol vytvorený pomocou templatu vo Visual Studio 2022, vďaka čomu bolo možné ho spúšťať s Visual Studia a bol zahrnutý do jedného .sln súboru, čo nám zjednodušilo manipuláciu. Z používateľského hľadiska aj z hľadiska písania kódu je to ale ekvivalentné vygenerovaniu projektu pomocou príkazu `ng new`.

Jednou z hlavných výhod využitia frameworku ako napríklad Angular je, že aplikáciu je možné deliť na komponenty. Jeden komponent je možné si predstaviť ako

istú podčasť zobrazenej stránky. Každý komponent sa môže skladať s viacerých iných komponentov. Naša aplikácia obsahuje nasledovné komponenty:

- app.component - Základný komponent pre každú aplikáciu. Všetky ostatné komponenty sú jeho podkomponenty. Reprezentuje celé okno aplikácie. Framework Angular je defaultne nastavený aby vždy zobrazoval práve tento komponent.
- nav-menu.component - Komponent reprezentujúci vrchný panel s menu aplikácie.
- ip-address-map.component - Komponent reprezentujúci mapu s vyznačenými ip adresami zoskupenými podľa geografických súradníc.
- country-ping-map.component - Komponent reprezentujúci mapu krajín zafarbených podľa pingu.

4.4 Dôležité algoritmy a postupy riešenia

Pri implementácií aplikácie sme využívali niekoľko špecifických postupov a algoritmov, ktoré považujeme za vhodné vysvetliť a zdôvodniť.

4.4.1 CORS

„Cross-Origin Resource Sharing“ je mechanizmus na zdieľanie obsahu stránok z rôznych domén pracujúci na základe HTTP hlavičiek. Tento mechanizmus je nutné použiť na zdieľanie dát medzi frontendovou a backendovou časťou aplikácie. V systéme ASP.NET Core 6.0 je možnosť zadefinovať „CORS“ hlavičky pre všetky endpointy naraz v triede Program. definovať

4.4.2 Zoskupovanie IP adries na domovskej stránke

Na domovskej stránke sa nachádza mapa sveta a na nej sú vyznačené body znázorňujúce skupiny IP adries zoskupené podľa geografickej blízkosti. Adresy boli zoskupené podľa geografických súradníc takým spôsobom, aby skupín vzniklo co najviac, ale zároveň stránka zostala funkčná a responzívna. Jedna skupina je ekvivalentná jednému sektoru na mape. Stránka sa stane neresponzívnou v prípade, ak je zobrazených príliš veľa skupín. Vtedy knižnica Leaflet nestíha skupiny dostatočne rýchlo prekresľovať pri pohybe alebo zväčšovaní mapy. Dôležité bolo preto odhadnúť čo najväčši počet skupín tak, aby ich stránka bola stále schopná zobrazovať. Tento počet bol systémom pokus-omyl odhadnutý na . Na testovanie bola použitý kód v ukážke 4.3. Najskôr sme skúsili vygenerovať jeden kruh na každú rovnobežku a na každý poludník, čo znamená 64800 kruhov. To sa ukázalo ako priveľa a tak sme skúsili vygenerovať kruh len na každú

desiatu rovnobežku a každý desiaty poludník, čiže dokopy 648 kruhov, čo aplikácia v prehliadači zvládala. Skúsili sme preto každú piatku rovnobežku a každý piaty poludník, čiže 2592 kruhov. Aplikácia sa v prehliadači stále zvládala zobrazovať. Následne sme postupným pridávaním získali hodnotu približne 6000 kruhov, teda 6000 sektorov na mape. Toto číslo sme následne vynásobili tromi, pretože približne dve tretiny zemského povrchu tvorí voda, v ktorej neočakávame takmer žiadne odpovedajúce IP adresy. Teda z 18000 sektorov bude približne 12000 prázdnych a teda sa zmestíme do 6000 vykreslených kruhov.

Algoritmus 4.3: Ukážka kódu použitého na testovanie počtu možných skupín

```
1    getMarkers(mapPoints: MapIpAddressRepresentation[], zoom:
         number): Leaflet.CircleMarker[] {
2        let markers: Leaflet.CircleMarker[] = [];
3        for(let lat= -90; lat<=90; lat++){
4            for(let lon = -180; lon <=180; lon++){
5                markers.push(new Leaflet.CircleMarker(new Leaflet.
                     LatLng(lat, lon)))
6            }
7        }
8    }
```

Máme teda niekoľko možností ako rozdeliť mapu na 18000 sektorov, my sme sa rozhodli pre zaokrúhlenie zemepisnej šírky a dĺžky na najbližšiu menšiu rovnobežku deliteľnú tromi a na najbližší menší poludník deliteľný šiestimi. Aby body na mape vyzerali prirodzene, za ich stred sme určili aritmetický priemer všetkých bodov v danej skupine namiesto príslušnej zaokrúhlenej hodnoty.

Veľkosť daného kruhu znázorňuje počet IP adries patriacich do daného sektoru. Rozhodli sme sa, že závislosť medzi počtom adries a polomerom kruhu bude logaritmická, pretože v rôznych sektoroch boli rozdiely medzi počtami niekedy rádovo aj desaťtisícnásobné, čo má za následok že menšie sektory by boli prakticky neviditeľné. Vzorec na výpočet tiež obsahuje aj úroveň priblíženia, pretože pri nízkych úrovňach priblíženia sa kruhy príliš prelínali. Preto kruhy začínajú menšie a rastú s úrovňou priblíženia. Konkrétny vzorec bol upravený tak, aby kruhy na mape vyzerali prirodzene. Vzorec je uvedený v ukážke 4.4.

Algoritmus 4.4: Ukážka kódu na výpočet veľkosti kruhu z počtu IP adries v sektore

```
1    ipCountToCircleRadius(ipCount: number, zoom: number): number{
2        return 0.5 * Math.log(ipCount) * zoom;
3    }
```

Farba kruhu je zase určená priemernou dobou odozvy v danom sektore pre daný týždeň. Rozhodli sme sa, že chceme nízke hodnoty doby odozvy reprezentovať zelenou

farbou a vysoké hodnoty doby odozvy červenou farbou. Všetko medzi má byť adekvátne na škále medzi týmito dvoma farbami. Ako vhodné sa preto ukázalo určovať farbu pomocou RGB notácie (jazyk CSS podporuje aj HSL notáciu). Hodnotu B sme určili konštantne na 0. Cieľom je teda rovnomerne sa presunúť medzi RGB hodnotami v HTML notácií `#00FF00` do `#FF0000` s rastúcou dobou odozvy. Ako hornú hranicu pre úplne červenú farbu sme zvolili 500 ms. Túto hranicu sme zvolili preto, že odchýlky k najvyššej dobe odozvy pri niektorých týždňoch boli priveľké. Ako spodnú hranicuu pre absolútnu zelenú sme zvolili hodnotu 5 ms, pretože menšie hodnoty považujeme za zanedbateľné z hľadiska odchýlky merania. Presný algoritmus je uvedený v ukážke 4.5.

Algoritmus 4.5: Ukážka kódu na výpočet veľkosti kruhu z počtu IP adries v sektore

```
1     pingToColor(ping: number) {
2         const upperBound = 500;
3         const lowerBound = 5;
4         let pingInBounds = ping;
5         if(pingInBounds < lowerBound)
6         pingInBounds = lowerBound;
7         if(pingInBounds > upperBound)
8         pingInBounds = upperBound;
9         let percent = (pingInBounds - lowerBound)/(upperBound -
              lowerBound);
10        let r = Math.round(255*percent), g = Math.round((1-percent)
              *255), b = 0;
11        let h = r * 0x10000 + g * 0x100;
12        return '#' + ('000000' + h.toString(16)).slice(-6);
13    }
```

4.4.3 Mapa krajín sveta zafarbených podľa priemernej doby odozvy

Keďže knižnica Leaflet nepodporuje jednoduchý spôsob zafarbovania celých krajín, namiesto toho sme sa rozhodli použiť iný postup. Pomocou vyhľadávača sme našli vzor mapy sveta vo fromáte SVG. Na prácu s dokumentmi SVG existuje v jazyku C# niekoľko knižníc, my sme sa rozhodli pre knižnicu GrapeCity. Formát SVG má tú výhodu, že ak vieme priradiť ku konkrétnej krajine daný objekt jej tvaru na obrázku, vieme v programe jednoducho tento objekt uchopiť a prefarbiť. SVG vzor, ktorý sme použili [18] mal predpoklady na takéto priradenie, pre optimálne použitie ich však bolo treba upraviť. Hlavný problém nezmeneného templatu bolo nekonzistenté pomenovávanie prvkov `path`, kde boli náhodne pre rôzne prvky pomenované ISO 3166-1 Alpha 2 kôdom v atribúte id, menom krajiny v atribúte name alebo menom krajiny v atribúte class či ľubovoľnou kombináciouv daných atribútov. Pre nás bolo najvhodnejšie tento

Obr. 4.1: Pôvodný vzor so SVG mapou sveta, ktorý sme upravili pre lepšie použitie.

systém zjednotiť pre všetky prvky rovnako. Z toho dôvodu sme vylúčili možnosť ISO 3166-1 Alpha 2 kódu v atribúte id, pretože rôzne krajiny sa môžu skladať z viacerých prvkov `path` (napríklad rôzne ostrovy), no id musí byť jedinečné pre konkrétny SVG dokument. Atribút name je netypický pre SVG dokumenty a použitá knižnica v C# s ním nevie pracovať, preto sme si zvolili identifikáciu podľa atribútu class. Ďalší problém je, že niektoré krajiny majú viacero dlhých názvov, no my potrebujeme jeden jednoznačný reťazec na identifikáciu. Preto sme sa rozhodli, že prvky `path` budeme identifikovať podľa ISO 3166-1 Alpha 2 kódu v atribúte class. Preto pri spracovávaní dát pre tento graf ukkladáme ISO 3166-1 Alpha 2 kód krajiny. Súbor sme programom upravili tak, aby všetky prvky `path` mali v atribúte class aj hodnotu adekvátneho ISO 3166-1 kôdu. Pôvodné triedy sme ponechali tiež, pretože jeden prvok môže patriť do viacerých tried, stačí ich oddeliť medzerou.

Algoritmus 4.6: Ukážka kódu neupravenej SVG mapy.

```
1    ...
2    <path class="Canada" d="M 665.9 203.6 669.3 204.5 674 204.3
        670.7 206.9 668.7 207.3 663.2 204.6 662.6 202.5 665.1 200.6
        665.9 203.6 Z">
3    </path>
4    ...
5    <path d="M1633.1 472.812.2-2.4 4.6-3.6-0.1 3.2-0.1
        4.1-2.7-0.2-1.1 2.2-2.8-3.3z" id="BN" name="Brunei
        Darussalam">
6    </path>
7    ...
8    <path class="Azerbaijan" d="M 1229 253.2 1225.2 252.3 1222
        249.4 1220.8 246.9 1221.8 246.8 1223.7 248.5 1226 248.5
        1226.2 249.5 1229 253.2 Z">
9    </path>
10   ...
```

Algoritmus 4.7: Ukážka kódu upravenej SVG mapy.

```
...
<path class="Canada CA " d="M665.9,203.6 669.3,204.5 674,204.3
    670.7,206.9 668.7,207.3 663.2,204.6 662.6,202.5 665.1,200.6
    665.9,203.6Z"/>
...
<path id="BN" class=" BN" d="M1633.1,472.8l2.2,-2.4 4.6,-3.6
    -.1,3.2 -.1,4.1 -2.7,-.2 -1.1,2.2 -2.8,-3.3z" name="Brunei
    Darussalam"/>
...
<path class="Azerbaijan AZ " d="M1229,253.2 1225.2,252.3
    1222,249.4 1220.8,246.9 1221.8,246.8 1223.7,248.5H1226L1226
    .2,249.5 1229,253.2Z"/>
...
```

Na základe upravených tried sme boli schopní identifikovať krajiny aj pri pridávaní prvkov `title`. Tento prvok má za následok to, že pri podržaní kurzora myši nad danou krajinou sa zobrazí vyskakovacie okno s informáciou o názve krajiny a konkrétnou hodnotou doby odozvy. Na prevod medzi ISO 3166-1 Alpha 2 kódom, ktorý je v atribúte class a názvom krajiny používame slovník, ktorý sme zostavili ako konštantu v triede GeographicUtilities. Ako zdroj pre tento tento zoznam sme použili voľný súbor vo formáte JSON [4]. Pre úpravu do formátu, v akom sa v jazyku C# definujú slovníky, sme použili nástroj pre nájdenie a nahradenie v programe Visual Studio, ktorým sme reťazce `"Name":` a `"Code":` nahradili prázdnym reťazcom.

Algoritmus 4.8: Ukážka kódu CountryCodeToNameDictionary

```
public static Dictionary<string, string>
    CountryCodeToNameDictionary = new Dictionary<string, string
    >()
{
    { "AF", "Afghanistan" },
    { "AX", "\u00c5land Islands" },
    { "AL", "Albania" },
    { "DZ", "Algeria" },
...
};
```

4.4.4 Priebeh priebežného spracovávania dát

Súčasťou nášho systému sú aj dve služby, ktoré majú za úlohu na pozadí priebežne spracovávať dáta pribúdajúce do tabuľky. Tieto služby sa spustia raz za týždeň pomocou zabudovaného Linuxového nástroja cron, ktorý slúži ako plánovač úloh. Pre

spustenie služby stačí spustiť kontainer príslušnej služby, služba sa spustí spolu s ním. Po dokončení spracovávania sa služba sama ukončí.

Dáta sa pri každom novom spustení len doplnia, spracovanie sa nerobí odznova. Programy dopĺňajú dáta nasledovným spôsobom:

- Program skontroluje, či existujú všetky potrebné tabuľky v databáze, ak nie tak ich vytvorí.
- Program získa dátum posledných spracovaných dát. Ak tabuľka neobsahuje žiadne dáta, za posledný dátum sa určí 19.4.2008, čo je týždeň pred začatím meraní, aby sa dáta začali spracovávať od prvého týždňa.
- Z databázy sa vytiahne zoznam všetkých platných IP adries, pre ktoré máme geografické dáta.
- Tieto adresy sa následne zoskupia podľa určených parametrov pre daný program pomocou metódy `GroupBy` zo vstavanej C# knižnice LinQ. To znamená, že služba CountryPingInfoService zoskupí IP adresy podľa ISO 3166-1 Alpha 2 kódu príslušnej krajiny, kým služba MapPointsService zoskupí adresy podľa dvojice najbližšieho vyhovujúceho poludníka a najbližšej vyhovujúcej rovnobežky.
- Zavolá sa funkcia enumerujúca týždne od posledného spracovania po súčasnosť.
- Následne sa cez tieto týždne paralelne iteruje pomocou vstavanej metódy `Paralel.ForEachAsync`, ktorá podporuje iteráciu na viacerych vláknach naraz, čo má z následok výrazne skrátenú dobu spracovávania, pretože dáta pre jednotlivé týždne nie sú od seba závislé.
- Vrámci jednotlivej iterácie sa pre konkrétny týždeň vyberie zoznam všetkých meraní doby odozvy pomocou príkazu ping ktoré prebehli v daný týždeň.
- Následne sa vykoná projekcia pre všetky skupiny, pri ktorej sa na každú skupinu premietne jej výpočet priemernej doby odozvy v daný týždeň.
- Výpočet pre jednotlivú skupinu prebieha tak, že sa pre každú IP adresu v skupine nájde jej príslušná doba odozvy pre daný týždeň zo zoznamu všetkých meraní.
- Parametre danej skupiny a vypočítané hodnoty sa následne použijú na zostavenie výsledného objektu, teda CountryPingInfo alebo MapPoint.
- Výsledné objekty sa nakoniec vrámci jednej transakcie zapíšu do databázy. To má za následok, že pre daný týždeň sa dáta buď uložia všetky alebo v prípade chyby neuložia žiadne. Vďaka tomu sa vyhneme nekonzistencii dáta pre daný týždeň.

- Po dokončení behu sa aplikácia sama ukončí volaním príkazu `Environment.Exit(0)`. Služby v C# sa totiž bez zavolania tohto príkazu same neukončia.

Kapitola 5

Testovanie

Aplikáciu sme testovali dvoma spôsobmi. Najskôr sme nappísali a spustili unit testy pomocou knižnice xUnit. Následne sme sa pozreli na ručné používanie samotnej aplikácie a testovanie používateľského rozhrania.

5.1 Testovanie unit testami

Unit testy sú metóda testovania softvéru, kedy je softvér rozdelený na veľa malých jednotiek (z anglického units), a každá časť je otestovaná samostatne [13]. V našom kóde sme ako jednotky brali public metódy tried.

Pre unit testovanie v jazyku C# sa najčastejšie používajú riešenia MSTest, NUnit a xUnit [17]. Každá z nich ponúka všetky pre nás dôležité funkcionality, takže výber je možné spraviť podľa osobnej preferencie. Vybrali sme si knižnicu xUnit pretože nám najviac vyhovoval spôsob overovania výsledkov (všetky knižnice používajú triedu s názvom Assert, každá je však inako implementovaná).

Pri písaní unit testov sa najčastejšie postupuje tak, že triedam vložíme namiesto skutočných objektov sprostredkúvajúcich dáta, napríklad inštancie IpInfoViewerDbRepository, objekty vracajúce vymyslené, nami definované dáta. Tetno proces sa nazýva mockovanie. Mockovanie je možné vďaka použitiu návrhového vzoru vkladanie závislosti. Tento vzor určuje, že trieda by nemala vytvárať inštanciu inej triedy a teda byť od nej závislá, ale namiesto toho by mala len prijímať inštancie spĺňajúce definované rozhranie [8]. Vytváranie mockovacích objektov zabezpečjeme pomocou knižnice Moq. Tá nám šetrí prácu, pretože nemusíme pre každý mockup vytvárať novú triedu, ale len zadefinujeme dáta, ktoré chceme aby nám inštancia vrátila. Knižnica Moq nám takúto inštanciu spĺňajúcu potrebné rozhranie vytvorí [14].

Pri písaní unit testov pomocou knižnice xUnit sa testovacie metódy označujú atribútmi `Fact` a `Theory`. Kým metódy s atribútom `Fact` slúžia ako testy bez parametrov, metódy s atribútom `Theory` doplnené parametrom `InlineData` prijímajú aj parametre

[6].

Naším cieľom bolo vytvoriť sadu unit testov pre každú metódu z tried, ktoré sa starajú o logickú časť backendovej časti, teda `Week`, `DateTimeUtilities`, `GeographicUtilities`, `CountryPingInfoFacade`, `IpAddressInfoFacade` a `MapPointsFacade`. Triedy `MFileDbRepository` a `IpInfoViewerDbRepository` úmyselne netestujeme pretože v jazyku C# neexistuje dobrý spôsob na simuláciu volania SQL databázy. Taktiež nepíšeme testy na kontrolery, pretože tie v našej implementácií dáta čisto len predávajú z nižšej vrstvy, preto nie je potrebné ich testovať unit testami. Chyby v takomto predávaní dát by sa v každom prípade rýchlo prejavili pri používaní webovej aplikácie, čomu sa venujeme v sekcii 5.2.

5.2 Testovanie používateľského rozhrania

V tejto časti otestujeme používanie aplikácie podľa scenárov definovaných v kapitole 2.2.1.

Kapitola 6

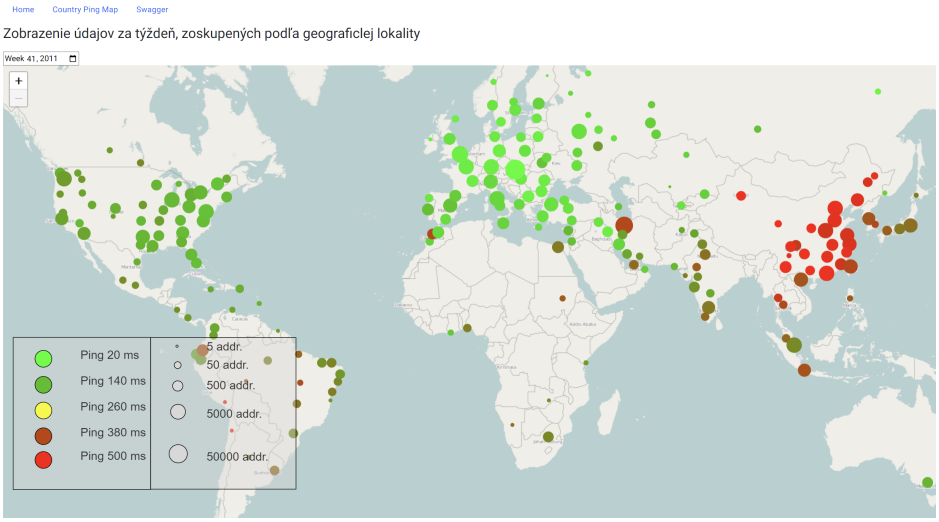
Používanie aplikácie

6.1 Úvodná strana a bodová mapa

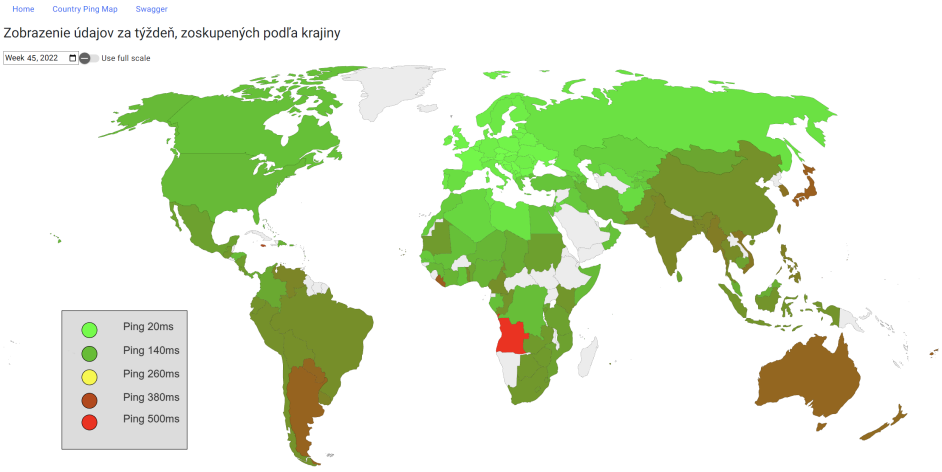
Aplikácia je prezentačne nasadená na adrese https://bp.trencansky.com. Po otvorení aplikácie sa ako prvá zobrazí stránka obahujúca nástroje na zobrazovanie mapy sveta s bodmi označujúcimi IP adresy zoskupené podľa lokality. Vo vrchnej časti aplikácie sa nachádza menu pomocou ktorého je možné prechádzať medzi podstránkami aplikácie. Pod ním sa nachádza nadpis popisujúci dané zobrazenie. Vľavo pod nadpisom je umiestnený selektor týždňov, pomocou ktorého je možné určiť týždeň, pre ktorý chceme dáta zobraziť. Po zmene hodnoty v selektore sa dáta zmenia okamžite. Prednastavená hodnota pri zobrazení aplikácie je posledný týždeň, pre ktorý existujú spracované dáta. V hlavnej sekcii aplikácie vidíme samotné zobrazenie dát pomocou mapy. V mape v ľavo dole sa tiež nachádza legenda, ktorá bližšie opisuje koľko konkrétne IP adries je znázornených kruhom určenej veľkosti a aká priemerná doba odozvy príkazu ping je zobrazená danou farbou. Detaily výpočtu sú opísané v kapitole 4.4.

6.2 Mapa krajín zafarbených podľa doby trvania odozvy

Po kliknutí na podstránku Country Ping Map sa zobrazí mapa sveta s krajinami zafarbenými podľa priemernej doby pingu pre zvolený týždeň. Vo vrchnej časti zostalo nezmeneené menu a rovnako aj selektor týždňov. Vedľa selektora pribudol prepínač škály. Ten rozhoduje, či je pre maximálnu hodnotu použitá hodnota 500 milisekúnd a všetko ostatné už je len rovnako červené, alebo je ako maximum použitá maximálna hodnota z daného týždňa. Po prepnutí prepínača sa mapa prekreslí okamžite, rovnako ako po prepnutí týždňa. V hlavnej časti sa nachádza samotná mapa s legendou, tentoraz opisujúcov len prevod farieb na konkrétnu dobu odozvy. Po prejdení myšou nad krajinu sa zobrazí okno s názvom krajiny a konkrétnou hodnotou doby odozvy.



Obr. 6.1: Na úvodnej stránke aplikácie vidíme menu, pod ním nadpis a selektor týždňa. Nakoniec vidíme mapu s bodmi znázorňujúcimi IP adresy ktoré úspešne odpovedali na ping v danom týždni zoskupené podľa lokality. V ľavo dole je legenda znázorňujúca prevod veľkosti kruhu na počet IP adries a farby kruhu na ping v milisekundách.



Obr. 6.2: Na podstránke je vidno hlavné menu, taktiež pod ním selektor týždňov a samotnú mapu. Oproti domovskej stránke pribudol prepínač škály.

Záver

Výsledkom práce je webová aplikácia, ktorá umožňuje vizualizovať výsledky meraní internetu v grafickej forme pomocou máp. Tento systém by mal pomôcť používateľom lepšie pochopiť dlhodobé trendy a charakteristiky vývoja rýchlosti a dostupnosti internetu.

V úvodnej kapitole sme zhrnuli teóriu potrebnú k pochopeniu meraných dát. Ukázali sme si štruktúru meraní a definovali sme si ktoré merania sa dajú považovať za platné. Nakoniec sme sa pozreli na iné práce v podobnej oblasti.

V ďalšej časti sme navrhli, ako by mala aplikácia vyzerať. Navrhli sme, čo všetko by mal používateľ vedieť v aplikácií robiť a ako presne by mala aplikácia fungovať na architektonickej úrovni. Určili sme základné funkčné požiadavky z používateľskej strany, navrhli sme rozhranie, ktoré by malo byť používané na komunikáciu medzi frontendovou a backendovou časťou. Tiež sme navrhli štruktúru databázových tabuliek a ich indexáciu.

V ďaľšej časti sme si prešli možnosti spôsobu implementácie. Prešli sme si výberom programovacieho jazyka zvlášť pre backendovú a frontendovú časť a procesom výberu databázového poskytovateľa. Tiež sme sa zamysleli nad možnosťami výberu použitých knižníc a všeobecnou architektúrou programu.

V kapitole 4 sme riešenie implementovali podľá návrhu. V kapitole sme rozobrali detaily implementácie od štruktúry programového systému, cez detaily tried backdendového riešenia, analýzu použitých techník a algoritmov až po opis frontendových komponentov. V ďalšej kapitole sme aplikáciu otestovali automatickými unit testami aj ručným skontrolovaním podľa špecifikácie.

V poslednej kapitole sme si prešli skutočné používanie aplikácie.

V práci sa nám podarilo splniť cieľ vybudovania aplikácie na vizualizáciu dlhodobých meraní internetu. Aplikáciu sme prezentačne nasdili do používania a je možné ju použiť na analýzu nameraných hodnôt. K aplikácii je dodaný návod na inštaláciu na ľubovoľnom počítači spĺňajúcom minimálne kritériá a tiež používateľská príučka.

Ako možné budúce rozšírenie práce sa naskytá možnosť zobraziť dáta o dobe odozvy vo forme grafov. Aplikáciu je možné rozšíriť aj o analýzu meraní topológie siete. Aplikácia by sa tiež dala rozšíriť o možnosť definície chcených zobrazení zo strany používateľa. Momentálne sa dá aplikácia jednoducho rozšíriť implementáciou spracovávacieho

servisu v ľubovoľnom jazyku a dodefinovaním zobrazenia vo frontendovej časti.

Literatúra

[1] RIPE Atlas. Rtt measurements to fixed destinations. Dostupné z https://atlas.ripe.net/results/maps/rtt-fixed/.

[2] Ignacio L. Bisso. What is a database index?, 2020. Dostupné z https://learnsql.com/blog/what-is-an-index/.

[3] KC Claffy. Nae99: Internet measurement and data analysis:topology, workload, performance and routing statistics. Technical report, University of California, San Diego, 1999. Dostupné z https://www.caida.org/catalog/papers/1999_nae/.

[4] DataHub.io, 2018. Dostupné z https://datahub.io/core/country-list.

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

[6] .NET Foundation. Getting started with xunit.net, 2018. Dostupné z https://xunit.net/docs/getting-started/netcore/visual-studio.

[7] Jason Gerend, Robin Harwood, and contributors. Documentation - ping, 2023. Dostupné z https://learn.microsoft.com/en-us/windows-server/administration/windows-commands/ping.

[8] Alexander S. Gillis. dependency injection, 2020. Dostupné z https://www.techtarget.com/searchapparchitecture/definition/dependency-injection.

[9] The PostgreSQL Global Development Group. Postgresql: Documentation: 15: 8.9. network address types. Dostupné z https://www.postgresql.org/docs/current/datatype-net-types.html.

[10] Red Hat. What is a rest api?, 2020. Dostupné z https://www.redhat.com/en/topics/api/what-is-a-rest-api.

[11] IBM. What is a data warehouse? Dostupné z https://www.ibm.com/topics/data-warehouse.

[12] Goran Jevtic. Linux ping command tutorial with examples, 2019. Dostupné z https://phoenixnap.com/kb/linux-ping-command-examples.

[13] Mike Jones, Amaury Levé, and contributors. Unit test basics, 2022. Dostupné z https://learn.microsoft.com/en-us/visualstudio/test/unit-test-basics?view=vs-2022.

[14] Sean Killeen. Quickstart, 2022. Dostupné z https://github.com/Moq/moq4/wiki/Quickstart.

[15] Bojan Petrovic. Sql index overview and strategy, 2018. Dostupné z https://www.sqlshack.com/sql-index-overview-and-strategy/.

[16] Sagar Sharma. traceroute command examples in linux, 2022. Dostupné z https://linuxhandbook.com/traceroute/.

[17] Himanshu Sheth. Nunit vs. xunit vs. mstest: Comparing unit testing frameworks in c#, 2021. Dostupné z https://www.lambdatest.com/blog/nunit-vs-xunit-vs-mstest/.

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

[19] Dennis Vita. Analýza dát výsledkov dlhodobých meraní vybraných charakteristík internetu. Technical report, 2022.

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 `https://github.com/Trenky2122/IpInfoViewer`.

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.