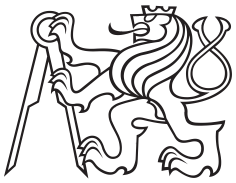
Bakalářská práce



České
vysoké
učení technické
v Praze

F3

Fakulta elektrotechnická
Katedra telekomunikační techniky

Vizualizace dat naměřených platformou F-Tester®

Miroslav Halamka

Vedoucí: Ing. Zbyněk Kocur, Ph.D.
Obor: Softwarové inženýrství a technologie
Květen 2022

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

I. OSOBNÍ A STUDIJNÍ ÚDAJE

Příjmení: **Halamka** Jméno: **Miroslav** Osobní číslo: **487603**

Fakulta/ústav: **Fakulta elektrotechnická**

Zadávající katedra/ústav: **Katedra počítačů**

Studijní program: **Softwarové inženýrství a technologie**

Studijní obor: **bez oborů**

II. ÚDAJE K BAKALÁŘSKÉ PRÁCI

Název bakalářské práce:

Vizualizace dat měřených platformou F-Tester

Název bakalářské práce anglicky:

F-Tester Measured Data Visualisation

Pokyny pro vypracování:

Analyzujte data, která sbírá měřící plaftorma F-Tester
[1]. Následně nalezněte vhodný formát vizualizace
[2,3,4]. Dbejte na možnosti interaktivity - filtrování či změna dat v závislosti na poloze či čase
[5]. Navržené řešení musí být autonomní - nesmí být závislé na externích systémech. Proveďte výkonnostní i uživatelské testování.

Seznam doporučené literatury:

[1] F-Tester - Efficient, reliable and accurate testing of data networks - https://f-tester.fel.cvut.cz
[2] Claus O. Wilke. Fundamentals of Data Visualization: A Primer on Making Informative and Compelling Figures, ISBN: 978-1492031086
[3] Stephanie Evergreen. Effective Data Visualization: The Right Chart for the Right Data, ISBN: 978-1544350882
[4] Steve Wexler. The Big Book of Dashboards: Visualizing Your Data Using Real-World Business Scenarios, ISBN: 978-1119282716
[5] D3.js - Data-Driven Documents - https:///d3js.org/

Jméno a pracoviště vedoucí(ho) bakalářské práce:

Ing. Zbyněk Kocur, Ph.D. katedra telekomunikační techniky FEL

Jméno a pracoviště druhé(ho) vedoucí(ho) nebo konzultanta(ky) bakalářské práce:

Datum zadání bakalářské práce: **07.02.2022** Termín odevzdání bakalářské práce: **20.05.2022**

Platnost zadání bakalářské práce: **30.09.2023**

Ing. Zbyněk Kocur, Ph.D.	podpis vedoucí(ho) ústavu/katedry	prof. Mgr. Petr Páta, Ph.D.
podpis vedoucí(ho) práce		podpis děkana(ky)

III. PŘEVZETÍ ZADÁNÍ

Student bere na vědomí, že je povinen vypracovat bakalářskou práci samostatně, bez cizí pomoci, s výjimkou poskytnutých konzultací. Seznam použité literatury, jiných pramenů a jmen konzultantů je třeba uvést v bakalářské práci.

. Datum převzetí zadání

Podpis studenta

Poděkování

Děkuji vedoucímu Ing. Zbyňku Kocurovi, Ph.D. za vstřícnost, trpělivost, odbornou pomoc a za čas strávený nad kontrolou práce. Dále bych chtěl poděkovat panu Ing. Ondřeji Votavovi a panu Ing. Ondřeji Vondroušovi, Ph.D. za věcné připomínky k implementaci a cenné rady týkající se práce.

Prohlášení

Prohlašuji, že jsem předloženou práci vypracoval samostatně, a že jsem uvedl veškerou použitou literaturu.

V Praze, 20. května 2022

Abstrakt

V práci je provedena analýza měřených dat měřící platformy F-Tester®, současných technologií pro tvorbu interaktivních map a způsobů vizualizace. Jsou navržena autonomní řešení ve vybraných technologiích se zvolenou vizualizací. S kandidáty je provedeno výkonnostní a uživatelské testování. Dle výsledků z testování je nejlepším kandidátem zvolena vizualizace pomocí bodu v knihovně Leaflet.

Klíčová slova: F-Tester®, Leaflet, OpenLayers, Mapbox GL, Mapy.cz API/SDK, OpenStreetMap

Vedoucí: Ing. Zbyněk Kocur, Ph.D.

Abstract

The thesis analyses the measured data of the F-Tester®measurement platform, current technologies for creating interactive maps, and visualization methods. Also suggests autonomous solutions in chosen technologies with selected visualization. Performance testing and usability testing are conducted with the candidates. According to the results from the testing, the best candidate is the visualization using circles in Leaflet library.

Keywords: F-Tester®, Leaflet, OpenLayers, Mapbox GL, Mapy.cz API/SDK, OpenStreetMap

Obsah

Obrázky

Tabulky

Seznam zkratek

4G Fourth Generation of Mobile Telecommunications Technology. 3

5G Fifth Generation of Mobile Telecommunications Technology. 3

API Application programming interface. 5, 7

BSD Berkeley Software Distribution. 17

CSS Cascading Style Sheets. 22, 27

FEL Fakulta elektrotechnická. 2

GPS Global Positioning System. 26

HSL Hue, Saturation, Lightness. 21, 22

HTML HyperText Markup Language. 22, 27

HTTP Hypertext Transfer Protocol. 22

JSON JavaScript Object Notation. 3, 7, 16, 28

RAM Random-access memory. 16, 31

RGB Red, Green, Blue. 22

RM ISO/OSI Referenční model International Organization for Standardization/Open System Interconnection. 3

RSRP Reference Signal Receive Power. 21, 28

RSRQ Reference Signal Receive Quality. 21, 28

RSSI Received signal strength indication. 3

RTT Round-trip time. 3

SDK Software development kit. 5

Seznam zkratek

SNR Signal-to-noise ratio. 3, 21, 28

TCP Transmission Control Protocol. 2, 3

TCP/IP Transmission Control Protocol/Internet Protocol. 1, 2

UDP User Datagram Protocol. 2

WI-FI Wireless Fidelity. 3

ČTU Český telekomunikační úřad. 33

ČVUT České vysoké učení technické. 2

Kapitola 1
Úvod

Hlavním cílem práce je návrh vizualizace vybraných dat měřených platformou F-Tester®. F-Tester® je zařízení vyvinuté na katedře Telekomunikační techniky Českého vysokého učení technického sloužící pro měření parametrů komunikačních sítí založených na protokolech TCP/IP. Dalším cílem je výběr a ověření nástrojů pro vizualizaci dat na mapových podkladech.

Hlavním akceptačním kritériem je navržení interaktivní vizualizace, která je autonomní. Je očekáváno více možných přístupů, proto dalším kritériem je provést s kandidáty uživatelské a výkonností testování, podle kterého je vybrána technologie pro konečné řešení.

V úvodní kapitole je představeno zařízení F-Tester®. V dalších kapitolách je definována metodika výběru technologií, představení současného řešení, analýza požadavků a technologií, návrh vizualizace a výběr technologií. Poté následuje popis implementace ve vybraných technologiích a průběhu testování. V poslední kapitole je provedeno shrnutí výsledků a diskuze nad možným rozšířením pro diplomovou práci.

Kapitola 2
Současné řešení

V první polovině této kapitoly je představena platforma F-Tester®, její schopnosti měření, naměřená data a jejich vizualizace v mapách. Ve druhé polovině jsou popsány aktuální přístupy k tvorbě interaktivních map.

2.1 Měřicí platforma F-Tester®

F-Tester® [8]je zařízení vyvinuté na Katedře telekomunikační techniky FEL ČVUT v Praze, které měří parametry komunikačních sítí založených na protokolech TCP/IP. Zařízení existuje v několika hardwarových modifikacích s operačním systémem Linux v distribuci OpenWrt [11]. OpenWrt je operační systém pro vestavěná zařízení vytvořený na bázi Linuxu. Uživateli umožňuje plnou kontrolu nad souborovým systémem. Díky tomu je možné postavit systém přímo pro potřeby zařízení bez dalších aplikací okolo.

Koncepce je taková, že veškerá měření probíhají na samotném zařízení. Následné zpracování a analýza probíhají ve webovém rozhraní založeném na technologii Vue.js. Vue.js je JavaScriptový framework sloužící k vytváření single-page aplikací.

2.1.1 Představení měřicí platformy

Zařízení F-Tester® je univerzální měřicí systém pro TCP/IP sítě s plnou konfigurací generovaného měřicího provozu.

Dokáže měřit krátkodobě (ověření funkčnosti, ověření parametrů), ale také dlouhodobě v řádu hodin, dnů či týdnů díky čemuž získáme přehled o stabilitě sítě.

Pro protokol TCP můžeme konfigurovat parametry přenosu včetně počtu toků, měnit posloupnosti testů, volit varianty TCP/IP algoritmů, vyhodnocovat přenosové rychlosti a zpoždění ve smyčce.

U testů propustnosti protokolu UDP lze ověřovat jeho minimální propustnost a stabilitu, vyhodnocovat zpoždění a ztrátovost paketů.

Zařízení dokáže generovat UDP tok s různou konstantní, ale i v čase proměnlivou, velikostí paketů (schody, pila, dávky a jiné časové průběhy) nebo simulovat zahlcení linky. Také dokáže míchat TCP a UDP toky.

Platforma F-Tester® pracuje v topologiích bod-bod, bod-více bodů či mesh.

F-Tester® disponuje plně konfigurovatelným editorem testů a scénářů, díky kterým lze realizovat cílená měření na určité efekty a události v síti či chování aplikací [8].

2.1.2 Měřená data

Zařízení je schopné měřit řadu telemetrických údajů komunikačních sítí od fyzické vrstvy RM ISO/OSI až po transportní vrstvu. F-Tester® tato data ukládá spolu s polohou měřicího terminálu ve formátu JSON. K podporovaným přenosovým technologiím patří: Ethernet WI-FI, Mobilní sítě 4G/5G.

Údaje získávány na fyzické vrstvě se liší podle typu přenosové technologie. Lze tedy říct, že se získávají vždy ty parametry, které definují kvalitativní parametry komunikačního rozhraní. Pro technologii 5G se jedná o následující parametry: RSSI - intenzita přijímaného signálu v dBm a SNR - odstup signálu od šumu udávaný v dB.

Na 3. a 4. vrstvě RM ISO/OSI dokáže zařízení měřit aktuální přenosovou rychlost, zpoždění ve smyčce RTT a ztrátovost. Měření lze provádět, jak ve vzestupném (Upstream), tak i sestupném (Downstream) směru komunikace. U protokolu TCP lze pak měřit a analyzovat počty znovu přeposlaných paketů nebo vývoj velikosti okna zahlcení v průběhu komunikace.

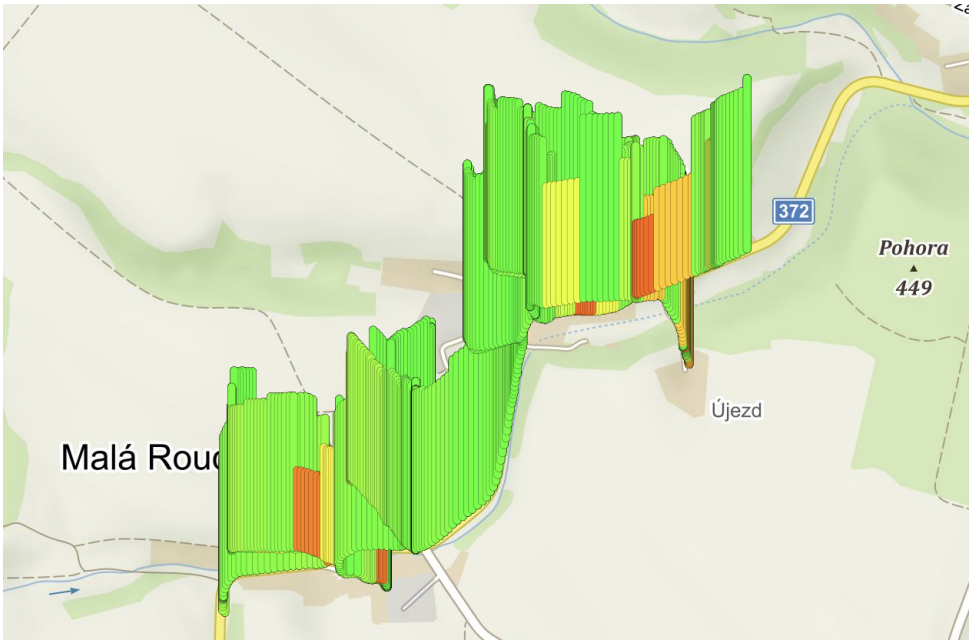
Všechna data jsou měřena moduly, které čas od času mohou naměřit neplatná data a je nutné aby byla odfiltrována.

Množství dat měřených a zaznamenaných platformou F-Tester® je široké a jejich kompletní výčet je uveden na stránce projektu.

2.1.3 Zobrazení naměřených dat v mapě

Současné řešení využívá vizualizaci pomocí obdélníků, které mění svoji velikost a barvu dle dat. Tato reprezentace není ideální, protože může docházet ke ztrátě informace. Zejména při klikatých cestách a při zobrazení velkých množství dat na malém prostoru se obdélníky přes sebe překrývají.

Z tohoto důvodu bylo rozhodnuto upustit od tohoto stylu zobrazení a hledat reprezentaci, u které nebude docházet ke ztrátě informace. Viz obrázek 2.1.



Obrázek 2.1: Současná vizualizace se ztrátou informace.

2.2 Vizualizace dat do map

Přístupů k vizualizaci dat do map je mnoho. Prvním a nejjednodušším řešením je použití nástroje, ve kterém lze nadefinovat požadovaná kritéria vizualizace. Zástupcem tohoto přístupu je například ArcGIS [7].

Další možností je vytvoření nativní aplikace, která by byla podobná výše zmíněným nástrojům, ale byla by vytvořena dle požadavků ze zadání. V Javě je možné takovou aplikaci vytvořit například pomocí knihovny Unfolding [14].

Poslední možností je vytvoření aplikace běžící v prohlížeči pomocí JavaScriptových knihoven. Jelikož F-Tester® využívá webové rozhraní, soustředil jsem se výhradně na tento přístup. Představeny jsou zástupci podporující offline mapové podklady z databáze OpenStreetMap a aktuálně využívané řešení.

Díky tomu je možné výstup této práce integrovat do současného grafického rozhraní zařízení F-Tester®.

2.2.1 Leaflet

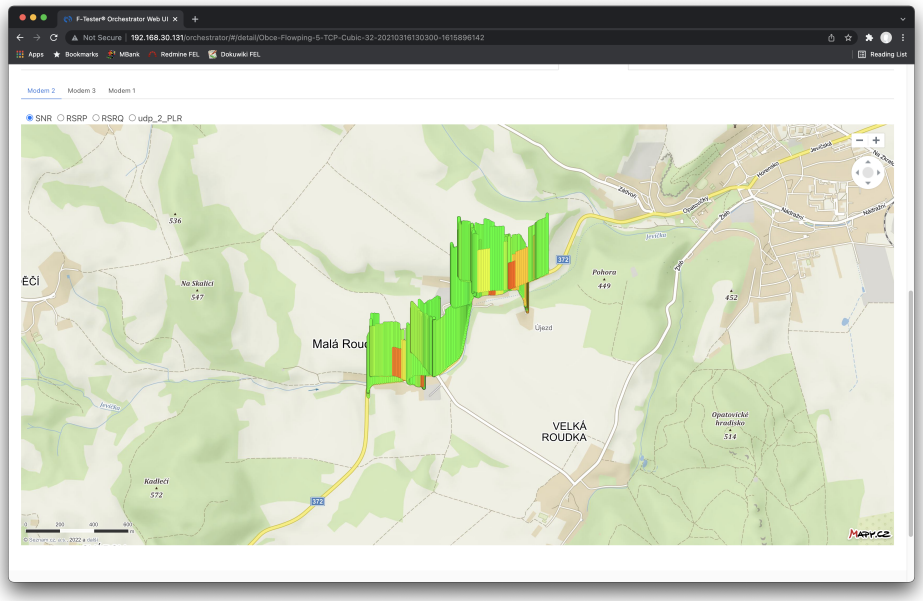
Leaflet je moderní open-source JavaScriptová knihovna pro tvorbu interaktivních map. Jak sami autoři uvádějí, je designován s citem pro *„jednoduchost, výkon a použitelnost“* [1] pro všechny platformy. Knihovnu skvěle doplňuje nespočet pluginů, které dokáží z Leafletu udělat mocný nástroj. Jedním ze zajímavějších je například plugin, který implementuje jednoduchou navigaci [2]. Knihovna má, dle dostupných zdrojů, dokázat zobrazit více jak 100 000 bodů bez ztráty odezvy [17].

2.2.2 OpenLayers

OpenLayers je open-source knihovna vytvořená už v roce 2006, ale v žádném případě nemůže být nazývána zastaralou. Nejnovější verze 6.14.1 byla vydána 26.3.2022 [19] tudíž je jasné, že je stále vyvíjenou knihovnou. OpenLayers jsou oproti Leafletu robustní a nabízí vše, co si programátor může přát, ale také jsou méně přehledné a pro nezkušeného programátora matoucí [10].

2.2.3 Mapbox GL

Mapbox poskytuje, jak vlastní vyrenderované mapy z databáze OpenStreeMap, tak také knihovnu Mapbox GL (dále jen Mapbox). Mapbox přináší mnoho zajímavých snadno použitelných věcí jako jsou například 3D mapy či adaptivní projekce. Je ale od 50 000 načtení za měsíc placený. Cena je 5$ za 1000 načtení s navyšující se slevou při překročení určitých počtů načtení [13]. Zajímavostí je, že v roce 2013 do Mapboxu nastoupil zakladatel Leafletu Vladimir Agafonkin a propojil tak Mapbox a Leaflet [12]. První verze Mapboxu byla dokonce vytvořena jako Leaflet plugin, a i v dnešní době je patrné, že jádro Mapboxu je postavené na Leafletu. Největší výhodou tohoto propojení je kompatibilita Mapboxu s Leaflet pluginy [9].



Obrázek 2.2: Aktuální grafické rozhraní zařízení F-Tester®běžící v prohlížeči.

2.2.4 Mapy.cz API/SDK

Momentálně na platformě F-Tester® je používána implementace využívající Mapy.cz API/SDK od společnosti Seznam.cz [18], která je nevyhovující. Úvodní načítání mapy pro testovací data je v v rozmezí od 2,73 sekund

do 2,93 sekund, což je nepřijatelné. Další nevýhodou je rychlost vývoje a rozšíření, které se nemůže rovnat výše uvedeným knihovnám. Ukázka současného rozhraní je vyobrazena na obrázku 2.2.3.

Kapitola 3
Metodika

Pro výběr vhodné zobrazovací technologie a nalezení adekvátního způsobu vizualizace pro definované požadavky je vypracována metodika. Ta je složena ze dvou částí, které se opírají o požadavky definované v zadání a jsou detailněji rozepsány v kapitole 2.1. Jejich souhrn je pak proveden v následujícím textu v kapitoly 3.1. Samotný způsob výběru vhodné vizualizační technologie je navržen a popsán v kapitole 3.2. Vizualizací a jejím vhodným provedením se zabývá návrh popsaný v kapitole 3.3.

3.1 Požadavky

Následující kapitola seskupuje identifikované požadavky pro návrh a realizaci vizualizačního systému. Požadavky jsou rozděleny do 2 kategorií: funkční a nefunkční.

3.1.1 Funkční požadavky

1. Zařízení sbírá data z různých modulů a může docházet k výpadkům měření. Z tohoto důvodu je zapotřebí filtrovat poškozená a neplatná data.
2. Zobrazení připravených dat na mapových podkladech.

3.1.2 Nefunkční požadavky

1. S ohledem na API zařízení F-Tester® musí být řešení napsáno v JavaScriptu.
2. Zařízení F-Tester® generuje data ve formátuJSON, tudíž aplikace musí s tímto formátem pracovat a převést ho do vhodné podoby pro vizualizaci.
3. Pro zachování mobility zařízení, řešení nesmí být závislé na jiných systémech.
4. Řešení musí být interaktivní.
5. Je požadováno, aby bylo možné vykreslit více než 50 000 bodů

6. Doba mezi načtením stránky a vykreslením mapových podkladů musí být menší než 3 s.
7. Je požadováno, aby aplikace fungovala v prohlížeči (Chrome, Firefox, Safari a EDGE).
8. Při vizualizaci standardních měření (desítky tisíc záznamů) musí mít řešení odezvu v řádu desítek milisekund.
9. Aplikace musí být snadno rozšiřitelná, tudíž čitelná i pro programátory, kteří nemají zkušenost s vybranou knihovnou.
10. Navržená vizualizace musí být přehledná a musí být omezen stav, kdy dochází ke ztrátě informace při zobrazování velkého množství dat na malém prostoru.

Dle požadavků definovaných výše je sestavena metodika, která popisuje proces výběru, realizace a ověření vizualizačního systému. Práce je složena ze dvou částí: Výběru technologií a Výběru vizualizace.

3.2 Výběr technologií

Při výběru technologie jsou testovány tři technologie pro tvorbu interaktivních map. Je provedena jejich analýza a implementace základních zobrazení dat na mapě. Poté je proveden test, který má za cíl otestovat chování knihoven při zobrazení velkého množství dat. Hodnocena je rychlost načtení a odezva při prohlížení mapy. Zohledněna je i licence dané knihovny, jednoduchost implementace a obecné chování knihoven (například zda knihovna zobrazí všechna data). Jsou vybráni dva nejvhodnější kandidáti, ve kterých je implementováno řešení, které splňuje všechny požadavky zadávajícího. Níže jsou popsány kroky, podle kterých probíhá výběr technologií:

1. Analýza vybraných technologií.
2. Implementace vizualizace jednoduchého grafického stylu.
3. Otestování schopnosti vizualizovat velký vzorek dat.
4. Porovnání vlastnosti knihoven v různých prohlížečích.

3.3 Výběr vizualizace

V této části je vybráno konečné řešení vizualizace. Nejprve je provedena analýza mapových podkladů. Je navržen objekt a legenda reprezentující zobrazovaná data a také je navrženo řešení offline mapových podkladů. Je implementována vybraná vizualizace do obou vybraných knihoven spolu s dalšími požadavky konečného řešení, jak jsou offline mapové podklady nebo

vložení více datových vrstev. Nakonec je provedeno testování na reálných datech a porovnání vizualizací.

Je měřena rychlost načtení, odezva, přehlednost vizualizace, celkové paměťové nároky a jednoduchost implementace. Níže je popsán postup, podle kterého proběhne výběr vizualizace.

1. Analýza mapových podkladů.
2. Návrh vizualizace a řešení offline mapových podkladů.
3. Implementace vizualizace s následujícími parametry:
   - Použití složitějších grafických stylů navržených dle zadání.
   - Použití více vrstev s různými typy dat.
   - Použití offline mapových podkladů.
4. Volba vizualizace dle následujícího testování:
   - Testování vybraných řešení na velkém vzorku reálných dat > 30000.
   - Porovnání v různých prohlížečích.
   - Porovnání různých způsobů vizualizace.

Kapitola 4

Výběr technologie

V této kapitole je provedena analýza vybraných technologií a navržena základní implementace, která otestuje schopnost zobrazovat velké množství dat. Poté je provedeno testování a výběr vhodných technologií, který podléhá stanovené metodice v kapitole 3.2.

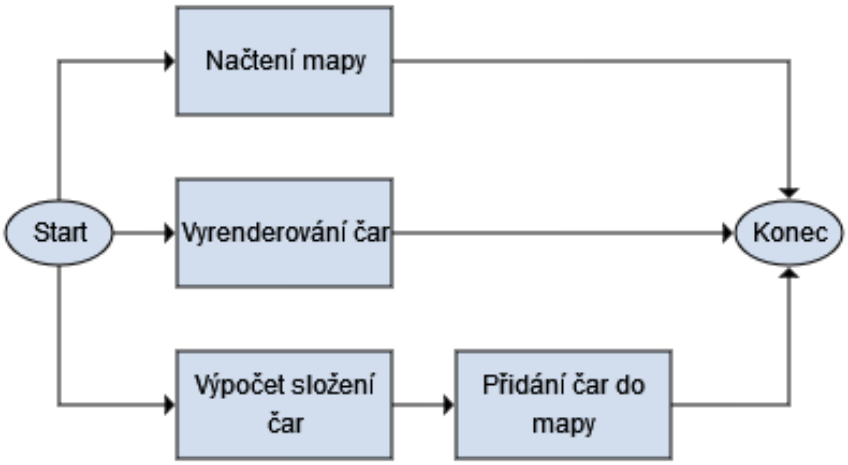
4.1 Analýza

4.1.1 Vybrané knihovny

Z požadavků vyplývá, že je nutné hledat řešení, které lze naprogramovat v JavaScriptu. V této sekci jsou rozebrány základní aspekty knihoven Leaflet, OpenLayers a Mapbox GL, které jsou momentálně nejpoužívanější a nejaktuálnější [15].

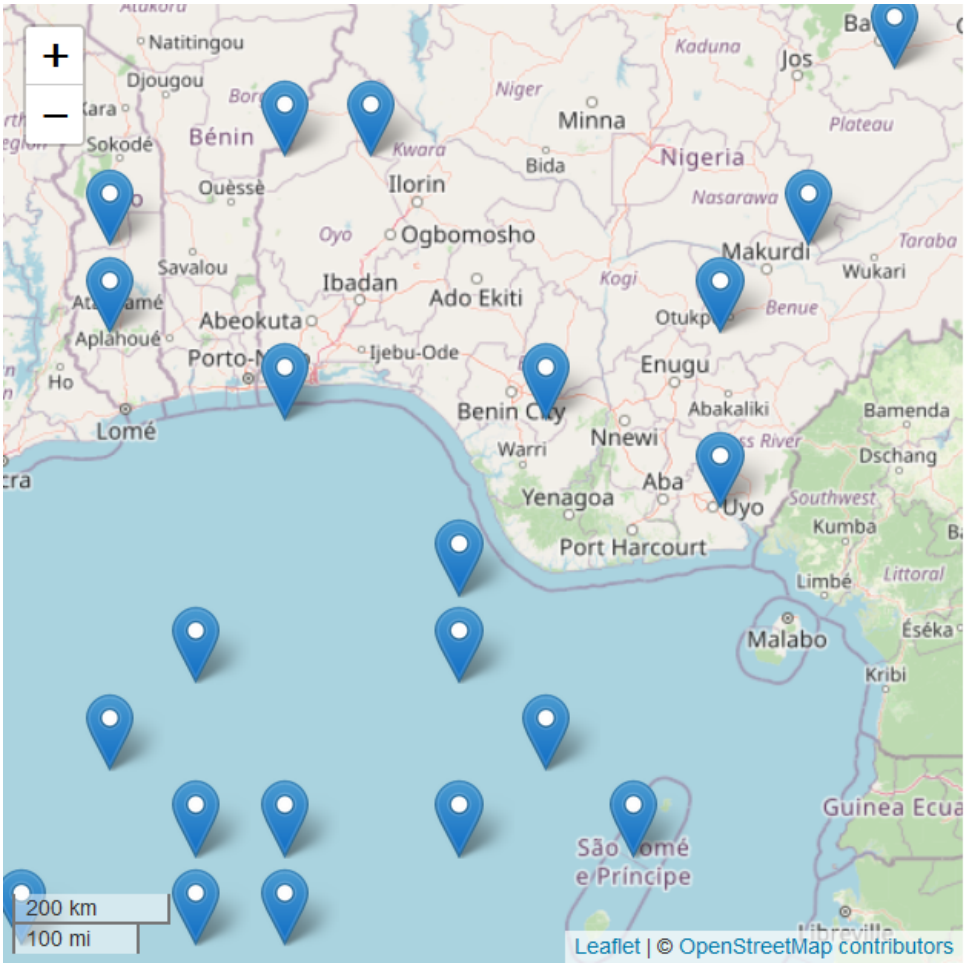
Leaflet

V Leafletu není nutné čekat na načtení mapy a je možné tedy provádět výpočty a vkládat objekty do mapy ještě před jejím načtením. Také lze vkládat do mapy objekty po jednom, tudíž načtení mapy, vytvoření objektů a vykreslení objektů lze provádět paralelně. Proces přidání objektů je vyobrazen na vývojovém diagramu 4.1.



Obrázek 4.1: Vývojový diagram reprezentující tvorbu objektů v knihovně Leaflet.

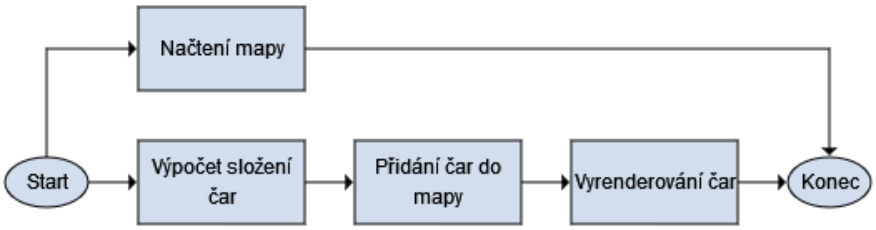
Markery v Leafletu jsou jednoduché na implementaci, ale nedostatečně upravitelné pro potřeby naší práce. Nelze jednoduše měnit barvu, jen změnit ikonu jako celek. Výchozí markery jsou poměrné velké, takže by mohlo dojít ke ztrátě informace při velkém oddálení. Příklad Leaflet Markeru můžete vidět na obrázku 4.2.



Obrázek 4.2: Leaflet markery vykreslené na mapě.

OpenLayers

OpenLayers přistupuje k načítání mapy stejně jako Leaflet tedy není potřeba čekat na načtení mapových podkladů pro vkládání objektů do mapy. Objekty se před vložením do mapy musí vložit do vektorové vrstvy. Tudíž z tohoto důvodu není možné vkládat objekty jednotlivě, ale je potřeba je vložit nejdříve do vrstvy a poté celou vrstvu do mapy. Pokud by bylo potřeba vkládat objekty jednotlivě, tedy každý do jeho vlastní vrstvy, bylo by vytvořeno tolik vrstev, kolik je objektů a nevystačila by paměť prohlížeče, protože vektorové vrstvy jsou velice náročné na operační paměť. Tudíž v OpenLayers nelze přidávat objekty do mapy a zároveň je vykreslovat. Proces přidání objektů je vyobrazen na vývojovém diagramu 4.3.



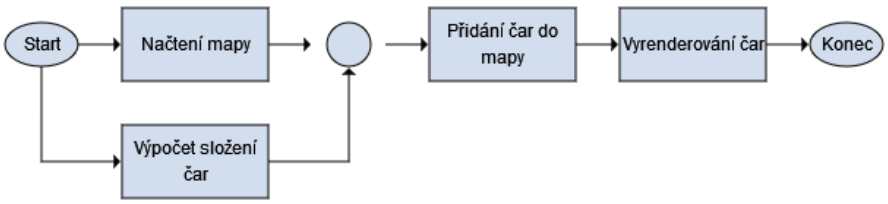
Obrázek 4.3: Vývojový diagram reprezentující tvorbu objektů v knihovně OpenLayers.

V nejnovější verzi OpenLayers je místo markerů objekt „Feature“, který plní funkci markeru. Objektu je možné téměř libovolně měnit styl a dokonce i jeho geometrický tvar. Na obrázku 4.4 je příklad OpenLayers Markerů.

Obrázek 4.4: OpenLayers markery vykreslené na mapě.

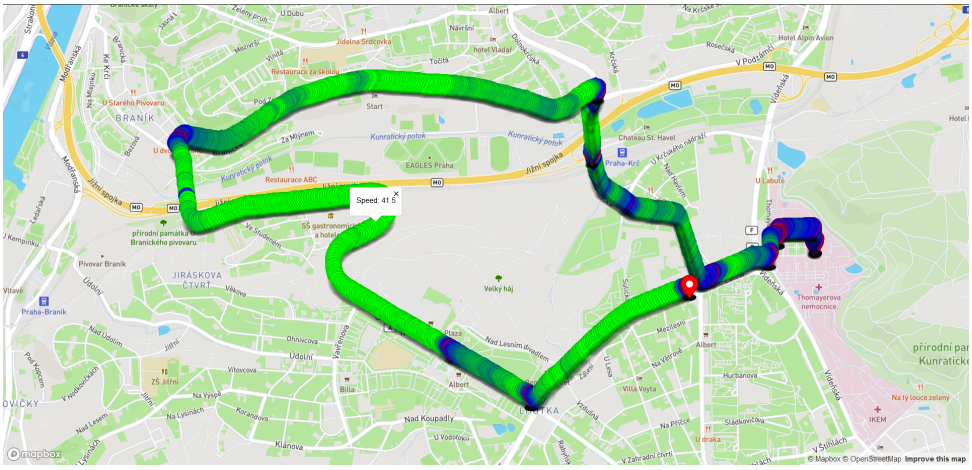
Mapbox GL

Mapbox vyžaduje načtení celé mapy, před jakoukoliv manipulací, protože až po načtení mapy budou i načteny základní styly, bez kterých nelze vkládat nic jiného než markery. Objekty lze do mapy vkládat, jak po jednom, tak jako pole více objektů. Přístup se liší podle typu objektu. Markery lze vkládat po jednom přímo do mapy, ale například čára se vkládá jako pole „Feature“. U markerů je přístup stejný jako u Leafletu, ale u čáry je možné pouze si připravit data a s jejich vykreslením počkat až na načtení celé mapy. Zobrazení čar je vyobrazeno na vývojovém diagramu 4.5.



Obrázek 4.5: Vývojový diagram reprezentující tvorbu objektů v knihovně Mapbox GL.

Mapbox markery umí to samé, co markery z Leafletu s tím rozdílem, že je možné jejich barvu a styl libovolně jednoduše měnit. Jejich implementace je zobrazena na obrázku 4.6.



Obrázek 4.6: Mapbox GL markery zobrazené na mapě.

4.2 Návrh

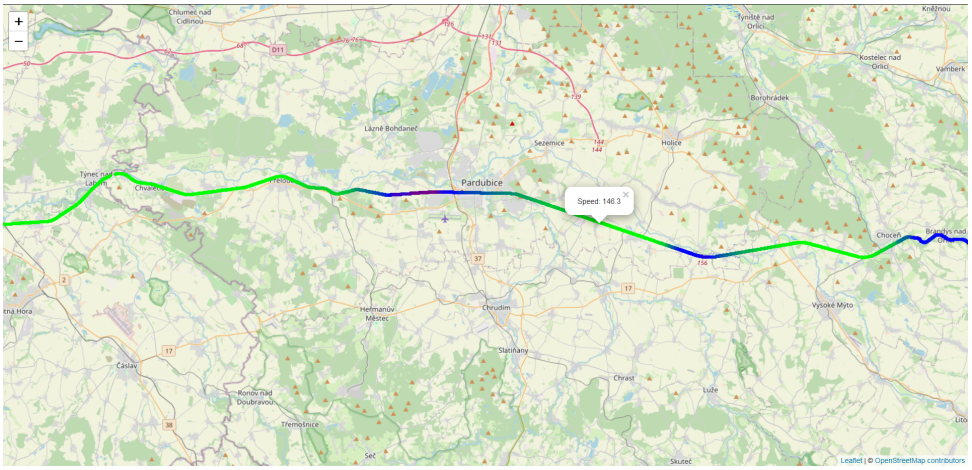
Cílem první fáze je otestovat schopnost knihoven zobrazit velké množství barevných objektů, Z analýzy vyplývá, že ne u všech knihoven je možné měnit barvu markerů, proto bylo rozhodnuto o použití barevné čáry. V první fázi nebyla řešena přehlednost vizualizace ani další požadavky.

4.3 Implementace

V první polovině této sekce jsou vysvětleny implementace základních prvků jednotlivých knihoven. V druhé je popsáno finální řešení ve vybraných knihovnách.

4.3.1 Leaflet

Objekt čáry má Leaflet nejlepší ze všech tří knihoven. Je možné čáru snadno přidat, upravit a navěsit na ní vyskakovací okno s informacemi. Výsledná čára je odolná proti chybám, jelikož se nevykresluje jako celek, ale po částech, pokud nastane neošetřený nedefinovaný stav, tak se nevykreslí pouze bod, ve kterém tento stav nastal. Příklad hotové implementace lze vidět na obrázku 4.7.



Obrázek 4.7: Výsledná vizualizace čáry v knihovně Leaflet.

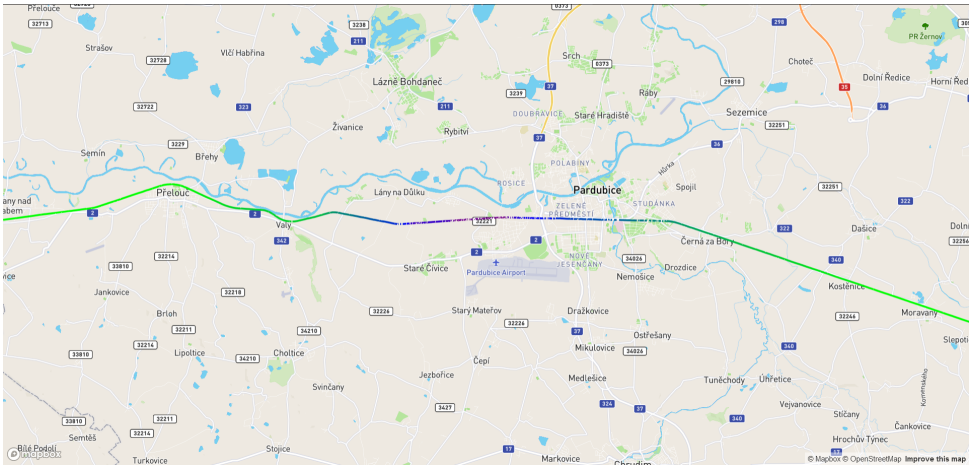
4.3.2 OpenLayers

Pro vytvoření čáry v knihovně OpenLayers je zapotřebí pouze změnit reprezentaci objektu Feature na geometrický objekt LineString a nastavit mu styl. Jelikož se všechny objekty vkládají do jedné vrstvy, jeden neošetřený chybný vstup způsobí, že se žádný objekt nevykreslí. Příklad hotové implementace lze vidět na obrázku 4.8.

Obrázek 4.8: Výsledná vizualizace čáry v knihovně OpenLayers.

4.3.3 Mapbox GL

Geometrické objekty mají v Mapboxu podobnou implementaci jako v OpenLayers a tedy je potřeba si nejdříve připravit Feature a ty poté přidat do vrstvy, která je vykreslena na mapu. Jako u OpenLayers, jeden neošetřený chybný vstup způsobí, že se nevykreslí žádný objekt. Příklad hotové implementace lze pozorovat na obrázku 4.9.



Obrázek 4.9: Výsledná vizualizace čáry v knihovně Mapbox GL.

4.4 Testování

Test byl proveden na notebooku s osmijádrovým procesorem AMD Ryzen 7 4800H 2.90 GHz, 16 GB RAM, grafickou kartou NVIDIA GeForce GTX 1660 Ti with Max-Q a na operačním systému Windows 11. Implementace byla testována v prohlížečích Mozilla Firefox verze 98.0.2, Microsoft Edge verze 99.0.1150.55 a Google Chrome verze 99.0.4844.84.

V první fázi testování byl proveden zátěžový test v knihovnách Leaflet, OpenLayers a Mapbox. Testování mělo za cíl otestovat, jak se budou knihovny chovat pro velké množství dat na různých prohlížečích. Byly implementovány jen základní funkce, jakými je filtrace a zobrazení dat pomocí barevné čáry, která je složitější na vykreslení a naprogramování než obyčejné body, aby bylo možné co nejefektivněji otestovat vybrané knihovny.

Pro první fázi bylo spojeno několik reálných měření zařízení F-Tester®. Data obsahovala cesty vlakem z Prahy do Blanska, z Kolína do Ústí nad Orlicí, ale také drobnější cesty po Praze, Brněnsku či Severní Moravě. Měření probíhalo v sekundovém intervalu. Záznamů v JSONu bylo 49808, neplatná data byla odfiltrována a na mapě nebyla zobrazena. Vizualizovanou veličinou byla rychlost pohybu.

Celý skript využívající Leaflet byl v průměru dokončen 552 ms v prohlížeči Chrome, což je velké zlepšení oproti AS-IS stavu. Takto nízký čas je způsoben tím, že skript a přidávání objektů do mapy se děje paralelně s načítáním mapy, proto Leaflet není omezován podklady a dokáže přidávat data i na

„prázdnou“ mapu. Při minimálním zoomu je mapa hůře ovladatelná, ale při přiblížení se odezva výrazně zlepší. Z hlediska čitelnosti kódu je Leaflet ze všech tří knihoven nejčitelnější. Přidání a změna stylu čáry je provedena jedním příkazem a kód je proto přehledný a snadno rozšiřitelný jiným uživatelem.

OpenLayers zvládl průměrně provést skript v prohlížeči Chrome 415 ms, což je také masivní zlepšení a oproti Leafletu je v tomto objemu dat dokonce rychlejší. Odezva je podobná jako u Leafletu, při oddálení byl pomalejší, ale při přiblížení byla odezva opět v normě. Jeho vkládání čar je nepraktické. Místo jednoho příkazu se musí vytvořit pole objektů (Feature), kde s každou Feature je zapotřebí vytvořit objekt Style a v něm objekty Fill a Stroke. Pole Feature se vloží do nového objektu source.Vector a z něho se vytvoří layer.Vector, který se přidá do mapy. Implementace je tak pracnější a hůře čitelná (místo jednoho příkazu je jich použito osm).

MapboxGL je jediná placená knihovna, která byla podrobena testování. Z tohoto důvodu lze očekávat, že bude lepší než ostatní knihovny. Už při spuštění skriptu nad testovacími daty je možné si povšimnout, že data byla vykreslena později. Přesněji v průměru za 550 ms v prohlížeči Chrome. Tento čas je způsoben tím, že Mapbox musí čekat na načtení mapy, než do ní začne vkládat objekty. S pomalejším nebo méně kvalitním internetem se dokonce stává, že se celý skript rozpadne kvůli dlouhému načítáni map (tento problém by byl vyřešen stažením offline map). Další chybou, která je vidět ihned po prvním načtení stránky je, že polovina čar není vykreslena. Tento jev závisí na velikosti přiblížení. Při větším oddálení se ztrácí většina čar, a naopak při přiblížení se opět zobrazují. Podle mého názoru to je způsobeno špatným zaokrouhlováním pozic na mapě, a proto Mapbox neumí vykreslit body, které jsou blízko sebe, resp. se překrývají. Nicméně to je jen domněnka a její potvrzení, či vyvrácení není náplní této práce. Implementace čar v Mapboxu je přehledná jako u knihovny Leaflet, ale výkon zůstává daleko za výše uvedenými knihovnami.

	Leaflet	OpenLayers	Mapbox
Firefox	710 ms	669 ms	1110 ms
Google Chrome	552 ms	420 ms	545 ms
Microsoft Edge	739 ms	415 ms	550 ms
Odezva	Ano	Ano	Ano
Přehlednost kódu	4/5	2/5	3/5
Licence	BSD	BSD	Placený

Tabulka 4.1: Výsledné hodnocení první testovací fáze.

4.4.1 Shrnutí

Z provedeného testování vyplývá, že Mapbox, ačkoliv je placený, nedisponuje žádnými pro nás nepostradatelnými funkcemi a může být plně nahrazen open-source knihovnami. Výběr Mapboxu by dokonce byl nešťasný, jelikož by bylo snadné přehlédnou klíčová data, protože některé zobrazené čáry při

menším přiblížení mizí. Leaflet a OpenLayers naplňují zadané požadavky. Co se týče klíčových parametrů, působily obě knihovny vyrovnaně. Pro vybrání vhodné knihovny bude zapotřebí implementace pokročilejších funkcí.

Tabulka 4.1 zachycuje kompletní výsledky první fáze testování a mé osobní hodnocení daných knihoven založené na zkušenostech z implementace. Přehlednost kódu je pro odlišení hodnocena na stupnici od 1 (nejhorší) do 5 (nejlepší), kdy známkou 3 byla označena knihovna, která byla svojí implementací mezi nejhorší a nejlepší knihovnou.

Zajímavostí je, že je znatelný rozdíl mezi JavaScript engine. Firefox, který využívá SpiderMonkey[21], je pomalejší než prohlížeče využívající engine V8[22].

Kapitola 5
Výběr vizualizace

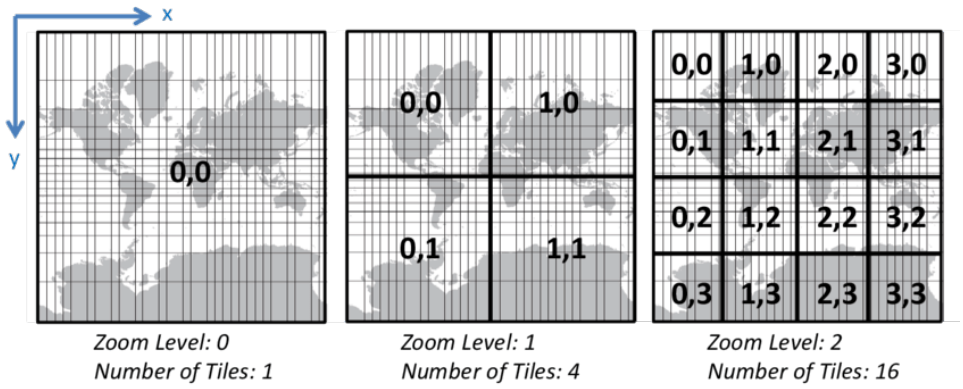
V kapitole je provedena analýza mapových podkladů a algoritmů pro mapování hodnot na barevné spektrum, vybrána reprezentace barevných objektů, vytvořen návrh mapování dat na barevnou škálu a je navrženo řešení offline mapových podkladů. Poté jsou ve zvolených knihovnách implementovány funkcionality splňující požadavky finálního řešení. Na závěr je provedeno testování a výběr nejvhodnějšího řešení, který podléhá metodice popsané v kapitole 3.3.

5.1 Analýza

V této sekci je provedena analýza mapových podkladů a algoritmů pro výpočet barvy. Vybrané technologie nebyly znovu podrobeny analýze, jelikož jejich analýza byla provedena při jejich výběru v kapitole 4.1.1.

5.1.1 Mapové podklady

Z funkčních požadavků vyplývá, že je nutné použití offline mapových podkladů. Kvůli omezeným paměťovým zdrojům bylo rozhodnuto o vykreslení vlastních map z databáze OpenStreetMap, což je volně dostupná databáze geografických dat, která jsou poté vizualizována do různě vypadajících map. Databázi používá více jak osm miliónů uživatelů, včetně velkých korporací jako jsou Facebook, Amazon či Apple. Data jsou poskytována pod licencí Open Database License, která je umožňuje svobodně upravovat a používat. Díky tomu jsou data neustále vylepšována, jak dobrovolníky, tak velkými společnostmi [6]. Vykreslená mapa je rozdělena na čtverce, které jsou dále děleny do skupin podle přiblížení a souřadnic. Složení mapy můžete vidět na obrázku 5.1.



Obrázek 5.1: Schéma Google Map čtverců, které používají stejné dělení na čtverce jako OpenStreetMap. Převzato z [20]

Běžná služba poskytující mapy z databáze OpenStreetMap funguje tak, že přijímá požadavky na mapové čtverce. Po přijmu požadavku webový server prohledá mezipaměť, když nalezne požadovaný čtverec, pošle ho zpátky klientovi, pokud ne připojí se k vykreslovacímu serveru. Vykreslovací server má přístup k surovým datům a vykreslí nový čtverec, který je zaslán zpátky webovému serveru. Ten si tento čtverec uloží do mezipaměti a pošle klientovi. Vykreslovací server ukládá surová data do databáze, kterou můžeme napojit na nástroj osm2pgsql, který dokáže data synchronizovat s oficiální databází OpenStreeMap a tím zajistit, že vykreslené mapy obsahují aktuální data. Tato konfigurace je nicméně velmi náročná na zdroje. Pro obsluhu mapy města je zapotřebí 10 až 20 GB paměti, 4 GB mezipaměti a moderní dvou jádrový procesor[16].

5.2 Návrh

Jedním z hlavních cílů práce je nalézt způsob zobrazení dat naměřených platformou F-Tester®. V první fázi bylo provedeno testování pouze řešení používající objekt čáry. Ve druhé fázi jsou testovány i ostatní druhy zobrazení.

5.2.1 Výběr reprezentace

Ze současného řešení viz obrázek 2.2.3 se nabízí zobrazení pomocí tvaru obdélníku, které je snadno viditelné a nepřehlédnutelné. Obdélník může ale být zbytečně vysoký. Data, která jsou sbírána, se váží k určité souřadnici na mapě a obdélník zasahuje do větší plochy a při větší hustotě se stává zobrazení nepřehledné. Dalším úkolem je nalezení řešení, které je co nejvíce efektivní. Chceme tedy eliminovat vykreslování zbytečně velkých objektů a tím obdélník v porovnání například s kruhem jistě je.

Data jsou měřena a ukládána pro určité body na mapě. Okolo bodu můžeme sestrojit obarvený kruh. Tato vizualizace může být jednou z nejefektivnějších vizualizací, protože je možné vykreslit opravdové minimum.

Hotová měření lze chápat i jako cestu z bodu `A` do bodu `B`. Díky tomu, že měření lze chápat jako cestu, můžeme data reprezentovat objektem čáry, která bude měnit barvu. Oproti vizualizaci pomocí bodu získáme i přehled o tom, jak na sebe data navazují.

5.2.2 Výběr legendy

Ze současného řešení a povahy dat je patrné, že data lze namapovat na barevnou škálu. Bude použita sekvenční paleta, která nejlépe pracuje s kontinuálními daty [3].

Použit bude HSL model, který se skládá ze tří hodnot: odstínu, sytosti barvy a hodnoty jasu. Jelikož bylo možné sytost a jas zafixovat, k výpočtu zbývá pouze odstín, který je reprezentován úhlem na barevném spektru. Tudíž lze snadno namapovat na jakékoliv hodnoty a lze z tohoto spektra udělat výseč.

Toto barevné odlišení, dle mého názoru, stačí, aby uživatel snadno zjistil, zda jsou data v normě. Naopak další vlastnost současného řešení - změna velikosti - se ve spojení s barevným odlišením jeví jako redundantní a pro velké množství dat nepřehlednou. Toto odlišení nebude v novém řešení implementováno.

5.2.3 Mapování zobrazovaných dat na barevnou škálu

Pro všechna data nemůžeme použít stejné mapování na barevnou škálu. V této sekci jsou nastíněné barevné rozsahy některých měřených hodnot.

Síla signálu

Po dohodě s vedoucím bylo rozhodnuto o mapování parametrů signálu, protože jeho hodnoty jsou nejlépe popsané a vyskytují se na většině naměřených dat. Zobrazovaná barevná škála odpovídá hodnotám v tabulce 5.1, která vychází z obecně používaných rozsahů při vizualizaci parametrů mobilní sítě [5].

	RSRP	RSRQ	SNR
Zelená	>= -80	>= -10	>= 20
Žlutá	okolo -90	okolo -20	okolo 13
Červená	<= -100	<= -20	<= 0

Tabulka 5.1: Tabulka mapování hodnot na barevnou škálu.

Z tabulky 5.1 je patrné, že všechny hodnoty jsou typu Integer případně Float a není zapotřebí řešit složitější datové struktury.

5.2.4 Mapové podklady

Z informací ze sekce 5.1.1 vyplývá, že využití vlastních mapových podkladů je zdrojově velmi náročný proces. Jelikož jsou zdroje zařízení F-Tester® omezené, je zapotřebí tento proces zjednodušit.

Po konzultaci s vedoucím bylo rozhodnuto, že pro naše potřeby není nutné mít data neustále synchronizována. Je možné si mapové čtverce před vykreslit a pouze je uložit na webový server. Není nutné tedy používat databázi ani vykreslovací server.

Mapové čtverce byly vykresleny jen pro Českou republiku, do přiblížení 11. Tímto byla omezena velikost na 207 MB. V obou knihovnách bylo nastaveno minimální přiblížení pro načtení nových čtverců. Knihovny při překročení tohoto přiblížení přestanou požadovat nové čtverce a stávající se pouze roztáhnou.

Jako webový server byl použit Apache. Data jsou ukládána ve struktuře "ZXY", kterou dokáží využít všechny výše zmíněné knihovny. Když na webový server přijde následující HTTP požadavek "http://somedomain.com/z/x/y.png", kde z je úroveň přiblížení a x s y jsou souřadnice čtverce, tak server odešle zpět na klienta odpovídající čtverec.

5.3 Implementace

Z první fáze byly vybrány knihovny Leaflet a OpenLayers. V těchto knihovnách jsou implementovány pokročilé funkce splňující minimální požadavky zadání.

V obou knihovnách je vytvořeno řešení využívající zobrazení dat pomocí barevné čáry a bodů. Každé toto řešení obsahuje vyskakovací okno, které se zobrazí po kliknutí na objekt čáry či bodu a vyznačení vybraného objektu pro která jsou zobrazována data.

5.3.1 Algoritmus pro výpočet barvy

K výpočtu HSL hodnot je použito jednoduché a snadno rozšířitelné mapování odpovídající tabulce 5.1. Jediným problémem je, že Leaflet ani OpenLayers neumí pracovat s HSL modelem, a proto je zapotřebí výsledek převést do RGB. K tomu je použit upravený algoritmus z knihovny w3color.js [24]. Výsledný algoritmus je k vidění v kódu 5.1.

5.3.2 Leaflet

Leaflet dokáže přiřadit k objektu vyskakovací okno už při jeho inicializaci. Při najetí myši nad zobrazený objekt se změní kurzor na symbol ruky, který značí, že lze kliknutím zobrazit vyskakovací okno. Do vyskakovacího okna lze vkládat i HTML značky a měnit obsah pomocí CSS či JavaScriptu.

Jak bylo zmíněno v analýze možných řešení, tak implementace v knihovně Leaflet je intuitivní a oproti knihovně OpenLayers jednoduchá. Na algoritmu 5.2 je možné vidět implementaci čáry s předem vypočítanou barvou. Hlavní výhodou je, že už při inicializaci čáry je vytvořeno vyskakovací okno a styl dále je čára vložena do vrstvy, která je vykreslena na mapu.

Leaflet sám řeší zobrazení vyskakovacího okna a jeho zavření. Při najetí kurzorem nad objekt se změní jeho reprezentace z šipky na ruku, což značí, že je k objektu přiřazeno vyskakovací okno a je možné ho rozkliknout. Toto

```
    function computeHSLColor(value, dataType){
        h = Math.floor(140 - (value) * 1.4);
        if (dataType == "snr"){
            h = (10 + value) * 4;
        }
        if (dataType == "rsrp"){
            h = (100 + value) * 7;
            if (h<0) h = 0;
            if (h>140) h =140;
        }
        if (dataType == "rsrq"){
            h = (20 + value)*12;
            if (h<0) h = 0;
            if (h>140) h =140;
        }
        return h;
    }

    function hueToRgb(t1, t2, hue) {
        if (hue < 0) hue += 6;
        if (hue >= 6) hue -= 6;
        if (hue < 1) return (t2 - t1) * hue + t1;
        else if(hue < 3) return t2;
        else if(hue < 4) return (t2 - t1) * (4 - hue) + t1;
        else return t1;
}

    function HSLtoRGB(hue){
        sat = 1;
        light = 0.5;

        var t1, t2, r, g, b;
        hue = hue / 60;
        if ( light <= 0.5 ) {
            t2 = light * (sat + 1);
        } else {
            t2 = light + sat - (light * sat);
        }
        t1 = light * 2 - t2;
        r = hueToRgb(t1, t2, hue + 2) * 255;
        g = hueToRgb(t1, t2, hue) * 255;
        b = hueToRgb(t1, t2, hue - 2) * 255;
        return "rgb("+r +", "+g +", "+b +")"
    }
```

Algoritmus 5.1: Funkce výpočtu barvy vizualizovaných objektů.

```
var polyline = L.polyline([
                          start,
                          end
                          ],{
                          color: "rgb("+r+", "+g+", "+b,
                          weight: 5,
                          lineJoin: 'round'
}).bindPopup("Snr: " + element.snr).addTo(snrLayer);
```

Algoritmus 5.2: Vytvoření objektu čáry, jeho vložení do vrstvy a přídání vyskakovacího okna v knihovně Leaflet.

```
var circle = L.circle(
start,
{
    color: "rgb("+r+", "+g+", "+b,
    weight: 5,
}
).bindPopup("Snr: " + element.snr).addTo(snrLayer);
```

Algoritmus 5.3: Vytvoření bodu, jeho vložení do vrstvy a přídání vyskakovacího okna v knihovně Leaflet.

může pomoci uživateli poznat, ke kterému místu na mapě se zobrazovaná data vztahují.

Implementace vizualizace pomocí bodu je stejná, jako implementace čáry. Liší se pouze počtem souřadnic, jelikož se jedná o bod, tak jsou zadány pouze jedny souřadnice, a použitím objektu L.circle. Ukázku této implementace lze vidět na algoritmu 5.3.

5.3.3 OpenLayers

OpenLayers nedisponuje výše zmíněnými funkcemi, tudíž bylo zapotřebí je naprogramovat. Těmito funkcemi byla implementace vyskakovacího okna na mapě, které zobrazovalo data náležící místu kliku a indikace možnosti otevření okna při najetí na objekt zobrazený na mapě.

Na algoritmu 5.4 je možné vidět implementaci čáry v knihovně OpenLayers. Po vytvoření nového objektu je nutné vytvořit styl a poté tento styl přidat. Následně je přidán objekt do pole objektů, ze kterého je vytvořen vektor. Tento vektor je přidán do vrstvy, která je vykreslena na mapu. Oproti Leafletu je toto méně intuitivní a více náchylné na chyby, jelikož pokud se jeden objekt špatně vytvoří (je nedefinovaný), tak nebude vykreslený žádný objekt z daného vektoru.

K obarvení bodů je nutné změnit zobrazení bodů, protože objektu Point nelze měnit barvu. Z tohoto důvodu je zapotřebí vytvořit objekt RegularShape, který definoval tvar, jímž byla nahrazena výchozí reprezentace. Tomuto tvaru je nutné, stejným způsobem jako u čáry, definovat barvu, dále počet vrcholů, rádius a úhel pod kterým má být výsledný objekt natočený. Ukázka této implementace je na algoritmu 5.5.

```
var snrfeature = new ol.Feature({
        geometry: new ol.geom.LineString([start,end]),
});
var style = new ol.style.Style({
        stroke: new ol.style.Stroke({
            color: "rgb("+r+", "+g+", "+b+")",
            width: 3,
        }),
            fill: new ol.style.Fill({
            color: "rgb("+r+", "+g+", "+b+")",
        }),
});
snrfeature.setStyle(style);
snr.push(snrfeature);

var vectorSource2 = new ol.source.Vector({
    features: snr
});

snrLayer = new ol.layer.Vector({
    source: vectorSource2,
});

map.addLayer(snrLayer);
```

Algoritmus 5.4: Vytvoření objektu čáry a její vložení do vrstvy v knihovně OpenLayers.

```
const style = new ol.style.Style({
    image: new ol.style.RegularShape({
        stroke: new ol.style.Stroke({
            color: "rgb("+r+", "+g+", "+b+")",
            width: 5,
        }),
        fill:  new ol.style.Fill({
            color: "rgb("+r+", "+g+", "+b+")"
        }),
        points: 4,
        radius: 1,
        angle: 1,
    })
});
var snrfeature = new ol.Feature({
    geometry: new ol.geom.Point(coord),
});
snrfeature.setStyle(style);
```

Algoritmus 5.5: Vytvoření objektu bodu v knihovně OpenLayers.

Zvětšení aktivního objektu

Algoritmus zvětšení aktivního objektu probíhá tak, že je zapotřebí vytvořit funkci, která naslouchá, kdy se na mapě pohne kurzor a vypočítá, který objekt je nejblíže od nových souřadnic kurzoru.

Výpočet spočívá v tom, že byla vypočítána vzdálenost v kilometrech mezi GPS souřadnicemi objektů a GPS souřadnicemi myši. Index objektu s nejmenší hodnotou je nejbližší objekt. Použitým vzorcem byl Haversinův vzorec. Algoritmus 1 je přepisem Haversinova vzorce do podoby algoritmu [23].

Algoritmus 1 Haversinův vzorec v podobě algoritmu

```
dlat = (lat2-lat1) * PI / 180
dlon = (lon2-lon1) * PI / 180
lat1 = lat1 * PI / 180
lon1 = lon1 * PI / 180
a = sin(dLat/2) * sin(dLat/2) + sin(dLon/2) * sin(dLon/2) * cos(lat1) *
cos(lat2);
c = 2 * atan2(sqrt(a), sqrt(1-a))
return  6371 * c
```

Po nalezení objektu s nejmenší hodnotu je index této hodnoty uložen. Jelikož v OpenLayers nelze objektům zvětšit pouze velikost, je nutné vytvořit nový styl a použít ho k přepsání stylu vybraného objektu. Je vypočítána barva pro zvolený objekt, zvětšena velikost a je objektu nastaven tento styl. Stejný postup je potřeba zvolit i při zpětném zmenšování. Uložený index nejmenšího prvku je použit ke zpětnému zmenšení objektu a pro zobrazení vyskakovacího okna. Ukázka zvětšeného objektu na 5.2.

Obrázek 5.2: Zvětšení nejbližšího objektu s vyskakovacím oknem od kurzoru myši v knihovně OpenLayers.

```
1  map.on('singleclick', function (evt) {
2    content.innerHTML = '<p> Rsrq je: ' + arrPopups[
       closestFeatureIndex].rsrq + '</p>';
3    content.innerHTML = content.innerHTML + '<p> Snr je: ' +
       arrPopups[closestFeatureIndex].snr + '</p>';
4    var coord = [arrPopups[closestFeatureIndex].x,arrPopups[
       closestFeatureIndex].y];
5              var coordinate = [ol.proj.fromLonLat(coord)[0],ol.
       proj.fromLonLat(coord)[1]];
6
7    overlay.setPosition(coordinate);
8         });
```

Algoritmus 5.6: Zobrazení vyskakovacího okna v knihovně OpenLayers.

Zobrazení popupu

K vytvoření vyskakovacího okna je zapotřebí vytvořit tři tagy, jeden pro kontejner do kterého je vkládán obsah a "tlačítko", které vyskakovací okno zavře. K zobrazení vyskakovacího okna je potřeba vytvořit objekt overlay, do kterého je nutné vložit identifikátor kontejneru a poté vytvořený overlay vložit do kontruktoru mapy. Zobrazení vyskakovacího okna probíhá tím, že se mu nastaví poloha, kde se má zobrazit. Pro schování vyskakovacího okna je zapotřebí nastavit pozici na null.

Na kódu 5.6 je zobrazena funkce odchytávající singleclick událost. Funkce naplní obsah vyskakovacího okna daty a poté přepočítá souřadnice a nastaví je objektu overlay, který zobrazí okno na daných souřadnicích. K obsahu okna lze kdykoliv přidávat další text. Stojí za povšimnutí, že je možné vkládat HTML značky, tudíž lze tomuto obsahu libovolně měnit styl pomocí CSS.

5.3.4 Přepnutí vrstev

OpenLayers ani Leaflet nedisponují přepínačem vrstev, který by bylo možné jednoduše vložit do mapy, proto je pro zjednodušení implementován přepínač vrstev vně mapy. Při prvním načtení se pro každý typ zobrazovaných dat vytvoří vrstva s objekty, které lze pak vykreslit na mapu. Tyto vrstvy jsou vloženy do pole a s tímto polem je ukládán index aktivní vrstvy pro jednoduché přepínání, protože při odebírání vrstvy z mapy je nutné v obou knihovnách vložit referenci vrstvy, která má být odebrána.

V HTML je vytvořený seznam, který má indexy stejné jako mají vrstvy v poli. Po výběru požadované vrstvy a zmáčknutí tlačítka je odebrána aktuální vrstva z mapy a je zobrazena se nově vybraná. Tímto je vytvořený jednoduchý přepínač vrstev.

5.3.5 Shrnutí

Zde je také možné vidět, Leaflet je uživatelsky přívětivější a jednodušší na pochopení. U OpenLayers je křivka učení pomalejší a většinu funkcí nechává na kreativitě vývojáře. Což může být velké plus, protože vývojář není zatíženi

věcmi, které nemusí nikdy použít. Na druhou stranu je vývoj v Leafletu díky tomu rychlejší a tyto před připravené funkcionality jsou optimalizovány.

5.4 Testování

Druhé kolo testování probíhalo po celou dobu implementační fáze. Při testování byl kladen důraz na jednoduchost implementace, přehlednost způsobů vizualizace, rychlost prvotního načtení a odezva mapy při prohlížení velkého množství dat a paměťové nároky.

Pro testování je, oproti prvnímu kolu, použit nový JSON ze zařízení F-Tester®, který zahrnuje reálná měření veličin 5G signálu. V souboru je 31 877 měření. Tato měření byla vložena do tří vrstev, podle veličin (SNR, RSRQ, RSRP). Byly použity stažené mapové podklady, pro zajištění prostředí podobné tomu, které bude použito v produkci. Výchozí úroveň přiblížení byla nastavena na 9.

Na začátku testování bylo zjištěno, že implementace bodu v knihovně OpenLayers zabírá velké množství RAM paměti a kvůli tomu nebylo možné dokončit skript. Je vytvořeno alternativní řešení využívající objekt LineString, kde je začátek roven konci a tím vznikl bod.

5.4.1 Přehlednost vizualizace

Při porovnání vizualizací si lze povšimnout, že reprezentace pomocí obdélníků je méně přehledná a více náročnější než řešení pomocí čar nebo bodů. Barevné čáry a zvláště body dokáží zřetelněji předat informaci o jaká data se jedná a tudíž jsou ideálními kandidáty.

Čára se může zdát více uživatelsky příjemnější, neboť lze snadno vidět přechod mezi různými úrovněmi dat. Body, ale dokáží lépe vystihnout místo, kde došlo např. k výpadku. Čára je dlouhá podle rychlosti sběru dat, tudíž není přesně (jako u bodu) určeno, kde k výpadku došlo.

Z výše uvedených důvodů je možné se domnívat, že bod je ideálnější pro vizualizaci než ostatní způsoby.

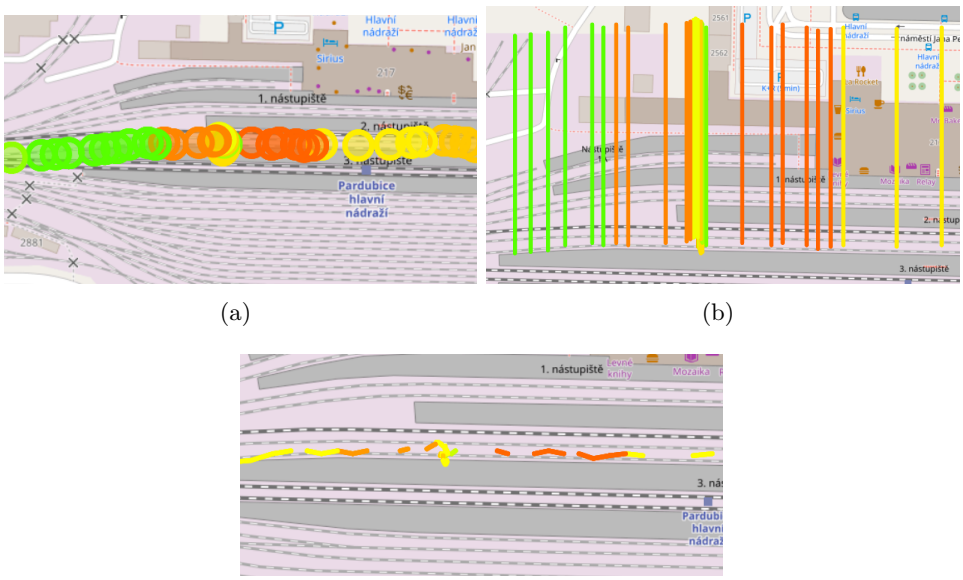
Na obrázku 5.3 je vytvořené porovnání vizualizací pomocí knihovny Leaflet v nejhustším místě měření.

Knihovna Leaflet při změně přiblížení zřetelně přepočítává a mění velikost objektů. Důsledkem je, že například reprezentace čáry je přehledná, jak pro velké přiblížení, jako je zobrazeno na obrázku 5.3, tak i při oddálení.

Pro výběr mezi vizualizací pomocí bodu nebo čáry musel být osloven vedoucí, který určil, která reprezentace je nejpřehlednější a vyhrálo řešení využívající body s průhledným středem viz 5.3a.

5.4.2 Rychlost načtení

Z úvodní testovací fáze vyplynulo, že doba načtení obou knihoven pro velká množství dat byla řádově nižší než u současného řešení. Nyní bude měřeno, jak se změní tato doba s přidáním více vrstev a vyskakovacích oken. Rychlost



(a) (b)

(c)

Obrázek 5.3: Porovnání vizualizací v knihovně Leaflet: (a) Zobrazení pomocí bodu. (b) Zobrazení pomocí obdélníku. (c) Zobrazení pomocí čáry.

načtení byla měřena desetkrát a poté byl vypočítán průměr, který byl použit v tabulce 5.2.

Značně se, mezi knihovnami, liší směrodatná odchylka rychlosti načtení. Například řešení pomocí bodu v knihovně Leaflet mělo směrodatnou odchylku 23,49 a tatáž reprezentace v knihovně OpenLayers měla směrodatnou odchylku 99,85 (měřeno v prohlížeči Chrome). Směrodatné odchylky v prohlížeči Edge a vizualizace pomocí čáry byly 84,45 pro Leaflet a 177,31 pro OpenLayers. Z krabicového grafu na obrázku 5.4 lze vyčíst, že Leaflet má menší rozptyl hodnot, než OpenLayers.

Z výše zmíněných skutečností je možné usoudit, že knihovna Leaflet má konzistentnější čas načtení, než OpenLayers.

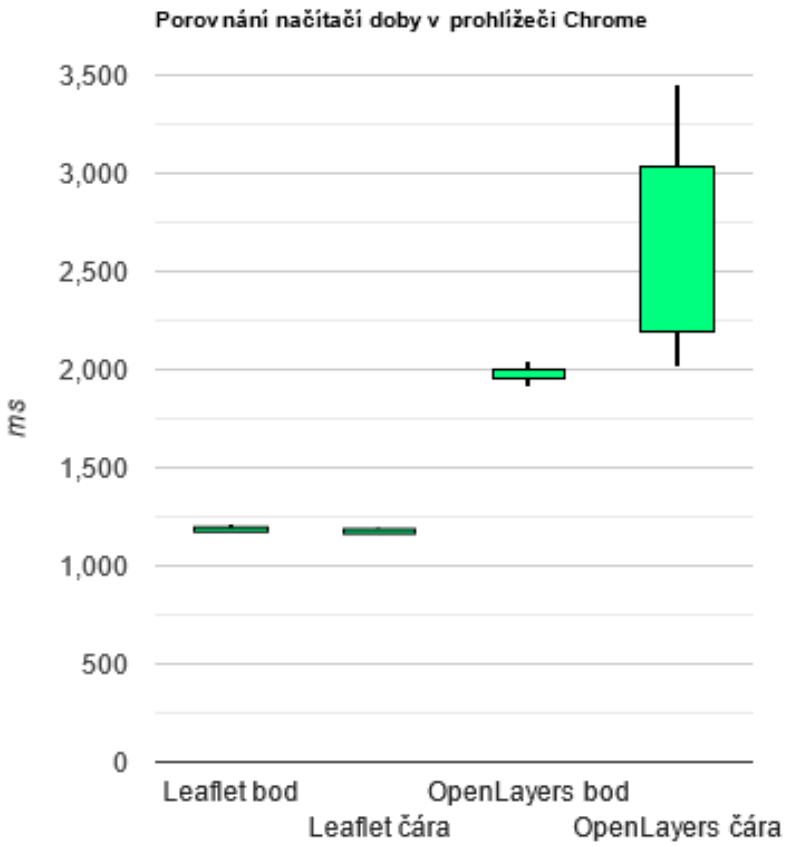
Každá z knihoven má jiný způsob načítání mapy a zobrazení dat. Leaflet nejprve zobrazí mapu. Zároveň, ale provádí výpočet a vytváří objekty vkládané do mapy. Po načtení mapy ihned vloží vytvořenou vrstvu do mapy.

OpenLayers nejprve vypočítá data, která následně zobrazí. Až po zobrazení dat se začne načítat mapa. Důsledkem je, že pro velký objem dat, vidíme u OpenLayers pouze šedou zónu, naopak Leaflet ihned začne načítat mapu a uživatel takto pozná, že aplikace pracuje a například nespadla.

Z tabulky 5.2 je patrné, že i s implementací vyskakovacích oken a více vrstev se dostaneme pod hranici současného řešení. Knihovna Leaflet předčila OpenLayers. Také se potvrdilo, že prohlížeče využívající V8 pracují s knihovnami efektivněji než Firefox využívající SpiderMonkey.

Knihovna	Leaflet		OpenLayers	
Implementace	Body	Čára	Body	Čára
Firefox	1733 ms	1466 ms	3680 ms	4100 ms
Google Chrome	1206 ms	1206 ms	1939 ms	2712 ms
Microsoft Edge	1271 ms	1625 ms	1795 ms	2029 ms

Tabulka 5.2: Výsledky měření rychlosti načtení pro různé implementace.



Obrázek 5.4: Krabicový graf doby načtení pro všechny druhy vizualizace v prohlížeči Chrome.

5.4.3 Odezva mapy

U Leafletu i OpenLayers je odezva mapy, při maximální oddálení, poněkud trhavá, ale nezamezuje interaktivitě. Po přiblížení je odezva opět plynulá.

Při porovnání knihoven je Leaflet plynulejší než OpenLayers. Obě knihovny pracují lépe v prohlížeči Chrome a Edge než v prohlížeči Firefox.

5.4.4 Paměťové nároky

Paměťové nároky celkového řešení se oproti standardnímu využití podařilo několikanásobně snížit. Jsou použity odlehčené mapy a služba, která tyto mapy obstarává je také zredukována na přístupné minimum. Samotné knihovny nezabírají na disku místo.

Nicméně implementace bodu v knihovně OpenLayers není optimální vzhledem k RAM paměti. Ta při tomto objemu dat sahá až k jednotkám gigabajtů a takové řešení není proveditelné.

5.4.5 Shrnutí testování

Z testování vyplývá, že nejvhodnější vizualizací je reprezentace pomocí bodu s průhledným středem umožňující rozeznatelnost hustých dat.

Ve všech testech, ač někde mírně, vítězila knihovna Leaflet. Implementace byla intuitivnější a přehlednější, což otevírá možnost jednoduché rozšířitelnosti kódu novými vývojáři. Dobu načtení má Leaflet rychlejší ve všech vybraných prohlížečích a je konzistentnější. Způsob, jakým Leaflet provádí načítání, je uživatelsky příjemnější než přístup OpenLayers. Posledním důvodem jsou vlastnosti implementace bodu. Pro velké množství dat je standardní řešení v knihovně OpenLayers příliš náročné na RAM paměť a proto je zapotřebí využít alternativu. U Leafletu je pro bod možné použít standardizované řešení.

Ze všech výše zmíněných důvodů je Leaflet lepším kandidátem než OpenLayers, protože splňuje všechny požadavky.

Kapitola 6

Závěr

Cílem práce bylo analyzovat data sbíraná zařízením F-Tester®, navrhnout vhodný formát vizualizace a implementovat jej.

Nejprve proběhla analýza měřící platformy F-Tester®a způsobů vizualizace dat do map. Z této analýzy byly sesbírány požadavky a vytvořena metodika, podle které probíhal výběr konečného řešení. Byly definovány dvě fáze: „Výběr technologie“ a „Výběr vizualizace“.

V rámci výběru technologie byly analyzovány přístupy pro tvorbu interaktivních map a jejich napojení na současné řešení využívané zařízením F-Tester®. Poté byl proveden návrh a implementace základních funkcionalit v perspektivních technologiích. Následně bylo provedeno testování jejímž výstupem byli dva vhodní kandidáti: OpenLayers a Leaflet.

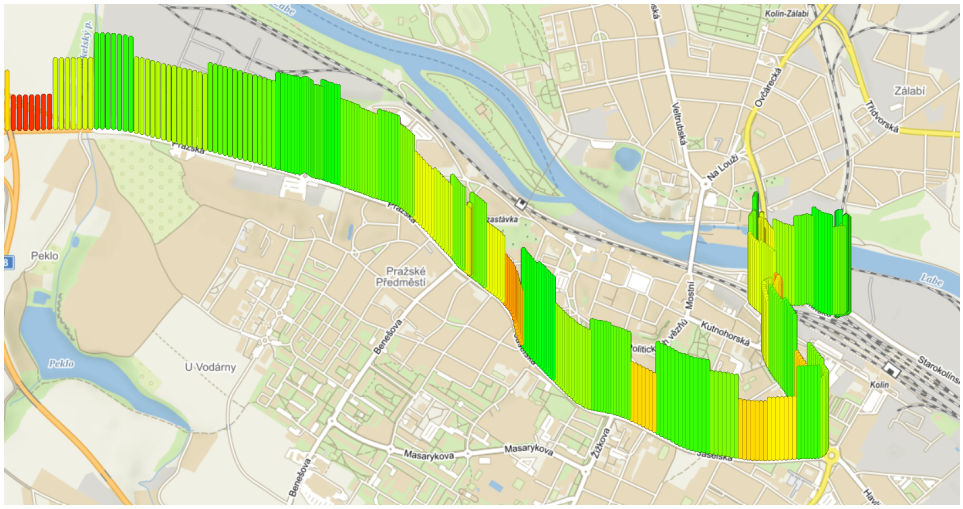
Při výběru vizualizace bylo cílem vytvořit plnohodnotné řešení. Byly analyzovány mapové podklady a proveden návrh vhodných legend, reprezentace dat, mapování měřených dat na barevné spektrum a služby pro mapové podklady. Následně byla provedena implementace dle návrhu. V knihovně OpenLayers bylo zapotřebí naprogramovat onClick funkce pro zobrazení vyskakovacích oken a pro uživatelský komfort bylo implementováno zvětšení aktivního objektu. Nakonec bylo provedeno výkonnostní testování se všemi kandidáty.

Vybraným způsobem vizualizace byl barevný objekt bodu v knihovně Leaflet, který byl vybrán jako nejpřehlednější reprezentace. Leaflet konzistentně zvládal vykreslit velké množství dat do 3s od načtení stránky, byl autonomní, interaktivní, s přijatelnou odezvou a implementace byla přehledná. Také z pohledu uživatele je Leaflet přívětivější než OpenLayers.

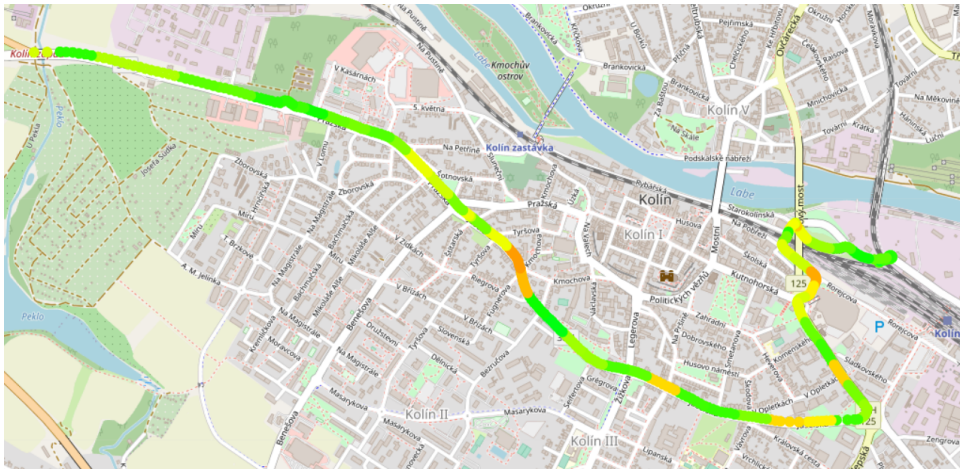
Na obrázku 6.1 je porovnání vizualizace stejné datové sady pomocí současného řešení a vybrané nové implementace. Pro tento dataset byl čas načtení mapy zlepšen z 2,5 s, pro současné řešení, na 0,5 s, pro řešení v knihovně Leaflet.

Vytvořený kód lze s menšími úpravami, zakomponovat do grafického rozhraní zařízení F-Tester®.

V práci se povedlo dosáhnout všech vytčených cílů definovaných v kapitole 1. Navržené řešení může sloužit jako solidní základ pro další úkoly spojené s vizualizací a rozšířením funkcionality celého zobrazovacího systému naměřených dat platformou F-Tester®. Jedním z možných rozšíření je vhodné zobrazení



(a)



(b)

Obrázek 6.1: (a) Současné řešení v knihovně Mapy.cz API/SDK. (b) Nové řešení v knihovně Leaflet.

grafu dat dle výběru plochy na mapě případně implementace návrhového vzoru Lazy loading pro velký objem dat, nebo implementace měřících čtverců dle metodiky ČTU pro měření mobilních sítí [4]. Velkou výzvu by také tvořila vizualizace v reálném čase při pohybu měřicího vozu.

Bibliografie

[1] Vladimir Agafonkin. *Leaflet overview*. URL: https://leafletjs.com/. (accessed: 08.01.2022).

[2] Vladimir Agafonkin. *Leaflet Plugins database*. URL: https://leafletjs.com/plugins.html. (accessed: 08.01.2022).

[3] Ihaka et al. *HCL-Based Color Palettes*. URL: https://colorspace.r-forge.r-project.org/articles/hcl_palettes.html. (accessed: 1.05.2022).

[4] Ihaka et al. *Metodika pro měření a vyhodnocení datových parametrů mobilních sítí elektronických komunikací*. URL: https://www.ctu.cz/sites/default/files/obsah/stranky/186507/soubory/metodikamerenidatovychparametruvmobilnichsitichv2.1.pdf. (accessed: 19.05.2022).

[5] Telco Antennas. *4G LTE Signal Strength Reference Guide*. URL: https://www.telcoantennas.com.au/blog/guide-to-mobile-networks/4g-lte-signal-strength-reference-guide/. (accessed: 1.05.2022).

[6] Corey Dickinson. *Inside the 'Wikipedia of Maps,' Tensions Grow Over Corporate Influence*. URL: https://www.bloomberg.com/news/articles/2021-02-19/openstreetmap-charts-a-controversial-new-direction. (accessed: 08.01.2022).

[7] Esri. *ArcGIS*. URL: https://www.esri.com/en-us/arcgis/about-arcgis/overview. (accessed: 26.04.2022).

[8] F-Tester. *Dokumentace*. URL: https://f-tester.fel.cvut.cz/technicka-podpora/dokumentace. (accessed: 08.01.2022).

[9] John Firebaugh. *Using Leaflet plugins with MapBox.js: A Showcase*. URL: https://blog.mapbox.com/using-leaflet-plugins-with-mapbox-js-a-showcase-ad60146044e1. (accessed: 08.01.2022).

[10] Geoapify. *Leaflet vs OpenLayers. What to choose?* URL: https://www.geoapify.com/leaflet-vs-openlayers. (accessed: 08.01.2022).

[11] hauke. *Welcome to the OpenWrt Project*. URL: https://openwrt.org/start. (accessed: 13.04.2022).

[12] Tom MacWright. *Leaflet Creator Vladimir Agafonkin Joins MapBox*. URL: https://blog.mapbox.com/leaflet-creator-vladimir-agafonkin-joins-mapbox-99cc83588a76. (accessed: 08.01.2022).

[13] Mapbox. *Mapbox GL JS*. URL: https://www.mapbox.com/mapbox-gljs. (accessed: 08.01.2022).

[14] Till Nagel. *Unfolding*. URL: http://unfoldingmaps.org. (accessed: 26.04.2022).

[15] OpenBase. *10 Best JavaScript Map Libraries*. URL: https://openbase.com/categories/js/best-javascript-map-libraries. (accessed: 11.01.2022).

[16] OpenStreetMap a contributors. *Serving Tiles*. URL: https://switch2osm.org/serving-tiles/. (accessed: 14.04.2022).

[17] Nataly Otairr. *Best Data Visualization Tools 2021*. URL: https://techresearchonline.com/blog/best-data-visualization-tools-2021/#5_Leaflet. (accessed: 08.01.2022).

[18] a.s. Seznam.cz. *Mapy.cz API/SDK*. URL: https://vyvojari.seznam.cz/mapy. (accessed: 21.04.2022).

[19] Tim Schaub. *OpenLayers release notes*. URL: https://github.com/openlayers/openlayers/releases/. (accessed: 08.01.2022).

[20] Emmanuel Stefanakis. *Web mercator and raster tile maps: two cornerstones of online map service providers*. URL: https://www.researchgate.net/publication/321064657_Web_mercator_and_raster_tile_maps_two_cornerstones_of_online_map_service_providers. (accessed: 23.04.2022).

[21] V8. *SpiderMonkey*. URL: https://spidermonkey.dev/. (accessed: 11.01.2022).

[22] V8. *What is V8?* URL: https://v8.dev/. (accessed: 11.01.2022).

[23] Chris Veness. *Calculate distance, bearing and more between Latitude/Longitude points*. URL: https://www.movable-type.co.uk/scripts/latlong.html. (accessed: 26.04.2022).

[24] w3school. *W3Color JavaScript Library*. URL: https://www.w3schools.com/lib/w3color.js. (accessed: 13.04.2022).