Progresivní Webová Aplikace pro záznam, analýzu a zobrazování geografických dat

Bc. Michal Kovařík

Diplomová práce
2019

Univerzita Tomáše Bati ve Zlíně
Fakulta aplikované informatiky

Univerzita Tomáše Bati ve Zlíně
Fakulta aplikované informatiky
akademický rok: 2018/2019

ZADÁNÍ DIPLOMOVÉ PRÁCE

(PROJEKTU, UMĚLECKÉHO DÍLA, UMĚLECKÉHO VÝKONU)

Jméno a příjmení:	**Bc. Michal Kovařík**
Osobní číslo:	**A17304**
Studijní program:	**N3902 Inženýrská informatika**
Studijní obor:	**Informační technologie**
Forma studia:	**kombinovaná**
Téma práce:	**Progresivní Webová Aplikace pro záznam, analýzu a zobrazení geografických dat**
Téma anglicky:	**A Progressive Web Application for Recording, Analysing and Displaying Geographic Data**

Zásady pro vypracování:

1. **Nastudujte moderní způsoby vyvíjení aplikací, především standardy PWA (Service Workers, IndexedDB, Local Storage, App Cache) a srovnejte s běžnými způsoby tvorby aplikací.**
2. **Pojednejte o bezpečnosti webové platformy, integritě souborů a dat aplikace a možnostech autentizace uživatele.**
3. **Prozkoumejte možnosti záznamu či jiných forem získávání GPS dat a shrňte možná řešeni mapových podkladů a knihoven pro práci s nimi.**
4. **Zaměřte se na vývoj PWA aplikace, která tyto data dokáže zaznamenat, upravovat, katalogizovat, zobrazovat a ukládat.**
5. **Na základě nabitých poznatků a funkčních a nefunkčních požadavků aplikace naimplementujte podpurné knihovny pro tvorbu PWA, práci s daty, GPS, apod.**
6. **Proveďte sběr dat a demonstrujte běh aplikace na reálném zařízení a diskutujte také případná omezení či technologické překážky.**

Rozsah diplomové práce:

Rozsah příloh:

Forma zpracování diplomové práce: **tištěná/elektronická**

Seznam odborné literatury:

1. **ATER, Tal. Building progressive web apps: bringing the power of native to the browser. Sebastopol, CA: O'Reilly Media, 2017. ISBN 978-149-1961-650.**
2. **SHEPPARD, Dennis. Beginning progressive web app development: creating a native app experience on the web. New York, NY: Springer Science+Business Media, 2017. ISBN 978-1484230893.**
3. **HUME, Dean Alan a Addy OSMANI. Progressive web apps. Shelter Island, New York: Manning Publications, 2018. ISBN 1617294586.**
4. **MEW, Kyle. Learning Material Design. Packt Publishing, 2015. ISBN 1785289810.**
5. **WARGO, John M. Apache Cordova 4 programming. Upper Saddle River, NJ: Addison-Wesley, 2015. ISBN 0134048199.**
6. **ANDERSON, Paul. Web 2.0 and beyond: principles and technologies. Boca Raton: CRC Press, 2012. Chapman & Hall/CRC textbooks in computing. ISBN 9781439828670.**
7. **KASTANAKIS, Bill. Mapbox Cookbook. Packt Publishing, 2016. ISBN 1784397350.**
8. **HOLDENER, Anthony T. HTML5 Geolocation. Sebastopol, CA: O'Reilly, 2011. ISBN 14-493-0472-9.**

Vedoucí diplomové práce: **Ing. Radek Vala, Ph.D.**
Ústav informatiky a umělé inteligence

Datum zadání diplomové práce: **3. prosince 2018**

Termín odevzdání diplomové práce: **15. května 2019**

Ve Zlíně dne 7. prosince 2018

doc. Mgr. Milan Adámek, Ph.D.
děkan

prof. Mgr. Roman Jašek, Ph.D.
garant oboru

Prohlašuji, že

- beru na vědomí, že odevzdáním diplomové práce souhlasím se zveřejněním své práce podle zákona č. 111/1998 Sb. o vysokých školách a o změně a doplnění dalších zákonů (zákon o vysokých školách), ve znění pozdějších právních předpisů, bez ohledu na výsledek obhajoby;
- beru na vědomí, že diplomová práce bude uložena v elektronické podobě v univerzitním informačním systému dostupná k prezenčnímu nahlédnutí, že jeden výtisk diplomové/bakalářské práce bude uložen v příruční knihovně Fakulty aplikované informatiky Univerzity Tomáše Bati ve Zlíně a jeden výtisk bude uložen u vedoucího práce;
- byl/a jsem seznámen/a s tím, že na moji diplomovou práci se plně vztahuje zákon č. 121/2000 Sb. o právu autorském, o právech souvisejících s právem autorským a o změně některých zákonů (autorský zákon) ve znění pozdějších právních předpisů, zejm. § 35 odst. 3;
- beru na vědomí, že podle § 60 odst. 1 autorského zákona má UTB ve Zlíně právo na uzavření licenční smlouvy o užití školního díla v rozsahu § 12 odst. 4 autorského zákona;
- beru na vědomí, že podle § 60 odst. 2 a 3 autorského zákona mohu užít své dílo – diplomovou práci nebo poskytnout licenci k jejímu využití jen připouští-li tak licenční smlouva uzavřená mezi mnou a Univerzitou Tomáše Bati ve Zlíně s tím, že vyrovnání případného přiměřeného příspěvku na úhradu nákladů, které byly Univerzitou Tomáše Bati ve Zlíně na vytvoření díla vynaloženy (až do jejich skutečné výše) bude rovněž předmětem této licenční smlouvy;
- beru na vědomí, že pokud bylo k vypracování diplomové práce využito softwaru poskytnutého Univerzitou Tomáše Bati ve Zlíně nebo jinými subjekty pouze ke studijním a výzkumným účelům (tedy pouze k nekomerčnímu využití), nelze výsledky diplomové/bakalářské práce využít ke komerčním účelům;
- beru na vědomí, že pokud je výstupem diplomové práce jakýkoliv softwarový produkt, považují se za součást práce rovněž i zdrojové kódy, popř. soubory, ze kterých se projekt skládá. Neodevzdání této součásti může být důvodem k neobhájení práce.

Prohlašuji,

- že jsem na diplomové práci pracoval samostatně a použitou literaturu jsem citoval. V případě publikace výsledků budu uveden jako spoluautor.
- že odevzdaná verze diplomové práce a verze elektronická nahraná do IS/STAG jsou totožné.

Michal Kovařík, v. r.

Ve Zlíně, dne 17.5.2019 …….

podpis diplomanta

ABSTRAKT

Práce řeší problematiku vývoje mobilních aplikací a programů skrze webovou platformu za pomocí moderních, experimentálních standardů PWA. Cílem je vytvořit webovou aplikaci, jevící se jako běžná stránka, kterou je ale možno si nainstalovat přímo z prohlížeče. Aplikace se pak bude chovat jako nativní, ve vlastním okně, s hlubší integrací do operačního systému a přístupem k hardwarovým sensorům. Poslouží k zobrazování a správě geografických dat vynesených na interaktivní vektorové mapě.

Klíčová slova: web, aplikace, PWA, Progresivní Webová Aplikace, Service Worker, offline cache, exif, gps, mapa, geolokace, souřadnice

ABSTRACT

This thesis focuses on the development of mobile applications programs through web platform with the use of modern, experimental PWA standards. The objective is to create a web page, which on first sight looks like a regular website, but one that can be installed from the browser. The application then acts like a native one, with deeper integration into the operating system and access to hardware sensors. It will then display and allow modification of geographic data rendered on an interactive vector map.

Keywords: web, application, PWA, Progressive Web App, Service Worker, offline cache, exif, gps, map, geolocation, coordinates

Chtěl bych poděkovat vedoucímu práce Ing. Radku Valovi Ph.D. za vedení, vstřícné jednání, čas a cenné rady poskytnuté při tvorbě práce.

Prohlašuji, že odevzdaná verze diplomové práce a verze elektronická nahraná do IS/STAG jsou totožné.

OBSAH

ÚVOD **9**
I TEORETICKÁ ČÁST **10**
1 PROGRESIVNÍ WEBOVÉ APLIKACE (PWA) **11**
1.1 SINGLE PAGE APLIKACE 11
1.2 PROGRESSIVE ENHANCEMENT 11
1.3 SOUČASNÁ PODPORA PWA 13
1.4 ALTERNATIVY 15
2 PWA STANDARDY **16**
2.1 WEB APP MANIFEST 16
2.2 SERVICE WORKER 17
2.2.1 Charakteristické vlastnosti 18
2.2.2 Životní cyklus 19
2.3 INDEXEDDB A OSTATNÍ PERSISTENTNÍ ÚLOŽIŠTĚ 20
2.4 CACHE STORAGE API 21
2.5 SDÍLENÉ ZDROJE PROHLÍŽEČE A PWA 22
3 GEOGRAFICKÉ INFORMAČNÍ SYSTÉMY **24**
3.1 DLAŽDICE 24
3.2 RASTROVÉ DLAŽDICE 25
3.3 VEKTOROVÉ DLAŽDIC 27
4 ZABEZPEČENÍ **28**
4.1 ZABEZPEČENÍ A RIZIKA WEBOVÉ PLATFORMY 28
4.1.1 Bezpečnost z pohledu uživatele 28
4.1.2 Omezení Webových API 29
4.2 ZABEZPEČENÍ PWA APLIKACE 30
4.2.1 Základní předpoklady instalace PWA 30
4.2.2 Bezpečnostní model PWA 31
4.2.3 Integrita lokálních dat aplikace 32
4.2.4 Zabezpečení dat za běhu aplikace a prohlížeče 32
4.3 MOŽNOSTI NAPADENÍ 33
4.4 AUTENTIZACE A AUTORIZACE 34
II PRAKTICKÁ ČÁST **36**
5 PRVOTNÍ KROKY VÝVOJE **37**
5.1 MOTIVACE VYTVOŘENÍ MAPOVÉ APLIKACE 37
5.2 PROTOTYPOVÁNÍ 38
5.3 KOSTRA APLIKACE 39
5.4 ROZLOŽENÍ APLIKACE 40
6 IMPLEMENTAČNÍ DETAILY **44**

6.1 Cachování statických dat 44
6.2 LocalStorage a IndexedDb 45
6.3 Plugin pro knihovnu Mapbox 46
7 PROHLÍŽENÍ A ÚPRAVA GEOGRAFICKÝCH DAT 48
7.1 Prohlížeč, adaptivní hitbox 48
7.2 Adaptivní hitbox 49
7.3 Editor 50
7.3.1 Výkonnostní optimalizace 51
7.3.2 User experience 53
8 ZÍSKÁVÁNÍ A ZAZNAMENÁVÁNÍ GEOGRAFICKÝCH DAT 56
8.1 Nahrávání skrze Geolocation API 56
8.2 Získávání souřadnic z EXIFu fotografií 58
8.3 Rekonstrukce trasy z náhodné posloupnosti souřadnic 60
9 GEOPROSTOROVÁ ANALÝZA A ZPRACOVÁNÍ DAT 62
9.1 Začištění a komprimace 62
9.2 Detekce překryvů a duplicitních tras 65
9.3 Výpočty ovlivněné kartografickou projekcí 67
9.3.1 Lambertovo zobrazení 67
9.3.2 Mercatorovo zobrazení 68
9.3.3 Výpočty zkreslující geografickou přesnost ve prospěch UI 68
10 ADAPTIVNÍ PŘIZPŮSOBOVÁNÍ AKTUÁLNÍ PLATFORMĚ 71
10.1 Tvorba knihovny detekující platformu a prostředí 71
10.2 Experimentování s dostupností na TV 72
10.3 Experimentování s dostupností na herních konzolích 72
ZÁVĚR 74
SEZNAM POUŽITÉ LITERATURY 76
SEZNAM POUŽITÝCH SYMBOLŮ A ZKRATEK 78
SEZNAM OBRÁZKŮ 79

ÚVOD

Dnešní doba chytrých telefonů sice vkládá uživateli do rukou více možností, větší tlak je však na vývojáře. Dříve stačilo vyvinout jeden program pro jeden operační systém, dnes jich je celá řada, z nichž každá vyžaduje svůj vlastní programovací jazyk, vestavěné nástroje a knihovny. Namísto vývoje jednoho produktu, tak vývojář, nebo spíše celý tým skončí s vývojem čtyř samostatných projektů, z nichž každý má identickou funkcionalitu, ale je vyvinut v odlišeném prostředí a jazyku.

Každá aplikace navíc potřebuje svou webovou verzi. V současné době však vzniká řada standardů, které rozšiřují technologické možnosti webové platformy, které přibližují schopnosti webových stránek k nativním aplikacím a rozmazávají tak rozdíl mezi webem a aplikacemi, jak je známe dnes. Umožnují tak na první pohled běžnou webovou stránku nainstalovat jako aplikaci, přímo z webového prohlížeče. Jako by to byla nativní aplikace, s vlastním oknem a ikonou, bez adresy a ovládacích prvků prohlížeče. Zato schopnou fungovat i bez připojení k internetu. Této technologii se říká Progresivní Webové Aplikace, zkráceně PWA a teprve v době psaní této práce se tato funkcionalita dostává do webových prohlížečů.

Autor se v práci bude věnovat zkoumání nově vzniklým, či stále vznikajícím standardům rozšiřující schopnosti webových stránek. Konkrétně se bude věnovat rodině standardů PWA, jejich schopnostem, současné podpoře ve webových prohlížečích, dále rozebere jejich výhody i obtížnosti. Na základě nabytých zkušeností bude vyvinuta řada experimentů opírající se o tyto standardy. Z těch pak bude vytvořena demonstrativní aplikace představující schopnosti PWA, včetně existujících schopností webu, jako je získávání GPS souřadnic, vykreslování 3D grafiky pomocí WebGL s akcelerací GPU, nebo skladování strukturovaných dat do persistentního úložiště.

Cílem bude dokázat, že web je dostatečně pokročila platforma na vývoj nejen webových stránek, ale především aplikací, jež se svou funkcionalitou, výkoností a vzhledem mohou rovnat nativním aplikacím.

Záměrem je také dát práci přesah do světa open source software. Kvůli své experimentální podstatě bude výstupní aplikace především demonstrovat schopnosti platformy. Avšak na základě vzniklých požadavků k vyvinutí aplikace vzniknou podpůrné knihovny, jež budou vyseparovány a publikovány k užití i v jiných projektech mimo tuto práci.

I. TEORETICKÁ ČÁST

1 PROGRESIVNÍ WEBOVÉ APLIKACE (PWA)

Běžnou realitou dnešního vývoje uživatelského software je prioritizace doby potřebného pro vývoj, který je zásadně ovlivněn přívětivostí a dostupností rozličných API na cílové platformě. Práci na desktopovém programu, kterou dříve obstarávaly celé týmy programátorů, dnes nahradily mobilní aplikace jejichž vývoj zastanou i jednotlivci. Díky abstraktnějším programovacím jazykům, rozmachu open source a tím i dostupnosti velké škály frameworků a knihoven. Rozšíření mobilních telefonů zapříčinil posuv ve strategii. Mantrou dnešní doby se stalo mobile-first. Webové stránky se tak předně vyvíjejí pro mobilní telefony a až poté jsou responzivně rozšiřovány pro větší displeje. Mobilní aplikace se vyvíjejí dříve než případný desktopový program. Jindy je na desktopovém OS pouze emulována mobilní aplikace, viz. Chrome OS s podporou Javových aplikací z Androidu.

1.1 Single Page Aplikace

V myslích uživatelů je možné nakreslit dělící čáru mezi webem a mobilními aplikacemi, kde web představuje statický obsah a mobilní aplikace, nebo klasické desktopové programy, jsou interaktivní, nabízejí nástroje pro tvorbu obsahu, modifikaci a produktivní činnost. Díky schopnostem moderních webových technologií začaly být webové stránky nejen rozšiřovány o interaktivní prvky, ale velká část logiky aplikace se přesunula z backendu do webového prohlížeče. Stránky tak již nejsou server-side rendered. Klient namísto hotové HTML stránky obdrží řadu zdrojových souborů, šablon, dat a zpracovává je za chodu.

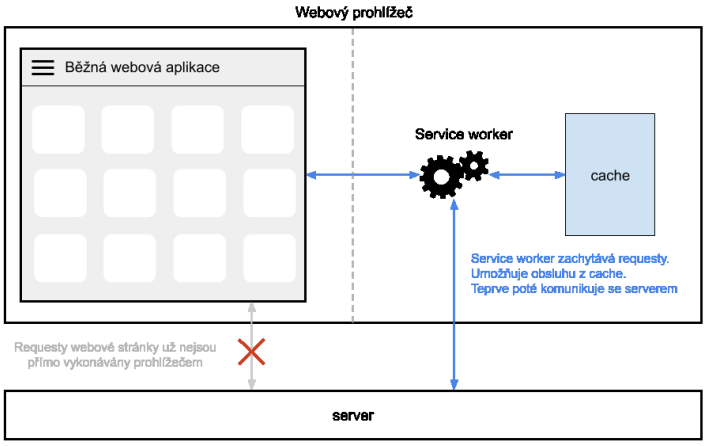
Standard HTML5 Navigation umožňuje zachytávat změny v URL adrese a reagovat na ně, případně zabraňovat v navigaci, aby současná stránka nebyla opuštěna a načtena jiná. AJAX HTTP requesty pak zase umožňují v pozadí načíst dodatečný, nebo aktualizovaný obsah a vložit jej v aktuálně otevřené stránky [1]. Takové webové stránky již plně replikují chování nativních aplikací a programů, zcela ve webovém prohlížeči, v rámci životního cyklu jedné návštěvy. Nazýváme je proto Single Page Aplikace, neboť se veškeré dění odehrává na jedné webové stránce, která je průběžně překreslována, aniž by ji uživatel opustil. Zpravidla bývají doplněny některým z aktuálně populárních MVC frameworků, jako např. Angular, React, Vue, Aurelia apod.

1.2 Progressive enhancement

Spolu s rozšiřujícími se schopnostmi webové platformy se také začaly objevovat pojmy progresivní rozšiřování z anglického progressive enhancement, který jde ruku v ruce s graceful

degradation [1]. Lze si je vyložit jako rozšiřování funkční služby o přidanou hodnotu zvyšující uživatelský komfort a zároveň zachování základní funkcionality na zařízeních a platformách, které nedisponují dostatečnou technologií, aktuální verze prohlížeče apod. Mezi příklady z každodenního života patří například e-shop nabízející nejbližší výdejní místo vzhledem k uživatelově poloze, či sociální sítě zobrazující systémové notifikace po obdržení nepřečtených zpráv.

Běžné webové stránky by na hlavním UI vlákně prováděly mimo vykreslování aplikace také složitější výpočty, které by mohly UI blokovat a zpozdit tím reakční dobu na interakci uživatele, jako stisk tlačítek, scrollování apod. Takové operace mohou a měly by být přesunuty do samostatného skriptu, který není do hlavního vlákna importován běžným `<script>` tagem, ale je spuštěn v samostatném vlákně, s odděleným kontextem. To lze vytvořením instance třídy `Worker` s cestou ke skriptu. Následně mezi oběma vlákny započne komunikace, umožňující předávání instrukcí a dat z hlavního vlákna na Workeru a nazpět. Obdobně lze Service Workerem přesměrovávat komunikaci se serverem a statické assety načítat z cache namísto stahování ze serveru.



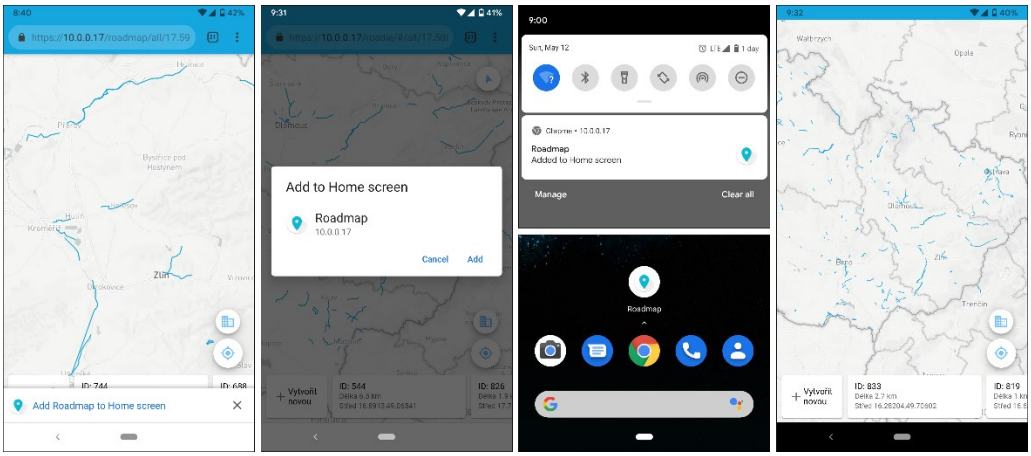
Obrázek 1. Diagram progresivního rozšiřování webové stránky Service Workerem [vlastní tvorba].

Uživatelské prostředí je odstíněno od negativních vlivů, funkcionalita stránky je zachována, načítání zrychleno, a výsledný komfort užívání, z anglického User Experience, je navýšen. Na tomto principu pak staví Progresivní Webové Aplikace [2] [3], což je soubor webových standardů, přinášející vývojářům nové API umožňující své webové stránky dále rozšířit a hlouběji integrovat do OS, aby byly nerozeznatelné od nativních aplikací.

1.3 Současná podpora PWA

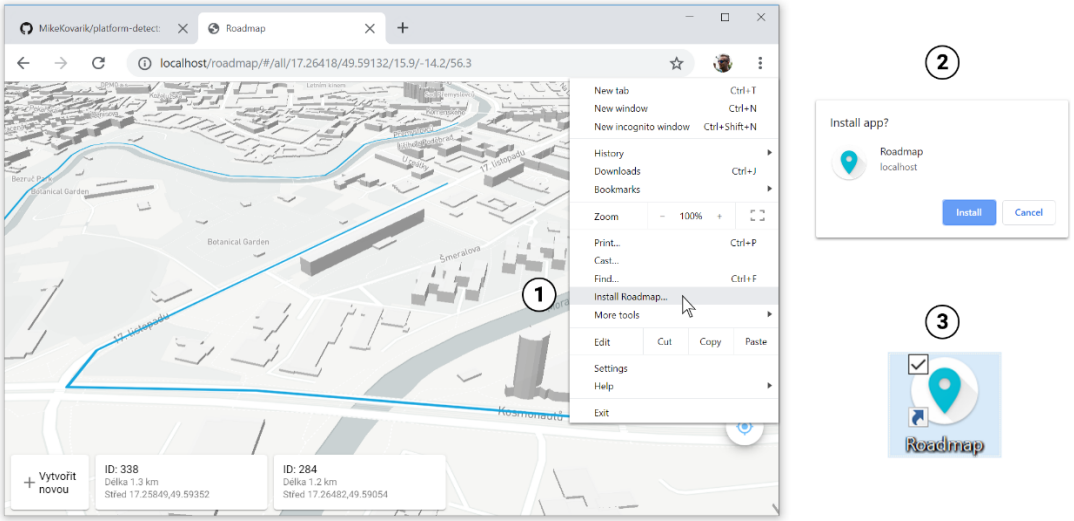
Prvotní implementace technologií umožňující vývoj a instalování PWA se datuje k roku 2013 [4], kdy Google začal standardizovat své experimenty z mobilní verze prohlížeče Chrome. Dnes již za standardy stojí nadnárodní výbor W3C a velká část z nich je postupně implementována v prohlížečích. Ačkoliv velká část standardů byla stabilizována již před lety a experimentálně plně implementována v prohlížeči Chrome, dostupná byla vždy jen pro vývojáře, se specifickými instrukcemi, jak tuto funkcionalitu v prohlížeči aktivovat. Postupem času se začaly přidávat i další společnosti a prohlížeče. Teprve na přelomu let 2018 a 2019 začíná být implementace ucelená, veřejná a funkční. Prozatím však jen na mobilních telefonech.

Mobilní operační systémy je umožňují PWA webové stránky nainstalovat jako aplikaci, na plochu telefonu, kde se chová jako nativní aplikace. Po spuštění se otevře v samostatném okně, bez rozhraní prohlížeče, tj. bez adresového řádku, tlačítka zpět, záložek atd. Rozpozná ji také systémový přepínač aplikací, kde je vyobrazena s korektní ikonkou. Běžnému oku uživatele tak unikne, že nejde o nativní aplikaci, ale jen o webovou stránku.



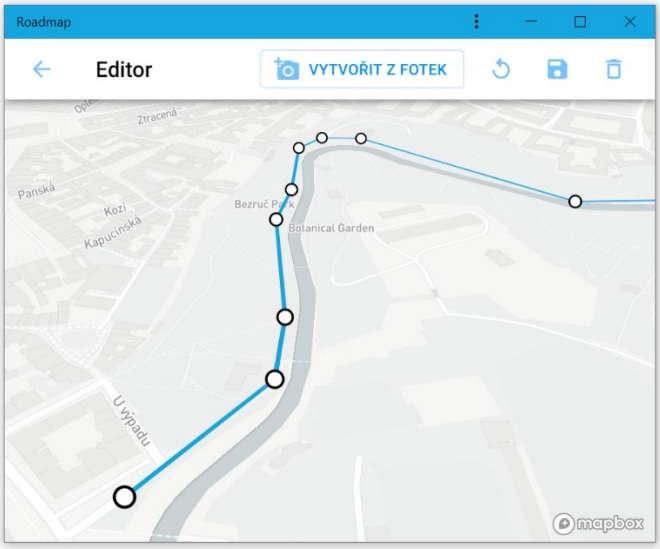
Obrázek 2. Instalace PWA z mobilního prohlížeče.

Výjimkou je společnost Apple, která donedávna využívala své dominance na trhu k potlačení rozvoje PWA ve prospěch své vlastní platformy. Pod tíhou kritiky a tlaku komunity společnost podlehla a v posledních verzích svého mobilního iOs 11.3 začala implementovat Service Worker do prohlížeče Safari.



Obrázek 3. Proces instalace PWA z okna prohlížeče Google Chrome, na desktopovém systému Windows 10.

Desktopová podpora PWA aplikací je v době psaní této práce v raném stádiu. Desktopová verze prohlížeče Chrome umožňuje instalování PWA aplikací jen po zapnutí příznaku experimentálních technologií v nastavení prohlížeče. Společnost Microsoft se dříve snažila prosadit svou platformu UWP, která se nesešla s úspěchem u vývojářů, následkem čehož MS zásadně změnil strategii a začal se angažovat v implementaci PWA a integraci do OS Windows. Vybrané PWA aplikace již dnes společnost sama zařazuje do svého Microsoft Store, dostupného v operačním systému Windows.



Obrázek 4. Desktopová PWA.

Důležitým milníkem bylo nedávné rozhodnutí MS ukončit vývoj svého vlastního vykreslovacího jádra, které bylo silně motivováno snahou lepé podporovat Progresivní Webové Aplikace. Operační systém Windows v sobě integruje vykreslovací jádro EdgeHTML, které pohání nejen prohlížeč Edge, ale slouží také k vykreslování UWP aplikací naprogramovaných v jazycích JS a HTML, nebo jako Webview komponenta pro C# aplikace. EdgeHTML je zmodernizovaný Trident, datující se až k dobám Internet Exploreru. Přestože Edge již podporuje Service Worker, absence řady jiných moderních webových standardů vedla společnost k ukončení vývoje prohlížeče Edge, spolu s vykreslovacím jádrem EdgeHTML a JavaScriptovým enginem Chakra. Na přelomu let 2018 a 2019 Microsoft oznámil přechod na open source jádro Chromium, vyvíjené Google pro prohlížeč Chrome, mimo jiné využívané i prohlížečem Opera. Jde o zásadní krok signalizující, že Microsoft uznává nedostatky své více než dvacet let vyvíjené proprietární technologie a plně se spoléhá na konkurenční řešení, aby tak zajistil hladkou a funkční podporu PWA aplikací na systémech Windows. V době psaní této práce je známo jen několik málo informací, mezi nimiž je i příslib rozšíření Chromiové verze Edge na další platformy jako Xbox One, macOS, a Linux. Bude zajímavé sledovat jestli, a jak hluboce nakonec bude Chromium integrováno do samotného Windows.

1.4 Alternativy

Samozřejmou alternativou je tvorba nativních aplikací v nativních jazycích a knihovnách zvolené platformy, tedy Android a Java, nebo iOs a Objective C nebo Swift. Zde však nemůžeme sdílet zdrojový kód aplikací.

Doposud velmi populární krok bylo wrapování SPA aplikace balíčku nativní aplikace a vydat ji v Google Play nebo App Store. Umožňují to projekty PhoneGap, nebo Cordova [5]. Principiálně jde o archiv s HTML a JS soubory, které jsou načteny do WebView, jediné viditelné nativní komponenty. Nevýhodou je nepatrně nižší výkon, způsobený abstrakcí stojící mezi aplikací a operačním systémem. Pro vývojáře pak jde o nutnost instalace všech nástrojů spojených s vývojem pro danou platformu, včetně všech závislostí a následně udržování vydaných balíčků. PWA umožňuje, nebo v dohledné době bude umožňovat stejný výsledek, bez zmíněných negativ.

2 PWA STANDARDY

Pojem progresivní aplikace je už z podstaty slova vágní, subjektivní a lze jej interpretovat různými způsoby. Komunita vývojářů spolu s tvůrci prohlížečů se proto shodla na těchto základních požadavcích, jež by měla každá aspirující webová stránka splňovat, aby mohla být označena jako progresivní aplikace. Respektive aby ji tak považoval i prohlížeč a vybídl uživatele k instalaci.

1. **Bezpečná** – servírovaná přes šifrovaný protokol HTTPS.
2. **Responzivní** – přizpůsobuje se všem displejům různých velikostí.
3. **Cross-broswser** – funguje a vykresluje se stejně ve všech prohlížečích.
4. **Metadata** – poskytuje prohlížeči skryté informace, ikonu a instrukce pro instalaci a běh v samostatném okně.
5. **Nezávislost na připojení** – všechny sekce aplikace lze spustit bez ohledu na rychlost připojení, a to i off-line.

Bod první až třetí jsou zcela v rukou autora aplikace, z nichž první bod pasivně řeší prohlížeč a webový server, nebo poskytoval. Pro body čtyři a pět bylo potřeba vyvinout nové API, kterým by autoři aplikací signalizovali prohlížeči záměry aplikace. Tyto API byly popsány a vyvinuty ve standardech Web App Manifest[1] a Service Worker[2], které spolu s dalšími webovými technologiemi tvoří rodinu standardů PWA.

2.1 Web App Manifest

Manifest standard zavádí jednoduchý soubor formátu JSON, poskytující základní informace o aplikaci, zejména název, popisek, ikony a barva. Odkaz na `manifest.json` je vkládán do hlavičky HTML stránky prostřednictvím `<link>` tagu, podobně jako CSS soubory.

```
<link rel="manifest" href="./manifest.json">
```

Vložení aplikačního manifestu do hlavičky v html.

[1] https://www.w3.org/TR/appmanifest/
[2] https://www.w3.org/TR/service-workers-1/

Informace definované v manifestu nemají žádný primární dopad na stránku, slouží prohlížečům jako silný indikátor, že nejde pouze o webovou stránku, ale webovou aplikaci. Ke kvalifikaci jako PWA je nutno splnit řadu dalších kritérií, mezi kterými je i implementace Service Workeru, nebo aspoň jedna ikona. Poté prohlížeč přistupuje k zobrazování nabídek instalace stránky jako aplikaci.

```
{
  "name": "My Progressive Web App",
  "short_name": "My PWA",
  "start_url": "/index.html?type=pwa",
  "display": "standalone",
  "theme_color": "#4285f4",
  "icons": [
    {
      "src": "/images/icons-192.png",
      "type": "image/png",
      "sizes": "192x192"
    },
    {
      "src": "/images/icons-512.png",
      "type": "image/png",
      "sizes": "512x512"
    }
  ]
}
```

Ukázka obsahu manifest.json.

Důležité je pole `display`. To by mělo mít hodnotu `standalone`, pokud se má aplikace spouštět v samostatném okně bez adresového řádku a jakýchkoliv prvků uživatelského rozhraní prohlížeče.

2.2 Service Worker

Klíčovým prvkem PWA je takzvaný Service Worker [7]. Jak již z názvu vyplývá, vychází z běžných Web Workerů [8], což jsou skripty běžící mimo hlavní vlákno aplikace a slouží k vykonávání náročnějších výpočtů, které by mohly blokovat vykreslování uživatelského rozhraní a způsobit zamrznutí prohlížeče, nebo aplikace.

```
<script src="./intensive-calculations.js">
<script>
var worker = new Worker('intensive-calculations.js')
</script>
```

Spuštění skriptu v samostatném vlákně Worker v porovnání s běžným importem.

2.2.1 Charakteristické vlastnosti

Service Worker běží na pozadí, samostatně od hlavního vlákna aplikace, ale na rozdíl od Web Workerů není vázán na její životní cyklus a jednu instanci aplikace. Neukončuje se při zavření stránky, nebo aplikace, a nadále běží v pozadí, pod kontrolou prohlížeče. Může tak zajišťovat obstarávání cache, podporu fungování aplikace offline, push notifikace, synchronizaci dat neodeslaných před ukončením aplikace a další funkcionalitu běžných nativních aplikací.

Vždy existuje pouze jeden Service Worker. Ten je zároveň sdílen mezi všemi otevřenými záložkami a instancemi ze stejného originu, tedy domény a portu z níž je aplikace dostupná. Pokud se aplikace pokusí zaregistrovat novou instanci, tak je předchozí SW ukončen a nahrazen novým.

Registrace a provoz Service Workeru je omezen pouze na šifrovaný protokol HTTPS. Vynucuje si to schopnost alterovat odchozí i příchozí komunikaci. Pokud by se útočníkovi podařilo odposlechnout komunikaci mezi klientem a serverem ve chvíli stahování zdrojových kódů stránky, stačilo by nahradit pouze Service Worker. Tím by získal kontrolu nad aktuální a všemi budoucími instancemi aplikace, včetně schopnosti poskytovat prohlížeči podvržené soubory a data.

Přístup k DOMu není povolen. Platí i pro Web Workery. Worker je sice součástí aplikace, která má i uživatelské rozhraní psané v HTML vykresleného do DOMu. Tyto dvě vlákna a jejich kontexty jsou od sebe ale odděleny. Proto ani proměnné definované v UI kontextu nejsou přístupné z Workeru. Jediný způsob předávání dat je komunikace zprávami skrze funkci `self.postMessage()` a události `message`.

Service Worker skript musí být hostován ze stejného originu jako aplikace. Nelze ho vystavovat z CDN, jako knihovnu. Pole působnosti SW jsou sandboxovány prohlížečem, díky čemuž nemá SW přístup ke cache a HTTP requestům jiných originů, resp. aplikací.

Veškerá operace SW by měla být navázána na události životního cyklu. Prohlížeč se může kdykoliv rozhodnout potlačit, nebo ukončit vykonávání Service Workeru z důvodů šetření baterie, dopadů na výkon CPU, paměti apod. SW může být několikrát restartován. Nelze tak spoléhat na globální proměnné, nebo globálně definované a okamžitě prováděné příkazy

2.2.2 Životní cyklus

Životní cyklus SW začíná registrací. Ta je vyvolána v UI vlákně aplikace při jejím spuštění. Následně prohlížeč stáhne a spustí daný skript v samostatném vlákně, izolovaném od aplikace, nezávislém na jejím životním cyklu.

```
<script>
if ('serviceWorker' in navigator) {
    navigator.serviceWorker.register('/sw.js')
}
</script>
```

Registrace Service Workeru z hlavního vlákna aplikace.

K registraci dochází při každém spuštění, ale úspěšná registrace SW ještě nezaručuje jeho instalaci. Ta probíhá jen pokud ještě nebyl žádný předchozí SW nainstalován, nebo pokud došlo ke změně ve zdrojovém kódu již nainstalovaného SW. V takovém případě je zastaralému a doposud aktivnímu SW vystavena událost `terminated`, což je možnost zpravidla odinstalovat starou verzi aplikace, vyčistit cache a následně stáhnout nové zdrojové kódy prostřednictvím nově aktivovaného SW.

Při instalaci je v SW vytvořena událost `install`. Teprve po vyvolání této událost může SW začít provádět pomocné úkony. Zároveň umožňuje pozastavit další instalační procesy ze strany prohlížeče, dokud není vykonána veškerá požadovaná instalační logika.

```
self.addEventListener('install', event => {
    var appFiles = ['/index.html', '/style.css', '/app.js']
    var promise = caches.open('staticAssetsCache')
        .then(cache => cache.addAll([appFiles]))
    event.waitUntil(promise)
})
```

Zachycení události install a následné cachování souborů aplikace.

Proběhne-li instalace úspěšně, dostává se do stavu installed, kdy ale ještě nemá kontrolu nad aplikací. Zpravidla zde nově instalovaný service worker čeká, než předešlá verze SW vykoná potřebné operace. Ačkoliv není SW závislý na běhu aplikace a zůstává spuštěn i po uzavření aplikace, jediná výjimka je právě čekání při nahrazování staré verze SW. Novější SW vyčkává ukončení aplikace, nebo refreshe starší verze stránky s navázanou starší verzí SW. Teprve poté dojde k aktivaci nového SW.

Aktivaci lze vynutit zavolání `self.skipWaiting()` čímž se nová verze SW násilně ujme kontroly nad všemi aktuálně běžícími instancemi aplikací i webových stránek.

Po aktivaci získá SW kontrolu nad životním cyklem rodičovské aplikace, a především může zachytávat a alterovat odchozí HTTP requesty a následně odpovědi na ně.

```
self.addEventListener('fetch', event => {
    var promise = caches.match(event.request)
        .then(response => response || fetch(event.request))
    event.respondWith(promise)
})
```

Zachycení HTTP requestu a vystavení z cache, je-li odpověď cachována.

2.3 IndexedDB a ostatní persistentní úložiště

Častým, nikoliv však nezbytným, požadavkem aplikací je persistentní úložiště pro data aplikace. Na poli webových standardů se jich vystřídalo několik, mezi nejznámější patří Local Storage a IndexedDB. Nejde přímo o nejnovější standard z rodiny PWA, tak jako Manifest nebo Service Worker. Potřeba uchovávat data napříč restartům webové stránky existovala již dříve. Prvotní implementace IndexedDB v prohlížečích sahá až do roku 2012 ve verzích Chrome 20 a dokonce také IE 10.

Prvním zmíněný standardem je Local Storage. Jedná se o prostý objekt akceptující stringové data formátu klíč:hodnota. Čtení a zápis je vázán na přímý přístup na disk. Proto je LS vhodný k ukládání drobných dat, jako konfigurační preference uživatele. Větší objemy strukturovaných dat je nutné ukládat do některé z prohlížečových databází. Populární, ale dnes již deprekovaná, byla do posledních let WebSQL. Jednalo se o zjednodušenou verzi SQL databáze pro web, s možností vytváření tabulek a zápisu dat skrze SQL query. WebSQL se neuchytil kvůli nižšímu výkonu a obtížnější práci s daty, oproti konkurenčnímu standardu IndexedDB z rodiny NoSQL databází. Technicky přesně nejde o tabulkovou databází, namísto tabulek se zde používá koncept takzvaných store. Princip práce s nimi je podobný tabulkám. Skladují číselně unikátně identifikovatelné záznamy, obdobné řádkům tabulek. Store však nemusí mít předem definované schéma, protože NoSQL databáze jsou objektové a přijímají libovolně tvarované, zanořené a komplexní JavaScriptové objekty. Pro přístup do databáze se nepoužívají dotazy, ale volání metod vestavěného API.

2.4 Cache Storage API

Souběžně se standardem SW vznikala i specifikace Cache Storage API, která slouží k uschovávání souborů a odpovědí na HTTP requesty. Přestože byla navržena především pro použití uvnitř Service Workeru v návaznosti na událost `fetch`, lze ji použít i z hlavního vlákna aplikace odkud může být vynuceno stažení a nacachování vybraných souborů.

```
var appFiles = ['./index.html', './style.css', './main.js']
window.caches.open('app-files').then(cache => cache.addAll(appFiles))
```

Otevření cache a uložení zdrojových souborů aplikace.

Na rozdíl od některých starších standardů je rozhraní všech metod Promise-based. Každá metoda tedy vrací instanci třídy Promise, tedy objekt s metodou `.then()`, která přijímá funkci, jež bude vykonána po dokončení operace. Díky nové syntaxi async/await lze operace zřetězit i bez volání této metody, za pomocí klíčového slova `await`. Alternativně lze použít `Promise.race()` a postavit proti sobě dvě souběžné operace a request obsloužit tou rychlejší. Naopak `Promise.all()` umožňuje vyčkat do vykonání všech souběžných promis. Tím je usnadněna synchronizace a manipulace s větším počtem operací. Následující příklad popisuje chování hypotetického Service Workeru, který zachytí odchozí HTTP request a následně jej obslouží rychlejším ze dvou souběžných handlerů. Prvním je `getFromCache()`, který vyčká do otevření a zpřístupnění cache s názvem `app-files` a následně se v ní pokusí najít předem nacachovanou odpověď na aktuální request. Druhou větví je `fetchAndCache()`, který podstoupí vykonání síťového požadavku a teprve po obdržení odpovědi ji zároveň uloží do cache, pro budoucí použití, a zároveň vrátí obsluze requestu.

```
var cacheOpenPromise = caches.open('app-files')

async function getFromCache({request}) {
    var cache = await cacheOpenPromise
    return cache.match(request)
}

async function fetchAndCache({request}) {
    var response = await fetch(request)
    cacheOpenPromise.then(cache => cache.put(request, response))
    return response.clone()
}

self.addEventListener('fetch', event => {
    var promise = Promise.race([getFromCache(event), fetchAndCache(event)])
    event.respondWith(promise)
})
```

Tento příklad je jeden z mnoha možných způsobů obsluhy a liší se podle potřeb aplikace. Zde je prioritizována rychlost obsluhy a vykreslené aplikace a je ideální pro fungování offline. V případě aplikace, u níž se nepočítá s příliš častými aktualizacemi může jít o zbytečné plýtvání mobilními daty, neboť na telefonech s rychlými paměťmi bude rychlejší vždy vystavení z cache. Naopak u počítačů s pomalejšími disky může být síťový požadavek mnohdy rychlejší než roztočení ploten pevného disku. Další možností je kaskádový fallback, kdy se nejprve pokusí provést síťový požadavek o aktuální zdrojové kódy a v případě neúspěchu je záložní variantou obsluha z cache. To však není vhodné pro offline-first aplikace, jelikož čekání na selhání síťového požadavku může trvat i dlouhé sekundy.

Možná je i opačná varianta cache-first s fallbackem na síťovou operaci. Zde však může dojít k fatálnímu rozsynchronizování verzí různých souborů aplikace, které mohou být dostahovány a cachovány až ve chvíli kdy je uživatel potřebuje. Při práci s cache je proto kriticky důležité pamatovat na retenci starých verzí aplikace u uživatelů. Přístup „rychlejší vyhrává“ sice zdrojové kódy aplikace v cache udržuje relativně aktuální, neboť se pokaždé pokusí stáhnout aktuální verzi ze sítě, ale zároveň bude aplikace vždy jeden refresh vzdálená od aktuální verze, protože rychlejší bývá obsluha z cache. Pokud aplikace změní své backendové API, mnoho uživatelů může skončit s nefungující aplikací i přesto, že je updatován i frontendový kód. A to právě kvůli načtení aplikace z cache. Nejen kvůli těmto případům je vhodné cache verzovat, podobně jako je možné verzovat databáze, kdy jsou tabulky rozšířeny o další sloupce. Často se tak setkáváme s aktualizovaným Service Workerem otevírajícím novou verzi cache pomocí `caches.open('app-files-v2')`.

2.5 Sdílené zdroje prohlížeče a PWA

Service Worker není výlučně exkluzivní pouze pro PWA aplikace. Lze ho používat i na běžných webových stránkách, které chtějí zajistit širší funkcionalitu uživateli skrze např. Push notifikace. K registraci SW dochází při první návštěvě stránky. Nainstalování stránky jako PWA je umožněno teprve poté co prohlížeč zjistí, že webová stránka splňuje všechny předpoklady pro fungování jako standalone aplikaci. V prohlížeči Chrome jde o heuristiku zjišťující, zda je zaregistrován SW spolu s handlerem událostí fetch, a zároveň pokud existuje soubor `manifest.json` spolu s definovanými povinnými poli a minimálně jednou ikonkou.

Instalace PWA je tedy přerod z běžné webové stránky na samostatnou aplikaci. Ta však sdílí všechny zdroje s původní webovou stránkou, přestože je PWA spouštěna v samostatném

okně a jeví se jako samostatná nativní aplikace. Na pozadí PWA však stále stojí prohlížeč, se svým vykreslovacím jádrem a JavaScriptovým enginem, který ke zdrojovému kódu aplikace přistupuje jako k jakékoliv jiné webové stránce. Rozdílný je způsob vykreslení do samostatného okna bez ovládacích prvků webového prohlížeče. Technicky vzato tak jde o pokročilejší webové záložky se zástupci na ploše, nikoliv o zcela samostatnou aplikace.

Soubory aplikace i stránky jsou načítány ze stejného originu. Proto je sdílen Service Worker, cache, databáze a zápis do ní skrze webovou stránku se projeví i v PWA aplikaci a naopak. Debugovat spuštěné instance Service Workerů je možné prostřednictvím vývojářských nástrojů webového prohlížeče, ze kterého bylo PWA nainstalováno.

3 GEOGRAFICKÉ INFORMAČNÍ SYSTÉMY

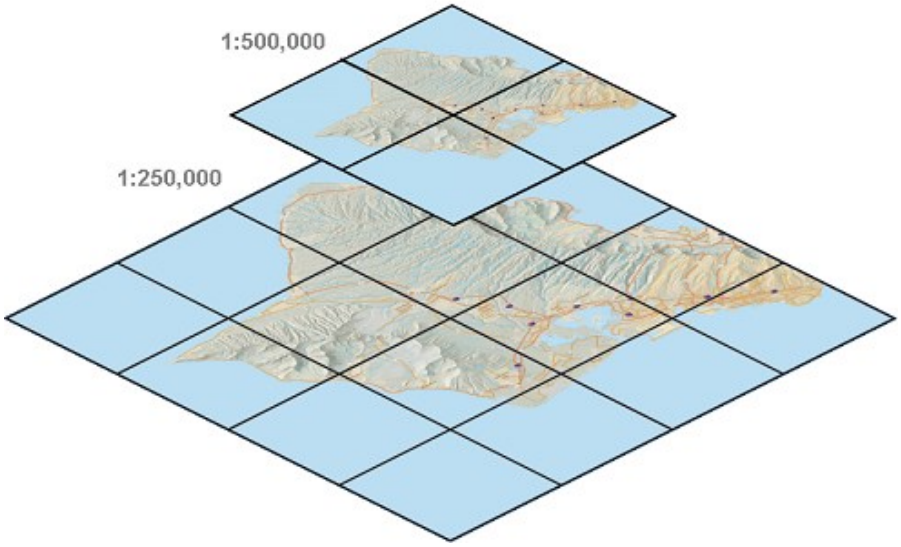
Mapa je abstraktní pojem, pod kterým si lze představit surová kartografická data, informace o polohách budov, cest, lesů, jejich rozlohám, vzájemným vzdálenostem, nebo také podrobnější vlastnostech jako třeba kvalita povrchu cesty a počet jízdních pruhů silnice.

3.1 Dlaždice

Základním stavebním kamenem každé mapy jsou tzv. dlaždice. Ty se seskupují do mřížky, které pak utvoří výslednou mapu. Bylo by nejen nepraktické, ale také nemožné obsáhnout veškeré informace o světě do jednoho obrovského obrázku. Ten by musel mít milióny pixelů jak na šířku, tak i na výšku a jeho datový objem by překračoval objemy hudebních alb i filmů. Proto byla zavedena mřížka dlaždic.

Dlaždice nahrazují prvotní webové mapy, kde se vybraná část mapy vyrenderovala na serveru jako jeden statický obrázek. Takový přístup byl nejen neefektivní kvůli nárokům na výpočetní výkon serveru, ale také nepraktický pro uživatele z hlediska user experience. Při každém posunu nebo přiblížení mapy musela být aktuálně zobrazený výřez mapy zahozena a načten nový. Takový posun nemohl být prováděn intuitivním drag and drop ovládáním jako dnes, ale mapa byla ze všech stran ohraničena šipkami po jejímž kliknutí došlo o skok ve zvoleném směrem v uniformní vzdálenosti. S příchodem nových webových technologií, především AJAXu, vznikla cesta pro efektivnější využívání zdrojů. Serverových i uživatelských.

Zatímco staré mapy byly generovány on-demand pro jedno použití uživatele a jeho okamžité potřeby, dlaždice umožňují celý svět rozdělit do předvídatelných oddílů, před generovávat je ještě, než první uživatel stránku s mapou navštíví, a následně se stejnými daty univerzálně obsluhovat velké množství návštěvníků. Tyto úkoly mohou být plánovaný, paralelizovány, a omezeny na pře generování pouze pozměněných dlaždic podle delty změn mezi aktuálními daty a naposledy renderovanými. Všechny dlaždice jsou tak jednoznačně označeny, podle své pozice, uloženy na serveru nebo v databázi, a univerzálně a opakovaně využívány pro obsluhu. Ze serveru se tak vlastně stává CDN server, tj. Content Delivery Network. Zde je zaveden další pojem, takzvaný tileserver, odvozený z anglického slova tile, tj. dlaždice.



Obrázek 5. Mapové Dlaždice. [9]

Po technické stránce jsou dlaždice rastrové obrázky načtené ze serveru. Každá dlaždice má své pořadové číslo ve formátu `z/x/y`, které je dané souřadnicemi X, Y a měřítkem přiblížení Z neboli zoom. Každá dlaždice přesně navazuje na své přilehlé sousedy, kteří pak navazují na své sousedy atd... Díky tomu lze zobrazit libovolný výřez světa.

3.2 Rastrové dlaždice

Přestože dlaždice znamenaly značný pokrok na poli interaktivních mapových služeb, ve světle nových technologií začínají na povrch stále častěji vystupovat nevýhody rastrových dlaždic. Jelikož se ale jedná o rastrové obrázky, nelze s nimi příliš dobře pracovat na displejích s jemnější pixelovou mřížkou (pixel ratio). Takové displeje bývají známější pod pojmem Retina display.

U běžných monitorů stolních počítačů je jemnost pixelů zpravidla 1, čili jeden softwarově vykreslený pixel je zobrazen jedním fyzickým pixelem na monitoru. S příchodem chytrých mobilních telefonů se jemnost začala zvyšovat. Nyní běžně narážíme na jemnosti 1.5, 2, 3, nebo 4. Například mobilní telefon s pixel ratio 3 a rozlišením 1080x1920 zobrazí jeden softwarový pixel na třech fyzických pixelech. Operační systém se chová jako by byl displej třikrát menší a pracuje s rozlišením 360x640. Ve skutečnosti se ale jedná o logické pixely z pohledu aplikace. Ta má šířku 360px a například písmo má velikost 16px. Vykreslovací jádro OS se pak postará o upscalování na fyzickou šířku 1080px, díky čemuž se vektorový obsah vykreslí třikrát jemněji. Písmo se tak sice logicky chová stále jako 16px, ale ve skutečnosti je vykresleno jako 48px. Text vypadá jemněji a fotografie mohou být ostřejší a

detailnější. Avšak pouze za předpokladu že do aplikace byla načtena fotografie větší, než je logické rozlišení, v tomto případě 3x. Jinak dochází k běžnému upscalování, roztahování obrázku, přičemž nové pixely jsou vytvořeny zprůměrováním barev sousedních pixelů. Následkem je nekvalitní, rozmazaný obraz.

Proto i na moderních displejích některý obsah vypadá nepěkně a rozmazaně. Chyba není na straně hardware ani OS, ale u autora aplikace, nebo webové stránky. Jedná se o mnohem komplikovanější téma, které netřeba rozvádět vice než konstatováním, že protipólem líbivější grafiky je efektivita vytváření a načítání zdrojů. Proto velká spousta webů stále poskytuje obsah pro logické rozlišení displeje a nebere v potaz pixel ratio displeje.

Dotýká se to především rastrových map od Google, nebo předchozí generace map od Mapboxu postavených na knihovně Leaflet. Mapy se kvůli tomu zdají být rozmazané a nevhodné pro použití na mobilních zařízeních. Jsou sice rychleji načtené, neboť není potřeba u uživatele renderovat. Z hlediska user experience však neobstojí.

Druhým zdrojem potíží pro displeje s vyšší jemností je překryv, respektive nedokrývající se mezery mezi dlaždicemi. Jak již bylo zmíněno, logické pixely jsou přepočítávány vykreslovacím jádrem prohlížeče na fyzické pixely displeje. U sudého pixel ratia jde o triviální zdvojnásobení, čtyřnásobení apod. Avšak liché, nebo desetinná čísla je nutno transponovat. Následkem toho se pixely původního obrazu vzájemně prolínají a misí do sebe, podobně jako při upscalování méně kvalitních obrázků. A to i v případě že downscalujeme dlaždici o větším rozlišení. Výsledný obraz je mírně deformovaný, místy rozmazaný. Poté dochází k zaokrouhlení šířky a výšky dlaždice. Zde může dojít k zaokrouhlení směrem dolů u jedné či více dlaždic. Ty k sobě nepřiléhají natolik těsně aby nešla vidět mezera v mřížce čímž vznikne nevzhledný šev po celé délce dlaždice nebo i mapy.

Tento problém je častější na OS Windows s vykreslovacím jádrem Webkit, tedy v prohlížečích Chrome a Opera. Dále je podstatně výraznější na zařízeních s pixel ratio, které není celé číslo. Například Surface Pro 3, s pixel ratio 1.5. Podobné problémy byly pozorovány i na mobilních telefonech Nexus 5 s pixel ratio 3.0 a Google Pixel s jemností 2.66. Lze konstatovat že jde o neodstranitelný problém způsobený optimalizací vykreslovacího, které upřednostňuje výkon před kvalitou. To samozřejmě nelze označit za špatné rozhodnutí ze strany tvůrců webových prohlížečů. Čímž je podkopáván user experience spolu s kvalitou některých aplikací, což nutí hledat jiné možnosti vykreslování map.

3.3 Vektorové dlaždic

Nástup technologie WebGL umožnil vykreslování náročnější grafiky přímo v prohlížeči, Webové aplikace tak již nejsou omezeny pouze obdélníkovými elementy `div`, ale mohou prostřednictvím grafické karty efektivně vykreslovat jakýkoliv obraz přímo do elementu `canvas`. Toho využívá společnost Mapbox [7], která se svou knihovnou Mapbox GL vykresluje takzvané vektorové mapy. Ty řeší veškeré nevýhody rastrových map vykreslováním výsledné mapy přímo v zařízení, na míru danému zařízení. Jedná se o Client Side Rendering ve stejném slova smyslu jako když se aplikační logika přesunula ze serveru na stranu klienta, tedy aplikace běžící na zařízení, což umožnilo vznik Single Page Aplikací. Kvůli tomu je sice zvýšen nárok na výkon zařízení, ale objem dat nutný k vykreslení mapy je zlomkový.

Umožňují to vektorové dlaždice, které, stejně jako ty rastrové, rozdělují mapu na čtvercové segmenty. Ty lze libovolně stahovat a skládat dohromady. Na rozdíl od rastrových nejsou jejich data tvořeny barvami pixelu, nýbrž jde o logický popis objektů v daném segmentu a souřadnice míst kde se objekty nachází. Aplikace tak může libovolně zobrazovat silnice, domy, řeky a další prvky v různých barvách a lemováních, dle potřeby je měnit, nebo zcela ignorovat. A to na zařízení u uživatele, bez nutnosti generování a stahování nových rastrových obrázků ze serveru. Konkrétním příkladem může být tzv. dark mode, kdy ve večerních hodinách dojde k přepnutí světlých barev na tmavé.

Vykreslování vektorových map probíhá v samostatném vlákně odděleném od hlavního vlákna, ve kterém se vykresluje uživatelské rozhraní. Díky tomu nedochází k zamrznutí aplikace a veškeré prvky uživatelského rozhraní jsou tak stále interaktivní, přestože je procesor maximálně vytížen vykreslováním mapy.

Na tomto principu stojí nejen webová knihovna Mapbox GL JS, ale také nativní Javová knihovna Google Maps pro tvorbu Android aplikací, kde je postavena na proprietárním vykreslovacím enginu postaveném na OpenGL ES 2. Bohužel se jedná technologii dostupnou pouze na OS Android, s využitím jazyka Java, a tedy nepřenositelnou a nedostupnou pro vývoj webových aplikací, resp. PWA. Google sice poskytuje i webovou knihovnu Google Maps pro jazyk JavaScript, ta je ale značně zastaralá, omezená a především rastrová. Tudíž nevhodná pro mobilní aplikace a Retina displeje s vyšším rozlišením a jemností.

4 ZABEZPEČENÍ

Abychom porozuměli úrovni zabezpečení PWA aplikací musíme se nejprve zaměřit na léty prověřený bezpečnostní model webových prohlížečů, aplikovaný na webové stránky, spolu s omezeními, kterým se musí vývojáři podrobit.

4.1 Zabezpečení a rizika webové platformy

Původním účelem webových prohlížečů bylo zobrazování volně šiřitelných textových dokumentů. Ty později mohly obsahovat i drobné skripty, jež ale měly vliv pouze na samotnou stránku, v níž jsou obsaženy. Je kriticky důležité, aby potenciálně škodlivý kód neopustil sandbox webového prohlížeče a nezískal přístup k počítači a datům uživatele. Tato bezpečnostní filozofie se stala klíčovou pro rozvoj webu do podoby, jaké známe dnes.

4.1.1 Bezpečnost z pohledu uživatele

Dnes prohlížeče dosahují ochrany uživatelů mnoha způsoby, z nichž hlavní je izolace každé načtené webové stránky, resp. každé otevřené záložky, do samostatného procesu. Ten tak nesdílí paměť s žádným jiným procesem a ani není možné, aby se procesy ovlivňovaly navzájem. Microsoft se svým prohlížečem Edge jde dokonce tak daleko, že pro enterprise zákazníky poskytuje virtualizaci každé záložky prostřednictvím Hyper-V [8]. Prohlížeč je tedy hypervisorem izolovaných a anonymizovaných kontejnerů z nichž by nemělo dojít k napadení nebo úniku.

V neposlední řadě byla zavedena řada bezpečnostních mechanizmů pro zachycení XSS a script inject útoků. Stahování skriptů a souborů webové stránky podléhá tzv. Content Security Policy [9] a Cross-Origin Resource Sharing [10]. Obě tyto policy hlídají tok stahovaných skriptů, povolují, které z nich se odpovídají politice aplikace a serveru který ji poskytuje. Dále monitorují HTTP GET a POST requesty, povolují lokální komunikaci v rámci stejného originu, nebo udělují výjimku bezpečným datovým tokům mimo pocházející mimo origin aplikace. Také zakazují, invalidují, či skrývají obsah komunikace podezřelé, nebo odchozí na neznámy server. Bezpečnost tudíž souvisí i s vhodnou konfigurací serveru, který servíruje soubory aplikace a diktuje míru striktnosti CORS a CSP.

Ovšem je potřeba zmínit že tak jako každý software, i webové prohlížeče podléhají občasným nálezům bezpečnostních děr. Ne však více než tímto problémem notoricky známá Java. Naštěstí vysoká kadence aktualizací a bounty programy vývojářů prohlížečů, na odhalování chyb, podchytávají tyto attack vectory v jejich prvopočátku, než je stihnou útočníci zneužit.

Webovou platformu tak můžeme označit za stejně bezpečnou, ne-li bezpečnější než platformy jiné.

4.1.2 Omezení Webových API

S rostoucí popularitou mobilních aplikací začal vznikat tlak na webovou platformu, aby umožňovala webovým aplikacím přístup k datům, senzorům a anténám zařízení. Na jedné straně tak stojí potřeby aplikací, které se bez těchto přístupů nemohou funkcionalitou rovnat s nativními instalovatelnými mobilními aplikacemi. Nemohly by tak vzniknout např. mapové navigace, aplikace zobrazující data z chytrých hodinek monitorujících sportovní aktivitu, nebo dálkový ovladač světel a obecně chytrých domácností. Na druhé straně pak stojí uživatelova bezpečnost, která tím může být narušena, neboť je nežádoucí, aby navštívená webová stránka začala nevědomky monitorovat uživatelovu GPS pozici, nebo ovládat okolní připojené Bluetooth zařízení.

Nakonec API pro přístup k, a manipulaci s GPS, Bluetooth, USB atd. vznikly. Jsou však navrženy s velkým omezením a častým doptáváním se uživatele o povolení. Například Web Bluetooth API [11] umožňuje komunikaci s Bluetooth zařízením, jeho výběr ale není kódu aplikace umožněn. Aplikace tak prohlížeč zažádá o spuštění vyhledávání s určitými parametry a filtry, např. pouze připojení k fitness náramkům, ale namísto odevzdání dat o všech okolních zařízeních aplikaci, je uživateli zobrazen dialog s tímto seznamem. Výběr tak spočívá na uživateli. Prohlížeč se posléze se zařízením spáruje, naváže spojení a aplikaci předá pouze otevřený komunikační kanál. Kontrolu nad spojením má po celou dobu uživatel a může aplikaci kdykoliv zakázat přístup k API.

Stejně je podchycena i IPC komunikace. Dřívější tvrzení že procesy prohlížeče se navzájem neovlivňují je potřeba rozšířit o upřesnění. Technicky vzato se procesy nemohou ovlivnit, protože každý z nich spouští renderovací engine a JavaScriptový interpret prohlížeče, v nichž je interpretována webová stránka, a ani jeden ze zmíněných enginů nemá důvod ke komunikaci s jiným renderovacím procesem. Nicméně některé webové aplikace umožňují otevření více oken s téže aplikací a pro tyto případy je žádoucí vzájemně komunikovat mezi všemi okny aplikace. Pro tyto potřeby existuje omezené webové API `window.postMessage()`, které sice umožňuje komunikaci, ale pouze regulovaně a napříč origin lokací. Tedy pouze mezi okny a worker procesy dané aplikace, resp. webové stránky, načtené z identické adresy.

Webové aplikace tedy mají přístup k celé řadě hardwarových součástí zařízení i do uživatelského prostoru v operačním systému. Přístup je ale striktně řízen prohlížečem a kontrolován

uživatelem. A to o dost striktněji než v případě nativních aplikací. Nemůže (resp. podhoubí pro to je značně nižší) tedy dojít k situacím kdy jednoduchá aplikace s jediným úkolem, např. svítilna, odposlouchává uživatelova osobní data, jak se tomu v minulosti na mobilních platformách několikrát stalo.

4.2 Zabezpečení PWA aplikace

PWA neboli Progresivní Webové Aplikace, jsou, jak již z názvu vyplývá, je v jádru jen a pouze webová stránka. Obohacená o skript dodávající stránce dynamičnost vykreslování obsahu a zpracování vstupů uživatele. To ze stránky činí tzv. aplikaci. Především je ale obohacena o několik souborů jako ikony a definiční manifest.json s metadaty, které se opírají o řadu nových webových standardů. Ty pak umožní takovou webovou aplikaci instalovat do zařízení jako jakoukoliv jinou aplikaci nebo program a následně ji spouštět prostřednictvím renderovacího enginu prohlížeče, avšak bez adresového řádku a záložek, které jsou pro prohlížeče tak typické. To aplikací činí Progresivní.

Bezpečnost takové aplikace je odvozena z bezpečnosti samotné webové platformy, na níž staví. Ale zmíněné nové standardy s sebou přináší několik dalších článků příslovečného řetězu a mohou otevřít prostor k jeho přetržení a tím pádem kompromitaci aplikace a dat.

4.2.1 Základní předpoklady instalace PWA

Tak jako každá webová stránka je i PWA doručena prostřednictvím protokolu HTTP. V posledních letech vznikl tlak na používání zabezpečených kanálů díky čemuž se upouští od používání běžného HTTP provozovaném na protokolu TCP a portu 80 a stále více se tlačí na zavádění alternativního HTTPS. Je to stále ten samý webový protokol, ale doručovaný skrze asymetricky šifrovaný SSL/TLS socket. Proto nemůže dojít k odposlechu komunikace, či dokonce záměně dat.

Standardy PWA přikazují použití výhradně protokolu HTTPS a zakazují použití HTTP. V praxi to znamená, že navštíví-li uživatel webovou aplikaci za použití HTTP, prohlížeč bude zcela ignorovat přiložený manifest.json a vše se bude jevit pouze jako obyčejná webová stránka s JavaScriptovou Single Page Aplikací. Naopak navštívení webu skrze HTTPS způsobí zobrazení „Add to Home screen“ banneru a zpřístupnění instalace webu uznaného jako progresivní webová aplikace. A díky instalaci po HTTPS je zajištěna autenticita zdrojových souborů aplikace.

To vše za předpokladu, že vyloučíme Man-In-The-Middle útoky. Ty už ale spadají do kategorie útoků na páteřní technologii asymetrického šifrování a TLS socketů. Alternativně může být uživatel ošálen k navštívení webové stránky skrze pochybný proxy server, například skrze odkaz v podvodném emailu. To ale opět přesahuje rámec zabezpečení PWA.

4.2.2 Bezpečnostní model PWA

Do rodiny PWA standardů patří i tzv. Service Worker, což je nadstavba specifikace Web Workers umožňující webovým aplikacím vytvářet dočasné background thready, obdoby vláken ve více vláknových programů. Workery lze využívat k náročným výpočtům, které by jinak blokovaly hlavní vlákno obsluhující uživatelské rozhraní. Každá PWA aplikace musí obsahovat přesně jeden Service Worker. Tento skript s rozšířenými pravomocemi je ovládán nezávisle na aplikaci a jeho hlavním účelem je dohled nad procesem instalace aplikace z internetu, selektivní cachování jejich souborů, obsluha běžící instance aplikace a správa souborové cache.

Implementace je volitelná a závisí na potřebách každé aplikace, ale nejzajímavější schopností je zachytávání a alterace HTTP requestů. Nikoliv všech, pouze své vlastní aplikace, z pochopitelných důvodů. A to nejen dodatečné XHR/Ajax GET/POST komunikace se serverovým API, ale kompletní kontrolu nad načítání souborů aplikace, včetně js, html, css, obrázku apod. Díky tomu získává aplikace kontrolu nad sama sebou a v případě špatného připojení k internetu se může aplikace rozhodnout k načtení zdrojů z cache, nebo se pokusit o souběžné získaní zdrojů jak z cache, tak i z internetu a obsloužit request nejrychlejším výsledkem.

To však nejsou jediné schopnosti Service Wokeru. Každý aplikace si může prostřednictvím Budget API naplánovat časově omezený běh pro synchronizaci se serverem, refresh cache, vytváření notifikací, odeslání zpráv, které nebylo možné dříve doručit v offline modu atd... A to vše na pozadí, bez nutnosti běhu aplikace.

Spouštění Service Workeru je plně v režii prohlížeče, a ačkoliv má více pravomocí nejedná se o bezpečnostní riziko. Souborová cache je opět pouze abstrakcí prohlížeče, nikoliv konkrétní složka na disku se kterou by mohl SW nakládat dle libosti. Není povolen přístup k filesystému ani k TCP či UDP komunikaci. Řada API navíc není ani v SW přístupná. Geolocation API je přístupné pouze z kontextu rendereru stránky. Nikoliv z Worker vláken. Jedná se o samozřejmý požadavek na ochranu soukromí uživatele. SW by se kvůli tomu mohl stát nástrojem pro špehování, byť je v omezeném časovém rámci. To bohužel činí problémy

aplikacím, které potřebují sledovat uživatelovu aktivitu právě při vypnutém displeji a s minimalizovaným oknem aplikace. Sportovní, běžecké a fitness monitorující aplikace kvůli tomu nelze vyrobit pro platformu PWA.

4.2.3 Integrita lokálních dat aplikace

Důležitou součástí každé aplikace jsou data a jejich perzistentní úložiště pro využití i při příštím spuštění. Dříve se k uchovávaní dat napříč sessiony zneužívaly Cookies, původně drobné zprávy směřované od serveru ke klientovi určené k snadné identifikaci sessionu, nebo např. předvolby jazyka v jaké se má aplikace servírovat. Z hlediska čistě klientského využití se ve své době jednalo o ne příliš šťastné řešení, ale nutné zlo.

Posléze byl uveden standard Local Storage přímo určený k ukládání dat aplikace pro využití v budoucnu i po zavření a znovuotevření prohlížeče. Jde o jednoduché nestrukturované key-value uložiště stringových hodnot a při každém přístupu k objektu `window.localStorage` dochází ke čtení nebo zápisu na disk. Z těchto důvodů není vhodné Local Storage využívat ke skladování strukturovaných objektů, polí, dat větších objemů a také k častým přístupům. Proto byl zaveden i standard IndexedDB integrující plnohodnotnou NoSQL databázi do prohlížeče. Přístup k ní je per-origin neboli každá doména může nakládat pouze se svými vlastními databázemi. Na stejné bázi funguje i Local Storage. Pokud tedy stránka například `example.com` zapíše `localStorage.lang='cs'`, tato hodnota bude přístupná i na `example.com/appv2`, nikoliv už však na `subdomain.example.com`, natož na úplně jiné doméně. Nemůže se proto stát, že by data jedné aplikace unikly a byly zpřístupněny jiné aplikaci nebo stránce, neboť každá je izolována v rámci originu. To zajišťuje bezpečnostní model webových prohlížečů

Prohlížeče tyto data ukládají jako soubory do zařízení uživatele, v místě instalace prohlížeče nebo v místě k tomu určené. Veškeré data jsou pak uložena prohlížečem v proprietárním binárním a zabezpečeném formátu k tomu určenému, nikoliv v plain textu. V případě Google Chrome se jedná o databázový formát LevelDb, jehož soubory nelze mimo samotný prohlížeč snadno otevřít a pozměnit, aniž by došlo k poškození a ztrátě. Tento způsob napadení pro přístup k datům aplikace je složitý a tudíž nepravděpodobný.

4.2.4 Zabezpečení dat za běhu aplikace a prohlížeče

Jelikož se útočník nedokáže snadno zmocnit dat externě, pokusí se o to otevřením vývojářských nástrojů po spuštění aplikace. Tento debugovací nástroj se tak bohužel stává

dvousečným mečem a dává útočníkovi náhled do databáze skrze šikovný prohlížeč. Samotná databáze IndexedDB není šifrovaná a neposkytuje žádné možnosti uzamčení obsahu ve chvíli kdy je načtena do prohlížeče. Doporučení pro nakládání s citlivými daty jsou v této situace dva.

1. Neukládat lokálně žádné citlivé informace. Ty by se měly vždy stáhnout ze serveru teprve poté co je zajištěný zabezpečený kanál mezi aplikaci a serverem a uživatel je autentizován. A tyto citlivé informace by měly zůstat pouze v paměti zařízení, a to pouze po dobu běhu aplikace.
2. Programátor aplikace si sám musí obstarat šifrování veškerých dat která potřebuje lokálně uložit. Využít k tomu může například Web Cryptography API [12], nebo šifrovacích knihoven třetích stran. Důležité ale je, aby se neukládaly šifrovací klíče čímž by došlo ke znehodnocení snahy o zabezpečení dat. Stejně tak by se neměly předvídatelně, pseudo-náhodně generovat uvnitř aplikace. Získání klíčů je pak možné například:
    a. Aplikace při spuštění vyzve uživatele k přihlášení. Po navázání zabezpečeného spojení server poskytne klientovi klíče k dešifrování lokálně uložených dat.
    b. Aplikace při spuštění vyzve uživatele k přihlášení pomocí biometrických sensorů zařízení či jiných důvěryhodných prostředků poskytovaných operačním systémem. To jsou například, čtečky otisky prstů na telefonech, rozpoznání obličeje Windows Hello, přihlášení se systémovým uživatelským účtem, pinem apod. K tomu je třeba použít webového standardu Web Authn [13]. Tato varianta zaručuje provoz i v offline.

4.3 Možnosti napadení

Tak jako IndexedDB, Local Storage i souborová Cache podléhá stejnému riziku napadení z prostředí vývojářských nástrojů prohlížeče. Webové stránky a Progresivní Webové Aplikace jsou interpretované za běhu a dev-tools poskytují JavaScriptovou REPL konzoli a interaktivní prohlížeč IndexedDB databází. Toho mohou útočníci zneužít, ale může k tomu dojít pouze v případě, že se počítače útočník fyzicky zmocní.

Druhou možností je phishing. Uživatel může být pod různými záminkami donucen, aby následovat postup, který ho zavede do dev-tools a vykonal úkony benefutující útočníka. Většinou se jedná o script injection útoky, kdy se útočníkův skript spustí v kontextu aplikace a

opět se tak otevře přístup k IndexedDB, cache apod. Proti phishingu se lze bránit jedině osvětou. Společnosti Google a Facebook nechávají do konzolí svých webových aplikací vypsat velký červený varovný nápis k odrazení nevědomého uživatele.

Co se týče narušení souborové cache, nebo databáze aplikace, ať už úmyslné, nebo způsobené náhodným výpadkem. Vždy záleží na kvalitě kódu, předvídání autora aplikace a rozsahu ošetřených chybových stavů. Každý vyspělý software by měl být schopen self-healingu. Neboli nahrazení chybějících dat. V případě PWA znovu stažením ze serveru. Patří to ke good practises ve vývoji jakéhokoliv software. I běžné desktopové programy mohou přijít o svá data, například agresivními utilitami pro očistu počítače od zastaralých a neužitečných souborů. V tom se PWA nijak neliší, ale mají jednou nespornou výhodu. A sice snazší distribuce zdrojových kódu díky umístění celé aplikace online. Pokud tedy dojde k nedostupnosti některého souboru z cache, dojde k fallbacku – stažení a případně znovu zacachování souboru z internetu.

4.4 Autentizace a autorizace

Komunitně směřované aplikace musí nad rámec svých funkcí zvládat i přihlašování uživatelů a jejich správu. Dříve běžným způsobem bylo ukládání hesel spolu s uživatelským jménem a případně i emailem do databáze. Jde však o častý zdroj problémů a úniků dat, navzdory dobrým snahám o zabezpečení hesla hashováním a skladováním pouze hashe hesla, nikoliv samotného plaintexotévho hesla.

Díky rozmachu sociálních sítí a jednotných účtů pro online služby a aplikace je dnes časté mít účet u velkým internetových společností jako Google, Facebook, nebo Microsoft. Protokol Oauth2 umožňuje drobným aplikacím využít serverů těchto velkých společností k autorizaci uživatelů. Tím aplikaci odpadá nutnost implementace registračních formulářů na frontendu, odesílání emailů a zpracovávání aktivací účtů na backendu. Především odpadá nutnost skladování hesel a citlivých informací, což je obzvláště velkým plusem po zavedení směrnice GDPR.

Do praxe je Oauth uváděn dnes již notoricky známými tlačítky „login with Facebook“ nebo „sign-up with Google“ a další, které nejen webové aplikace nabízejí vedle, nebo namísto svých vlastních přihlašovacích formulářů. Pro uživatele jde o zásadní zvýšení komfortu jelikož namísto vyplňování registračního formuláře, aktivace účtu a nutnosti pamatování si hesla, již stačí mít pouze jeden účet. Například s Microsoft účtem se tak uživatel může

přihlašovat do OS Windows, do dokumentů na office.com, nebo také do řady aplikací třetích stran, využívajících Oauth.

Princip Oauth spočívá v přihlášení se do účtu tzv. providera, který posléze poskytne třetí straně informaci o uživateli, o které si aplikace zažádá. Pokud to však uživatel schválí. Po započetí autorizačního procesu proběhne třístranná komunikace mezi zařízením uživatele, serverem aplikaci a serverem autorizačního providera. Uživateli se v pop-up okně zobrazí přihlašovací formulář providera, např. již jmenovaného Microsoftu, do nějž se uživatel přihlásí, provider poté v okně provede přesměrování na adresu v jejíž URL je umístěn token, který dle potřeby využívá backend či frontend aplikace k získání informací o uživateli, či k aktualizaci tokenu před jeho expirací. Výhodou je také, že pokud je již uživatel autentizován u providera, například pokud si ve vedlejší záložce prohlížeče prohlíží poštu na outlook.com, autorizace proběhne okamžitě bez nutnosti opakovaného přihlašování.

II. PRAKTICKÁ ČÁST

5 PRVOTNÍ KROKY VÝVOJE

Jedním z cílů této práce bylo vytvořit jednoduchou aplikaci implementovanou prostřednictvím technologií PWA a vhodně zvolených, či naimplementovaných podpůrných knihoven. Tato aplikace má demonstrovat výhody, ale i nevýhody oněch moderních webových standardů umožňující tvorbu aplikací, které jsou svým fungováním a funkcionalitou nerozeznatelné od aplikací nativních.

5.1 Motivace vytvoření mapové aplikace

Webové technologie a jazyky ke tvorbě webových stránek a aplikací byly historicky považovány za méně výkonné a dodnes sklízí kritiku od konzervativních programátorů. Ze zkušeností přecházejících, ale i výzkumem [17] a vývojem aplikací v rámci této práci chci toto tvrzení vyvrátit. Namísto vytvoření běžných "hello world" aplikací, např. zápisník poznámek, To-Do list apod., jsem se proto rozhodl vytvořit mapovou aplikaci.

Mapové prohlížeče nejsou ve světě webových stránek přílišnou novinkou. Doposud byly tvořenou rastrovou mřížkou před-generovaných JPG obrázků mapových podkladů neboli dlaždic, zatímco nativní mobilní aplikace využívaly dynamickou vektorovou grafiku díky technologiím jako OpenGL. Hlavní předností vektorových map je jejich flexibilita. Uživatel není svázán do skokových celočíselných měřítek zoomu, mapu může libovolně otáčet, přibližovat a naklánět. Při zvětšování se prvky na mapě plynule posouvají a prolínají.

Takto vykreslovaná vektorová grafika byla donedávna doménou nativních aplikací, protože je náročná na výpočetní výkon. Hlavním argumentem kritiků je údajná neschopnost jazyka JavaScript vykonávat časové a výkonově náročně úlohy. Částečně musím dát za pravdu, protože při práci s čísly je nezbytná obezřetnost, neboť datový typ Number pracuje pouze s 53 bitovými čísly. Na druhou stranu musím kontrovat, že výkon značně omezuje spíše jazyk HTML, respektive API pro manipulaci s jeho vyrenderovanou reprezentací – DOM. V případě mapy se můžeme od HTML ve velké míře oprostit a vykreslovat ji do speciálního elementu `<canvas>`, který umožňuje rychlou a výkonnou práci s grafikou. Ve spojení s dalšími z řady nových webových standardů vzniklých v posledních letech, jmenovitě WebGL, `requestAnimationFrame()`, Web Workers, můžeme náročné grafické operace vykonávat na pozadí ve vlákně neovlivňující hlavní UI vlákno. Dále lze plánovat, synchronizovat a seskupovat vykreslovací operace do stejných okamžiků, opakující se každých 16.6 milisekund, čímž lze dosáhnout překreslování v tzv. 60fps neboli šedesát-krát za sekundu. To lidskému oku stačí na to, aby se aplikace jevila jako plynulá.

Díky těmto novým pokrokům na poli webových standardů jsem se proto rozhodl vytvořit jednoduchou aplikaci, která uživatelům umožní získat svou aktuální polohu, sledovat ji v delším časovém úseku a tento záznam uložit do paměti telefonu. Dále umožní tyto GPS data vykreslit ve vektorové mapě a prohlížet si je spolu s jinými trasami [17]. Tím je pokryto několik klíčových aspektů

1. PWA – aplikace je naprogramována v jazyce JavaScript, HTML a CSS, za použití standardů, které umožňují vytvořit mobilní aplikaci z webové stránky navštívené internetovým prohlížečem.
2. Offline & perzistentnost – Veškeré zdrojové soubory aplikace a uživatelova data jsou po prvním spuštění staženy z internetu a uloženy na mobilním zařízení, kde přetrvávají bez nutnosti připojení internet.
3. Rozšířená funkcionalita – Aplikace bude přistupovat k GPS anténě mobilního telefonu a získávat uživatelovu polohu. Schopnost, která byly donedávna přístupná jen nativním mobilním aplikacím, nikoliv webovým stránkám.
4. Výkonnost – Mapa není rastrová, ale vektorová. Klade vysokou laťku na výkonnost a rychlost překreslování, podobně jako u nativních aplikací.

5.2 Prototypování

První krůčky při vývoji této práce, kromě studia standardů PWA, směřovaly k výběru vhodných knihoven pro realizaci požadavků na aplikaci. Web, na rozdíl od nativních aplikací, nedisponuje vestavěnými komponenty komplexních ovládacích prvků, jako toolbar, výsuvné menu, záložky, nebo třeba i takové banality jako obrázkové ikony. Prvním krokem proto byla instalace UI frameworku Flexus, mé dřívější práce, který poskytuje základní stavební prvky uživatelských rozhraní a snaží se věrně implementovat specifikaci Material Design [18], coby vzhled Android aplikací a webových služeb Google.

Poté jsem pro separaci logiky jednotlivých částí aplikace zvolil MVC Framework Aurelia, který abstrahuje práci s daty a jejich vykreslováním do šablon aplikace. Díky tomu jsem se mohl zaměřit především na vývoj mapové části aplikace, geoprostorové operace a rozšiřování aplikace o nové vlastnosti za pomocí standardů PWA.

Významnou součástí každé mapové aplikace je samotná mapa. Tu si vývojář nevytváří sám, ale využije poskytovatele mapových podkladů a knihovnu na jejich zobrazení od třetí strany. Při výběru jsem kladl důraz na čtyři hlavní faktory.

1. Kvalita mapových pokladů
2. Kvalita knihovny a API
3. Náklady na provoz
4. Otevřenost dat a platformy

Po implementaci řady experimentů nad knihovnou Google Maps jsem narazil na mnoho nepříjemných požití. Mezi ně patří například zastaralost knihovny, omezené vykreslování dodatečné grafiky a dat do map, rastrové dlaždice a absence dlaždic vyššího rozlišení pro mobilní telefony a displeje Retina s hustější pixelovou mřížkou atd. Tyto okolnosti mne přiměly od Google Maps upustit a přepsat aplikaci s pomocí knihovny Mapbox GL. Ta je modernější, pravidelně aktualizovaná, flexibilnější, a především a postavena na WebGL umožňující vykreslování map vektorově, s podporou GPU akcelerace. Společnost Mapbox sice stejně jako jiné mapové služby vydělávají na poskytování dat ze svých tile serverů, ale knihovnu Mapbox GL vyvíjí jako Open-Source a mapové podklady přejímá z veřejné služby Open Street Maps, což umožňuje budoucí rozvoj aplikace s využitím vlastního tile serveru.

5.3 Kostra aplikace

Prvotní prací bylo vytvoření základní kostry Single Page Aplikace se schopností routování. To znamená zobrazení adekvátního fragmentu aplikace a dat dle navštívené URL. Ty jsou ve formátu `#/view/id`, kde view je název načtené části aplikace a `id` je jednoznačný identifikátor zobrazeného obsahu, což je v případě mapové aplikace zaznamenaná trasa zakreslená do mapy.

Samozřejmostí jsou poziční ovládací prvky. V pravém dolním rohu umístěno tlačítko pro získání aktuální polohy, jehož stisknutím se vyvolá interakce s webovým Geolocation API [16], načež aplikace získá souřadnice zařízení a zaznačí je do mapy. Nad geolokačním tlačítkem se nachází tlačítko pro aktivaci a deaktivaci 3D vrstvy, která vykreslí budovy jako trojrozměrné obdélníky vystupující z mapy.

Pokud je aplikace zobrazena na dotykovém zařízení, kde je prostřednictvím dotykovým gestem dvou prstu možno pootočit uhel mapy, zobrazí se v pravém horním rohu také tlačítko s šipkou směřující k severu, čímž supluje kompas. Při jeho stisknutí se náklon vyresetuje a mapa se natočí zpět do výchozího úhlu. Dále bylo do aplikace doprogramováno dotykové gesto z aplikace Google Maps, které mění náklon mapy tahem dvou prstů

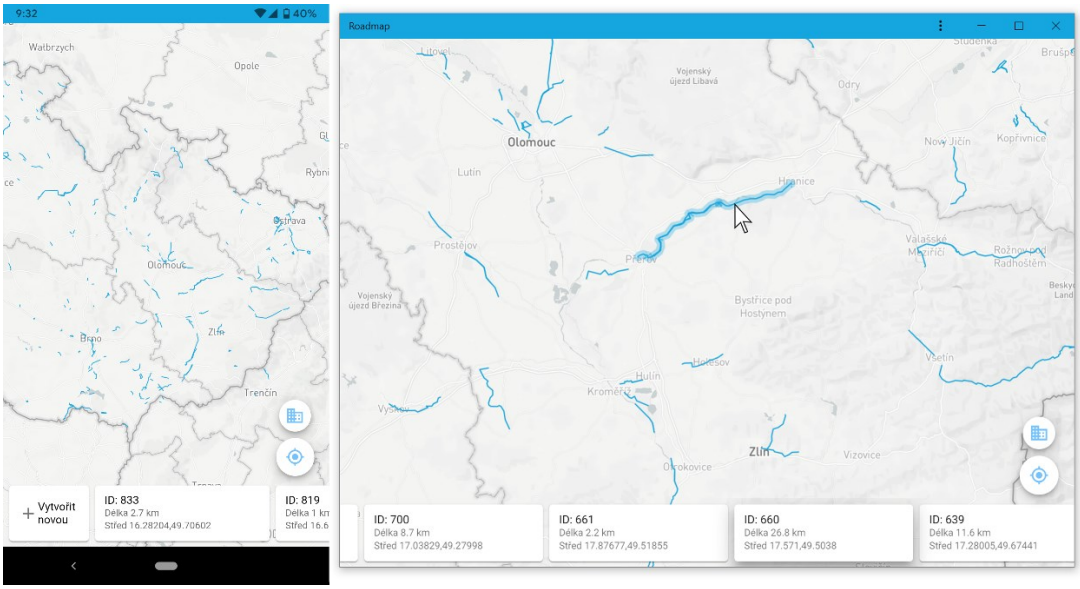
Následně bylo možné začít s rozšiřováním webové aplikace na Progresivní Webovou Aplikace skrze Service Worker a manifest, čímž je prohlížeči signalizováno, že se nejedná o obyčejnou webovou stránku. Prostřednictvím SW byla naimplementováno cachování zdrojových souborů i mapových podkladů, čímž aplikace získala offline mód. Dále byly odděleně vyvíjeny experimenty, z nichž na závěr vznikly ucelené view aplikace. Z práce také vzešla řada knihoven potřebné k realizaci aplikace. Jmenovitě `platform-detect`[3] poskytující informace displeji a zařízení, `mapbox-toolbox`[4] umožňující snazší práci s mapou a `exifr`[5] pro extrakci GPS souřadnic z fotografií.

Rozhodl jsem se neimplementovat žádnou komunitní funkcionalitu, ani backend, na kterém by byla aplikace závislá. A to z důvodu upřednostnění schopností standardů PWA, díky čemuž aplikace není závislá na specifickém backendu, ale všechny data a soubory si ukládá do lokální cache a IndexedDB. Vyjma první instalace z běžného statického web serveru.

5.4 Rozložení aplikace

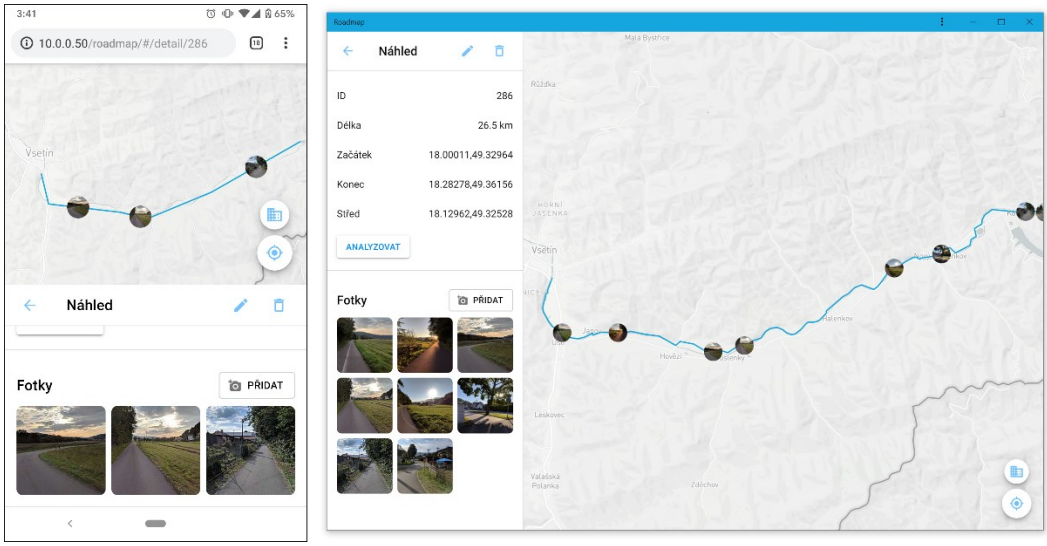
Úvodní stranou aplikace je prohlížeč všech tras zakreslených do jedné globální mapy. Při každém posuvu, nebo přiblížení v mapě se do URL zapíše aktuální poloha a zoom. Při opětovném spuštění tak dojde k otevření stejného místa, na které se lze i odkazovat zkopírováním a sdílením URL. Například `#/all/17.28798/49.61403/9.4` se zobrazí okolí Olomoucka. Spodní část prohlížeče tras je tvořena horizontálním seznamem viditelných tras. Seznam je aktualizován po každém přesunu či přiblížení. Lze tak snadno katalogizovat trasy podle jejich polohy pouhým pohybováním se po mapě.

[3] https://www.npmjs.com/package/platform-detect
[4] https://github.com/MikeKovarik/mapbox-toolbox
[5] https://www.npmjs.com/package/exifr



Obrázek 6. Úvodní strana aplikace, view #/all, na mobilu a desktopu.

Jak seznam, tak i vykreslené trasy jsou interaktivní. Reagují na najetí myší, čímž se dotyčná trasa, spolu s její kartou ze seznamu, zvýrazní. Při najetí myší na karty vespod také dojde k jemnému posunutí mapy směrem k označené trase. Nejde vycentrování trasy. Uživatel tedy není teleportován na zcela jiné místo a neztrácí kontext. Naopak, získává lepší přehled. Spolu se zvýrazněním trasy jde o vizuální pomůcku, která uživateli pomáhá nasměrovat pozornost ve změti velkého množství vykreslených tras. Po kliknutí na trasu či kartu se otevře detail konkrétní trasy.

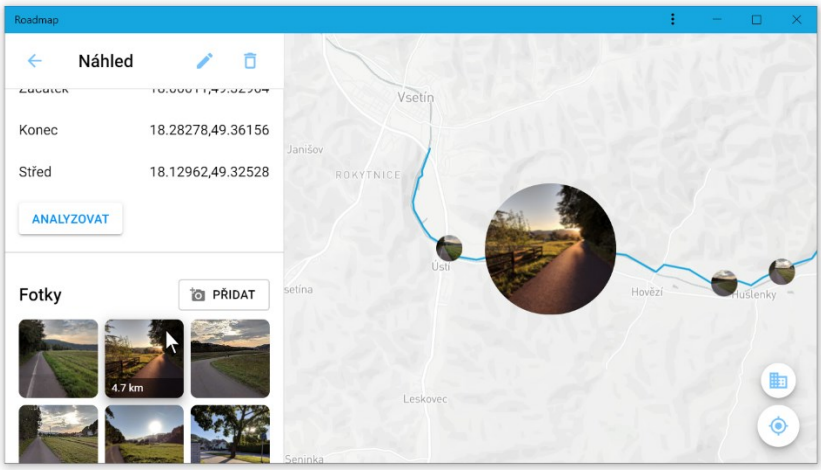


Obrázek 7. Detail trasy, coby mobilní webová stránka a desktopová PWA.

Detail disponuje responzivním panelem s informacemi a fotografiemi, který podle velikosti displeje mění velikost a umístění. Na menších displejích je panel přilepen ke spodní části

obrazovky, na větších displejích je vlevo. Do trasy lze vložit fotografie, z nichž je pomocí knihovny `exifr`[6] vytažena poloha pořízení a ta je pak vyznačena spolu náhledem fotografie na mapu. Tato problematika je dále rozebírána v kapitole 8.2. Po najetí myší na náhled fotky v panelu, či její miniaturu na mapě se obě položky zvýrazní a zobrazí vzdálenost fotografie od začátku trasy. Také dochází k jemnému posuvu mapy směrem k pozici fotografie, stejně jako na úvodní straně aplikace. Kliknutím na fotku se otevře ve vyšší kvalitě přes celou obrazovku. Kvůli povaze práce a absenci backendu se fotografie nikam neukládají a zůstávají přítomny jen po dobu běhu aplikace.

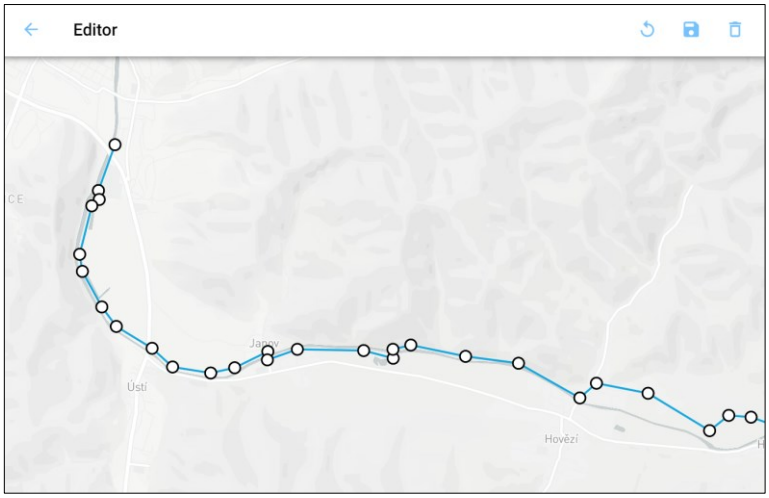
Náhledy fotografií vykreslené na mapě lze dle libosti přesouvat podél trasy, zůstávají však fixně k trase přilepena. Od kurzoru je při tažení vždy spočítána nejbližší vzdálenost k trase, kde je tažená fotka umístěna. Nemůže se tak stát, že by se octla mimo trasu. Po přesunu se zároveň přepočítá aktuální vzdálenost fotografie od začátku trasy a dle potřeby se seřadí náhledy fotografií v panelu. Vždy jsou tak popořadě.



Obrázek 8. Náhled na fotografie v detailu trasy.

Kliknutím na ikonou tužky se lze přepnout do editace současné trasy. Ten disponuje toolbarem s tlačítky k vrácení úprav, uložení změn, nebo smazání celé trasy. Editor rozpoznává kontext kurzoru a inteligentně přepíná mezi nástroji pro posun mapy, vkládání bodů, prodlužování trasy apod. Implementační detaily jsou přiblíženy v kapitole 7.

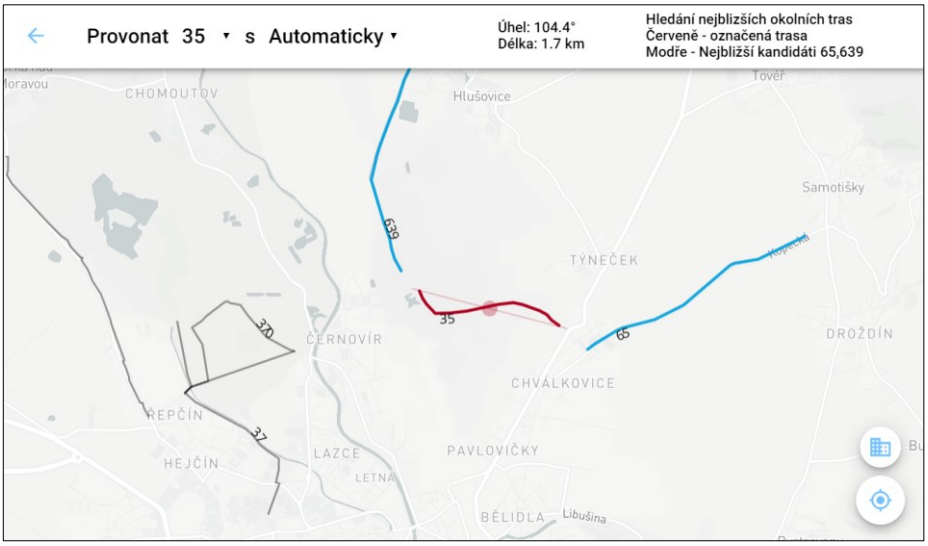
[6] https://github.com/MikeKovarik/exifr/



Obrázek 9. Editor trasy.

Aplikace je dále doplněna skrytým vývojářským view `#/dev` s informacemi o mapě, zařízení a výkonu aplikace, společně s nástroji k manipulaci mapových podkladů. Odsud lze zapínat a vypínat vrstvy vykreslené na mapě, jako cesty, řeky a zviditelnit hitboxy. Také je zde možno zjistit velikost cache a vymazat ji.

Za účelem demonstrace geografických výpočtů z kapitoly 9 byla také vytvořena sekce `#/comparison/id1/id2?/` přijímající ID jedné, nebo dvou tras k porovnání. Pokud je dodána jen jedna trasa, jsou k ní nalezeny nejbližší sousední trasy. Při zadání i druhé trasy jsou vzájemně porovnány a zjišťuje se zda se jedná o protínající se duplikáty.



Obrázek 10. Comparison view vyhledávající sousední trasy.

6 IMPLEMENTAČNÍ DETAILY

6.1 Cachování statických dat

Zásadní pasivní vlastností mapových aplikací je cachování, které překvapivě není ani v roce 2019 standardem. Potkáváme se s ní pouze v klientských aplikacích mapových poskytovatelů. Tedy pouze v aplikaci Google Maps pro Android. Nikoliv už však v žádných aplikacích třetích stran využívající Google Maps API. Osobně jsem se nevěnoval Javové knihovně, které pohání drtivou většina současných Android aplikací. Stačilo zjištění, že dle licenčního ujednání Google zakazuje ukládání dat na delší dobu a zároveň implmentuje no-cache direktivu spolu se striktním CORS omezením. Naštěstí přechod na knihovnu Mapbox mi umožnil cachování implementovat.

Samotná knihovna Mapbox GL toto však neumí a je potřeba vlastní implementace. Dříve by bylo nutné kvůli tomu dělat zásahy do jejího zdrojového kódu, tak aby odpověď každého zaslaného http requestu byla přijata jako datový typ `Blob` a následně převedena na `Array-Buffer`. Aby tento přístup fungoval, musel by navíc tile server u odpovědí přikládat hlavičku `access-control-allow-origin`. Bez ní by prohlížeč přijatou odpověď označil za nebezpečnou a zahodil ji, protože přišla z jiného serveru, než z kterého je načtená a spuštěná aplikace. Jedná se o zabezpečení CORS neboli Cross-Origin Resouce Sharing. To má přecházet nechtěným útokům na uživatele injekcí skriptu z jiného serveru. Bohužel také blokuje legitimní situace jako právě zmíněné cáchování map v prohlížeči.

Dnes lze cachování řešit mnohem snadněji díky Service Workeru. Tento skript běží na pozadí webové aplikace, je instalován při prvním spuštění a poté je kontrolován prohlížečem nezávisle od životního cyklu aplikace. Proto jsem byl schopen naimplementovat skript, který odchytává a alteruje veškerou HTTP komunikaci aplikace. Mohu tak ovlivňovat jaké data a za jakých podmínek jsou aplikaci předloženy.

V praxi to vypadá tak, že při načítání aplikace, nebo jakéhokoliv souboru, či odchozího HTTP requestu je v SW vygenerován event `fetch`, který zpracovávám a následně rozhoduji. Pokud jde o obrázek, nebo soubor s příponou .js, .css, .html a zároveň pokud je načítán z aktuálního originu, tedy serveru, což je v případě vývoje localhost, potom je request předán prohlížeči k vykonání. Následně po odeslání a stažení odpovědi ji zároveň vrátím aplikaci, ale také vložím do cache aplikace. Díky tomu je poté každý budoucí identický request obsloužen z cache, a nikoliv stahováním ze serveru. Aplikaci tak lze načíst i bez přístupu

k internetu. Uživatel může spustit internetový prohlížeč a vstoupit na stránku aplikace, která se mu načte jako by byl připojen k internetu. Stejně bude aplikace fungovat i když ji nebude navštěvovat skrze prohlížeč, ale pokud si jí před-instaloval na svém zařízení.

Obdobným způsobem jsou zpracovávány i žádosti na server mapbox.com. Pokud bychom spustili nacachovanou aplikaci v offline módu, ale neměli bychom přístup k mapovým dlaždicím, nebylo by možné aplikaci používat. Proto každý odchozí GET request vedoucí na server api.mapbox.com po vykonání také uložím do oddělené cache. Díky tomu můžeme odděleně promazávat, či aktualizovat cache aplikace v případě zastaralosti, aniž bychom přišli o mapová data. A naopak, uživatel může promazat mapovou cache, aniž by přišel o aplikaci. Při zjišťování přítomnosti mapové dlaždice v cache navíc z URL odstraňuji access token svázaný s osobním vývojářským účtem u Mapboxu. Pokud by došlo k jeho změně, nebylo by možné již možné přistoupit k nacachovaným datům, kvůli odlišné URL, byť by je nebylo třeba aktualizovat.

6.2 LocalStorage a IndexedDb

Poté co aplikace disponuje statickými daty nutným k jejímu spuštění je potřeba ukládat i data generované uživatelem. K tomu jsem mohl využít dvou, respektive třech standardů. Prvním je LocalStorage a z něj vycházející SessionStorage, avšak jak již z názvu vyplývá, SessionStorage data ukládá pouze po dobu trvání aktuální session, tedy po dobu běhu aplikace. Ta nepoběží ve více instancích, a proto všechny data potřebné pro běh aplikace zůstávají v paměti aplikace. Pokud tyto data potřebuji uložit, zejména nastavení aplikace, ukládám je do LocalStorage, dostupný skrze `window.localStorage`. Jedná o objekt skladující jednoduché stringové hodnoty pod stringovým indexem. Toto omezení lze obejít serializací složitějších dat do formátu JSON a teprve ten do storage objektu umístit.

Elegantnějším řešením u komplikovanějších, a především sériových dat je využití databáze IndexedDB. Jde o takzvanou NOSQL databázi, kde k vykonávání operací nad ní není potřeba vytváření SQL query. Namísto toho jsou volány metody. Také interní struktura je odlišná a oproti tabulkám z SQL zde používáme tzv. store, které nemají předem definované sloupce a typy. Lze tak do nich ukládat libovolně strukturované objekty a každému z nich přiřadit ID. Podobně jako u SQL databází se tak ke každému objektu můžeme přistupovat jako k řádku tabulky.

Tento typ úložiště je ideálním pro skladování tras vytvořených uživatelem. Kvůli knihovně Mapbox a zájmu využívat otevřených standardů jsem se rozhodl každou trasu skladovat

v objektu předepsaným standardem GeoJSON. Je proto snadné jej kdykoliv serializovat do textového formátu JSON a uložit na disk, nebo poslat serveru. Díky tomu, že IndexedDB je schopen přijímat celé objekty, je snadné celou trasu bez serializace přímo uložit do databáze, nebo naopak relativně rychle přistoupit ke kterémukoli objektu z databáze.

6.3 Plugin pro knihovnu Mapbox

Knihovna Mapbox má v porovnání s populárními Google Maps podstatně širší možnosti pro vykreslování dat a práci s nimi. API knihovny sice tvůrcům do rukou vkládá mocné nástroje s velkým množstvím možností, nastavení, ale bohužel také příkazy, jak své požadavky strukturovat. Výsledkem je nutnost volání několika komplikovaných, často se opakujících funkcí k vytvoření alespoň základních prvků, jako je úsečka, nebo bod. Na následujících příkladech je vidět kód nezbytný k vytvoření šedé trasy. Poté je pozměněna tloušťka, barva je nahrazena přechodem červené, zelené a modré barvy, a následně je celá vrstva schována.

Prvním krokem je vytvoření tzv. zdroje dat, který musí být formátu GeoJSON, což samo o sobě není příliš intuitivní v průběhu vývoje a experimentování, kdy nejčastější formou dat je pole souřadnic. Často tak vytvoření zdroje dat předchází ještě přetransformování do formátu GeoJSON.

```
var geojson = {
  type: 'Feature',
  geometry: {
    coordinates: [[14.81893, 50.29038], [14.81893, 50.29038], ...]
  }
}

var id = 'the-line'
map.addSource(id, {
  type: 'geojson',
  lineMetrics: true,
  data: geojson
})
```

Po vytvoření zdroj dat můžeme přistoupit v k vytvoření grafické vrstvy, která vykresluje data z předem definovaného zdroje. Pro jejich propojení si musíme udržovat ID zdroje. V grafické vrstvě je pak potřeba definovat položky v objektech paint a layout. Oba sice ovlivňují vizuální podobu vrstvy, ale ne vždy je jasné, které položka náleží které skupině.

```
map.addLayer({
  type: 'line',
  source: id,
  id: 'the-line-layer',
  paint: {
```

```
    'line-color': 'black',
    'line-width': 2,
  },
  layout: {
    'line-cap': 'round',
    'line-join': 'round'
  }
})
```

V třetím kroku je pozměněn vzhled trasy voláním metod na změnu paint a layout vlastností. Pokud chceme upravit místa kudy trasa prochází, musíme měnit zdroj dat. Při změně barvy měníme položky paint v grafické vrstvě, ale pro dočasné schování vrstvy musíme měnit položku layout. Ta navíc ani není typu boolean.

```
map.setPaintProperty('line', 'line-width', 6)
map.setPaintProperty('line', 'line-gradient', [
      'interpolate', ['linear'], ['line-progress'],
      0, 'red', 0.5, 'green', 1, 'blue'
])
map.setLayoutProperty('line', 'visibility', 'none')
```

Obdobné problémy se týkají změn souřadnic a polohy vykreslené vrstvy. Zde je třeba zasáhnout do datové vrstvy. Tento kód byl velice dlouhavý, repetetitivní a náchylný k rozbití kdykoliv jsem si nebyl jist přesným zněním a posloupností příkazů a položek. Výsledkem pak byla nejen nízká efektivita práce, ale především útlak kreativity, protože vyzkoušení každého, byť banálního nápadu, mohlo trvat desítky minut hledání v dokumentaci. Z těchto důvodů jsem nejprve pomocné wrapper funkce zabalil do samostatného modulu, z nějž jsem vytvořil knihovnu, resp. plugin `mapbox-toolbox`[7], který rozšiřuje existující instance třídy Mapbox map a zjednodušuje práci s nimi. Jde především o vykreslování čar, bodů, textů a snadnou manipulaci s existujícími vrstvami a dat v nich. Předešlý příklad s nativním API knihovny Mapbox lze prostřednictvím mého pluginu přepsat následovně.

```
var coords = [[14.81893, 50.29038], [14.81893, 50.29038], ...]
var line = map.renderLine(coords, 'black', 2)
line.width = 6
line.gradient = ['red', 'green', 'blue']
line.hide()
```

[7] https://github.com/MikeKovarik/mapbox-toolbox

7 PROHLÍŽENÍ A ÚPRAVA GEOGRAFICKÝCH DAT

Pro svou komplexnost je těžké navrhnout uživatelské rozhraní editoru v GIS aplikacích. Na rozdíl od běžných tabulkových nebo slovních procesorů, či jiných kancelářských nástrojů, geografické systémy nepoužívají klávesnicí ovladatelný kursor. Tím je prostor interakce s uživatelem omezen pouze na pohyb myší, případně pár tlačítek klávesnice jako ESC nebo DELETE pro dokončení, či potvrzení akce předem vyvolanou právě myší.

Například nakreslení trasy spočívalo ve vytvoření několika propojených bodů. S myší je to snadný úkol, neboť jde o velmi přesnou periferii a výběr místa můžeme korigovat s přesností na pixel až do chvíle, než místo označíme stiskem tlačítka na myši. Rychlý přesun umožňuje vytvořit sérii takových bodů v krátkém čase bez nutnosti dalších úprav. Na druhou stranu dotyková zařízení zpravidla neumožňují přesný výběr místa. Průměrný lidský prst má velikost přibližně jeden centimetr v průměru. Při dotyku na displej to odpovídá asi 40 až 50 pixelům. Material Design Language od Google apeluje na vývojáře mobilních aplikací, aby každá interaktivní plocha měla alespoň 48 pixelů na šířku i výšku. Pro statické prvky uživatelského rozhraní menší, než 48px existují možné řešení ve formě neviditelného hitboxu. Výběr konkrétního místa na mapě je ale velmi obtížný, ne-li nemožný. Dotykem prstu na místo určení vzniká oblast již zmíněných asi 40 pixelů v průměru, odkud je interpolován jeden konkrétní bod ze středu dotyku. To může být až 20px od původně zamýšleného místa, což může v závislosti na měřítku činit odchylku klidně i stovek metrů.

7.1 Prohlížeč, adaptivní hitbox

Nejprve bylo nutné vyřešit označování vykreslených tras. Na úvodní stránce aplikace se do mapy vykreslují uživatelovy zaznamenané trasy. Budeme-li chtít otevřít detail jedné z nich, je potřeba na ni kliknout. Pro obsluhu dotyků na obrazovkách mobilních telefonů jsem zvolil již zmíněné hitboxy. Tento přístup spočívá ve vytvořené neviditelné oblasti okolo prvku uživatelského rozhraní, který má být klikatelný. Teoretické znění je až příliš ideální. Narazil jsem však při praktickém zpracování, a to hned z několika důvodů.

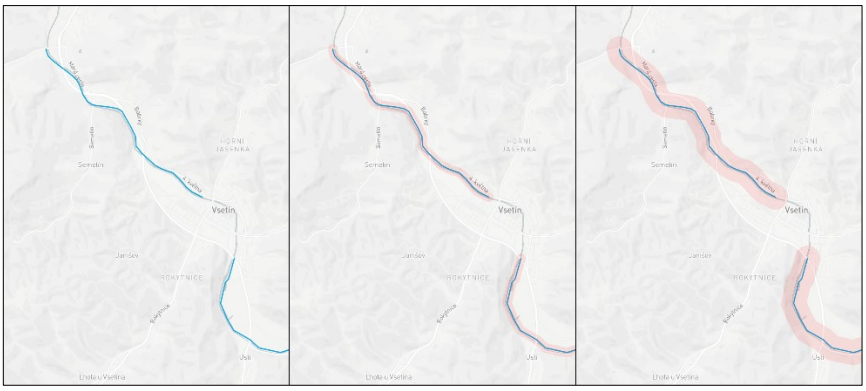
Je snadné vytvořit hitbox okolo obdélníku, jako například tlačítko, a obecně kolem statických prvků uživatelského rozhraní. Těm lze v css přidat pseudo selektor `::before` nebo `::after`, kterým do elementu injektujeme dodatečný obsah, což je v případě hitboxu průhledná plocha, absolutně napozicovaná negativně k okrajům elementu, čímž vznikne překryv přes okraj původního elementu o zvolený počet pixelů. Vhodně zvolenými výpočty tak

lze např. 32px vysoké tlačítko překlenout o -8px zespod i z vrchu, čímž vznikne klikatelná oblast o výšce 48, tedy ideální plocha pro dotyk prstu na dotykové obrazovce.

Avšak u mapových aplikací vykreslujeme polygony a trasy všemožných tvarů, na které nelze aplikovat předchozí postup. Mnohdy je vektorová grafika ve formátu SVG, které je s HTML a CSS kompatibilní jen částečně. V případě Mapboxu, ale nevznikají žádné hmatatelné elementy, protože tato knihovna vše vykresluje do jednoho rastrového obrázku uvnitř elementu `<canvas>`. Ten je z hlediska aplikace plochý, bez potomků a události zachycené na canvasu nelze granulárně rozlišit. Mapbox sice obsahuje mechanismy potřebné pro zachytávání eventů nad vykreslenou čarou, pokud je ale tři pixely tlustá, pak má problém spíše uživatel se do tak úzkého prostoru trefit myší. Proto bylo potřeba existující grafickou vrstvu trasy překrýt identickou vrstvou s dvěma rozdíly. Barva je průhledná, aby větší tloušťka hitboxové trasy negativně neovlivnila vzhled. Díky sdílenému zdroji dat pro obě vrstvy, je každá změna tvaru trasy promítnuta do obou grafických vrstev, přičemž vrchní hitboxová slouží pro odchytávání událostí.

7.2 Adaptivní hitbox

Na obrázku je vidět aplikace se zapnutým vývojářským módem, který v průsvitné červené barvě znázorňuje jinak neviditelný hitbox obalující trasu. Napravo optimalizovaný pro dotykové obrazovky, uprostřed pak smrsknutý na menší tloušťku pro zařízení ovládané myší. Takto jsou v aplikaci obalovány všechny trasy i na počítačích bez dotykové obrazovky, neboť samotná trasa je příliš tenká na to, aby komfortně posloužila pro naslouchání na události pohybu myši.



Obrázek 11. Hitbox skrytý a optimalizovaný pro myš a dotyk.

Velikosti uživatelských rozhraní se dnes běžně přizpůsobují pro dotykové obrazovky pouze prostřednictvím triviální detekce, zda je displej dotykový. To není ideální řešení. Zajímavý

problém představují hybridní zařízení 2v1 s oddělitelnou klávesnici, nebo laptopy s dotykovou obrazovkou. Většina aplikací hned při spuštění zkontroluje dotykovost a nebere v potaz, jestli je k zařízení připojena klávesnice, nebo myš. Automaticky předpokládají, že hlavním způsobem interakce uživatele se zařízení je dotyk, zatímco uživatel může v tu chvíli chtít aplikaci ovládat touchpadem na připojené klávesnici. Tento problém bychom mohli přejít, řekneme-li, že takové hybridní zařízení je z 50 % času používáno výhradně jako tablet. Fatální dopad to ale má na dotykové laptopy, kde klávesnici nelze oddělit a v 90 % času slouží jako primární input type, zatímco dotyková obrazovka je jen příjemným zpestřením interakce, kdy uživatel jednou za čas natáhne ruku pro stisk tlačítka. Pokud by aplikace zvolila automaticky velikosti uživatelského rozhraní podle přítomnosti dotykové obrazovky, byl by uživatel takového laptopu odkázán většinu času používat kurzor myši pro navigování po obrovské aplikaci plné prázdného místa, neboť byla navržena pro tablety, nikoliv pro hybridní zařízení.

Protože jsou PWA především webové stránky navštívitelné z jakéhokoliv zařízení, bylo pro mne důležité zajistit, že bude správně fungovat na libovolných zařízení rozličných velikostí. K tomu částečně slouží i knihovna `platform-detect` vyvinuta v rámci této práce. O ní ale vice v kapitole 10. V kontextu prohlížeče a editoru tras je potřeba vyzdvihnout adaptivnost hitboxu tras a klikatelných prvků vykreslených do mapy.

Na laptopech s dotykovou obrazovkou je nutné omezit velikost hitboxů na minimální tloušťku kvůli komfortnosti při používaní kurzoru myši. Pokud by byl hitbox příliš velký, způsoboval by překryv sousedních tras. I přesto je ale žádoucí ve chvíli dotyku prstem na obrazovku zanechat hitboxy široké, aby na nich byl dotyk zaregistrován. Pro tyto potřeby však neexistuje žádný prostředek, a proto jsem v aplikaci implementoval důmyslnou kombinaci naslouchání na `mouse*` a `touch*` událost v kombinaci s časovačem reagující na posuv kurzoru. Ve chvíli delší neaktivity myši je hitbox rozšířen v očekávání, že uživatel může zvednout ruku a natáhnout ji k displeji. Naopak v případě odchycení dotykových událostí `touch*` je okamžitě hitbox zvětšen.

7.3 Editor

Důležitou součástí každé pokročilé mapové aplikace je kromě zobrazovacího módu také mód, ve kterém je možné provádět úpravy. Mapbox obdobně jako Google maps poskytuje kreslící nástroj, ten ale není uzpůsoben pro dotykové ovládání. Obsahuje navíc řadu nadbytečných nástrojů jako kreslení polygonů, zatímco nástroj kreslení čar a tras se pro potřeby

aplikace jevil jako nedostatečný. A to jak z hlediska funkcionality, tak také z hlediska User Experience. Dále jsem díky předchozí znalosti knihovny Mapbox a jejím interním procesům se zpracovávání GeoJSON dat pojal podezření, že plugin MapboxGL Draw[8] není nikterak dobře optimalizován. Stačí srovnat ukázkové aplikace aktuální verze projektu oproti předchozí generaci[9] Leaflet Draw[10]. Proto jsem se rozhodl implementovat vlastní editor, aplikovat v něm adaptivní hitbox, pokusit se navrhnout praktičtější ovládání a navýšit tak uživatelský komfort, s důrazem kladeným na efektivnější práci s daty a snížení operací potřebných pro vykreslení.

7.3.1 Výkonnostní optimalizace

Záznam nových tras je poměrně snadný. Do mapy se postupným klikáním na prázdná místa zanese nový bod a ten se propojí s existující trasou. To lze provést I bez dedikovaného editoru, pouze za pomocí odchytávání událostí `click`, nebo více specifickými `mousedown` či `touchstart`. Kliknutím na mapu získáme pixelovou souřadnici X/Y uvnitř kontejneru mapu, z níž přepočítáme geografickou souřadnici LON/LAT. Tuto souřadnici pak můžeme zařadit na konec seznamu bodů popisující trasu. Mapová knihovna se následně postará o překreslení trasy rozšířením o nově popsanou úsečku mezi předposledním a posledním bodem. Pokud bychom chtěli provést změnu, vzniká nám celá řada problémů. Opravit nepřesnost posledního bodu je zdánlivé triviální. Stačí se sáhnout na vrchol zásobníku souřadnic, tuto hodnotu vzít, upravit a nahradit. Pokud by se jednalo o jakýkoliv jiný bod, přichází na řadu detekce, o který bod se jedná, dohledání jeho hodnoty a pozici v zásobníku souřadnic a nahrazení za novou. Pro vykreslení všech souřadnic, z nichž se trasa skládá bylo potřeba vyextrahovat všechny souřadnice z GeoJSON dat trasy, neboť je ve formátu `LineString`. Ten není kompatibilní s Mapboxovým API pro vykreslování kruhových bodů, a proto jsou všechny souřadnice obaleny do GeoJSON formátu `Point` a dále seřazeny v kolekci `FeatureCollection`. Vzniká tak potřeba neustále synchronizovat změny pozic v obou seznamech souřadnic, jak v datech trasy, tak v kolekci pointů, v případě že dojde ke změně odstraněním, nebo tažením jednoho z vrcholů trasy.

---

[8] https://github.com/mapbox/mapbox-gl-draw
[9] http://leaflet.github.io/Leaflet.draw/docs/leaflet-draw-latest.html
[10] Leaflet.js je rastrová mapová knihovna od tvůrců MapboxGL, která předcházela právě aktuální MapboxGL.

K tomu bylo potřeba vrcholy vykreslit a navěsit na ně event listenery, při jejichž vyvolání by se zmíněné změny prováděly. Zde se složitost problému dále větví. U rastrových mapových knihoven je tato nově vytvořená vrstva reprezentována samostatným DOM stromem. V něm každý bod můžeme reprezentovat samostatným elementem, např. SVG element `<circle>`, a na něj pak listenery navázat. Dotykem na element prohlížeč vytvoří event, který se provolá do správného listeneru a my tak získáme konkrétní prvek a na něj navázaný bod se souřadnicí. V případě Mapboxu je tomu jinak, protože si vše knihovna sama vyrenderuje do `<canvas>`. Naštěstí eventy nad jednotlivými prvky mapy virtualizuje tím, že si v paměti ukládá jejích místo, podle pozice myši nalezne vrstvu vykreslenou na dané pozici a do ní vyšle event. Zde tak pracujeme s událostmi vyvolanými nad celou vrstvou, a nejen jednotlivými body. Proto bylo možné všechny vrcholy trasy seskupit do kolekce `FeatureCollection` a tu následně zaregistrovat jako zdroj dat prostřednictvím `map.addSource()`. Z něj se vykresluje grafická vrstva bílých kruhových bodů s černým okrajem. Jejich velikost je ale malá, v průměru 16px, aby nezabíraly příliš mnoho místa v uživatelském rozhraní. Na druhou stranu je tato velikost nevyhovující pro dotykové zařízení. Proto jsem podobně jako v případě prohlížeče tras využil techniku hitboxů přidáním druhé vrstvy, sloužící k zachytávání eventů. Z jednoho sdíleného zdroje, se tak vykreslují dvě samostatné grafické vrstvy. V případě změn souřadnic v kolekci bodů, z níž je sdílený zdroj vytvořen, se změny projeví v obou grafických vrstvách a hitboxy tak vždy překrývají viditelný bod.

Toto řešení, vyjma hitboxu, je obdobné jako implementace v pluginu Mapbox Draw. Funguje obstojně s kratšími trasami s menším počtem. Důvodem je API Mapboxu, které neumožňuje granulární úpravy nad daty a vynucuje si kompletní nahrazování dat a následné překreslení, ačkoliv je změněna jen jedna souřadnice. Uvažujeme-li modelový případ trasy 10 km dlouhé, s rozlišením 10 bodů na kilometr, jedná se o 100 bodů. Každá souřadnice v textové a nezaokrouhlené formě zabírá 36 znaků, resp. Bajtů, celá trasa tak zabírá 3,6 kB, což je stále zhruba jen třetina přenesených dat, protože posíláme identická data, ale ve dvou rozdílných formátech. Jeden pro vykreslení úseček trasy, a druhý pro body vrcholů. Kolekce vrcholů ještě přibližně dvakrát tolik nabobtná kvůli syntaxi GeoJSON obalující data. Upravujeme-li v trase jeden bod tažením na jiné místo, vygenerujeme po sekundě asi 50 eventů, které zapříčiní 50 změn a překreslení. To celkem činí nepřijatelných 500 kB přenesených z hlavního UI vlákna do vykreslujícího Web Workeru knihovny Mapbox. Sdílení dat mezi procesy a vlákny nejde jednoduše obejít a stává se bottleneckem pro celou aplikaci. Pokud je špatně navržena. Vyšší programovací jazyky bez adresovatelné paměti skrze pointery se

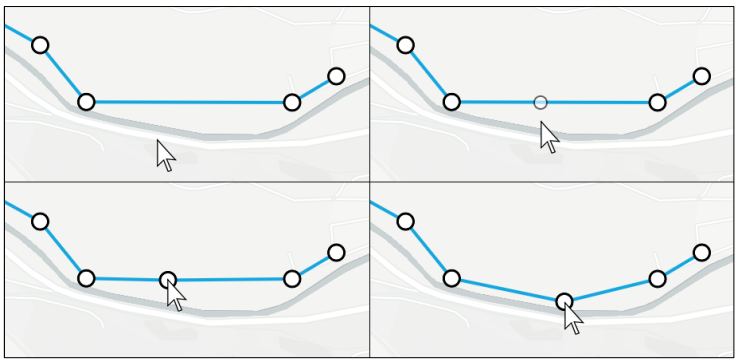
musí spokojit s vestavěnými prostředky pro IPC komunikaci. Ty jsou sice ve webovém API dostačující ale knihovna Mapbox nemá otevřené API pro efektivnější přenášení dat mezi uživatelským kódem a Web Workerem ovládaným knihovnou. Kvůli povaze struktury dat souřadnic je není možné efektivně přenášet prostřednictvím binárního typu ArrayBuffer. Navzdory tomu by ještě bylo nutné nad bufferem vytvořit Uint8Array náhled a každou souřadnici opakovaně transformovat z desetinného čísla na 32 bitů, resp. zaokrouhleně 24 bitů. Ačkoliv je to efektivnější způsob pro ukládání geografických dat, při práci s nimi je potřebná jednoduchá manipulace se souřadnicemi. Proto tento problém nemá jednoznačné řešení a záleží na potřebách aplikace.

Pro editor této aplikace jsem se rozhodl vyčleňovat neaktivní data do samostatných vrstev, se samostatnými GeoJSON zdroji dat. Namísto jedné vrstvy pro vykreslení trasy vzniknou hned tři, z nichž dvě jsou na počátku prázdné a neaktivní. Po stisku jakéhokoliv vrcholu trasy identifikuji index vrcholu, jeho lokaci, a především okolní dva sousedy. Z těchto třech bodů vzniknou dvě úsečky na samostatné vrstvě dedikované pro aktivní data. Zbylé dvě větvě trasy rozdělené ve vybraném vrcholu jsou pak vykresleny do zbývajících dvou vrstev, které již není potřeba překreslovat. Při tahu vrcholem se tak vždy přenášejí pouze tři souřadnice do jedné ze tří vrstev, oproti původním stovkám souřadnic do jedné vrstvy. Tím se zvýšil výkon aplikace snížením vizuální odezvy a velikosti přenesených dat mezi vlákny.

7.3.2 User experience

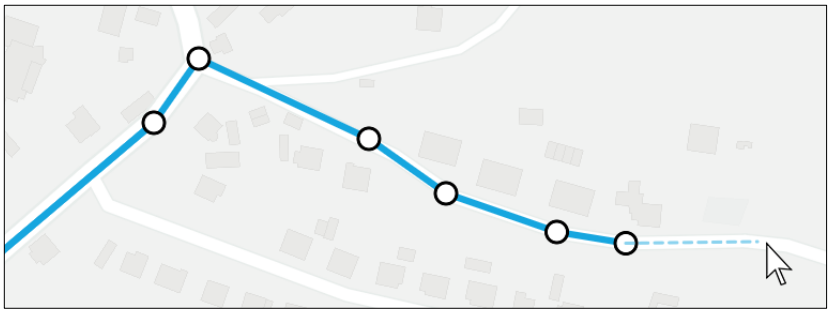
Zásadní problém s existujícími editory je z mého subjektivního hlediska jejich user experience, konkrétně zbytečná komplikovanost. Jedinou funkcí editoru je pouze nakreslit několik čar, ale k dosažení se musí uživatel přepínat mezi nástroji, nebo módy, pro posuv mapy, přidávání nových bodů, úpravu existujících apod. V této práci jsem se rozhodl implementovat intuitivnější řešení.

S každým pohybem myši se přepočítává poloha kurzoru vzhledem k trase. Na ní se vyhledá nejbližší bod a pokud je v dosahu, vytvoří se na ní duch. Poloprůhledný virtuální nástin, kde by mohl být umístěn nový bod, pokud by v tu chvíli uživatel klikl a bod vytvořil. Duch se posouvá spolu s kurzorem myši, ale zůstává stále přilepen k trase. Pokud se kurzor vzdálí od trasy, duch zmizí. Pokud se kurzor blíží existujícímu bodu, taktéž zmizí. Po kliknutí zůstává nově vytvořený bod v aktivním stavu a lze s ním manipulovat a dále jím táhnout. Díky tomu lze vkládat nové body nejen na konec trasy, ale i doprostřed již existujících úseků a ty tak dále zpřesňovat.



Obrázek 12. Vkládání bodu prostřednictvím ducha.

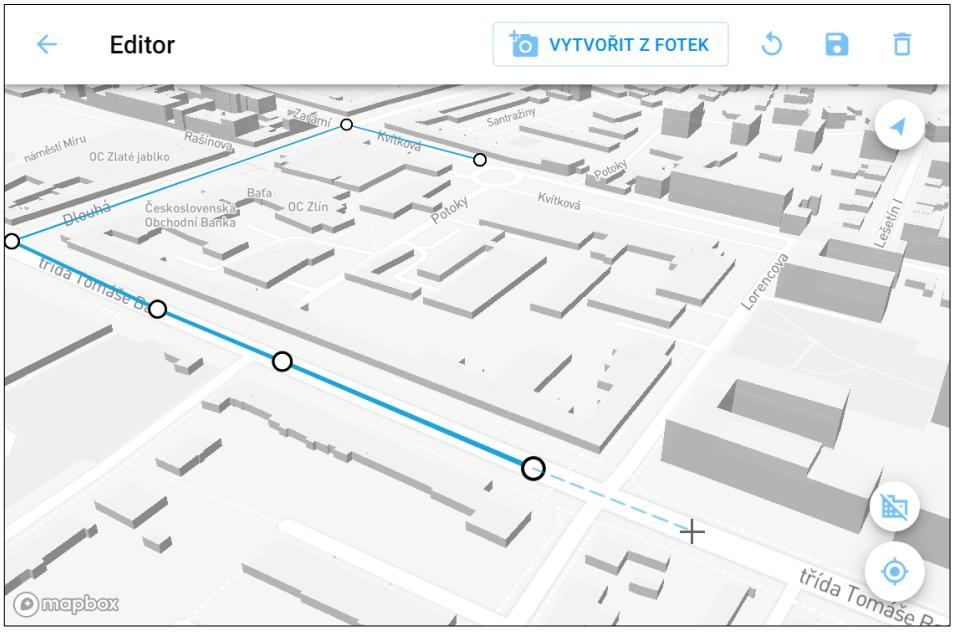
Pokud kurzor přesahuje začátek nebo konec trasy, začne se mezi daným okrajem a kurzorem vytvářet nástin navazujícího úseku. Po kliknutí se bod ukotví a trasa je prodloužena. Takto je možné trasu libovolně kreslit pouhým klikáním po mapě, namísto přepínání mezi nástroji. Editor podle kontextu inteligentně přepíná mezi posuvem mapy, manipulací bodů, vkládáním bodů mezi již existující body, nebo prodlužování trasy přidáváním nových bodů.



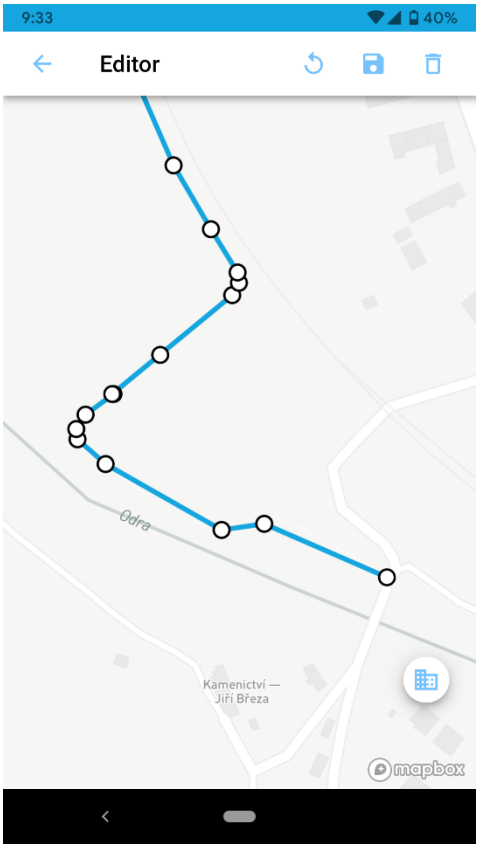
Obrázek 13. Prodlužování trasy.

Díky architektuře aplikace, editoru i použité knihovny Mapbox lze editor kombinovat se funkcionalitu, která by jinak byla oddělena do samostatné části aplikace. Například 3D budov, nebo libovolná rotace a náklon mapy nemusí sloužit jen pro prohlížení tras, ale i při vytváření. Souřadnice kurzoru myši na displeji se vhledem k úhlu a náklonu adekvátně přepočítají na geoprostorové souřadnice. Každý bod tak lze libovolně chytit a posouvat. Vizuální tloušťka vykreslených čar se mění, dle vzdálenosti a úhlu od kamery. Také vzdálenější body se jeví menší, ty ale na rozdíl od čar nemění náklon a zůstávají přesným kruhem.

Vytvářet trasy je také možno vložením několika fotografií, z jejichž exifu je extrahována souřadnice místa pořízení. Z jejich posloupnosti je následně rekonstruována celá trasa.



Obrázek 14. Editor v náklonu, se zapnutými 3D budovami.



Obrázek 15. Mobilní editor.

8 ZÍSKÁVÁNÍ A ZAZNAMENÁVÁNÍ GEOGRAFICKÝCH DAT

Vyjma ručního vytváření tras prostřednictvím editoru, o němž více v kapitole 7, jsem se také pokusil podívat na alternativní přístupy k vytváření tras.

8.1 Nahrávání skrze Geolocation API

Prvním a přirozeným způsobem sběru datu na mobilních telefonech je využití vestavené GPS antény k opakovanému získávání souřadnic aktuální pozice. Tyto potřeby naplňuje Geolocation API[11], datující se k roku 2013 a dnes již plně implementované ve webových prohlížečích. Za zmínku stojí také to, že API není zcela závislé na GPS anténě. Pokud ji zařízení nemá, může se pokusit získat alespoň přibližnou polohu jiným způsobem. Tudíž i na stolních počítačích či laptopech lze polohu určit. Jde však o proměnlivě spolehlivý způsob, ke kterému každý prohlížeč využívá svou vlastní heuristiku. Spolu s pozicí tak API poskytuje i hodnotu přesnosti pozice. U GPS antén se bavíme o přesnosti jednotek až desítek metrů, bez GPS jde spíše o jednotky až desítky kilometrů. Je však pochopitelné že k záznamu tras bude uživatel využívat pouze mobilní telefon, který anténou disponuje, zatímco desktopová aplikace slouží spíše k prohlížení a korekci existujících dat.

Záznam je možný skrze dvě asynchronní metody, které do převzatého callbacku předají získanou pozicí. `getCurrentPosition()` slouží k jednorázovému použití a `watchPosition()` startuje opakované sledování v intervalu jedné sekundy. Z hlediska šetrnosti k baterii zařízení by bylo vhodnější záznam pořizovat v intervalu spíše pěti až deseti sekund. Geolocation API bohužel neumožňuje interval nijak měnit. Ovlivnit můžeme pouze maximální požadované staří pozice prostřednictvím `maximumAge` v objektu předávaném jako třetí argument. To však vede ke cachování zastaralé pozice. API tak každou sekundu zavolá callback, ale se zastaralou pozicí. Nejen že tak nezískáme aktuální pozici, ale ani nesnížíme nárok na baterii kvůli krátkému a zcela zbytečnému intervalu.

[11] https://www.w3.org/TR/geolocation-API/

Na zvážení je pak vlastnoruční volání jednorázové metody `getCurrentPosition()` opakované uvnitř `setTimeout()` libovolného intervalu. Dle diskuzí na internetu jde o nepříliš efektivní způsob. Toto tvrzení však nemohu potvrdit ani vyvrátit.

Tento přístup by však bylo možné rozšířit o adaptivní získávání pozice v závislosti na rychlosti pohybu. Z předchozích dvou, či více pozic a jejich časových známek bychom mohli snadno interpolovat rychlost spočtením vzdálenosti obou bodu, respektive průměrnou vzdáleností mezi více body, a podělením rozdílů časů záznamu. Díky tomu bychom znali dosavadní rychlost pohybu, dle které lze předpokládat budoucí rychlost a čas potřebný k dosazení určené vzdálenosti. Mohli bychom tak interval záznamu vztáhnout a počítat dle uražené vzdálenosti a nikoliv času. Tím bychom mohli libovolně určit rozlišení trasy. Tudíž by například 10 km dlouhá trasa, uražená za hodinu, nemusela čítat 3600 souřadnic, pořízených každou sekundu, ale například jen 200 souřadnic zaznamenaných každých 50 metrů, či více, dle potřeby.

Dříve než jsem se však dostal k implementaci tohoto nápadu jsem se rozhodl Geolocation API opustit pro jeho současné omezení znemožňující multitasking a přirozené nakládání s mobilním telefonem. Konkrétně:

- Nedostupnost ze Service Workeru, za běhu v pozadí.
- Nedostupnost při zhasnutém, či zamknutém displeji

Jde o pochopitelnou snahu zabezpečit uživatele před potenciálně nebezpečnými webovými stránkami, které by špehovaly uživatele bez jeho vědomí. Osobně jsem zastánce názoru, že by stačilo dostatečně viditelně a často zobrazovat systémovou notifikaci, s informací o záznamu pozice na pozadí, kterou by nebylo možno odstranit, dokud není sledování ukončeno. Bohužel však dnešní prohlížeče v okamžiku zhasnutí displeje ukončí jakýkoliv záznam, probíhající, či naplánovaný. Znamená to tedy, že na platformě PWA nelze realizovat aplikace sledující sportovní aktivitu, jejichž use case je zapnutí záznamu následované zhasnutím displeje a schováním telefonu do kapsy. Možným řešením by mohl být nadcházející rozpracovaný standard Sensor API[12], jehož součástí je i poziční sensor[13].

[12] https://www.w3.org/TR/generic-sensor
[13] https://www.w3.org/TR/geolocation-sensor

Z důvodu těchto omezení jsem tedy v aplikaci implementovali pouze jednorázové získání souřadnic pro vykreslení aktuální polohy na mapě. Nikoliv už žádnou formu záznamu. Uživatelé na mobilních telefonech však mohou i přesto vkládat nové trasy vložením fotografií pořízených při jejich cestě.

8.2 Získávání souřadnic z EXIFu fotografií

Běžnou součástí každé digitální fotografie jsou metadata zachycující základní informace o fotografii, jejím rozlišení, fotoaparátu, rychlosti uzávěrky, clony a podmínkách při kterých byla zachycena. Tyto metadata souhrnně nazýváme EXIF a jsou neviditelnou binární součástí většiny fotografických formátů, z nichž nejpopulárnější je JPEG [17] [18]. Málokdo ale tuší, že mobilní telefony a dnes již i velké množství fotoaparátů, při zachycení fotografie do EXIFu zapisují také souřadnici pozice, kde byla pořízena.

Přirozeným use-casem mapové aplikace může být i vkládání fotek pořízených po čas výletu, přičemž právě souřadnice z EXIFu mohou hrát klíčovou roli pro umístění fotografie na vhodné místo na mapě. Přestože mezi volně publikovatelnými knihovnami na NPM existuje velké množství věnující se EXIFům, většina je implementována pouze pro serverové prostředí Node.JS. Zbylý zlomek je naopak spjatý pouze s webovými API. Velkým prohřeškem těchto knihoven je i nehospodárnost s výpočetním výkonem při zpracovávání souboru.

Ačkoliv parsování binárních metadat není triviální záležitostí, kvůli potřebě vyššího výkonu a multiplatformnosti jsem se rozhodl implementovat vlastní knihovnu `exifr`[14] [15]. S ohledem na potenciální rozšiřování aplikace a implementací backendu pro mě bylo důležité aby byl knihovna isomorphická. To znamená fungování v odlišných runtime prostředí, a sice:

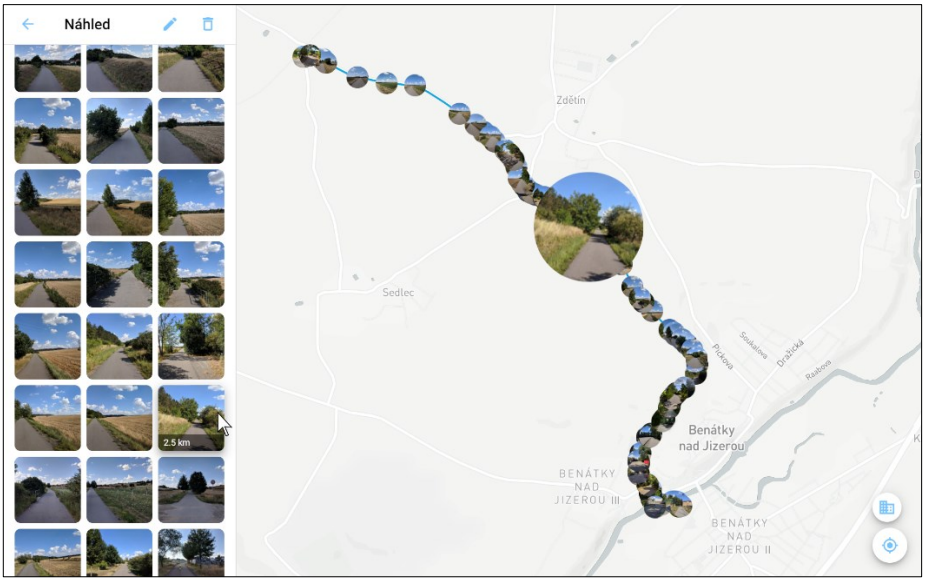
1. Webový prohlížeč s API `Blob`, `FileReader`, `Uint8Array`, `ArrayBuffer`, apod.
2. Node.js s vestavěnou knihovnou `fs` a binárním formátem `Buffer`.

Díky tomu by bylo možné přesunout část existujícího kódu z frontendové části aplikace do potenciální backendové části bez zásadních změn v kódu aplikace. Výhodou tohoto hypotetického rozšíření je vyšší zabezpečení backendu proti nechtěným zásahům a zároveň

[14] https://www.npmjs.com/package/exifr
[15] https://github.com/MikeKovarik/exifr

zachování funkcionality na frontendu, bez nutnosti odesílání velkého objemu dat fotografií ke zpracování na server.

Jelikož je ale knihovna v rámci aplikace využita pouze na frontendu, zaměřil jsem se obzvláště na vysoký výkon. Důvodem je umožnit uživateli vkládat nejen jednotky fotek, ale několik desítek až stovek, při zachování rozumného času na zpracování. Dosaženo je to selektivním čtením pouze prvních několika segmentů souboru a hledání známek naznačující přítomnost a rozsah EXIFových dat. V případě positivního nálezu se dále rekurzivně vkračuje hlouběji so souboru. Namísto načítání celého souboru o velikosti několika megabajtů, se tak přečte nanejvýše několik stovek bajtů. Ušetří se jak čas vykonání filesystémových operací přístupu na disk, tak i výkon nutný na parsování těchto binárních dat.



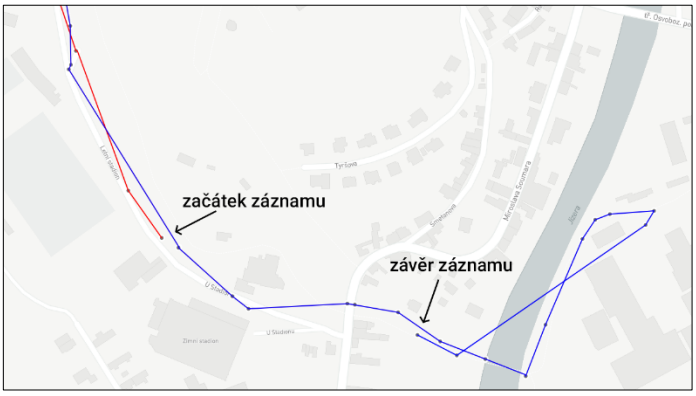
Obrázek 16. Detail trasy s vynesenými fotografiemi.

Na obrázku je zobrazen náhled na přípravek a zkušební vzorek devadesátí fotografií pořízených v rozmezí hodiny, při projížďce po cyklostezce o vzdálenosti 4.5 km. Každý soubor je v průměru 5 MB velký, což dohromady činí 450 MB. Po vložení všech souborů najednou do aplikace, byly předány knihovně `exifr`, který soubory zpracovala a vrátila GPS souřadnici každé fotografie. Experiment byl prováděn opakovaně, na běžném počítači s dvou jádrovým procesorem i5 o frekvenci 2.6 GHz a diskem SSD. Průměrná doba zpracování byla 173ms, odpovídající necelým dvěma milisekundám na každou fotografii, což je nad očekávání skvělý výsledek.

8.3 Rekonstrukce trasy z náhodné posloupnosti souřadnic

Některé specializované fotografické a editační programy ke zpracování fotografií, např. Adobe Lightroom, jsou schopny pozici fotografie zobrazit v malé mapce, jenže v kontextu účelu programu jde o nadbytečnou informaci.

Zajímavějším využití je však spojení souřadnic kolekce fotografií za účelem rekonstrukce trasy, na níž byly vytvořeny. Na základě vzorku dat z minulé kapitoly jsem provedl řadu iterací, z nichž prvním naivním řešením bylo seřazení fotografií chronologicky podle data pořízení. Tento přístup je zjevně problematický u míst vyfotografovaných opakovaně, podél cesty tam i zpět. Jak je vidět na následujícím obrázku, místa pořízení jsou protkána barevným přechodem z červené barvy od začátku focení, až po modrou barvou ke konci výletu. Výsledná trasa je nekoherentní, duplicitní, některá místa se překrývají, ale nejsou propojeny, protože se od sebe liší časem pořízení.

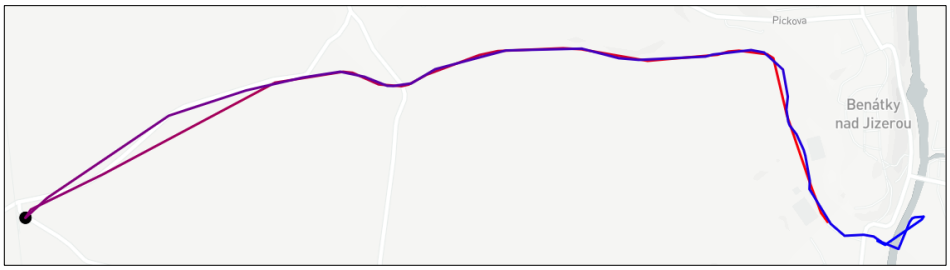


Obrázek 17. Chronologicky propojené body.

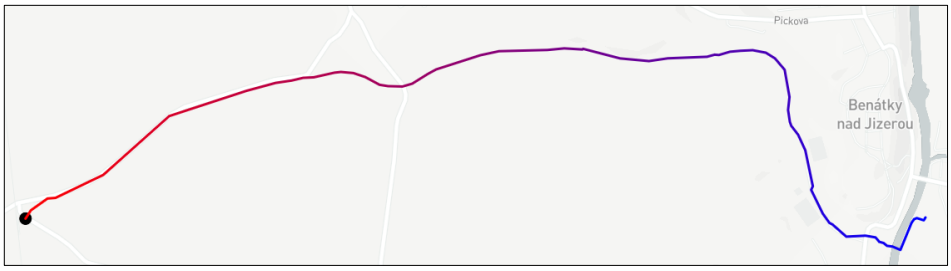
Rekonstrukce trasy je možná pokud se zcela oprostíme od časové značky a přistupujeme k souřadnicím jako k náhodné množině a spolehneme se na geoprostorové počítání vzdáleností mezi body. Výsledná implementace všechny body ohraničí do bboxu, z nějž vypočítá střed, vyhledá nejvzdálenější bod od středu a ten označí za počátek trasy. Následně prochází od iterativně od počátečního bodu, přes jeho nejbližší sousedy až dokud se nepropracuje k poslednímu. Jde o naivní, ale efektivní implementaci, která z náhodných dat, nejen

fotografických, sestaví trasu. Tento algoritmus však funguje pouze na přímých trasách, nikoliv na okruzích. Zdrojový kód je volně šiřitelný v repozitáři na GitHub.com[16].

Obrázek 18. Místa pořízení fotografií.



Obrázek 19. Chronologické propojené bodů.



Obrázek 20. Náhodné body propojené implementovaným algoritmem.

[16] https://github.com/MikeKovarik/geoutil

9 GEOPROSTOROVÁ ANALÝZA A ZPRACOVÁNÍ DAT

Naměřené data zpravidla potřebují začistit, zbavit duplikátů, případně zkomprimovat, aby je bylo možno lépe ukládat, nebo doručovat. Po prozkoumání chování uživatelů mapových aplikací jsem došel ke dvěma majoritním způsobům interakcí uživatelů s mapovou aplikací.

Prvním přístupem je precizní vytváření map, nebo tras nad mapami prostřednictvím sofistikovaného GIS software, s použitím klávesnice, myši a stolního počítače. V tomto případě není potřeba provádět žádný z procesů začišťování, protože se očekává že uživatel je proškolený odborník a výstup jeho práce je kvalitní.

Druhým přístup je častý u mobilních aplikací měřící sportovní výkony. Uživatel, před aktivitou spustí nahrávání GPS souřadnic a po celou dobu sportovní činnosti, mnohdy i několik hodin, jsou v pravidelném intervalu přidávány nové souřadnice. Výsledný záznam je pak plný duplicit a nadbytečných dat, které je potřeba pročistit.

V rámci aplikace jsem naimplementoval řadu algoritmů a pomocných funkcí pro práci s, či analýzu geografických dat. Tomuto kódu lze najít i jiné aplikace, proto jsem jej také rozhodl publikovat v otevřeném repozitáři `geoutil`[17].

9.1 Začištění a komprimace

Základním krokem je odstranění duplicitních souřadnic. K jejich vzniku většinou dochází, když se sportovec zastaví, nebo chybou v měření kdy dočasně není dostupné spojení se satelity a aplikace tak nezíská novou hodnotu. Poté následuje stlačení objemu dat na minimum komprimací. Nejsnazším krokem je zaokrouhlováním souřadnic, aby jejich textová podoba po zápisu na disk nebo do databáze zabírala co nejméně bajtů. Ve své surové formě nejsou souřadnice zaokrouhlovány. Tyto dvě desetinná čísla v praxi mívají až patnáct číslic za desetinnou čárkou. To dohromady činí 18 znaků na číslo, respektive 36 znaků na souřadnici. Pracujeme s formátem GeoJSON, který je nadstavbou formátu JSON, což je textový formát, kde jeden znak odpovídá jednomu bajtu, jedna souřadnice může zabírat až 36 bajtů. V případě trasy složené z pětiset souřadnic už jde o 18 000 bajtů neboli 18 kB. Takový objem dat jsme schopni vygenerovat při zhruba půl hodinovém měření s intervalem 4 s. Při vývoji

[17] https://github.com/MikeKovarik/geoutil

aplikace jsem se snažil najít způsob jakým takový objem dat zmenšit a nejsnazším způsobem aplikovatelným na jakákoliv souřadnicová data je zaokrouhlení čísel na menší počet desetinných míst.

```
function calculateCoordPrecision(coord, precision) {
      var decimal = Math.round(coord * (10 ** precision))
      var rounded = coord.toFixed(precision)
      var bytes = rounded.length
      var bits  = decimal.toString(2).length
      var bytesBefore = coord.toString().length
      var bitsBefore  = (coord * (10 ** 16)).toString(2).length
      console.log('precision   ', precision)
      console.log('coord       ', coord)
      console.log('rounded     ', rounded)
      console.log('decimal     ', decimal)
      console.log('string bytes', `${bytes} (${bytesBefore})`)
      console.log('int bits    ', `${bits} (${bitsBefore})`)
}
```

Tento snippet představuje funkci počítající ze vstupního čísla, jeho zaokrouhlenou variantu a počet bajtů a bitů, kolik by po zaokrouhlení zabralo. Pokud tento snippet zavoláme s libovolnou souřadnicí a počtem desetinných míst na které lze zaokrouhlovat, dostaneme data podobné jako v následující tabulce, kde srovnávám rozdíl mezi pěti a šesti desetinnými místy.

Precision	5	Precision	6
coord	15.094324350357056	coord	15.094324350357056
rounded	15.09432	rounded	15.094324
decimal	1509432	decimal	15094324
string bytes	8 (18)	string bytes	9 (18)
int bits	21 (58)	int bits	24 (58)

Zaokrouhlením na pět desetinných míst získáme výsledné číslo skládající se z osmi znaků, tedy osmi bajtů v textové podobně. V případě binární reprezentace jako integerové číslo pak zabírá 21 bitů. Zaokrouhlením na šest desetinných míst získáme obdobné výsledky. Devět bajtů textové reprezentace, nebo 24 bitů neboli 4 bajty v binární reprezentaci.

Experimentováním jsem došel ke zjištění, že i při zaokrouhlení na pět desetinných míst nedojde ke ztrátě přesnosti. Respektive dojde ke zkreslení do maximálně jednoho metru. Tento závěr jsem si ověřil i překreslením zaokrouhlené trasy přes původní nedotčené data, mezi nimiž nebyl viditelný rozdíl. K tomu začalo docházet až při zaokrouhlováním na místa čtyři. Aplikujeme-li tuto optimalizaci na předešlý příklad s pěti-sty souřadnicemi dostaneme se na výslednou velikost 8 kB což je více než polovina původní velikosti.

Souřadnice se dají dále zapisovat v alternativních formátech bez ztráty kvality, tzv. lossless. Například algoritmy pro převod číselných souřadnic na posloupnost znaků připomínající base64 kódování, ale efektivnější. Jedním z nich je Encoded Polyline[18] od Google. Najde své využití v případech, kde je třeba tlačit na efektivitu datového přenosu. Jako třeba při dotazování se na výškový profil trasy, které je prováděno přes URL jejíž délka omezena na určitý počet znaků. Jedna dvoučíselná souřadnice by tak mohla zabrat i dvacet znaků, což by při omezení na 200 znaků odpovídalo asi 10 souřadnicím. Enkódování do stringové formy je v takových případech značně efektivnější. Nevýhodou takového přístupu je vyšší výpočetní výkon nutný pro převod z nebo do tohoto formátu.

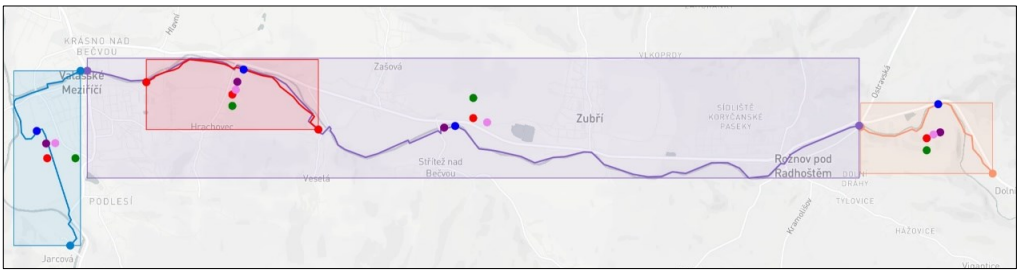
Po bližším zkoumání a experimentování s různými formami komprimace a skladování dat jsem se rozhodl pro potřeby aplikace zůstat u formátu GeoJSON, kde souřadnice jsou uloženy jako desetinná čísla zaokrouhlená na pět desetinných míst. Encoded Polyline jsem zavrhl kvůli zvýšeným výpočetním nárokům na klientské straně aplikace. To by vedlo k pomalejším načítacím časům a větší spotřebě baterie. Především však proto, že toto kódování není kompatibilní s formátem GeoJSON a při každém načtení a změně dat by muselo dojít k přepočítání, aby mohla být data vykreslena.

Za zmínku stojí také transformace prováděné interně knihovnou Mapbox GL. Rastrové mapové knihovny fungují pouze jako prohlížeče již existujících obrázkových dlaždic a klientská část kódu knihovny se stará jen o zajišťování interaktivity skrze mouse eventy. Naopak Mapbox GL veškeré mapové dlaždice vykresluje až na straně klienta, kde se také nachází kód ke zpracování a samotnému vykreslení souřadnicových. Proto je mnohem snazší do renderovací pipeline zařadit i vlastní data. Mapbox pak dle potřeby, zobrazeného výřezu, úrovně přiblížení apod. sám ořízne a zjednoduší veškeré data, které se rozhodne vykreslit. Proto jsem schopen v aplikaci vykreslit nejen body symbolizující zaznačené trasy, jako je tomu u mnohých podobně zaměřených konkurenčních aplikací, nýbrž zároveň vykresluji všechny trasy s veškerými souřadnicemi na globální mapě na úvodní straně aplikace.

[18] https://developers.google.com/maps/documentation/utilities/polylinealgorithm

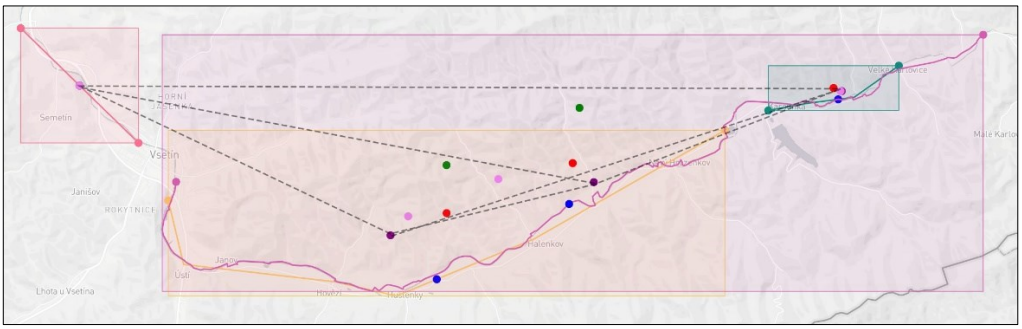
9.2 Detekce překryvů a duplicitních tras

Pracovat s libovolnou dvojící tras, souřadnici po souřadnici, bez žádných podpěrných informací by bylo velmi výpočetně náročné. Nejprve proto všem trasám vypočítám meta informace popisující začátek, konec, střed a ohraničující obdélník neboli bbox. Tyto meta informace se následně staly užitečnými i pro jiné využití v aplikaci. Bbox je dále využíván při zobrazování tras pro pohotovější zobrazení a předejití nadbytečně opakovaného výpočtu.



Obrázek 21. Znázornění bodů a bboxů testovaných tras.

Nicméně pro vypočet vztahu dvou tras jsou důležité body počátku, středu a konce. Získáme tak pole třech souřadnic, které zhruba popisuje umístění trasy, namísto původních několik desítek až stovek. Vstupní podmínka výpočtu vztahu dvou tras se tak omezí na spočtení vzdáleností třech bodů oproti třem jiným bodům, tedy celkem devět výpočtů vzdálenosti. Tato metoda má samozřejmě své omezení a není v žádném případě přesná. Jde ale pouze o první krok, kdy získáváme přibližnou představu, jak vzdálené si trasy jsou.



Obrázek 22. Vzájemné srovnání centroidů tras.

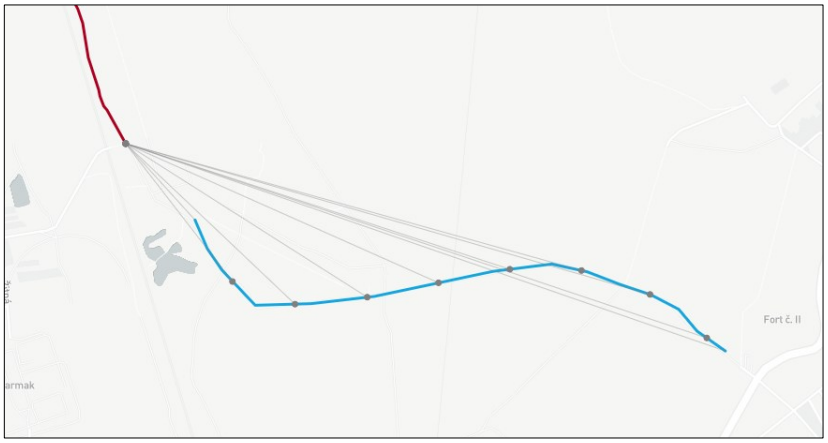
Vhodný je v situacích kdy uživatel vytvoří novou trasu do databáze existujících tras, která může čítat desítky až stovky tras, z nichž každá se skládá z desítek až stovek souřadnic. Namísto přesného porovnání všech bodů, tak porovnáme jen vzdálenosti od okrajů a středů. Z nich si ponecháme nejnižší hodnotu, čímž získáme aproximaci vzdálenosti nově přidané trasy vzhledem k libovolné jiné trase z databáze. Pokud bychom porovnávali více než tři

body, získáme vyšší přesnost, ale navýšíme výpočetní nároky. Ideální počet vzorkovacích souřadnic nelze jednoznačně určit, protože se vždy odvíjí od okolností, dat a aplikace. Z průběžných experimentů se nejvíce osvědčily právě tři.

Obrázek 23. Vizualizace srovnání dvou tras.

Následně vybereme několik nejbližších kandidátu, mezi kterými provedeme pokročilejší výpočty s vyšším rozlišením. Zde je z trasy A vybráno několik vzorků v uniformním rozestupu dvěstě metru a pro každý je hledán nejbližší bod na trase B. Neporovnává se vůči předem vybraným bodům z trasy B, ale aktivně vyhledáváme nejbližší bod. Poté opět spočteme vzdálenosti mezi vzorky a nalezenými protějšky a pokud alespoň v polovině měření nepřesahují více než 150 metrů, označíme trasy za duplikáty.

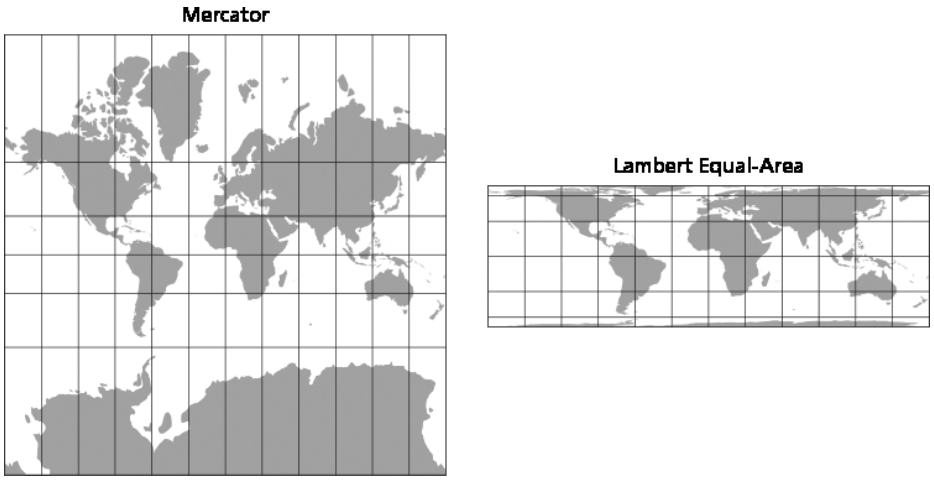
Celé rozhodování je založeno na fuzzy logice, kterou ovlivňuje ještě přesnost tras a počet průniků. Oněch 150 metrů je počáteční práh, který je dále upravován dle počtu souřadnic na kilometr. Pokud je např. jedna trasa velmi detailní, s dvaceti body na kilometr, a druhá nepřesná, s ani ne jedním bodem na kilometr, nemusel by být práh 150 dostačující.



Obrázek 24. Srovnání návazných tras.

9.3 Výpočty ovlivněné kartografickou projekcí

Potřeba zobrazit na ploché dvourozměrné ploše prostorový objekt existující ve třech dimenzích představuje problém. Zeměkoule ze své podstaty nemůže být plochá, a proto mapy světa představují jen její zdeformovanou reprezentaci. K její vytvoření se používá projekce jejíž úkol je ke každému bodu z referenčního zakřiveného povrchu přiřadit právě jeden bod na zobrazovací ploše. Podle potřeb a účelu dané mapy se tedy musíme smířit s určitou měrou zkreslení vzdáleností, úhlů, nebo zanedbání přesnosti. Zobrazit na mapě světa velikosti států a kontinentů tak, aby co nejvíce odpovídaly realitě je téměř nemožné.



Obrázek 25. Srovnání Mercatorova a Lambertova zobrazení [20]

9.3.1 Lambertovo zobrazení

Manipulaci se zobrazením zeměkoule si můžeme představit jako kůru pomeranče, který rozkrájíme na pravidelné měsíčky a slupky vyskládáme na stůl vedle sebe. Každý řez představuje poledník. Jednotlivé dílky k sobě ale nepřiléhají, protože jsou zakroucené kvůli původnímu kruhovému tvaru pomeranče. Nejdou zploštit a jejich tvar není obdélníkový. Připomíná spíše dva rovnoramenné trojúhelníky spojené k sobě podél svých základen. Právě v tomto středě se nachází rovník. Nahoře i dole se šířka postupně zužuje a strany se sbíhají do špičky. Jediným způsobem zploštění je konečky roztřepit na menší trojúhelníky vybíhající od středu. Tím se narovná zakřivení a výsledný tvar svou rozlohou připomíná obdélník. Nicméně povrch slupky není i nic větší, pouze přibylo volného místa v mezerách mezi zářezy. Na papíře, nebo v mapových softwarech jsme schopni mezery zaplnit roztažením okolních konečků slupky. V podstatě deformujeme skutečný tvar světa s přihlédnutím na to, že čím dále od rovníku jsme, tím více se mění i měřítko mapy pro přepočet centimetrů mapy na odpovídající kilometry na zeměkouli. Státy blíže severnímu, nebo jižnímu pólu, jako Rusko, nebo

Kanada, se tak zdají být širší, než doopravdy jsou. Tomuto zobrazení se říká Lambertova válcová projekce.

9.3.2 Mercatorovo zobrazení

Dnes nejpoužívanější mapy jsou postaveny na systému zobrazení vytvořeného Gerhardtem Mercatorem, po němž je pojmenován. Mercator projection, jak již z názvu vyplývá je také promítán z koule na válec, rovnoběžný se zemskou osou, dotýkající se glóbu na rovníku. Princip je podobný jako u Lambertovy projekce s tím rozdílem, že zeměkoule posazena do pomyslného válce, na který se vykresluje promítáním paprsků ze středu zeměkoule. Vzniká tak pravoúhlá síť rovnoběžek a poledníků. Poledníky jsou sice zobrazeny ve stejných rozestupech, ale rovnoběžky promítnuté na válci nejsou kolmé k rovnoběžkám na zeměkouli. Svírají stejný úhel, jako zemská rovnoběžka se středem zeměkoule. Jediná kolmá rovnoběžka je tedy rovník. Čím více se rovnoběžka vzdaluje od rovníku tím větší svírá úhel a tím se zvyšuje i vzdálenost od předešlé rovnoběžky. Zkresluje však Evropu tak, aby středem byla Evropa a vypadala větší, než ve skutečnosti je. Grónsko vypadá stejně velké jako Afrika. Ve skutečnosti je ale čtrnáctkrát menší.

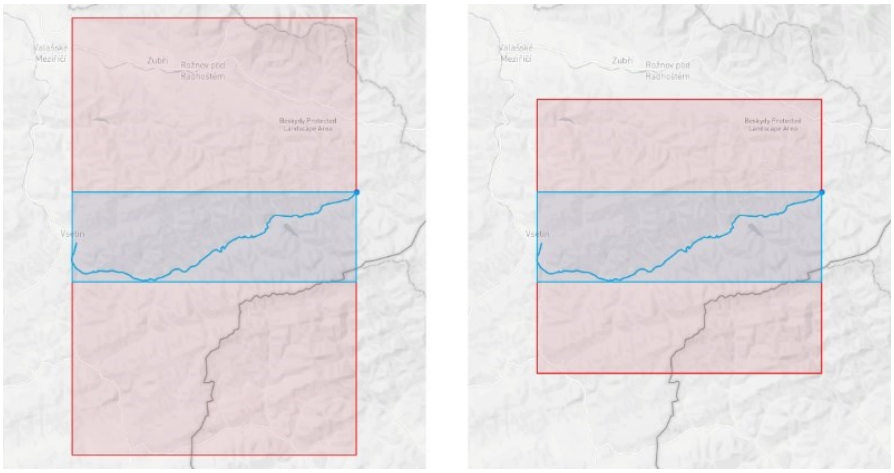
9.3.3 Výpočty zkreslující geografickou přesnost ve prospěch UI

Po celou dobu vývoje aplikace jsem měl pojem o Mercatorově zobrazení, ale domníval jsem se, že díky povaze vyvíjené aplikace budu od potíží spojených s touto projekcí dostatečně odstíněn. Přeci jen aplikace pracuje primárně se zeměpisnými souřadnicemi, nad kterými provádí pár typů běžných geoprostorových výpočtů. Ty nejsou ovlivněny typem zobrazení, neboť jejich algoritmy berou v potaz zakřivení země a pracují s běžnými matematickými operacemi. Čili práce se souřadnicemi dostatečně abstrahuje od problémů se zobrazením, o které se pak stará mapová knihovna. Komplikace s Mercatorovu projekcí tak přišly paradoxně až při práci na uživatelském rozhraní aplikace.

Dobrým zvykem při zobrazení trasy je nepracovat pouze s jejím středovým bodem, ale spočítat kolem trasy ohraničující obdélník, ve kterém jsou obsaženy všechny body trasy. Tento tzv. bbox neboli bounding box, pak slouží k přesnému zobrazení trasy vzhledem k velikosti displeje na kterém je trasa zobrazována. Vhodné měřítko zoomu se automaticky spočítá tak, aby žádný z okrajů bboxu nepřesáhl viditelné pole mapy.

Ke spočítání bboxu je použita funkce `bbox()` z geoprostorové knihovny Turf.js, které funguje podle očekávání. Ten je na následovném screenshotu vyobrazen modrým obdélníkem

a perfektně kopíruje hraniční body trasy po všech čtyřech stranách. Avšak potíže se dostavily při požití `square()`. Ten přejímá data z `bbox()` a slibuje vytvoření čtverce do jehož středu bbox umístí. Problémem je, že Turf vypočítává geograficky přesný čtverec, který je zkreslen svou vzdáleností od rovníku.

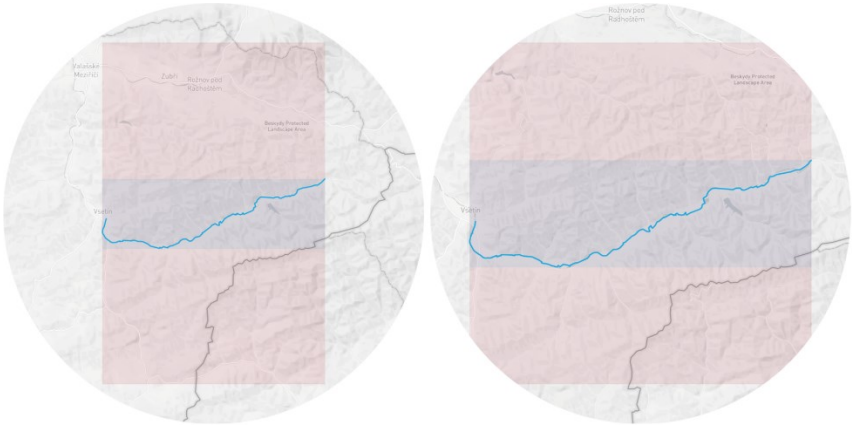


Obrázek 26. Rozdíl zkreslenosti funkce square.

Na obrázku je vidět trasa mezi městem Vsetín a obcí Velké Karlovice. Obě se nachází na pomezí čtyřicáté deváté a padesáté rovnoběžky. Dvourozměrné mapy nejsou z principu schopné věrně zobrazit trojrozměrnou kouli. Mercator promítá povrch země na válec, který se pak rozloží do roviny, díky čemuž vzniká pravoúhlá síť poledníků a rovnoběžek. Současně to ale znamená, že čím více se rovnoběžka vzdaluje se od rovníku k pólům, tím více se oblast pod nebo nad rovnoběžkou jeví vyšší, než ve skutečnosti je.

Obrázek 27. Progrese funkce square od rovníku.

Síť je tedy sice pravoúhlá, nikoliv však jednotková. Turf.js své výpočty zakládá se jednoduchém součinu a podílu zeměpisných délek a šířek, bez ohledu zkreslení ve vyšších zeměpisných výšek blíže k severnímu a jižnímu pólu. U padesáté rovnoběžky je tento problém již velmi zřejmý, viz. červený obdélník na levém obrázku. Pro potřeby vizualizace v aplikaci jsem tak musel implementovat vlastní výpočty vytvoření čtverce z hlediska uživatelského rozhraní, ne však nutně z hlediska geografického.



Obrázek 28. Vhodnější ohraničení a zobrazení trasy v UI.

10 ADAPTIVNÍ PŘIZPŮSOBOVÁNÍ AKTUÁLNÍ PLATFORMĚ

Kvůli univerzální povaze PWA aplikací a podporovaných platforem je někdy potřeba zjišťovat vlastnosti platformy a adekvátně přizpůsobit uživatelské rozhraní, nebo chování aplikace. Ta se, jak již bylo v předchozích kapitolách představeno, přizpůsobuje velikosti displeje, mění rozložení aplikace a přepíná velikost adaptivního hitboxu.

10.1 Tvorba knihovny detekující platformu a prostředí

Zajímavým podnětem pro budoucí rozšiřování jak technologie PWA, tak i této aplikace je i zohledňování nových typů zařízení, jako chytré televize a herní konzole. Ty dnes již disponují internetovým prohlížečem, díky čemuž je možno na nich otevřít PWA aplikace přinejmenším jako webové stránky, ale interakce s obsahem je omezena pouze na jeden emulovaný kurzor. Obdobně je tomu u herních konzolí s gamepadem.

Navíc v průběhu vývoje této práce ještě nebyl znám krok Microsoftu ukončit vývoj svého vykreslovacího jádra EdgeHTML a kvůli rozdílnému chování Edge a Chrome v některých případech jsem byl nucen detekovat vykreslovací engine, typ prohlížeče a další vlastnosti.

Další netriviální detekcí je rozlišování kontextu, ve kterém je skript spuštěn. Nachází-li se v samostatném vlákně Web Worker, nemá přístup k některým API volně dostupným z hlavního UI vlákna. Díky Node.JS již dnes můžeme programovat i servery jazykem JS. Na toto prostředí je dále aplikována další sada rozšíření a omezení, kterými se musí vývojář řídit k čemuž je potřeba terminálové prostředí detekovat a odlišit od prohlížečového okna.

Zmíněné podněty mne v průběhu vývoje aplikace vedly k implementaci řady opatření a detekcí uvnitř aplikace. Ty jsem později vyseparoval do několika souborů. Jelikož jde o praktickou utilitu, která se neomezuje jen na potřeby aplikace, a zároveň přerůstala rozsah aplikace, rozhodl jsem se tento kód vyčlenit do samostatné knihovny `platform-detect`[19], která je spolu s dokumentací a licencí MIT dostupná v repozitáři na GitHub.com[20].

[19] https://www.npmjs.com/package/platform-detect
[20] https://github.com/MikeKovarik/platform-detect

10.2 Experimentování s dostupností na TV

Nové televize posledních několika let již neslouží jen k zobrazování televizního signálu, ale obsahují také internetové prohlížeče a své vlastní aplikace. Mezi nejrozšířenější patří série Smart TV od výrobce Samsung, které jsou postaveny na OS Tizen. Ten je podobně jako Firefox OS nebo Chrome OS postaven čistě na HTML. Tizen a jeho webový prohlížeč je podobně jako Chrome postaven na vykreslovacím jádře Webkit. Jde tedy o perfektní prostředí, kam se mohou PWA přirozeně rozšířit. Mnou testovaný model televize Samsung UE55KU6000, ale bohužel zaostal na starší, ve které schází část páteřních standardů PWA. Toto je možné ze strany Samsungu opravit jednoduchou softwarovou aktualizací.

Samsung má sice větší kupní sílu a schopnost protlačit Tizen na trh, ale kvůli pomalému nárůstu a nejisté budoucnosti jsem se dostupností PWA na OS Tizen nemohl hlouběji zabývat. Proto jsem se v rámci této práce přímo nevěnoval optimalizaci aplikace pro TV, ale zohlednil jsem ji v knihovně platform-detect a snažil jsem se navrhovat všechny kroky při vývoji tak, aby do budoucna nevznikla žádná překážka pro rozšíření aplikace právě na televize a jejich uživatelské rozhraní.

10.3 Experimentování s dostupností na herních konzolích

Poněkud mimo rozsah této práce, ale přeci hodno zmínky je současná podpora PWA na herních konzolích. Ještě před chytrými televizemi s vlastním OS to byly právě herní konzole k televizím připojené, které je činily interaktivní. Herní konzole byly vždy vedle hraní také domácím multimediálním centrem. Přehrávání filmů z DVD a BluRay nahradily aplikace jako Netflix a Youtube. Webový prohlížeč je dnes standartní výbavou každé konzole. Avšak kvůli odlišnému ovládání uživatelského prostředí pomocí gamepadu je přizpůsobování aplikací pro konzole podstatně obtížnější.

V první fázi bylo nutné identifikovat rozdíly mezi běžným desktopovým prohlížečem a tím herním. Na první pohled ze strany vývojáře webových stránek nejde o žádné velké změny, nebo zásadně odlišné API. Jde o běžný prohlížeč MS Edge, ve stejné verzi a se stejnou funkcionalitou jako je na desktopovém Windows 10, nebo jaká byla na mobilních Windows. Xbox je sice ovládán herním kontrolérem s dvěma joysticky a řadou tlačítek, ale v běžném prohlížení internetových stránek je na displeji emulován kurzor. Ten má napodobovat kurzor myši, spolu s emulovanými eventy `mouseover`, `mousemove`, atd… Pohyb myši po stránce je ovládán jedním z joysticků, scrollování druhým. Tlačítko B je mapováno na tlačítko zpět, X je mapováno na backspace a klikání je ovládáno pravým trigger buttonem.

Tento přístup je nutným zlem potřebným k elementární navigaci po internetu a webových stránkách, není však dostačující z hlediska User Experience pro aplikace. Naštěstí již od roku 2012 existuje webový standard Gamepad API. Sice se stále nachází v pracovní verzi Draft. Po letech je již v této podobě implementován do prohlížečů Edge i Chrome.

Pomocí Gamepad API, konkrétně voláním `navigator.getGamepads()`, jsem tak byl schopen detekovat přítomnost připojeného gamepadu a označit tak zařízení na herní konzoli. Komplikací bylo, že volání této funkce zároveň zapříčinilo odstranění emulovaného kurzoru. Pokud by aplikace využívající knihovnu `platform-detect`, kde je tato volána, nepřizpůsobila své UI a neimplementovala individuální handlery pro herní kontrolér, přišla by o veškeré ovládání. To je nepřijatelné, proto jsem se musel najít jinou alternativu. Bohužel události `gamepadconnected` jsou nejen asynchronní, ale také vyvolané až po první interakci uživatele se stránkou. Avšak pro včasné zobrazení alternativního UI již při startu aplikace je nutné mít předvídatelnou a úplnou statickou analýzu zařízení a platformy dostupnou již ve chvíli importu knihovny. Spoléhání na asynchronní události je nepřípustné. Proto jsem musel přistoupit na méně elegantní řešení detekování pomocí dotazování se na user-agent string. Stejně se obstarává i detekce TV. Řešení to není ideální, zato funkční.

ZÁVĚR

Cílem diplomové práce bylo prozkoumání současných technologických možností vytváření webových aplikací pomocí standardů PWA, následně takovou aplikaci implementovat a demonstrovat na ni přednosti PWA.

V teoretické části jsem nastudoval a představil jednotlivé standardy a uvedl je do kontextu vývoje současných webových stránek. Pojednal jsem o zabezpečení, výhodách i rizicích těchto technologií. Také jsem přiblížil možnosti práce s mapovými podklady.

V praktické části jsem implementoval několik experimentů, z nichž vzešla PWA aplikace a podpůrné knihovny. Jedná se o pokročilou webovou stránku, která v prohlížeči vykresluje, zpracovává a ukládá obsah bez potřeby nepojení na backend. Na venek se jeví jako běžná stránka, ale díky Web App Manifestu umožňuje instalaci přímo z webového prohlížeče. Vznikne tak samostatná aplikace, s ikonou na ploše a vlastním oknem bez adresového řádku, či ovládacích prvků prohlížeče. Dále je implementován Service Worker, skrze nějž je zachytávaná HTTP komunikace ukládána do cache. Zdrojové soubory aplikace, databáze i mapové podklady jsou tak dostupné i bez připojení k internetu. Funkcionalitou, vzhledem i chování je tedy PWA aplikace k nerozeznání od nativních aplikací.

Vyvinutá aplikace je tvořena velkou interaktivní mapou, do níž jsou průběžně zakreslovány data, trasy a ovládací prvky. Mapa je vektorová, využívá moderní technologii WebGL a její vykreslování proto probíhá přímo na zařízení uživatele, bez nutnosti stahování rastrových mapových dlaždic. Prací jsem tak dokázat tvrzení, že webová platforma se svou výkonností, GPU akcelerací a plynulostí vykreslování již může rovnat nativním aplikacím psaných v jazycích Java, C# apod.

Zároveň jsem chtěl zjistit nakolik jsou technologie PWA připraveny a schopny napodobit funkcionalitu nativních aplikací. Má teze o budoucnosti a prospěšnosti platformy PWA byla zásadně podpořena rozhodnutím společnosti Microsoft, která v průběhu vývoje této práce ukončila vývoj svého prohlížeče Edge a vykreslovacího jádra EdgeHtml, ve prospěch konkurenčního projektu Chromium od Google. Rozhodnutí je motivované potřebou lepší podpory páteřních technologií pro tvorbu právě PWA.

Z hlediska této práce to však představilo drobné komplikace, neboť záměr experimentovat s prohlížečem Edge a skrze něj s integrací PWA do Windows 10 nebylo možno nadále uskutečnit. Dále jsem musel upustit od záměrů pokročilejšího záznamu souřadnic

prostřednictvím Geolocation API. Neumožňuje to současné omezení toho standardu. Práce se namísto toho blíže zabývala vývojem jiných funkcí aplikace.

Výstupem práce je tak kromě aplikace i řada veřejně publikovaných knihoven. Konkrétně knihovna pro detekci schopností zařízení. Knihovna pro analýzu a práci s geografickými daty. A také knihovna pro zpracování exif metadat z fotografií za účelem zanesení souřadnic do mapy, k následné rekonstrukci trasy, podél níž byly fotografie pořízeny.

Aplikace splňuje všechny funkční požadavky. Disponuje řadou funkcí od prohlížení, přes záznam, úpravu, až po analýzu geografických dat. Práce naplnila všechny body zadání a tvrzení o praktičnosti platformy byly potvrzeny. Pro autora měla tato práce přínos ve všech krocích studia i implementace. Práce představuje nové možnosti vývoje aplikací a zároveň má přesah i do Open Source komunity, díky publikaci užitečného kódu jako knihovny pro užití v jiných projektech a pracích. Autor kladně hodnotí průběh práce i její výstup a má v plánu se dále věnovat rozvoji aplikace i knihoven.

SEZNAM POUŽITÉ LITERATURY

[1] ANDERSON, Paul. Web 2.0 and Beyond: Principles and Technologies. CRC Press, 2012. ISBN 9781439828670.

[2] PARKER, Todd, Scott JEHL, Maggie Costello WACHS a Patty TOLAND. Designing with Progressive Enhancement: Building the Web that Works for Everyone. Pearson Education, 2010. ISBN 9780321659491.

[3] ATER, Tal. Building Progressive Web Apps: Bringing the Power of Native to the Browser. Sebastopol, CA: O'Reilly Media, 2017. ISBN 9781491961650.

[4] SHEPPARD, Dennis. Beginning Progressive Web App Development: Creating a Native App Experience on the Web. Apress, 2017. ISBN 9781484230893.

[5] HUME, Dean a Addy OSMANI. Progressive Web Apps. Manning Publications, 2018. ISBN 1617294586.

[6] WARGO, John M. Apache Cordova 4 programming. Upper Saddle River, NJ: Addison-Wesley, [2015]. ISBN 0134048199.

[7] AMARASINGHE, Sean. Service Worker Development Cookbook. Packt Publishing, 2016. ISBN 9781786469526.

[8] GREEN, Ido. Web Workers. O'Reilly Media, 2012. ISBN 9781449322137.

[9] What is map caching?. In: ArcGIS Enterprise [online]. [cit. 2019-05-14]. Dostupné z: https://enterprise.arcgis.com/en/server/latest/publish-services/windows/what-is-map-caching-.htm

[10] KASTANAKIS, Bill. Mapbox Cookbook. Packt Publishing, 2016. ISBN 1784397350.

[11] HALL, Justin. Windows Defender Application Guard overview [online]. [cit. 2019-05-16]. Dostupné z: https://docs.microsoft.com/en-us/windows/security/threat-protection/windows-defender-application-guard/wd-app-guard-overview

[12] Content Security Policy (CSP). Mozilla Developer Network [online]. 2018 [cit. 2019-05-16]. Dostupné z: https://developer.mozilla.org/en-US/docs/Web/HTTP/CSP

[13] Cross-Origin Resource Sharing (CORS). Mozilla Developer Network [online]. 2018 [cit. 2019-05-16]. Dostupné z: https://developer.mozilla.org/en-US/docs/Web/HTTP/CORS

[14] BEAUFORT, François. Interact with Bluetooth devices on the Web. Google Developers [online]. 2017 [cit. 2019-05-16]. Dostupné z: https://developers.google.com/web/updates/2015/07/interact-with-ble-devices-on-the-web

[15] Web Cryptography API: 2017. W3C [online]. [cit. 2019-05-16]. Dostupné z: https://www.w3.org/TR/WebCryptoAPI/

[16] Web Authentication: An API for accessing Public Key Credentials Level 1. W3C [online]. 2018 [cit. 2019-05-16]. Dostupné z: https://www.w3.org/TR/webauthn/

[17] SOUDERS, Steve. Even Faster Web Sites: Performance Best Practices for Web Developers. O'Reilly Media, 2009. ISBN 9780596555849.

[18] OWINGS, Rich. GPS mapping: make your own maps. Fort Bragg, Calif.: Ten Mile Press, c2005. ISBN 978-097-6092-636.

[19] MEW, Kyle. Learning Material Design. Packt Publishing, 2015. ISBN 1785289810.

[20] HOLDENER, Anthony. HTML5 Geolocation. O'Reilly Media, 2011. ISBN 1449304729.

[21] HAYES, Darren. A Practical Guide to Computer Forensics Investigations. Pearson IT Certification, 2014. ISBN 9780132756150.

[22] MILLER, Preston a Chapin BRYCE. Learning Python for Forensics. Packt Publishing, 2016. ISBN 9781783285242.

[23] PATTERSON, Tom, Bojan ŠAVRIČ a Jenny BERNHARD. Introducing the Patterson Cylindrical Projection. In: Cartographic Perspectives [online]. 2014 [cit. 2019-05-16]. Dostupné z: http://cartographicperspectives.org/index.php/journal/article/view/cp78-patterson-et-al/1362

SEZNAM POUŽITÝCH SYMBOLŮ A ZKRATEK

PWA Progresivní Webová Aplikace

GIS Geografický Informační Systém

OSM Open Street Maps

DOM Document Object Model

API Application Programming Interface

SPA Single Page Aplikace

UI User Interface

UX User Experience

NPM Node Package Manager

MS Microsoft

OS Operační systém

UWP Universal Windows Platform

SW Service Worker

SEZNAM OBRÁZKŮ

Obrázek 1. Diagram progresivního rozšiřování webové stránky Service Workerem [vlastní tvorba]. 12
Obrázek 2. Instalace PWA z mobilního prohlížeče. 13
Obrázek 3. Proces instalace PWA z okna prohlížeče Google Chrome, na desktopovém systému Windows 10. 14
Obrázek 4. Desktopová PWA. 14
Obrázek 5. Mapové Dlaždice. [9] 25
Obrázek 6. Úvodní strana aplikace, view #/all, na mobilu a desktopu. 41
Obrázek 7. Detail trasy, coby mobilní webová stránka a desktopová PWA. 41
Obrázek 8. Náhled na fotografie v detailu trasy. 42
Obrázek 9. Editor trasy. 43
Obrázek 10. Comparison view vyhledávající sousední trasy. 43
Obrázek 11. Hitbox skrytý a optimalizovaný pro myš a dotyk. 49
Obrázek 12. Vkládání bodu prostřednictvím ducha. 54
Obrázek 13. Prodlužování trasy. 54
Obrázek 14. Editor v náklonu, se zapnutými 3D budovami. 55
Obrázek 15. Mobilní editor. 55
Obrázek 16. Detail trasy s vynesenými fotografiemi. 59
Obrázek 17. Chronologicky propojené body. 60
Obrázek 18. Místa pořízení fotografií. 61
Obrázek 19. Chronologické propojené bodů. 61
Obrázek 20. Náhodné body propojené implementovaným algoritmem. 61
Obrázek 21. Znázornění bodů a bboxů testovaných tras. 65
Obrázek 22. Vzájemné srovnání centroidů tras. 65
Obrázek 23. Vizualizace srovnání dvou tras. 66
Obrázek 24. Srovnání návazných tras. 66
Obrázek 25. Srovnání Mercatorova a Lambertova zobrazení [20] 67
Obrázek 26. Rozdíl zkreslenosti funkce square. 69
Obrázek 27. Progrese funkce square od rovníku. 70
Obrázek 28. Vhodnější ohraničení a zobrazení trasy v UI. 70