FAKULTA
INFORMAČNÍCH
TECHNOLOGIÍ
ČVUT V PRAZE

ZADÁNÍ DIPLOMOVÉ PRÁCE

Název:	Mobilná aplikácia kvalita ovzdušia v Prahe
Student:	Bc. Igor Rosocha
Vedoucí:	Ing. Josef Gattermayer, Ph.D.
Studijní program:	Informatika
Studijní obor:	Webové a softwarové inženýrství
Katedra:	Katedra softwarového inženýrství
Platnost zadání:	Do konce letního semestru 2019/20

Pokyny pro vypracování

Cieľom práce je navrhnúť a implementovať mobilnú aplikáciu pre operačný systém iOS, ktorá bude na mapovom podklade vizualizovať kvalitu ovzdušia v Prahe. Dáta bude získavať z vlastného implementovaného servera, ktorý bude dáta o kvalite ovzdušia pripravovať na základe otvorených dátových zdrojov.

Pokyny:
1) Analyzujte existujúce aplikácie podobného zamerania
2) Analyzujte dátové zdroje merania znečistenia ovzdušia v Prahe
3) Analyzujte a zvoľte vhodné technológie na realizáciu serverovej časti
4) Navrhnite používateľského rozhranie
5) Implementujte navrhnutú aplikáciu pre operačný systém iOS
6) Otestujte výslednú aplikáciu

Seznam odborné literatury

Dodá vedoucí práce.

Ing. Michal Valenta, Ph.D.
vedoucí katedry

doc. RNDr. Ing. Marcel Jiřina, Ph.D.
děkan

V Praze dne 14. února 2019

FAKULTA
INFORMAČNÍCH
TECHNOLOGIÍ
ČVUT V PRAZE

Diplomová práca

Mobilná aplikácia kvalita ovzdušia v Prahe

Bc. Igor Rosocha

Katedra softwarového inženýrství
Vedúci práce: Ing. Josef Gattermayer, Ph.D.

7. mája 2019

Poďakovanie

Rád by som poďakoval Ing. Josefovi Gattermayerovi za jeho pomoc a odborné vedenie počas tvorby tejto diplomovej práce. Rovnako by som sa chcel poďakovať Bc. Tomášovi Krasnayovi za jeho čas a cenné rady pri tvorbe a testovaní mobilnej aplikácie. V neposlednom rade ďakujem svojej priateľke a celej svojej rodine za enormnú podporu počas celého štúdia.

Prehlásenie

Prehlasujem, že som predloženú prácu vypracoval(a) samostatne a že som uviedol(uviedla) všetky informačné zdroje v súlade s Metodickým pokynom o etickej príprave vysokoškolských záverečných prác.

Beriem na vedomie, že sa na moju prácu vzťahujú práva a povinnosti vyplývajúce zo zákona č. 121/2000 Sb., autorského zákona, v znení neskorších predpisov, a skutočnosť, že České vysoké učení technické v Praze má právo na uzavrenie licenčnej zmluvy o použití tejto práce ako školského diela podľa § 60 odst. 1 autorského zákona.

V Prahe 7. mája 2019 .....

Odkaz na túto prácu

Rosocha, Igor. *Mobilná aplikácia kvalita ovzdušia v Prahe*. Diplomová práca. Praha: České vysoké učení technické v Praze, Fakulta informačních technologií, 2019.

Abstrakt

Diplomová práca sa zaoberá problematikou vizualizácie kvality ovzdušia v Prahe prostredníctvom mobilnej aplikácie pre operačný systém iOS. Práca obsahuje podrobnú analýzu existujúcich aplikácií podobného zamerania, dátových zdrojov merania znečistenia ovzdušia a technológií na realizáciu serverovej časti, ktorá bude dáta aplikácii poskytovať. Súčasťou práce je návrh, implementácia, testovanie a reálne nasadenie vlastného servera a mobilnej aplikácie.

Klíčová slova kvalita ovzdušia, mobilná aplikácia, iOS, Swift, Vapor

Abstract

This master's thesis is focused on the issue of air quality visualization in the city of Prague through a mobile application for the operating system iOS. Thesis contains a detailed analysis of existing applications dealing with similar issue, data sources of air pollution measurement and technologies to implement the server, which will provide data for the application. Thesis also describes the entire process of design, implementation, testing and real deployment of own server and mobile application.

Keywords air quality, mobile application, iOS, Swift, Vapor

Obsah

Zoznam obrázkov

Úvod

Látky znečisťujúce ovzdušie unikajú do životného prostredia prostredníctvom rôznych prírodných či antropogénnych aktivít a môžu mať nepriaznivé účinky na ľudské zdravie a životné prostredie. Pravidelné vystavovanie znečistenému ovzdušiu môže mať na ľudské zdravie akútne aj chronické účinky [1]. Svetová zdravotnícka organizácia WHO (z ang. World Health Organization) odhaduje, že znečistenie ovzdušia prispieva každoročne k približne 800 000 predčasným úmrtiam, čím sa radí na 13. priečku medzi najčastejšími príčinami úmrtia na svete [2]. Je preto potrebné ľudí informovať o aktuálnom stave kvality a znečistenia ovzdušia, a vďaka svojej popularite sú na tento účel vhodnou platformou mobilné aplikácie.

Táto práca poskytuje dôkladnú analýzu existujúcich mobilných aplikácií, ktoré poskytujú informácie o kvalite a znečistení ovzdušia a dátových zdrojov merania znečistenia ovzdušia v meste Praha. Cieľom práce je navrhnúť a implementovať mobilnú aplikáciu pre operačný systém iOS, ktorá bude ľudí žijúcich v Prahe informovať o kvalite ovzdušia a server, ktorý bude dáta aplikácii poskytovať. Vďaka dostupným údajom budú môcť používatelia kedykoľvek zistiť aktuálnu kvalitu ovzdušia vo svojom okolí, ako aj na mapovom podklade v rámci celého mesta Praha a naplánovať tak svoje každodenné aktivity s ohľadom na aktuálne znečistenie ovzdušia a svoje zdravie.

Kapitola 1

Analýza existujúcich aplikácií podobného zamerania

Cieľom úvodnej kapitoly práce je analyzovať už existujúce mobilné aplikácie poskytujúce informácie o kvalite a znečistení ovzdušia za účelom identifikovania ich silných stránok, ako aj hlavných nevýhod a nedostatkov. Podmienkou analyzovaných aplikácií bola možnosť bezplatného stiahnutia z obchodu App Store[1] pre zariadenia s operačným systémom iOS. App Store slúži používateľom zariadení Apple ako univerzálne miesto pre nakupovanie, sťahovanie a aktualizovanie aplikácií, ktoré využívajú [3].

1.1 SmogAlarm 2.0

SmogAlarm 2.0[2] je mobilná aplikácia realizovaná českou všeobecne prospešnou spoločnosťou Čisté Nebe. Aplikácia zobrazuje aktuálne dáta meracích staníc Českého hydrometeorologického ústavu formou zoznamu. Ponúka prehľad indexu kvality ovzdušia a hodnôt koncentrácií znečisťujúcich látok v ovzduší v rámci jednotlivých staníc.

Aplikácia SmogAlarm 2.0 však nespokytuje údaje o kvalite ovzdušia s ohľadom na aktuálnu polohu používateľa, či podrobnejšiu vizualizáciu kvality ovzdušia v meste Praha. Na mapovom podklade vizualizuje iba umiestnenie jednotlivých meracích staníc. Ako možno vidieť na Obr. 1.1, jedná sa o jednoduchý, no zrozumiteľný prehľad dát získaných z jednotlivých meracích staníc, ktoré je možné zoradiť podľa abecedy či indexu kvality ovzdušia. Podľa dostupných údajov[2] o histórii verzií z obchodu App Store bola aplikácia naposledy aktualizovaná v októbri roku 2018 a aktualizácie sú vydávané raz ročne.

[1] `https://www.apple.com/ios/app-store/`

[2] `https://itunes.apple.com/us/app/smogalarm-2-0/id1212374819`

Obr. 1.1: SmogAlarm 2.0

1.2 Breezometer Air Quality Index

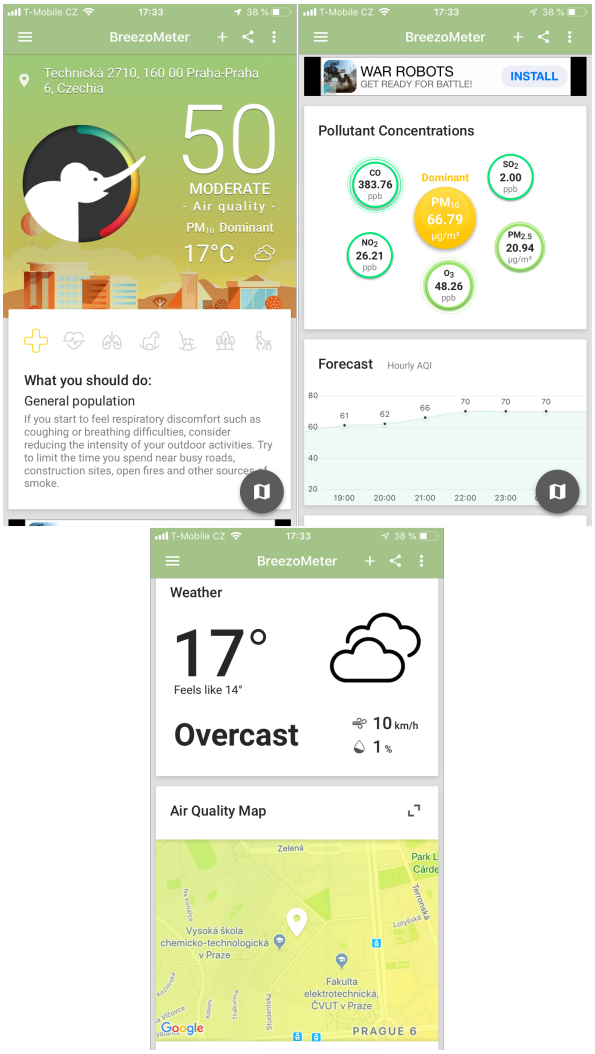
Spoločnosť Breezometer spracováva globálne dáta o kvalite a znečistení ovzdušia vo viac ako 90 krajinách sveta, a tieto dáta následne poskytuje vo forme plateného API (z ang. Application Programming Interface). Zadarmo však ponúka vlastnú mobilnú aplikáciu Breezometer Air Quality Index[3].

Mobilná aplikácia zobrazuje informácie o kvalite ovzdušia na základe polohy používateľa, rovnako ako aktuálne hodnoty znečisťujúcich látok v ovzduší. Navyše používateľom poskytuje univerzálne rady, aké aktivity je vhodné, resp. nevhodné vykonávať na základe aktuálneho stavu ovzdušia. Obsahuje taktiež údaje o aktuálnom počasí, predpoveď vývoja hodinového indexu kvality ovzdušia, či množstvo peľu v ovzduší. Na mapovom podklade vizualizuje kvalitu ovzdušia vo forme vlastného indexu kvality BAQI (z ang. Breezometer Air Quality). Ako uvádzajú v informáciách[3] o aplikácii, tento index je založený na skúsenostiach viacerých popredných environmentálnych agentúr a ich štandardov a na zdravotných vplyvoch súvisiacich s vystavením znečisťeniu ovzdušia.

Spoločnosť však nikde neuvádza, čo je zdrojom ich dát pre Českú republiku a akým spôsobom sú tieto dáta spracovávané. Naopak však vo svojich

[3] `https://itunes.apple.com/us/app/breezometer-air-quality-index/id989623380`

podmienkach používania [4] uvádza, že zobrazované informácie môžu byť oneskorené a nepresné, a preto nezaručuje, že obsah dostupný v aplikácii je presný, úplný, spoľahlivý, aktuálny alebo bezchybný. Mobilná aplikácia síce poskytuje predpoveď, no paradoxne neobsahuje žiadne údaje o historickom vývoji hodnôt jednotlivých znečisťujúcich látok v ovzduší. Rozhranie pôsobí neprehľadne, a to najmä vďaka množstvu reklám či rôznych informácií, ktoré nie sú rozdelené na viac logických častí, no zhromaždené v rámci jednej základnej obrazovky, ktorá je zachytená na Obr. 1.2. Aplikácii taktiež chýba lokalizácia do českého, či slovenského jazyka, čo môže spôsobovať niektorým používateľom v Českej republike problémy.



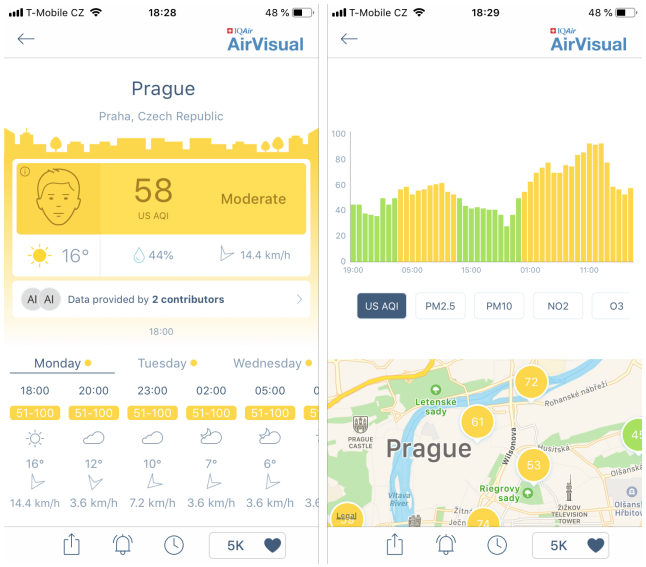
Obr. 1.2: Breezometer Air Quality Index

1.3 AirVisual Air Quality Forecast

Mobilná aplikácia AirVisual Air Quality Forecast[4] švajčiarskej spoločnosti IQAir pokrýva dáta o kvalite ovzdušia z viac ako 10 000 miest globálne. Ponúka aktuálne a historické dáta vývoja kvality ovzdušia a hodnôt jednotlivých znečisťujúcich látok, ako aj predpoveď na najbližších 7 dní či počet hodín denne strávených v znečistenom ovzduší. Existuje aj vo verzii pre chytré hodinky Apple Watch.

Dáta pre Českú republiku sú získavané z meracích staníc Českého hydrometeorologického ústavu. Aplikácia je však výnimočná v tom, že používateľ si môže priamo zakúpiť vlastný prístroj, ktorý meria okamžité a presné údaje o kvalite ovzdušia v interiéri aj exteriéri. Tento prístroj je následne možné prepojiť s aplikáciou a namerané dáta zdieľať s ostatnými používateľmi. V rámci Prahy však v čase vzniku tejto práce boli napojené iba 2 takéto stanice.

Aplikácia prezentuje namerané dáta takmer identickým spôsobom ako česká aplikácia SmogAlarm 2.0 (pozri Obr. 1.3). Taktiež neposkytuje žiadne podrobnejšie údaje o kvalite ovzdušia vzhľadom na aktuálnu polohu používateľa, či vizualizáciu kvality ovzduši v meste Praha iným spôsobom, ako zobrazením polohy meracích staníc na mapovom podklade a chýba jej lokalizácia do českého, či slovenského jazyka. Napriek tomu sa však aplikácia AirVisual Air Quality Forecast nachádza medzi pätdesiatimi najlepšími aplikáciami v kategórii *Počasie* v obchode App Store.



Obr. 1.3: AirVisual Air Quality Forecast

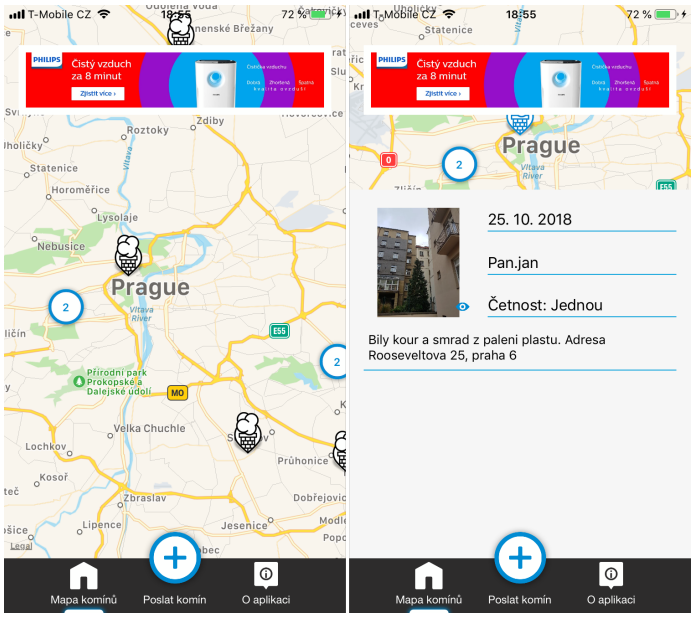
[4] https://itunes.apple.com/us/app/airvisual-air-quality-forecast/id1048912974

1.4 Čistý komín 2.0

Ďalším projektom, ktorý realizuje spoločnosť Čisté Nebe, je Čistý komín. Hlavným cieľom projektu je motivovať verejnosť k ekologickejšiemu spôsobu vykurovania v domácnostiach tým, že mapuje komíny, z ktorých pravidelne vychádza špinavý dym zhoršujúci kvalitu ovzdušia.

Za týmto účelom bola vytvorená mobilná aplikácia Čistý komín 2.0[5], v rámci ktorej môžu používatelia odfotiť komín znečisťujúci ovzdušie a pridať ho priamo na mapu, ktorá sa následne zobrazuje všetkým používateľom aplikácie (pozri Obr. 1.4), a tým upozorniť na znečisťovanie ovzdušia v ich okolí.

Aplikácia je jednoduchá a spĺňa svoj účel, no okrem spomínaných komínov neposkytuje žiadne podrobnejšie informácie o kvalite a znečistení ovzdušia. V rámci Prahy je počet takto nahlásených komínov príliš nízky, čo indikuje malú používateľskú základňu.



Obr. 1.4: Čistý Komín 2.0

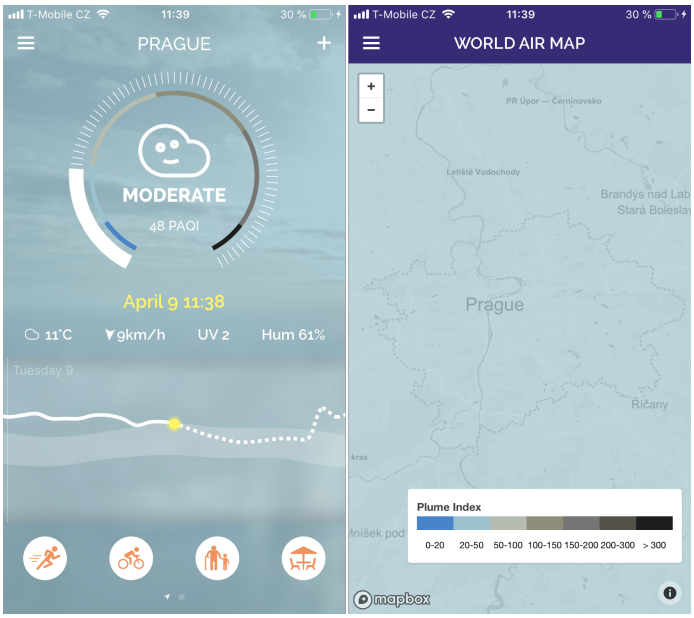
1.5 Plume Air Report

Mobilná aplikácia Plume Air Report[6] zobrazuje úroveň znečistenia ovzdušia v reálnom čase a predpoveď, ako sa bude kvalita ovzdušia vyvíjať v priebehu nasledujúcich 24 hodín.

[5] https://itunes.apple.com/cz/app/cisty-komin-2-0/id1164464855
[6] https://itunes.apple.com/us/app/plume-air-report-pollution/id950289243

Aplikácia disponuje vlastnou umelou inteligenciou, vďaka ktorej poskytuje používateľom časové odporúčania na vykonávanie ich každodenných aktivít bez toho, aby sa vystavovali nadmernému množstvu znečistenia. Pri každom meraní sú uvedené hodnoty jednotlivých znečisťujúcich látok a index dôveryhodnosti, ktorý udáva, do akej miery sú namerané dáta spoľahlivé.

Problém však nastal pri prvom spustení aplikácie, ktoré trvalo niekoľko minút, čo je pri spúšťaní mobilnej aplikácie neštandardne dlhý čas. Po spustení aplikácie je používateľovi prezentovaný index kvality ovzdušia v rámci mesta, v ktorom sa nachádza. Pre Prahu je tento index rovnaký v rámci celého mesta, nie je špecifický pre konkrétnu polohu. Vizualizácia dát na mapovom podklade je neprehľadná a málo špecifická pre jednotlivé zóny a celkové grafické spracovanie pôsobí zastaralým dojmom (pozri Obr. 1.5). Chýba taktiež lokalizácia do českého, či slovenského jazyka. Aplikácia bola podľa údajov[6] o histórii verzií z obchodu App Store naposledy akutalizovaná v auguste roku 2018, čo v čase vzniku tejto práce znamenalo viac ako 8 mesiacov bez aktualizácie.



Obr. 1.5: Plume Air Report

1.6 ČHMÚ Plus

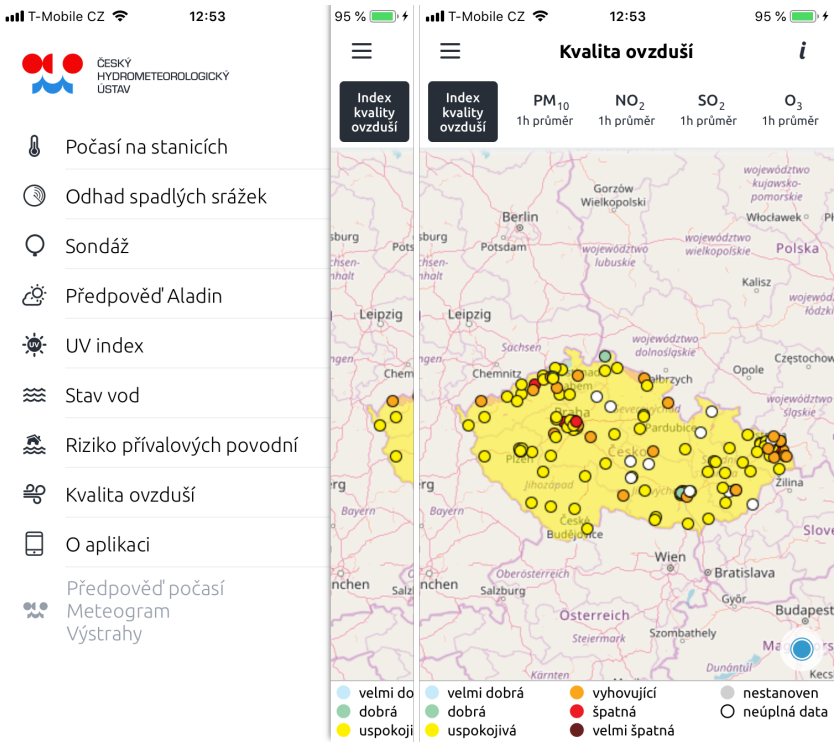
Poslednou analyzovanou mobilnou aplikáciou bola ČHMÚ Plus[7]. Hlavným dôvodom, prečo bola táto aplikácia analyzovaná ako posledná je fakt, že pri

[7] https://itunes.apple.com/cz/app/chmu-plus/id1417843679

vyhľadávaní aplikácií venujúcich sa kvalite a znečisteniu ovzdušia v obchode App Store sa aplikáciu nepodarilo nájsť vôbec. Skutočnosť, že takáto aplikácia vôbec existuje, sa podarilo identifikovať až na stránke Českého hydrometeorologického ústavu[8].

ČHMÚ Plus zobrazuje množstvo dát, ktoré poskytuje Český hydrometeorologický ústav, od aktuálneho počasia, odhadu spadnutých zrážok a UV indexu až po stav vôd či riziko prívalových povodní (pozri Obr. 1.6). Medzi tieto dáta patrí aj kvalita ovzdušia získavaná z meracích staníc pre celú Českú republiku. Jednotlivé stanice sú vizualizované na mapovom podklade, po rozkliknutí ponúkajú históriu vývoja jednotlivých znečisťujúcich látok a indexu kvality ovzdušia formou tabuľky.

Z hľadiska vizualizácie a poskytovania dát o kvalite a znečistení ovzdušia sa tak jedná o ďalšiu aplikáciu, ktorá okrem zobrazovania dát získaných z jednotlivých meracích staníc ČHMÚ neposkytuje žiadne ďalšie dáta, ktoré by boli podrobnejšie pre aktuálnu polohu používateľa či mesto Praha. Aplikácia síce poskytuje množstvo ďalších informácií z iných oblastí klimatológie, no jej dostupnosť pre používatelov vyhľadávajúcich aplikácie venujúce sa kvalite a znečisteniu ovzdušia v obchode App Store je nedostačujúca.



Obr. 1.6: ČHMÚ Plus

[8] http://portal.chmi.cz/informace-pro-vas/mobilni-aplikace/o-mobilni-aplikaci

1.7 Výsledok analýzy aplikácií

Ako vyplýva z vykonanej analýzy, aplikácií poskytujúcich informácie ohľadom kvality a znečistenia ovzdušia nie je málo. Väčšina aplikácií využíva v rámci územia Českej republiky dáta meracích staníc Českého hydrometeorologického ústavu, aplikácia Breezometer využíva vlastné API bez uvedenia zdroja dát.

Žiadna z analyzovaných aplikácií však neposkytuje podrobnejšie informácie ohľadom kvality a znečištenia ovzdušia pre mesto Praha. Zároveň jedinou aplikáciou, ktorá zohľadňuje aktuálnu polohu používateľa je Breezometer, ostatné aplikácie neposkytujú dáta na základe aktuálnej polohy. Vo väčšine prípadov si musí používateľ zvoliť mesto, prípadne stanicu, ktorej údaje ho zaujímajú. Väčšine aplikácií taktiež chýba lokalizácia do českého, či slovenského jazyka.

Kapitola 2

Dátové zdroje merania znečistenia ovzdušia v Prahe

V súčasnosti existujú 2 zdroje poskytujúce verejne dostupné dáta, ktoré informujú o kvalite a znečistení ovzdušia v hlavnom meste Praha:

1. otvorené dáta Inštitútu plánovania a rozvoja hlavného mesta Prahy [5],
2. dáta meracích staníc Českého hydrometeorologického ústavu [6].

2.1 Otvorené dáta Inštitútu plánovania a rozvoja hlavného mesta Prahy

Inštitút plánovania a rozvoja (IPR) hlavného mesta Prahy už od roku 2002 poskytuje geografické dáta formou webových služieb zdarma, ktorých využívanie sa rozšírilo najmä v roku 2011 v súvislosti s rozvojom projektu Pražský Geoportal Praha [7]. Od roku 2015 sú tieto geografické dáta poskytované taktiež formou otvorených dát.

Otvorené dáta sú dáta, ktoré sú voľne a bezplatne dostupné pre každého. Ich využitie nie je obmedzené žiadnými autorskými právami, patentmi či inými legislatívnymi prekážkami. Vďaka tomu môžu byť otvorené dáta voľne spracovávané a využívané v rámci výskumu, ale aj na komerčné účely či k tvorbe rôznych aplikácií [8]. Otváranie dát verejnej správy, podobne ako to robí plánovania a rozvoja hlavného mesta Prahy, so sebou prináša množstvo pozitívnych perspektív [7]:

- poskytovanie kvalitnejších služieb pre občanov,
- zvýšenie informovanosti obyvateľstva,
- podpora občianskej spoločnosti,

- zvýšenie ekonomického potenciálu,
- zvýšenie transparentnosti verejnej správy,
- efektívnejšia práca verejnej správy,
- redukcia administratívy a záťaže IT infraštruktúry verejnej správy.

Inštitút plánovania a rozvoja hlavného mesta Prahy publikuje otvorené dáta v rôznych kategóriách, medzi ktoré patrí aj ovzdušie a bonita klímy z hľadiska znečistenia ovzdušia. Dáta sú pravidelné aktualizované a mimoriadne dôkladné, obsahujú až 221 samostatných zón pre mesto Praha. Tie tvoria vrstvu, ktorá vznikla kombináciou rozloženia polí koncentrácií znečisťujúcich látok oxidu dusičitého (NO_2) a suspendovaných prachových častíc (PM_{10}). Výsledná vrstva bola následne preklasifikovaná do piatich kategórií 1 až 5, kde hodnota 1 indikuje najlepšiu, a hodnota 5 najhoršiu klimatologickú charakteristiku [5]. Dáta sú poskytované vo formátoch GeoJSON (z ang. Geographic JavaScript Object Notation), DXF (z ang. Drawing Exchange Format), GML (z ang. Geography Markup Language) a shapefile.

Vďaka množstvu poskytovaných formátov, no hlavne svojej presnosti a veľkému počtu unikátnych zón sú tieto dáta ideálne pre využitie v rámci navrhovanej aplikácie na účely podrobnej vizualizácie kvality ovzdušia v Prahe na mapovom podklade.

2.2 Dáta meracích staníc Českého hydrometeorologického ústavu

Český hydrometeorologický ústav[9] (ČHMÚ) je príspevkovou organizáciou zriaďovanou Ministerstvom životného prostredia Českej republiky, zaoberajúcou sa najmä predpoveďou počasia, ale aj poskytovaním odborných služieb v oboroch meteorológie a klimatológie.

Na ich stránke [6] môžeme nájsť v rámci Českej republiky údaje zo 143 meteorologických staníc, pričom v Prahe sa nachádza 17 z nich. Každá zo staníc poskytuje v rámci merania kvality ovzdušia nasledovné parametre:

- unikátny kód stanice,
- názov stanice,
- klasifikácia stanice,
- vlastník stanice,
- index kvality ovzdušia stanice,

[9] `http://portal.chmi.cz`

- hodinová koncentrácia oxidu siričitého (SO_2),
- hodinová koncentrácia oxidu dusičitého (NO_2),
- osemhodinová koncentrácia oxidu uhoľnatého (CO),
- hodinová koncentrácia suspendovaných prachových častíc (PM_{10}),
- denná koncentrácia suspendovaných prachových častíc (PM_{10}),
- hodinová koncentrácia ozónu (O_3),
- hodinová koncentrácia suspendovaných jemných prachových častíc ($PM_{2.5}$).

Veľkou výhodou týchto meracích staníc je najmä fakt, že dáta su aktualizované každú hodinu, čo v praxi znamená, že neustále poskytujú aktuálne dáta kvality a znečistenia ovzdušia. Zároveň sú jediným zdrojom dát v Prahe a Českej republike, ktorý poskytuje údaje o hodnotách a koncentráciách jednotlivých znečisťujúcich látok v ovzduší.

Dôležité je však poznamenať, že tieto dáta su označované ako neverifikované. Toto označenie indikuje, že namerané hodnoty z meracích staníc sú odosielané priamo, bez procesu verifikácie alebo úprav zamestnancami ČHMÚ a môžu obsahovať jemné odchýlky. Pri dlhodobom sledovaní vývoja kvality a znečistenia ovzdušia sa využívajú verifikované hodnoty, pre potreby navrhovanej aplikácie však drobné odchýlky nebudú spôsobovať žiaden problém, keďže aplikácia vyžaduje aktuálne hodnoty. Stále sa totiž jedná o jediné a najpresnejšie dáta ohľadom hodnôt a koncentrácií znečisťujúcich látok v ovzduší v Prahe a Českej republike, ktoré sú verejne dostupné.

Dáta sú poskytované vo formátoch JSON (z ang. JavaScript Object Notation) a XML (z ang. eXtensible Markup Language) a ponúkané pod licenciou Creative Commons CC BY-NC-ND 3.0, čo v praxi znamená, že je možné tieto dáta akýmkoľvek spôsobom využívať, rozmnožovať a distrubuovať, okrem možnosti priameho zásahu a úprav. Aktualizácia dát každú hodinu, ako aj hodnoty koncentrácií jednotlivých znečisťujúcich látok v ovzduší robia tieto dáta ideálnymi pre navrhovanú mobilnú aplikáciu pri zobrazovaní hodnôt koncentrácií znečisťujúcich látok používateľovi v jeho okolí, ako aj v rámci celého územia mesta Praha.

Kapitola 3

Technológie na realizáciu serverovej časti

Dôležitou časťou práce je analýza a voľba technológií vhodných na realizáciu serverovej časti. Keďže v súčasnosti existuje veľké množstvo jazykov na vytvorenie efektívnej serverovej časti pre mobilnú aplikáciu, základnými kritériami pri výbere boli najmä výkonnosť a požiadavky na pamäť.

3.1 Výber jazyka

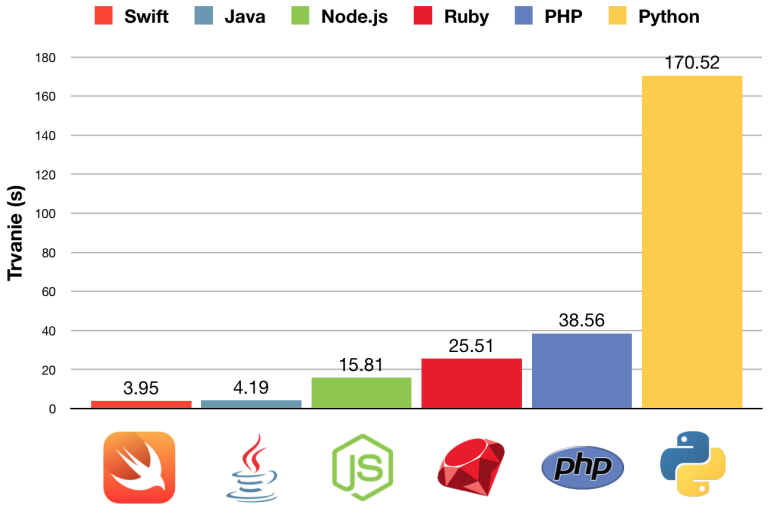
Pri výbere jazyka boli zohľadňované verejne dostupné merania, ktoré sú pravidelne realizované v rámci projektu Benchmarks Game[10] . Tento voľne dostupný softvérový projekt porovnáva celkový výkon, spotrebu pamäte, celkový procesný čas procesora na vykonanie algoritmu nad všetkými vláknami a veľkosti zdrojového kódu riešenia rôznych populárnych programovacích jazykov na danej podmnožine jednoduchých algoritmov. Všetky merania sú vykonávané na operačnom systéme Linux a procesore so štyrmi jadrami.

Porovnávané boli jazyky, ktoré sú v praxi využívané na realizáciu serverovej časti, konkréte Java, Node.js, Ruby, PHP a Python s jazykom Swift, ktorý čoraz viac naberá na popularite [9]. Na základe dostupných meraní na algoritme spectral-norm [10] boli vytvorené grafy, ktoré porovnávajú výkonnosť (pozri Obr. 3.1) a spotrebu pamäte (pozri Obr. 3.2) jednotlivých jazykov.

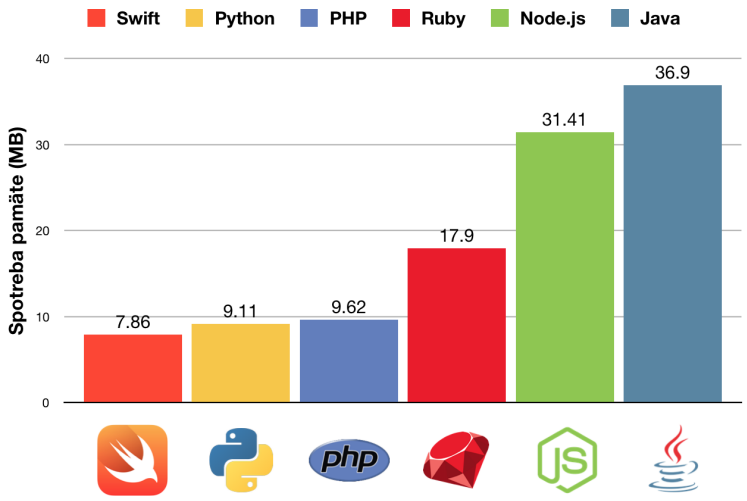
Ako môžeme vidieť na Obr. 3.1, jazyk Swift dosahuje v meraniach trvanie 3.95 sekundy. Java dosahuje približne rovnaký čas 4.19 sekundy. Kým časy jazykov Swift a Java sú takmer identické, Node.js dosahuje až 15.81 sekundy, čo je zhruba 4-krát pomalšie. Ostatné jazyky sú následne niekoľkokrát pomalšie. Zo strany výkonnosti môžeme vidieť, že Swift je skutočne rýchlym jazykom na implementáciu na serveri. Pozoruhodné sú však aj jeho nízke požiadavky na pamäť. Pri pohľade na Obr. 3.2, ktorý znázorňuje výsledky rovnakého me-

[10] `https://benchmarksgame-team.pages.debian.net/benchmarksgame/`

rania, vyžaduje Swift 7.86 MB pamäte, zatiaľ čo Java vyžaduje až 36.9 MB, čo v praxi znamená, že s jazykom Swift získavame za takmer 5-krát nižšiu spotrebu pamäte rovnaký výkon. Nízku spotrebu pamäte mali taktiež jazyky Python a PHP, no vzhľadom k tomu, že v meraniach výkonnosti dosahovali najhoršie výsledky môžeme usúdiť, že práve jazyk Swift poskytuje najvyššiu výkonnosť pri najnižšej spotrebe pamäte v porovnaní s ostatnými jazykmi.



Obr. 3.1: Merania výkonnosti



Obr. 3.2: Merania spotreby pamäte

Okrem najvyššiej výkonnosti pri najnižšej spotrebe pamäte poskytuje tzv. serverový Swift (z ang. server-side Swift) v našom prípade aj ďalšie výhody:

- samotná aplikácia bude implementovaná taktiež v jazyku Swift,
- znovupoužiteľnosť kódu - potenciálne opätovné využitie kódu medzi serverom a aplikáciou, ako aj možnosť vytvárať rámce a balíky, ktoré bude možné využiť aj priamo v aplikácii,
- možnosť využitia rovnakého vývojového prostredia Xcode pri vývoji servera a aplikácie,
- všetky serverové Swift frameworky (aplikačné rámce) sú open-source (otvorené),
- rozsiahla a aktívna komunita - Swift Server Work Group[11].

3.2 Frameworky jazyka Swift

Voľba jazyka Swift na vytvorenie serverovej časti bola na základe vykonanej analýzy jednoznačnou voľbou. Existuje hneď niekoľko Swift frameworkov, ktoré túto funkcionalitu ponúkajú, konkrétne Kitura, Perfect a Vapor.

3.2.1 Kitura

Kitura[12] je bezplatný open-source framework pre tvorbu serverových aplikácií v jazyku Swift, vydaný 9. februára roku 2016 a vyvinutý spoločnosťou IBM pod licenciou Apache 2.0. IBM sa aktívne zapája do vývoja jazyku Swift od jeho založenia [11].

Kitura začala postupne naberať na popularite ako technológia na strane servera najmä vďaka článku [12] z roku 2016, v ktorom sa viceprezident IBM Mike Gilfix vyjadril, že Swift je pripravený na využitie na podnikových produkčných serveroch. Hlavnou výhodou tohto frameworku je jednoznačne podpora IBM, z čoho možno predpokladať, že jeho vývoj bude v budúcnosti pokračovať a prinesie ďalšie vylepšenia a nové funkcionality.

Kitura v súčasnosti poskytuje štandardnú sadu funkcionalít, ktorú používateľ očakáva od akéhokoľvek frameworku na strane servera (HTTP protokol, URL smerovanie, podpora štruktúrovaných dát formátu JSON), ako aj niekoľko pokročilejších. Je unikátna vo svojom prístupe k databázam, pre SQL (z ang. Structured Query Language) databázy využíva databázovú abstraktnú vrstvu Kuery[13], ktorá zjednocuje drobné rozdiely medzi nimi do jednotného API. Kitura podporuje aj databázy, ktoré nie sú založené na jazyku

[11] `https://swift.org/server/`
[12] `https://www.kitura.io`
[13] `https://hub.com/IBM-Swift/Swift-Kuery`

SQL, konkrétne Redis, CouchDB, Apache Cassandra, a ScyllaDB, ale prostredníctvom natívnych balíkov, ktoré nesúvisia s Kuery.

Nasadenie Kitury je kompatibilné s prostredím macOS a Ubuntu. Rozdielom oproti ostatným frameworkom je však možnosť využitia natívnej macOS aplikácie, ktorú je vďaka podpore IBM možné integrovať priamo do ich platformy Bluemix[14]. Využitie Kitury má teda zrejmé výhody najmä pre projekty a aplikácie, ktoré využívajú IBM Bluemix pre potreby hostingu, či potrebujú integrovať iné IBM služby.

3.2.2 Perfect

Perfect[15] je open-source aplikačný server, webový a serverový framework napísaný v jazyku Swift, vytvorený kanadským startupom PerfectlySoft. Poskytuje nástroje na vývoj webových a iných REST (z ang. Representational State Transfer) služieb a štruktúry pre vývojárov pracujúcich s databázami, webovými službami a webovými stránkami [11]. Prvá verzia Perfect 1.0 bola vydaná 23. novembra 2015, krátko pred tým, ako sa jazyk Swift stal oficiálne open-source [13].

Perfect poskytuje veľmi široký súbor funkcionalít, vrátane takých, ktoré žiaden iný zo Swift frameworkov neposkytuje, ako napríklad podpora databázového jazyka MariaDB, natívne spracovanie súborov a adresárov, či možnosť ich rozbaľovania a zabaľovania. Perfect však nemá práve najjednoduchšiu syntax. Vzhľadom na jeho zameranie v prvom rade na výkonnosť boli mnohé z dostupných funkcionalít implementované za cenu zhoršenia čitateľnosti a zrozumiteľnosti kódu. Je veľmi technicky orientovaný a nie je vhodný pre vývojárov s nedostatkom skúseností s tvorbou serverovej časti, rovnako ako pre aplikácie, ktoré v rámci serveru okrem základnej REST funkcionality nevyžadujú žiadne komplexnejšie operácie [11].

Nasadenie je kompatibilné, podobe ako pri Kiture, s prostredím macOS a Ubuntu. Perfect navyše podporuje Docker kontajnery, ktoré môžu byť využité na nasadenie v rôznych iných prostrediach.

3.2.3 Vapor

Vapor[16] je open-source webový framework napísaný v jazyku Swift, slúžiaci na vytváranie REST API a webových aplikácií [11]. Vapor 1.0 oficiálne predstavili v januári roku 2016 jeho tvorcovia Tanner Nelson a Logan Wright. Vapor sa postupne vyvíja vo veľmi flexibilný a výkonný framework, ktorý predstavuje veľmi zaujímavú perspektívu do budúcnosti. Jeho súčasná verzia Vapor 3 je asynchrónna a má podporu verzie jazyka Swift 5.

[14] `https://www.ibm.com/cloud-computing/bluemix/`
[15] `https://perfect.org`
[16] `https://vapor.codes`

Hlavné výhody frameworku Vapor sú najmä veľmi jednoduchá syntax a obrovská a veľmi aktívna komunita. Vapor je taktiež najpopulárnejší spomedzi všetkých serverových Swift frameworkov, o čom svedčí aj fakt, že na portáli GitHub má v súčasnosti najvyšší počet hviezd (z ang. stars) - cez 16 000 [14]. Vapor je taktiež jediný z porovnávaných frameworkov, ktorý je celý napísaný v jazyku Swift a spolieha sa výhradne na Swift knižnice.

Exkluzivita jazyku Swift robí Vapor výrazne odlišným. Vapor tak ponúka jednoduché a zrozumiteľné asynchrónne API, ako aj odstránenie závislosti od mnohých tretích strán, narozdiel od frameworkov Kitura či Perfect. Má jednoznačne najčistejšiu a najjednoduchšiu syntax a najväčšiu komunitu, čo robí Vapor ideálnym pre projekty, ktoré si vyžadujú jednoduchosť, dobrú čitateľnosť a rýchlu implementáciu [11].

Databázová podpora frameworku Vapor je veľmi široká. Nielenže poskytuje natívne štruktúry pre prácu s databázami SQL, MySQL, Sqlite a PostgreSQL, ale obsahuje aj natívnu podporu pre Redis a MongoDB. Okrem toho má aj vlastný systém na objektovo-relačné mapovanie Fluent[17], ktorý v súčasnosti podporuje SQL databázové jazyky.

Široko dostupné vzdelávacie zdroje sú ďalšou prednosťou frameworku Vapor. Ich dokumentácia je veľmi dobre a jasne organizovaná a ľahko sa používa bez toho, aby pôsobila vyčerpávajúco. Vapor sa navyše od ostatných frameworkov odlišuje vlastnou výučbovou platformou[18], ktorá zhromažďuje videá prezentácií a články popisujúce implementáciu základných funkcionalít.

Nasadenie je opäť kompatibilné s prostredím macOS a Ubuntu, vrátane podpory Docker kontajnerov. Vapor navyše poskytuje vlastný cloud server[19], ktorý sľubuje výkon, efektívnosť a dostupnosť s minimálnou potrebou konfigurácie.

3.3 Výber frameworku

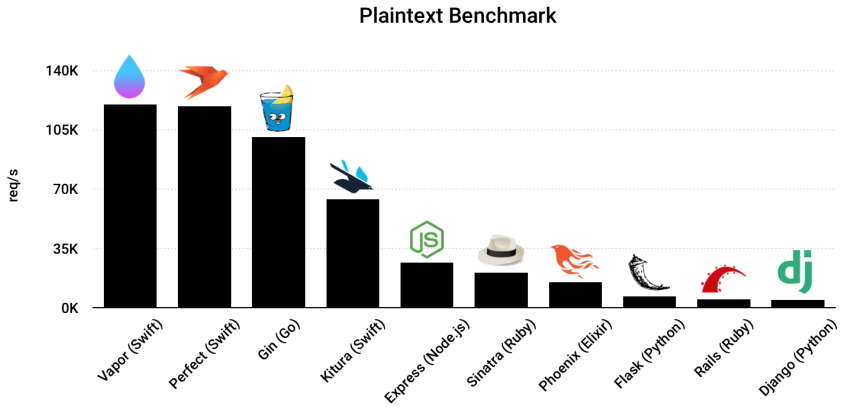
Napriek tomu, že všetky porovnávané frameworky poskytujú základnú funkcionalitu vyžadovanú na serveri, sú veľmi odlišné pokiaľ ide o ich schopnosti a zdroje. Každý z nich má svoje unikátne vlastnosti.

Dôležitým kritériom pri výbere frameworku však bola aj rýchlosť spracovania požiadaviek a priemerná latencia. Zdrojom týchto dát boli oficiálne dostupné merania [15], ktoré porovnávali frameworky Vapor, Perfect a Kitura spolu s inými populárnymi serverovými frameworkmi. Výsledky týchto meraní sú zobrazené na Obr. 3.3, ktorý znázorňuje celkový počet požiadaviek spracovaných za sekundu, a na Obr. 3.4, ktorý znázorňuje priemernú latenciu.

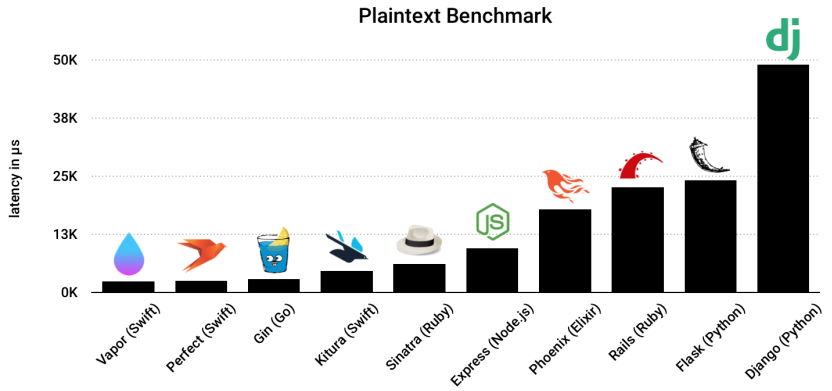
Z priložených meraní môžeme vidieť, že zaťial čo Vapor a Perfect dosahujú takmer identický výkon pri rovnakej latencii, Kitura výrazne zaostáva. Hlav-

[17] `https://github.com/vapor/fluent`
[18] `https://vapor.university`
[19] `https://vapor.cloud`

nou výhodou frameworku Kitura je možnosť integrácie so službami spoločnosti IBM. Žiadnu z ich služieb však navrhovaná mobilná aplikácia integrovať nebude. Z týchto dôvodov môžeme framework Kitura z výberu vylúčiť. Hoci možno Perfect považovať za najkomplexnejší, v rámci navrhovanej mobilnej aplikácie sa javí ako ideálna voľba Vapor, a to nielen vďaka svojej rýchlosti a nízkej latencii, ale najmä vďaka svojej nenáročnosti, asynchrónnej implementácii, aktívnej komunite a zameraniu výhradne na jazyk Swift.



Obr. 3.3: Počet požiadaviek spracovaných za sekundu [15]



Obr. 3.4: Priemerná latencia [15]

Kapitola 4

Návrh

Nasledujúca kapitola je zameraná na návrh mobilnej aplikácie a servera, ktorý bude dáta o kvalite ovzdušia aplikácii poskytovať. Popisuje základné funkčné a nefunkčné požiadavky, použitú architektúru, celý proces návrhu používateľského rozhrania a návrh dátových štruktúr serveru.

4.1 Funkčné požiadavky aplikácie

Pred začiatkom procesu implementácie je nevyhnutné jednoznačne definovať funkčné požiadavky. Funkčné požiadavky popisujú softvérovú funkcionalitu, ktorú musí vývojár aplikácie zabezpečiť tak, aby boli používatelia schopný splniť svoje úlohy [16]. Vo svojej podstate určujú, čo presne musí vývojár naprogramovať pre zabezpečenie základných funkcionalít aplikácie.

Medzi funkčné požiadavky navrhovanej aplikácie patria:

1. zobrazenie kvality ovzdušia na základe aktuálnej polohy používateľa,
2. zobrazenie hodnôt koncentrácií znečisťujúcich látok v ovzduší na základe aktuálnej polohy používateľa,
3. zobrazenie kvality ovzdušia na mapovom podklade v rámci mesta Praha,
4. zobrazenie zoznamu meracích staníc ČHMÚ v rámci mesta Praha spolu s aktuálnym indexom kvality ovzdušia,
5. zobrazenie detailu meracej stanice ČHMÚ spolu s históriou vývoja koncentrácií jednotlivých znečisťujúcich látok v ovzduší za posledných 24 hodín,
6. možnosť manuálnej aktualizácie hodnôt.

4.2 Nefunkčné požiadavky aplikácie

Dôležitú úlohu pri návrhu softvérových aplikácií zohrávajú okrem funkčných aj tzv. nefunkčné požiadavky aplikácie. Medzi ne môžeme zaradiť globálne požiadavky na vývoj, výkon, spoľahlivosť, udržateľnosť a prenositeľnosť [17].

Pre navrhovanú aplikáciu sú nefunkčné požiadavky definované nasledovne:

1. aplikácia bude dostupná pre všetky zariadenia s operačným systémom iOS,
2. aplikácia bude v základnej verzii dostupná pre chytré hodinky Apple Watch,
3. dáta, ktoré bude aplikácia prezentovať a vizualizovať, budú sťahované a pravidelne aktualizované zo vzdialeného servera,
4. aplikáciu bude možné využívať aj mimo územia Prahy na zobrazovanie informácií z meracích staníc ČHMÚ,
5. aplikácia bude lokalizovaná okrem anglického aj do českého a slovenského jazyka.

4.3 Funkčné požiadavky serveru

Na server sú kladené nasledujúce funkčné požiadavky:

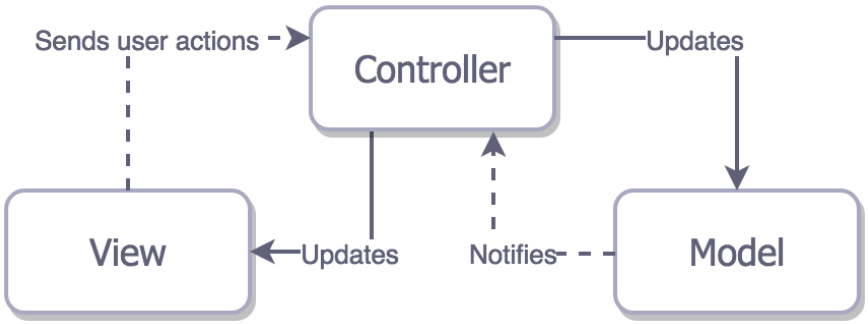
1. každú hodinu sťahovať aktuálne dáta meracích staníc ČHMÚ,
2. každých 24 hodín sťahovať otvorené dáta Inštitútu plánovania a rozvoja hlavného mesta Prahy o znečistení ovzdušia,
3. získané dáta spracovať a uložiť do databázy,
4. spracované a uložené dáta poskytovať formou API.

4.4 Návrh architektúry aplikácie

V procese návrhu aplikácie je dôležité zvoliť vhodnú architektúru, ktorá urýchli vývoj a sprehľadní výsledný produkt. Štandardom, ktorý firma Apple doporučuje vo svojej dokumentácii je architektúra Model-View-Controller (MVC) [18]. V súčasnosti je však pri návrhu mobilných aplikácií využívaná aj architektúra Model-View-Viewmodel (MVVM).

4.4.1 Model-View-Controller

Architektúra MVC definuje tri hlavné komponenty: Model, View a Controller a vzájomnú komunikáciu medzi nimi (pozri Obr. 4.1). Komponent Model obsahuje jadro dát aplikácie a slúži na manipuláciu medzi nimi. View získava dáta z Modelu a prezentuje ich používateľovi vo forme používateľského rozhrania, zatiaľ čo Controller funguje ako mediátor medzi Modelom a View, ktorý získava a interpretuje vstup od používateľa do požiadaviek pre Model alebo View [19].



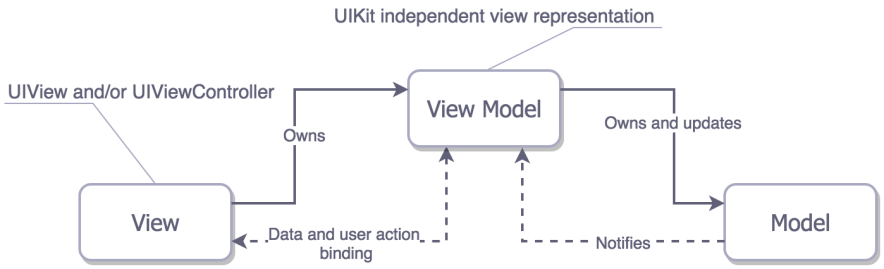
Obr. 4.1: Architektúra MVC [20]

Výhodou vzoru MVC je jasné oddelenie záujmov (z ang. separation of concerns). Každá vrstva architektúry MVC je zodpovedná za jasne definovaný aspekt aplikácie. MVC striktne oddeľuje Model od objektov používateľského rozhrania, a preto môže byť nad jedným Modelom vytvorených viacero komponentov View. Počas behu aplikácie tak môže byť súčasne otvorených viacero View, ktoré je možné tiež dynamicky otvárať a zatvárať. MVC taktiež umožňuje jednoduchú synchronizáciu jednotlivých View. Pri akejkoľvek zmene dát v Modeli sú všetky View, ktoré sú závislé na danom Modeli, okamžite informované, a používateľovi sú prezentované aktuálne dáta. Zmeny v Modeli taktiež neovplyvnia celú architektúru aplikácie, keďže Model nie je závislý na View. Vo väčšine aplikácií je tak jednoznačné, čo patrí do View a čo do Modelu. Pokiaľ je architektúra MVC implementovaná správne, komponenty Model a View sú navyše jednoducho znovupoužiteľné.

Nevýhodou architektúry MVC je však neustále sa zvyšujúca komplexnosť počas rastu aplikácie. Controller je navyše zodpovedný za veľkú časť funkcionality, pretože je využívaný ako priestor pre implementáciu všetkého, čo jednoznačne nepatrí ani do View, ani do Modelu. Výsledkom sú masívne a nečitateľné komponenty Controller a aplikácia, ktorú je náročné otestovať najmä kvôli skutočnosti, že väčšina logiky je implemetovaná v komponente Controller, čo z neho robí jediným možným miestom na vykonávanie testovania, a ktorého stav sa počas životného cyklu aplikácie môže meniť.

4.4.2 Model-View-Viewmodel

Architektúru MVVM možno považovať za akési vylepšenie MVC, ktoré zachováva hlavné komponenty MVC, no doménová logika je delegovaná na novú vrstvu nazývanú Viewmodel [21]. Ako možno vidieť na Obr. 4.2, celá komunikácia medzi View a Modelom je sprostredkovaná pomocou nového komponentu Viewmodel, čo prináša množstvo výhod najmä pri testovaní a rozširovaní aplikácie.



Obr. 4.2: Architektúra MVVM [20]

V prvom rade, Viewmodel tvorí jednosmernú komunikáciu, ktorá sa viaže na príslušný View. Výsledkom je, že View má referenciu na svoj Viewmodel, no Viewmodel nevie o danom View, čo znamená, že môže byť využívaný dalšími inými View. Viewmodel je navyše schopný pozorovať zmeny v Modeli a prispôsobovať sa na základe týchto zmien. Keďže existuje väzba medzi View a Viewmodelom, View je okamžite aktualizovaný.

Testovanie komplexných MVC aplikácií je často náročné. V architektúre MVVM je však Viewmodel nezávislý od používateľského rozhrania. Referencia na Controller je eliminovaná a Model zostáva izolovaný. Vykonávanie testov nad Viewmodelom je teda oveľa jednoduchšie a praktickejšie.

4.4.3 Výber architektúry

Na základe uvedených skutočností bola pri návrhu aplikácie zvolená architektúra MVVM. Napriek tomu, že obe architektúry sú podobné, a spoločnosť Apple odporúča využívanie MVC, MVVM prináša vďaka komponentu Viewmodel nesporné výhody počas celého procesu implementácie, ako aj pri následnom raste a zjednodušuje testovanie aplikácie.

4.5 Návrh používateľského rozhrania

Pre každú mobilnú aplikáciu je dôležitým prvkom používateľské rozhranie. Je preto potrebné zabezpečiť, aby bolo pre používateľa zrozumiteľné, intuitívne a prehľadné.

4.5.1 Wireframes

Pri návrhu používateľského rozhrania bola využitá metóda wireframe prototypovania. Wireframe definuje textový aj grafický obsah, základnú predstavu o rozmiestnení funkčných prvkov, ale aj navigáciu v rámci aplikácie. Je tvorený len pomocou čiar, textu a veľmi jednoduchých grafických prvkov. Môže byť vytváraný rôznymi spôsobmi, od klasického náčrtu ceruzkou na papier až po komplexnejšie návrhy použitím grafického softvéru. Jeho rýchla produkcia umožňuje v rámci návrhu experimentovanie s mnohými rôznymi vizualizáciami v ranom štádiu aplikácie bez potreby investovania veľkého množstva času, nákladov a úsilia. Aj napriek tomu, že wireframe má krátku životnosť, patrí medzi najvplyvnejšie a najvýznamnejšie metódy prototypovania, pretože z nich vychádzajú tie najinovatívnejšie nápady za nízke náklady [22].

Navrhnuté wireframes vychádzali z funkčných požiadaviek definovaných v kapitole 4.1 a boli vytvárané v súlade s Human Interface Guidelines [23]. Jedná sa o oficiálnu príručku spoločnosti Apple, ktorá popisuje optimálny návrh používateľského rozhrania iOS aplikácií. Na tvorbu wireframes bol použitý webový nástroj Balsamiq Cloud[20].

4.5.1.1 Navigácia

Základnými prvkami navigácie v rámci navrhovanej mobilnej aplikácie sú navigačná lišta UINavigationController [24], ktorá sa nachádza v hornej časti obrazovky a používateľovi zobrazuje názov aktuálnej obrazovky, a panel so záložkami UITabBarController [25] v dolnej časti obrazovky.

UITabBarController slúži ako hlavný prvok na navigáciu medzi hlavnými obrazovkami aplikácie. Každá z jeho záložiek obsahuje vlastný titulok a ikonu na intuitívnu navigáciu medzi nimi. Obsahuje 3 hlavné obrazovky:

1. úvodná obrazovka zobrazujúca aktuálnu kvalitu ovzdušia na základe aktuálnej polohy používateľa,
2. obrazovka obsahujúca mapový podklad znázorňujúci kvalitu ovzdušia a polohu meracích staníc ČHMÚ v rámci mesta Praha,
3. zoznam meracích staníc ČHMÚ.

4.5.1.2 Hlavné obrazovky

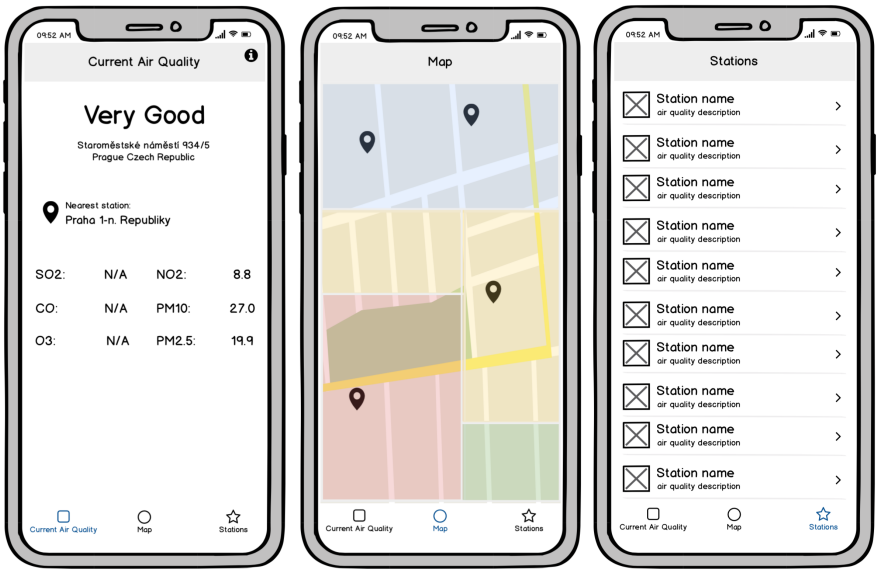
Návrh hlavných obrazoviek mobilnej aplikácie je znázornený na Obr. 4.3. Úvodná obrazovka je používateľovi zobrazená ihneď po spustení aplikácie a obsahuje informácie o aktuálnej kvalite ovzdušia na základe jeho polohy. Používateľa taktiež informuje o aktuálnych hodnotách koncentrácií znečisťujúcich látok v ovzduší nameraných na najbližšej meracej stanici ČHMÚ. Na

[20] `https://balsamiq.cloud/`

hornej navigačnej lište obsahuje tlačidlo, po ktorého stlačení sú používateľovi prezentované základné informácie o aplikácii.

Druhá obrazovka nad mapovým podkladom zobrazuje aktuálnu polohu používateľa a farebne odlíšené vrstvy, ktoré indikujú kvalitu ovzdušia v okolí používateľa, ako aj na celom území Prahy. V rámci Prahy taktiež zobrazuje polohu jednotlivých meracích staníc ČHMÚ.

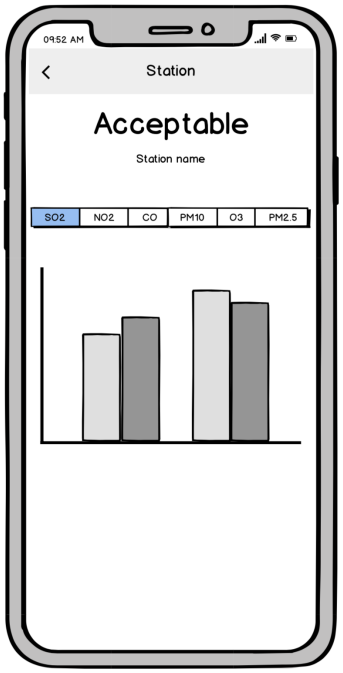
Tretia obrazovka obsahuje zoznam všetkých meracích staníc ČHMÚ, ktoré sa nachádzajú na území Prahy. Okrem ich názvu zobrazuje pre každú stanicu aj index kvality ovzdušia a jednoduchý popis kvality ovzdušia. Na každý prvok v zozname je možné kliknúť, čím sa zobrazí obrazovka obsahujúca detail stanice.



Obr. 4.3: Návrh hlavných obrazoviek

4.5.1.3 Detail stanice

Obrazovka detailu stanice, ktorej návrh je znázornený na Obr. 4.4, používateľa informuje o aktuálnej hodnote indexu kvality ovzdušia zvolenej stanice, rovnako ako o aktuálnych hodnotách koncentrácií znečisťujúcich látok v ovzduší. Navyše obsahuje jednoduchý stĺpcový graf, ktorý znázorňuje historický vývoj jednotlivých znečisťujúcich látok za posledných 24 hodín. Medzi jednotlivými grafmi je možné prepínať pomocou segmentovaného tlačidla obsahujúceho názvy všetkých znečisťujúcich látok meraních stanicami ČHMÚ. Pre navigáciu naspäť do zoznamu všetkých staníc slúži tlačidlo v ľavej hornej časti obrazovky.



Obr. 4.4: Návrh obrazovky detailu stanice

4.5.1.4 Informácie o aplikácii

Používateľovi je dôležité v rámci navrhovanej aplikácie poskytnúť informácie, ako s aplikáciou pracovať a ako vznikali dáta, ktoré aplikácia poskytuje. Na tento účel slúži tlačidlo s ikonou informácií na hornej navigačnej lište úvodnej obrazovky, po ktorého kliknutí sa objaví vyskakovacie okno obsahujúce základné informácie o aplikácii, zdrojoch dát a spôsobe merania a určovania indexu kvality ovzdušia využívaného v rámci jednotlivých obrazoviek. Pre návrat na domovskú obrazovku slúži tlačidlo v pravej hornej časti vyskakovaceiho okna. Návrh tohto vyskakovacieho okna je zobrazený na Obr. 4.5.

4.5.1.5 Apple Watch

Jednou z požiadaviek bola dostupnosť aplikácie pre chytré hodinky Apple Watch. Pre zachovanie hlavného účelu rýchleho poskytovania najpodstatnejších informácií je používateľské rozhranie tvorené 1 obrazovkou, ktorá zobrazuje aktuálnu hodnotu indexu kvality ovzdušia na základe polohy používateľa a názov najbližšej meracej stanice ČHMÚ. Realizácia ostatných obrazoviek, ktoré boli navrhnuté v rámci mobilnej aplikácie, by bola pre používateľa nepraktická najmä z dôvodu malej veľkosti obrazoviek zariadení Apple Watch (38 - 44 mm) [26]. Navrhnutá obrazovka je zobrazená na Obr. 4.6.

Obr. 4.5: Návrh obrazovky informácií

Obr. 4.6: Návrh hlavnej obrazovky aplikácie pre Apple Watch

4.5.2 Overenie návrhu používateľského rozhrania

Celý proces návrhu používateľského rozhrania bol realizovaný iteratívne s orientáciou na používateľa, v dvoch verziách prototypu a overený formou používateľského testovania. Obe verzie prototypu boli vytvorené a používateľ-

om zdieľané prostredníctvom online prototypovacieho nástroja InVisionapp[21], ktorý umožňuje vytvárať klikateľné verzie wireframe návrhu.

Používateľského testovania sa zúčastnilo osemnásť používateľov v oboch iteráciách, pričom každý z používateľov bol v čase vykonávania obyvateľom mesta Praha. Na testovanie bola použitá metóda priameho testovania použiteľnosti (z ang. usability testing) [27].

Nedostatkom prvého prototypu, ktorý bol odhalený v rámci prvej iterácie testovania, bola absencia obrazovky poskytujúcej základné informácie o aplikácii a spôsobe určovania indexu kvality ovzdušia. Tento nedostatok bol však v druhom prototype odstránený vytvorením vyskakovacieho okna, ktorého návrh je znázornený na Obr. 4.5. Používatelia sa taktiež sťažovali na spôsob zobrazovania grafov na obrazovke detailu stanice, keďže prvý prototyp obsahoval samostatný graf pre každú zo znečisťujúcich látok, pričom na obrazovke boli zobrazené všetky zároveň. Tento spôsob zobrazovania pôsobil na používateľov neprehľadným dojmom. V druhom prototype bol tento spôsob zobrazovania nahradený segmentovaným tlačidlom, pomocou ktorého si používateľ môže sám vybrať konkrétnu znečisťujúcu látku, o ktorej graf historického vývoja má záujem.

Vykonanie druhej iterácie testovania nepreukázalo žiadne ďalšie problémy či nedostatky. Pre implementáciu používateľského rozhrania aplikácie tak bol využitý druhý prototyp, ktorého návrh je popísaný v kapitole 4.5.1.

4.6 Návrh dátových štruktúr serveru

Návrh jednotlivých dátových štruktúr, ktoré bude server spracovávať a poskytovať mobilnej aplikácii, priamo vychádza zo štruktúry dát meracích staníc ČHMÚ a otvorených dát IPR hlavného mesta Prahy.

4.6.1 Feature

Otvorené dáta IPR Prahy obsahujú jednotlivé zóny znečistenia ovzdušia, ktoré budú v rámci serveru reprezentované objektom Feature, ktorý obsahuje:

- univerzálne jedinečný identifikátor,
- hodnotu klimatologickej charakteristiky kvality ovzdušia,
- dĺžku zóny,
- veľkosť zóny,
- tvar zóny,
- súradnice, ktoré daná zóna obklopuje.

[21] `https://www.invisionapp.com/`

4.6.2 Station

Station reprezentuje objekt meracej stanice tvorený dátami staníc ČHMÚ, ktorý bude obsahovať:

- jedinečný identifikátor (kód stanice),
- názov stanice,
- index kvality ovzdušia,
- zemepisnú šírku a dĺžku polohy stanice,
- hodnoty jednotlivých znečisťujúcich látok.

4.6.3 History

Objekt histórie stanice bude mať takmer rovnakú štruktúru ako objekt Station s tým rozdielom, že nebude obsahovať zemepisnú šírku a dĺžku svojej polohy, ktorá je v tomto prípade nepotrebná. Podstatným parametrom histórie stanice však bude časový údaj o vzniku daného historického údaju.

Kapitola 5

Popis implementácie

Kapitola približuje celý proces implementácie mobilnej aplikácie pre operačný systém iOS a API serverovej časti na základe vykonaného návrhu a popisuje použité technológie a knižnice.

5.1 Serverová časť

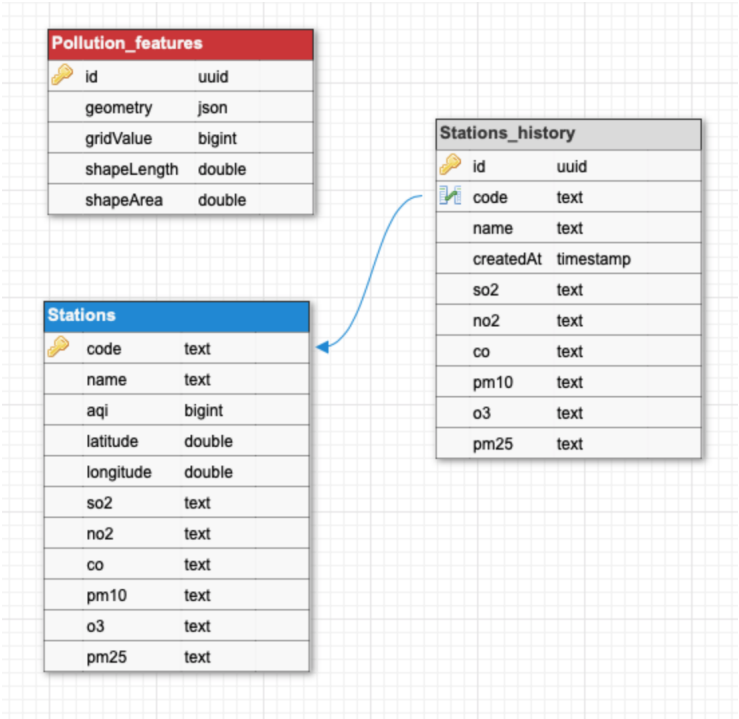
Na základe vykonanej analýzy bol na implementáciu serverovej časti zvolený jazyk Swift a framework Vapor 3. Ako vyplýva z definovaných funkčných požiadaviek serveru v kapitole 4.3, je potrebné, aby server dáta pravidelne sťahoval, ukladal do databázy a následne ich poskytoval aplikácii formou REST API.

5.1.1 Databáza

Aby bol server schopný neustále poskytovať ako aktuálne, tak aj historické dáta, je potrebné, aby boli tieto dáta ukladané v databáze. Vapor podporuje viacero databázových jazkov, oficiálna dokumentácia však odporúča využívanie jazyka PostgreSQL [28].

PostgreSQL je plnohodnotným objektovo-relačným databázovým systémom. Je známy svojou bohatou sadou poskytovaných funkcií a stabilnými verziami [29]. V rámci frameworku Vapor je navyše jednoduché pracovať s databázami PostgreSQL vďaka vlastnému systému na objektovo-relačné mapovanie Fluent, ktorý poskytuje širokú funkcionalitu a uľahčuje prácu pri ukladaní dát do databázy a získavaní uložených dát.

Implementácia databázy a jednotlivých dátových štruktúr vychádza z návrhu v kapitole 4.6. Na základe využívania týchto dátových štruktúr v aplikácii boli stanovené dátové typy jednotlivých záznamov a model implementovanej databázy je znázornený na Obr. 5.1.



Obr. 5.1: Model databázy

5.1.2 API

Representational state transfer (REST) predstavuje dátovo orientovanú architektúru rozhraní, ktorá je navrhnutá pre distrubuované prostredie. REST navrhol vo svojej dizertačnej práci [30] v roku 2000 Roy Fielding, jeden zo spolautorov protokolu HTTP. Implementuje štyri základné metódy, ktoré sú známé pod označením CRUD, teda vytvorenie dát (create), získanie požadovaných dát (read), zmena dát (update) a vymazanie dát (delete). Tieto metódy sú v rámci REST implementované pomocou zodpovedajúcich metód HTTP protokolu:

- POST - vytvorenie dát (create),
- GET - získanie dát (read),
- PUT - zmena dát (update),
- DELETE - vymazanie dát (delete).

Rozhranie REST je v rámci serverového API využité najmä pre vytvorenie jednoduchého a jednotného prístupu mobilnej aplikácie k dátam. REST explicitne nedefinuje formát reprezentácie dát, v prípade implementovanej aplikácie

bol zvolený formát JSON [31], ktorý je pre mobilné aplikáce považovaný za štandard, a v ktorom zároveň poskytujú dáta aj obe zdroje pre implementovanú aplikáciu. Implementované REST API obsahuje 5 hlavných koncových bodov. Každý z nich má jednoznačne definovanú cestu a metódu, ktorá sa v rámci koncového bodu využíva:

- GET pollution
- GET stations
- GET stations/{station_code}
- GET stations/{station_code}/history
- POST stations/nearest

5.1.2.1 GET pollution

Tento koncový bod slúži na získanie dát o znečistení ovzdušia, ktoré aplikácia využíva na vizualizáciu na mapovom podklade. Štruktúra jednotlivých záznamov zodpovedá návrhu dátovej štruktúry Feature v kapitole 4.6.1 a príklad odpovede servera je uvedený na ukážke 5.1.

```
{
  "id":"00198710-0D0D-42DD-9DC2-33C5C77BAA79",
  "gridValue":2,
  "shapeLength":0.126543741752413,
  "shapeArea":0.000540778513635,
  "geometry": {
    "type":"Polygon",
    "coordinates": [
      [
        [
          14.4809458980001,
          50.0295938880001
        ],
        [
          14.480963633,
          50.0295103730001
        ]
      ]
    ]
  }
}
```

Ukážka 5.1: Odpoveď koncového bodu pollution

5.1.2.2 GET stations

Koncový bod stations slúži na získanie aktuálnych dát všetkých meracích staníc ČHMÚ na území Prahy uložených na serveri. Aplikácia tento koncový bod využíva pri zobrazovaní zoznamu všetkých meracích staníc a pri ich vykresľovaní na mape. Štruktúra jednotlivých staníc zodpovedá návrhu dátovej štruktúry Station v kapitole 4.6.2. Odpoveď servera obsahuje hodnoty iba tých znečisťujúcich látok, ktoré boli v rámci aktuálneho merania na jednotlivých staniciach dostupné, ako možno vidieť na ukážke 5.2.

```
[
  {
    "code":"AREPA",
    "name":"Praha 1-n. Republiky",
    "aqi":3,
    "latitude":50.088066,
    "longitude":14.42922,
    "pm10":"52.0",
    "pm25":"33.4",
    "no2":"47.4"
  },
  {
    "code":"ARIEA",
    "name":"Praha 2-Riegrovy sady",
    "aqi":3,
    "latitude":50.081482,
    "longitude":14.442692,
    "pm10":"58.0",
    "pm25":"27.2",
    "no2":"42.7",
    "o3":"58.5",
    "so2":"3.7"
  }
]
```

Ukážka 5.2: Odpoveď koncového bodu stations

5.1.2.3 GET stations/{station_code}

Tento koncový bod slúži na získanie aktuálnych dát konkrétnej meracej stanice ČHMÚ, ktorá je definovaná svojím jedinečným kódom. V rámci aplikácie je tento koncový bod využívaný v prípade zobrazovania detailu konkrétnej stanice. Odpoveď má rovnakú formu ako odpoveď koncového bodu stations

s tým rozdielom, že neobsahuje pole všetkých meracích staníc, iba konkrétnu stanicu definovavú svojím jedinečným kódom.

5.1.2.4 GET stations/{station_code}/history

Koncový bod slúži na získanie historických dát konkrétnej meracej stanice ČHMÚ, ktorá je opäť definovaná svojím jedinečným kódom. Koncový bod je využívaný pri vykresľovaní grafu historického vývoja jednotlivých znečisťujúcich látok na obrazovke detailu stanice. Keďže dáta meracích staníc sú aktualizované každú hodinu, odpoveď servera obsahuje všetky tieto merania za posledných 24 hodín. Štruktúra jednotlivých historických záznamov zodpovedá návrhu dátovej štruktúry History v kapitole 4.6.3 a odpoveď koncového bodu je uvedená na ukážke 5.3.

```
[
  {
    "id":"6AD5B38F-771A-40D4-B384-3D7B11D26A9F",
    "code":"AREPA",
    "name":"Praha 1-n. Republiky",
    "createdAt":"2019-04-17T07:00:00Z",
    "pm10":"46.0",
    "pm25":"33.3",
    "no2":"71.5"
  },
  {
    "id":"E59BAC47-F437-4FB9-A098-E86FD96CFA8A",
    "code":"AREPA",
    "name":"Praha 1-n. Republiky",
    "createdAt":"2019-04-17T08:00:00Z",
    "pm10":"49.0",
    "pm25":"33.1",
    "no2":"35.0"
  }
]
```

Ukážka 5.3: Odpoveď koncového bodu stations/{station_code}/history

5.1.2.5 POST stations/nearest

Na rozdiel od ostatných koncových bodov, ktoré slúžia na získavanie dát pomocou HTTP metódy GET, posledný koncový bod implementovaného REST API využíva metódu POST. Aplikácia odošle na tento koncový bod aktuálnu polohu používateľa vo formáte, ako je uvedený na ukážke 5.4. Odpoveďou sú dáta

o najbližšej meracej stanici ČHMÚ vzhľadom k odoslanej polohe. Odpoveď má rovnaký formát ako odpoveď koncového bodu GET /stations/{station_code}.

```
{
  "latitude": 50.08804,
  "longitude": 14.42076
}
```

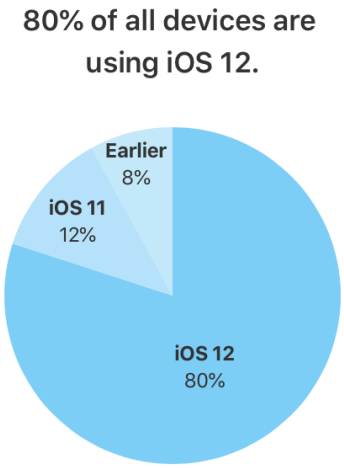
Ukážka 5.4: Formát dát odosielaných na koncový bod stations/nearest

5.2 Mobilná aplikácia

Mobilná aplikácia bola implementovaná pre zariadenia s operačným systémom iOS s ohľadom na návrh funkčných a nefunkčných požiadaviek v architektúre MVVM. Implementácia prebiehala v jazyku Swift 5 a vznikala vo vývojom prostredí Xcode, ktoré je zároveň jediným prostredím, v ktorom je možné natívne vyvíjať iOS aplikácie [32].

5.2.1 Podpora verzií iOS

Pri výbere podporovaných verzií operačného systému iOS boli do úvahy brané oficiálne štatistiky spoločnosti Apple [33] vykonávané 24. februára 2019, ktoré sú uvedené na obrázku 5.2. Zvolená bola podpora verzie 11 a vyššie, čo pokrýva viac ako 90% všetkých zariadení s operačným systémom iOS, pričom toto číslo má vďaka neustálemu a rýchlemu vývoju celej platformy tendenciu rásť.



Obr. 5.2: Percentuálne využívanie jednotlivých verzií iOS [33]

5.2.2 Architektúra aplikácie

Aplikácia bola na základe návrhu v kapitole 4.4 implementovaná v architektúre Model-View-Viewmodel (MVVM). Spôsob implementácie MVVM architektúry je popísany v tejto podkapitole.

5.2.2.1 Model

Jednotlivé modely (Feature, Station, History) zodpovedajú štruktúre dát, ktoré sú prijímané zo servera vo formáte JSON. Z tohto formátu sú prijaté dáta deserializované na objekty modelov vďaka natívnemu typu Codable [34].

Codable bol po prvýkrát predstavený s príchodom Swift 4. Slúži na uľahčenie serializácie a deserializácie dát formátu JSON v rámci jazyku Swift [35]. Stačí, ak daný model dedí po Codable a jeho premenné zodpovedajú štruktúre prijatých JSON dát, ktoré je potrebné deserializovať. Príkladom takéhoto modelu v implementovanej aplikácii je model Station reprezentujúci meraciu stanicu ČHMÚ, znázornený na ukážke 5.5.

```
1  struct Station: Codable {
2
3    //kód stanice
4    var code: String
5
6    //názov stanice
7    var name: String
8
9    //index kvality ovzdušia
10   var aqi: Int
11
12   //zemepisná šírka polohy stanice
13   var latitude: Double
14
15   //zemepisná dĺžka polohy stanice
16   var longitude: Double
17
18   //hodnoty koncentrácií jednotlivých znečisťujúcich látok
19   var so2: String?
20   var no2: String?
21   var co: String?
22   var pm10: String?
23   var o3: String?
24   var pm25: String?
25 }
```

Ukážka 5.5: Model Station

Po prijatí dát zo servera je následne možné ich jednoducho deserializovať na objekt modelu využítím objektu JSONDecoder [36], čím je získaný objekt zodpovedajúci definovanému modelu z prijatých JSON dát, s ktorým je možné ďalej pracovať a jeho premenné využívať v rámci aplikácie. Týmto spôsobom sú implementované všetky modely a deserializácia JSON dát prijatých zo servera v rámci mobilnej aplikácie.

5.2.2.2 View

Každý View predstavuje samostanú obrazovku aplikácie. Ako základ všetkých obrazoviek je implementovaná trieda BaseViewController, ktorá slúži na jednotné definovanie grafických prvkov, ktoré sú rovnaké pre celú aplikáciu, ako je napríklad farba pozadia či textu. Vďaka tejto triede je taktiež možné nastavovať viditeľnosť navigačnej lišty UINavigationController pre obrazovky, ktoré jej viditeľnosť nevyžadujú. V neposlednom rade slúži BaseViewController na vypisovanie všetkých alokovaných a dealokovaných obrazoviek do konzoly, čo je veľmi užitočné pri detekovaní pamäťových únikov (z ang. memory leaks).

5.2.2.3 Viewmodel

Rovnako ako každý View, aj každý Viewmodel má definovanú svoju základnú triedu BaseViewModel. Keďže jednotlivé Viewmodely nezdieľajú takmer žiadnu spoločnú logiku, táto trieda slúži najmä na vypisovanie ich alokácie a dealokácie do konzoly v procese vývoja. Každý View v rámci aplikácie má implementovaný svoj vlastný Viewmodel, ktorý slúži na umiestnenie logiky manipulácie s dátami, ktoré daný View zobrazuje.

5.2.3 Použité technológie a závislosti

Na manažment externých závislostí v rámci iOS aplikácií existujú dva manažéry - Cocoapods[22] a Carthage[23]. Jedným z kľúčových rozdielov medzi CocoaPods a Carthage je, že Carthage automaticky stiahne a vybuduje závislosti iba raz, bez potreby ďalšej indexácie alebo kompilovania, čo výrazne urýchľuje budovanie projektu počas jeho vývoja [37]. CocoaPods naopak vykonáva kompiláciu jednotlivých závislostí pri každom budovaní projektu. Počas implementácie aplikácie bol na základe uvedených dôvodov ako primárny manažér využívaný Carthage, no niektoré zo závislostí boli dostupné iba pre Cocoapods, čo vytvorilo potrebu využívat oba manažéry zároveň.

5.2.3.1 SnapKit

Základným funkčným prvkom pre vytváranie používateľského rozhrania iOS aplikácii je Storyboard. Storyboard je vizuálny nástroj na vytváranie jed-

[22] `https://cocoapods.org`
[23] `https://github.com/Carthage/Carthage`

notlivých obrazoviek aplikácie a definovanie prechodov medzi nimi. Je to v podstate plátno, ktoré je možné naplniť statickými grafickými prvkami a mať tak prehľad o rozhraní celej aplikácie [38]. Výhodou Storyboardu je jeho priamočiarosť - každý, kto sa na neho pozrie, vidí ako jednotlivé obrazovky vyzerajú a aké sú vzťahy medzi nimi. S rastom aplikácie a počtom obrazoviek sa však Storyboard stáva neprehľadným, no hlavne veľmi pomalým kvôli množstvu grafických prvkov, ktoré sa v ňom nachádzajú. Tieto prvky majú taktiež veľmi obmedzenú znovupoužiteľnosť, pri využívaní rovnakých prvkov na viacerých obrazovkách je potrebné ich znovu vytvárať nanovo, čo nie je práve ideálny prístup. Problémy nastávajú aj pri práci viacerých vývojárov, ktorý využívajú niektorý zo systémov pre správu verzií (ako napr. Git [39]), kedy je náročné zlúčiť (z ang. merge) viacero rôznych verzií Storyboardu.

Alternatívou voči Storyboardu je programovanie používateľského rozhrania, ktoré so sebou prináša množstvo výhod. Prvou výhodou je výkon. Storyboard musí byť pri každom budovaní projektu preložený do kódu, zatiaľ čo pri budovaní projektu s naprogramovaným rozhraním tento krok potrebný nie je. Vývojár taktiež získava oveľa väčšiu kontrolu nad rozložením grafických prvkov a dokáže ich jednoznačne prispôsobiť svojím predstavám. Ďalšou výhodou je znovupoužiteľnosť. Ak aplikácia používa prvky, ktoré vyzerajú v rámci viacerých obrazoviek rovnako, je môžné vytvoriť ich na jednom mieste a potom ich využívať v rámci celej aplikácie, napríklad textové polia, tlačidlá a podobne. Zlučovanie pri práci so systémom pre správu verzií je jednoduché, keďže celé používateľské rozhranie sa nachádza v kóde.

Za účelom programovania používateľského rozhrania aplikácie bola použitá knižnica SnapKit[24]. Tá umožňuje budovať grafické prvky a obmedzenia medzi nimi s minimálnym množstvom kódu a zároveň zabezpečiť, aby bol výsledný kód ľahko čitateľný a zrozumiteľný. Navyše je tento kód typovo bezpečný, čo zabraňuje vývojárovi vytvárať neplatné obmedzenia pre maximalizáciu produktivity. Kód celej hierarchie používateľského rozhrania jednotlivých obrazoviek sa nachádza v metóde `loadView()` [40], ktorá je volaná v rámci životného cyklu obrazovky pri jej vytváraní.

5.2.3.2 ReactiveSwift

Tradičým prístupom pri programovaní mobilných aplikácií je imperatívne programovanie. To spočíva v tvorbe kódu, ktorý jasne opisuje, akým spôsobom sa daný problém rieši. Každý riadok takéhoto kódu je vykonávaný sekvenčne za cieľom dosiahnutia požadovaného výsledku [41].

Veľmi populárnym trendom na mobilných platformách je však v súčasnosti funkcionálne reaktívne programovanie (FRP), ktoré je kombináciou funkcinonálneho a reaktívneho programovania. Je založené na asynchrónnom programování a práci s prúdmi dát a udalostí. Prúdom udalostí môže byť napríklad

[24] `https://snapkit.io`

vstup od používateľa, stlačenie tlačidla, gestá či aktualizácia polohy. Aplikácia je schopná sledovať tento prúd a adekvátne na neho reagovať. Funkcionálne reaktívne programovanie je paradigma, ktorá mení spôsob premýšľania nad aplikáciou a celú jej architektúru [42].

So zvyšujúcou sa popularitou FRP množstvo programovacích jazykov začalo vytvárať svoje vlastné frameworky, ktoré by umožnili vývojárom programovať reaktívne. Výnimkou nie je ani jazyk Swift. Prvým projektom bol ReactiveCocoa[25], ktorý vytvorili dvaja zamestanci GitHubu Josh Abernathy a Justin Spahr-Summers. Keďže projekt bol realizovaný v roku 2012, a jazyk Swift vtedy ešte neexistoval, framework bol založený na jazyku Objective-C, ktorý sa v tom čase využíval na vývoj mobilných aplikácií pre iOS. Po tom, čo jazyk Swift získal na popularite, bola vytvorená nová verzia ReactiveSwift[26], ktorá sa používa dodnes. ReactiveSwift ponúka skladateľné, deklaratívne a flexibilné objekty, ktoré sú postavené na prúdoch dát a udalostí v čase. Tieto objekty môžu byť použité na jednotnú reprezentáciu generických návrhových vzorov, ktoré sú založené na pozorovaní, ako napríklad delegáti, notifikácie alebo pozorovanie kľúčových hodnôt (z ang. key-value observation) [43].

Alternatívou k ReactiveSwift je RxSwift[27]. Hlavným rozdielom medzi nimi je, že RxSwift vznikol ako jedna z verzií knižnice Microsoft Reactive Extenstions pre programovací jazyk Swift. Je taktiež mladší ako ReactiveSwift, no postupne naberá na popularite. Oba frameworky sú vo viacerých ohľadoch podobné, no aplikácia bola implementovaná pomocou ReactiveSwift, ktorý je v súčasnosti populárnejší a z historického hľadiska viacej využívaný (cez 19 000 hviezd na portáli GitHub [44]).

Základom ReactiveSwift a implementovanej logiky mobilnej aplikácie sú tzv. väzby (z ang. bindings). Tie slúžia na naviazanie požadovanej funkcionality na konkrétny prúd dát či udalostí. Väzby sa vytvárajú pomocou špeciálneho operátora $<\sim$. Ľavá strana operátora predstavuje cieľ väzby, pravá strana zdroj. Jednoduchým príkladom z aplikácie, ktorý je uvedený na ukážke 5.6, je zmena textu zobrazujúceho aktuálnu adresu (addressLabel) stále, keď sa zmení hodnota premennej (address), ktorá obsahuje informácie o aktuálnej polohe používateľa.

```
1 addressLabel.reactive.text <~ viewModel.address
```

Ukážka 5.6: Príklad reaktívnej väzby

Všetky väzby sú implementované v metóde `setupBindings()` príslušného View, ktorá je následne volaná v metóde `viewDidLoad()` [45] životného cyklu aplikácie po načítaní View do pamäte.

[25] `https://github.com/ReactiveCocoa/ReactiveCocoa`
[26] `https://github.com/ReactiveCocoa/ReactiveSwift`
[27] `https://github.com/ReactiveX/RxSwift`

5.2.3.3 Dependency injection

Dependency injection alebo vkladanie závislostí je technika pre vkladanie závislostí medzi jednotlivými komponentmi programu, pri ktorej jeden objekt (alebo statická metóda) dodáva závislosti inému objektu [46]. Slovo dependency predstavuje v tomto kontexte závislosť, resp. službu. Injection je proces odovzdávania tejto závislosti objektu, ktorý je následne schopný ju použiť bez toho, aby zodpovedal za celý jej životný cyklus. Potrebuje iba referenciu na poskytovateľa závislostí, ktorý je schopný dodať viacero rôznych komponentov splňujúcich očakávania daného objektu. Každý z týchto komponentov môže objektu poskytovať rozdielne služby.

Pre uplatnenie dependency injection v iOS aplikáciách existuje framework Swinject[28]. Jeho využívanie je však počas procesu implementácie príliš zložité, keďže pri každom vkladaní novej závislosti je potrebné zdĺhavé prepisovanie použitých metód. Namiesto využívania Swinject bola implementovaná dependency injection založená na skladaní protokolov (z ang. protocol composition), ktorú vo svojom článku [47] popisuje Krzysztof Zabłocki. Skladanie protokolov dokáže zvýšiť čitateľnosť kódu a eliminovať viacnásobné prepisovanie použitých metód tým, že definuje generický protokol pre každú závislosť, ktorá je využívaná. Príklad takéhoto protokolu, ktorý implementovaná aplikácia využíva na komunikáciu s API je uvedený na ukážke 5.7.

```
1 protocol HasAppAPI {
2   var appAPI: AppAPIServicing { get }
3 }
```

Ukážka 5.7: Generický protokol HasAppAPI

Následne je aplikovanie závislosti v rámci Viewmodelu jednoduché. Swift navyše umožňuje spájať protokoly pomocou operátora &, čo znamená, že stačí deklarovať jednu premennú, ktorá bude mať definované všetky protokoly. Následne je možné pri inicializácii z tejto premennej získať potrebné závislosti. V rámci implementovanej aplikácie je využívaná premenná `Dependencies` spôsobom uvedeným na ukážke 5.8.

5.2.3.4 ScrollableGraphView

ScrollableGraphView[29] slúži na adaptívne zobrazovanie posuvných grafov. Hlavným cieľom tohto grafického komponentu je vizualizovať dátové súbory a umožniť používateľovi prechádzať celým grafom. Táto knižnica bola využitá na vizualizáciu historického vývoja jednotlivých znečisťujúcich látok na obrazovke detailu stanice formou grafu.

[28] `https://github.com/Swinject/Swinject`
[29] `https://github.com/philackm/ScrollableGraphView`

```
final class MapViewModel: BaseViewModel {
  typealias Dependencies = HasAppAPI & HasLocationProvider

  let appAPI: AppAPIServicing
  let locationProvider: LocationProviderServicing

  init(dependencies: Dependencies) {
    appAPI = dependencies.appAPI
    locationProvider = dependencies.locationProvider
  }
}
```

Ukážka 5.8: Aplikácia protokolovej dependency injection

5.2.3.5 Alamofire

Alamofire[30] je HTTP sieťová knižnica v jazyku Swift. Poskytuje elegantné rozhranie nad natívnou implementáciou sieťových funkcií pre iOS a macOS, ktoré zjednodušuje vykonávanie základných sieťových operácií. Alamofire umožňuje vykonávať HTTP požiadavky, serializovať a autentifikovať odpovede a mnoho ďalších funkcií.

Implementovaná aplikácia na získavanie dát zo serveru a odosielanie dát využíva práve knižnicu Alamofire a jej metódy. Tieto metódy sú implementované v triede `DataAPIServicing`, ktorá definuje celú komunikáciu mobilnej aplikácie s API.

5.2.3.6 SwiftGen

SwiftGen[31] je nástroj na automatické generovanie Swift kódu pre zdroje projektu, aby bolo ich využívanie typovo bezpečné. Táto knižnica bola implementovaná pre uľahčenie práce s obrázkami a prekladmi v rámci mobilnej aplikácie.

Lokalizácia do rôznych jazykov v rámci iOS je jednoduchá. Podporované jazyky stačí definovať vo vývojom prostredí Xcode, ktoré automaticky vytvorí súbory špecifické pre daný jazyk. Následne je úlohou vývojára, aby tieto súbory naplnil prekladmi jednotlivých názvov a textov, ktoré sa v aplikácii vyskytujú a SwiftGen z nich vygeneruje typovo bezpečné premenné, ktoré je možné využiť v kóde. Aplikácia automaticky rozozná primárny jazyk zariadenia a na základe toho zobrazuje používateľovi relevantný preklad. Implementovaná aplikácia je lokalizovaná do anglického, českého a slovenského jazyka. Podobným spôsobom je SwiftGen využívaný aj pri práci s obrázkami. Namiesto toho, aby bola cesta k obrázkom definovaná v kóde, SwiftGen auto-

[30] `https://github.com/Alamofire/Alamofire`
[31] `https://github.com/SwiftGen/SwiftGen`

maticky vygeneruje premenné z obrázkov, ktoré sa v nachádzajú v definovanej zložke projektu. Tieto premenné je následne možné využívať priamo v kóde.

5.2.3.7 Firebase

Firebase [32] je webová a mobilná platforma, ktorá bola v roku 2014 odkúpená spoločnosťou Google [48]. Jej portfólio pozostáva z množstva produktov, ako napríklad poskytovanie statického hostingu, vlastnej databázy či autorizačnej štruktúry pre prístup k databáze.

V rámci mobilnej aplikácie bola implementovaná ich služba Analytics, ktorá zdarma poskytuje informácie o počte aktívnych používateľov, demografické údaje a prehľad o používaní aplikácie a angažovanosti používateľov vo forme prehľadného webového rozhrania. Pomáha získať prehľad o veľkosti používateľskej základne a jednoznačne identifikovať správanie používateľov a ich interakciu s grafickým rozhraním, čo umožňuje v budúcnosti aplikáciu optimalizovať na základe ich potrieb

Implementovaná bola taktiež služba Crashlytics, ktorá poskytuje záznamy o pádoch aplikácie v reálnom čase. Služba zaznamenáva všetky dôležité informácie o páde - presný čas pádu, čo k nemu viedlo a na akom zariadení k pádu došlo. Tieto záznamy je možné využívať pri spravovaní produkčnej verzie aplikácie pre zachytenie a odstránenie chýb, ktoré neboli odhalené počas používateľského testovania.

5.2.4 Aktualizácia dát

Pre zachovanie výpovednej hodnoty dát, ktoré aplikácia poskytuje, je potrebné aby boli neustále aktuálne. Z tohto dôvodu aplikácia sťahuje všetky dáta z implementovaného servera a nevykonáva lokálnu perzistenciu dát, ktorá by mohla v prípade neúspešného stiahnutia aktuálnych dát zavádzať používateľa o aktuálnej kvalite ovzdušia v jeho okolí a bezdôvodne zaberať pamäť jeho zariadenia. Namiesto toho aplikácia informuje používateľa o prípadnom neúspešnom stiahnutí. Pri spustení aplikácie a každom pokuse o stiahnutie dát zo servera preto prebehne kontrola, a pokiaľ dáta nie sú k dispozícii, alebo používateľ nemá prístup k internetu, je na tento fakt upozornený s požiadavkou o vykonanie opätovného pokusu stiahnutia dát.

Napriek požiadavke na udržiavanie neustále aktuálnych dát je potrebné myslieť aj na úsporu dát. Sťahovanie aktuálnych dát pri každej malej zmene polohy používateľa, či každom prechode na novú obrazovku by bolo pre zariadenie používateľa náročné. Tvorilo by sa množstvo požiadaviek na server, ktoré v danom momente nie sú potrebné a bezdôvodne zvyšujú objem prenesených dát a spotrebu energie zariadenia. Z tohto dôvodu sú dáta o aktuálnej kvalite ovzdušia aktualizované pri každej zmene aktuálnej polohy používateľa aspoň o 100 metrov vzhľadom na predchádzajúcu polohu. Dáta jednotlivých

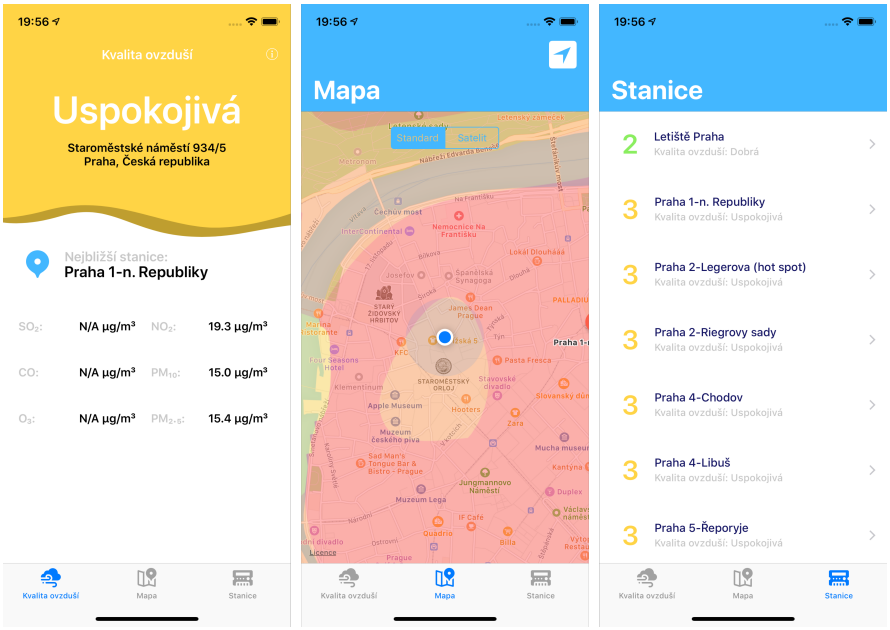
[32] `https://firebase.google.com`

meracích staníc ČHMÚ sú aktualizované pri spustení aplikácie, no používateľ má možnosť tieto dáta aktualizovať manuálne pomocou štandardného gesta - potiahnutím obrazovky nadol.

5.2.5 Grafické prevedenie

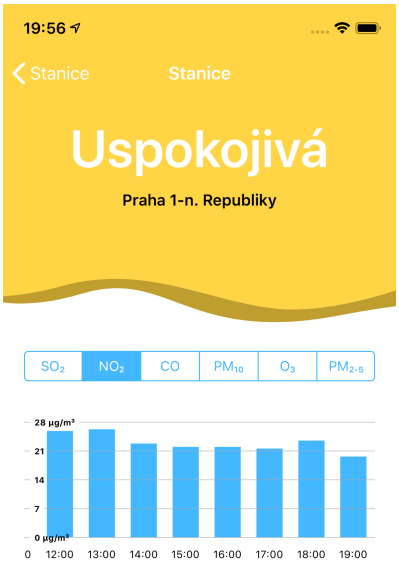
Grafika mobilnej aplikácie bola relizovaná vo vektorovom nástroji Adobe XD[33] a vychádza z návrhu používateľského rozhrania v kapitole 4.5. Snímky výslednej aplikácie sú znázornené na Obr. 5.3 a Obr. 5.4. Hlavným cieľom bolo zachovať prehľadnosť jednotlivých grafických komponentov a rešpektovať Human Interface Guidelines.

Hlavný indikátor je založený na sade 7 farieb a je základným prvkom pre zobrazovanie aktuálnej kvality ovzdušia. Mapový podklad je zobrazovaný vďaka natívnemu Apple frameworku MapKit [49] s možnosťou prepínania medzi štandardným a satelitným zobrazením. Mapový podklad prekrývajú vlastné definované vrstvy na základe kvality ovzdušia. Zoznam meracích staníc ČHMÚ je realizovaný formou komponentu UITableView [50], ktorý je súčasťou základného grafického balíka UIKit [51]. Detail stanice obsahuje taktiež hlavný indikátor a posuvný stĺpcový graf zobrazujúci historický vývoj jednolitvých znečisťujúcich látok, pričom voľba zobrazovaných látok je realizovaná prostredníctvom segmentovaného tlačidla UISegmentedControl [52].



Obr. 5.3: Hlavné obrazovky mobilnej aplikácie

[33] https://www.adobe.com/products/xd.html



Obr. 5.4: Obrazovka detailu stanice mobilnej aplikácie

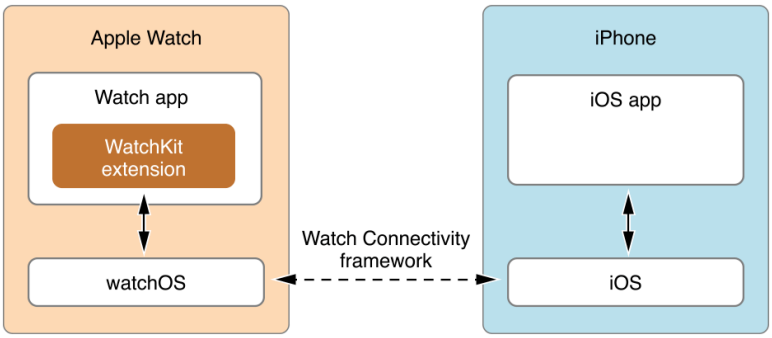
5.3 Apple Watch aplikácia

Chytré hodinky Apple Watch fungujú na vlastnej verzii operačného systému, ktorý sa nazýva watchOS [53]. Ten je založený na operačnom systéme iOS a zdieľa množstvo podobných funkcií. Bol vydaný 24. apríla 2015 spolu s prvým predstavením Apple Watch [54] a obsahuje vlastné API pod názvom WatchKit [55]. Súčasná verzia watchOS 5.2 bola vydaná 27. marca 2019.

Aplikácie watchOS sú vstavané do aplikácií pre systém iOS. Po stiahnutí a inštalácii iOS aplikácie do mobilného zariadenia, ktorá zároveň obsahuje aj svoju variantu pre watchOS, je táto aplikácia automaticky prenesená cez Bluetooth pripojenie na chytré hodinky Apple Watch. Ak hodinky momentálne nie sú v dosahu, aplikácia je nainštalovaná pri najbližšom úspešnom pripojení. Aplikácie watchOS sú nezávislé a ich funkcionalita je realizovaná priamo na hodinkách Apple Watch, od spracovávania dát až po správu pamäte. Pri získavaní dát, ktoré spracovávajú, sa však plne spoliehajú na mobilné zariadenie iPhone, ku ktorému sú pripojené.

5.3.1 Zdieľanie dát

Zdieľanie dát a obojstranná komunikácia medzi materskou aplikáciou a Apple Watch je implementovaná pomocou frameworku WatchConnectivity [56], ktorý realizuje celú komunikáciu medzi zariadeniami (pozri Obr. 5.5).



Obr. 5.5: Architektúra komunikácie medzi Apple Watch a iOS aplikáciou [56]

Na zahájenie komunikácie je potrebné ju inicializovať v oboch aplikáciách prostredníctvom triedy WCSession, pričom delegát musí byť priradený k triede WCSession a volať metódu `activate()`, aby bolo spojenie úspešné. Kód na inicializáciu takéhoto spojenia je uvedený na ukážke 5.9.

```
1 if WCSession.isSupported(){
2   let session = WCSession.default
3   session.delegate = self
4   session.activate()
5 }
```

Ukážka 5.9: Inicializácia spojenia medzi Apple Watch a iOS aplikáciou

Po úspešnom vytvorení spojenia môžu zariadenia využívať metódy na zdieľanie, odosielanie a prijímanie dát. Zvyčajne sú dáta medzi zariadeniami odosielané vo forme slovníka (z ang. dictionary) typu `[String: Any]`, kedy kľúč je definovaný reťazcom a typ odosielaných dát môže byť ľubovoľný. Výnimkou je prenos, ktorý zahŕňa URL cestu k súboru. Väčšina dát je odosielaných v pozadí, ale v prípade potreby je možné ich odoslať okamžite.

Implementovaná aplikácia odosiela dáta hodinkám pri každej zmene aktuálneho indexu kvality ovzdušia, resp. najbližšej stanice na základe aktuálnej polohy, a to aj v prípade, že je aplikácia v pozadí. Najprv si overí, či sú hodinky v dosahu prostredníctvom metódy `WCSession.default.isReachable` a následne dáta odošle metódou `WCSession.updateApplicationContext`, ktorá obsahuje odosielané dáta ako svoj parameter a slúži na synchronizáciu stavu medzi aplikáciou a hodinkami. Odosielané dáta obsahujú textový popis aktuálneho indexu kvality ovzdušia a názov najbližšej stanice.

Pre prijatie dát na strane watchOS aplikácie je potrebné implementovať metódu `didReceiveApplicationContext`. Po prijatí nových dát ich aplikácia ihneď zobrazí na hodinkách, a používateľa informuje jemným zavibrovaním hodiniek vďaka metóde `WKInterfaceDevice().play(.click)`.

5.3.2 Grafické prevedenie

Aplikácia pre chytré hodinky Apple Watch na základe návrhu v kapitole 4.5.1 obsahuje iba jednu hlavnú obrazovku, ktorá vizualizuje požadované informácie o kvalite ovzdušia spolu s jednoduchými grafickými prvkami (pozri Obr. 5.6).

Obr. 5.6: Hlavná obrazovka aplikácie pre Apple Watch

5.4 Dokumentácia

Neoddeliteľnou súčasťou implementácie je dokumentácia všetkých dôležitých častí. Dokumentácia serverovej časti obsahuje dokumentáciu zdrojového kódu vo formáte HTML, ktorá bola vytvorená prostredníctvom jazyka Markdown a vygenerovaná nástrojom Jazzy[34]. API je zdokumentované pomocou služby Apiary vo formáte API Blueprint, ktorý zrozumiteľným spôsobom popisuje celé existujúce API, obsahuje vzorové volania, povinné parametre a zodpovedajúce odpovede. Ten bol následne pre zvýšenie prehľadnosti prevedený do formátu HTML vďaka nástroju Aglio[35]. Zdrojový kód mobilnej aplikácie bol rovnako ako kód serverovej časti zdokumentovaný prostredníctvom nástroja Jazzy vo formáte HTML. Celá dokumentácia je súčasťou priloženého CD,

[34] `https://github.com/realm/jazzy`
[35] `https://github.com/danielgtaylor/aglio`

vrátane používateľskej príručky, ktorá slúži na oboznámenie sa s funkcionalitou implementovanej aplikácie, ako aj systémovej príručky, ktorá obsahuje popis štruktúry projektu a spôsob inštalácie mobilnej aplikácie a serverovej časti.

Kapitola 6

Testovanie a nasadenie

6.1 Testovanie

Táto podkapitola popisuje celý priebeh procesu testovania mobilnej aplikácie a serverovej časti, ktorého účelom bolo identifikovať problémy či prípadné nedostatky a eliminovať ich pred reálnym nasadením.

6.1.1 Mobilná aplikácia

Mobilná aplikácia bola testovaná priebežne počas vývoja prostredníctvom jednotkových (z ang. unit) testov. Jednotkové testy dokážu odhaliť prípadné implementačné chyby už počas vývoja aplikácie, čím môžu výraznou mierou ušetriť čas a predísť množstvu ďalších chýb [57]. Na realizáciu jednotkových testov bola použitá natívna knižnica XCTest [58]. Testami bola dôkladne pokrytá hlavná logika aplikácie implementovaná vo Viewmodeloch a služby, ktoré ju zabezpečujú.

Používateľské rozhranie bolo testované na všetkých zariadeniach podporujúcich iOS 11 a vyššie. Pre zariadenia iPhone sa jedná o modely 5S a novšie, pre zariadenia iPad modely mini 2 a novšie a všetky modely chytrých hodiniek Apple Watch. Testovanie prebiehalo prostredníctvom virtuálnej simulácie týchto zariedení, ktorú ponúka vývojové prostredie Xcode a cieľom bolo prispôsobiť používateľské rozhranie tak, aby sa korektne zobrazovalo na všetkých zariadeniach, keďže sa výrazne líšia vo veľkostiach obrazoviek.

Po implementácii aplikácie do stavu beta verzie bolo realizované používateľské testovanie na cieľovej skupine 12 používateľov, ktorí vlastnia aspoň jedno zariadenie s operačným systémom iOS a minimálne jeden rok žijú v meste Praha. Za týmto účelom bola využitá služba Apple TestFlight [59], pomocou ktorej je možné aplikáciu zverejniť na beta testovanie. Následne stačí, ak si každý z testovacích subjektov aplikáciu TestFlight stiahne do svojho zariadenia a prijme pozvánku, ktorú vývojár aplikácie odošle testovacím subjektom prostredníctvom mailu. Po prijatí pozvánky je aplikácia nainštalovaná do za-

riadenia a pripravená na vykonanie testovania. Testovacím subjektom boli zadané nasledujúce úlohy:

1. Zistite aktuálnu kvalitu ovzdušia na mieste, kde sa práve nachádzate.
2. Zistite, kde sa nachádza najbližšia meracia stanica ČHMÚ a aké sú jej akutálne namerané hodnoty koncentrácií znečisťujúcich látok.
3. Zistite kvalitu ovzdušia vo Vašom širšom okolí.
4. Zistite aktuálnu kvalitu ovzdušia na pražskom Karlíne a historický vývoj hodnôt koncentrácií znečisťujúcich látok, ktoré stanica meria.
5. Zistite, akým spôsobom je stanovený index kvality ovzdušia a v akom rozmedzí sa pohybuje.
6. Pokiaľ vlastníte hodinky Apple Watch, spustite na nich aplikáciu a zistite aktuálnu kvalitu ovzdušia.

Používateľské testovanie prebiehalo za prítomnosti autora tejto práce a počas jeho priebehu neboli odhalené žiadne kritické problémy alebo nedostatky. Každému z dvanástich testovacích subjektov sa úspešne podarilo splniť všetky zo zadaných úloh bez väčších problémov a potvrdiť tak funkčnosť a praktickú využiteľnosť aplikácie. Na záver testovania každý zo subjektov vyplnil SUS (z ang. System Usability Scale) dotazník, ktorý patrí medzi najznámejšie štandardizované hodnotiace merítka použiteľnosti systému a poskytuje dôležitú spätnú väzbu od používateľov [60]. Dotazník sa skladá z 10 výrokov ohľadom celkového využívania aplikácie, ktoré používatelia hodnotia na škále od 1 (s výrokom nesúhlasím) do 5 (s výrokom súhlasím). Jednotlivé hodnoty odpovedí testovacích subjektov boli zaznamenané a aplikácia dosiahla vysokú SUS hodnotu 92 z maximálnej možnej hodnoty 100, čo poukazuje na pozitívnu spätnú väzbu testovacích subjektov, veľmi dobrú použiteľnosť aplikácie a jej pripravenosť na reálne nasadenie.

6.1.2 Server

Implementovaný kód serverovej časti bol rovnako ako mobilná aplikácia testovaný prostredníctvom jednotkových testov počas vývoja, taktiež realizovaných natívnou knižnicou XCTest. Navrhnuté a implementované API bolo otestované pomocou programu Postman[36], pričom testované boli nasledujúce kritériá jednotlivých koncových bodov:

- stavové kódy a popis HTTP odpovedí (200 - OK pri úspešnej požiadavke, 400 - Bad Request pri neúspešnej),

[36] `https://www.getpostman.com`

- typ obsahu HTTP odpovedí (JSON),
- doba odozvy (do 200 milisekúnd),
- správna JSON schéma a dátové typy záznamov zodpovedajúcich HTTP odpovedí,
- počet záznamov, ktoré obsahujú jednotlivé HTTP odpovede (napr. počet staníc v rámci Prahy je 17, historické údaje sú dostupné za posledných 24 hodín),
- aktuálnosť časových známok záznamov (žiaden z historických údajov nie je starší ako 24 hodín),

Jednotlivé testy boli úspešne vykonané nad implementovaným API v 5 iteráciach. Záznam z testovania v HTML forme je dostupný na priloženom CD médiu.

6.2 Nasadenie

Táto podkapitola sa venuje spôsobou nasadenia serverovej časti a mobilnej aplikácie tak, aby bola dostupná všetkým používateľom, ktorý vlastnia zariadenie s operačným systémom iOS 11 a vyššie, prípadne zariadenie Apple Watch.

6.2.1 Server

Pri nasadení serverovej časti boli uvažované verejne dostupné serverové služby DigitalOcean[37], Heroku[38] a Vapor Cloud[39]. Kritériami boli čo najnižšie možné náklady na využívanie a spoľahlivosť pripojenia a komunikácie s mobilnou aplikáciou.

DigitalOcean je americký poskytovateľ serverových služieb, ktorý sa stal za rok 2018 treťou najväčšou svetovou hostingovou spoločnosťou [61]. Poskytuje spoľahlivé serverové riešenia na rôznych distribúciách Linuxu, ktoré je možné spravovať prostredníctvom webového rozhrania. Najlacnejšia alternatíva serverového riešenia s 1 GB RAM pamäte stojí 5 dolárov mesačne, pričom v cene je zarátaná aj prevádzka malej databázy.

Heroku je platforma založená v roku 2007, ktorá taktiež poskytuje serverové služby. V súčasnosti je na platforme Heroku nasadených viac ako tri milióny rôznych aplikácií v rámci ich infraštruktúry [62], čo svedčí o jej spoľahlivosti. Narozdiel od DigitalOcean má navyše vo svojich plánoch zahrnutý aj bezplatný plán s 512 MB RAM pamäte. Tento plán však prichádza

[37] `https://www.digitalocean.com`
[38] `https://www.heroku.com`
[39] `https://vapor.cloud`

s výraznými obmedzeniami. Ak je serverové riešenie nasadené na Heroku, je automaticky spustené v linuxovom kontajneri `dyno`. Každý takýto kontajner predstavuje spustenú inštanciu servera, ktorá je schopná prijímať požiadavky a odosielať odpovede, pričom jednotlivé kontajnery sú od seba izolované. Bezplatný plán ponúka jedno `dyno`, no počet hodín, počas ktorých môže byť využívané a počas ktorého beží reálne nasadený server, je obmedzený na 550. Pri prečerpaní týchto voľných hodín je následne potrebné zaplatiť 7 dolárov mesačne za jedno `dyno`. Server je navyše automatický uspávaný po 30 minútach bez aktivity, čo by v praxi mohlo viesť k situácii, kedy používateľ musí čakať, kým sa server znovu spustí z neaktívneho stavu, a až v tom momente obdrží aktuálne dáta. Tento proces pritom môže trvať aj dlhšie ako minútu. Možným riešením tohoto problému by bolo pravidelné odosielanie požiadaviek na server, aby sa vyhol stavu inaktivity, v tomto prípade by však bezplatný beh fungoval rádovo niekoľko dní, po uplynutí ktorých by bolo potrebné za server platiť 7 dolárov mesačne.

Treťou uvažovanou službou bol Vapor Cloud. Ten bol vytvorený špeciálne pre nasadzovanie serverových riešení implementovaných v jazyku Vapor tak, aby zjednodušil konfiguráciu serveru a celkovú správu nasadenia, čo bolo hlavným dôvodom, pre ktorý bol braný do úvahy. Podobne ako Heroku, aj Vapor Cloud poskytuje bezplatný plán. Jeho jediným obmedzením je maximálne 20 000 vykonaných požiadaviek na server mesačne, čo v prípade implementovanej aplikácie nespôsobuje žiaden problém. V prípade, ak by mesačný počet požiadaviek presahoval 20 000, je možné prejsť na platenú verziu, ktorá stojí 6 dolárov mesačne, prípadne ďalších 7 dolárov mesačne pre potreby prevádzky databázy s 1 GB pamäte. Počas doby vzniku tejto diplomovej práce však bola oznámená nová verzia Vapor Cloud 2[40], ktorá svojim prvotným zákazníkom poskytla neobmedzenú funkcionalitu úplne zadarmo, vrátane možnosti spustenia ľubovoľného počtu replík. Na základe tohoto faktu a svojmu určeniu pre framework Vapor bola pre nasadenie serverovej časti zvolená práve služba Vapor Cloud 2.

Výhodou nasadenia na Vapor Cloud 2 je, že službu je možné priamo prepojiť so systémom pre správu verzií GitHub, a celý kód serverovej časti nasadiť priamo z neho. Integrácia databázy je vykonaná automaticky, stačí iba vytvoriť jej inštanciu. Vapor Cloud 2 navyše poskytuje veľmi prehľadné a intuitívne webové rozhranie, v ktorom má používateľ prehľad o všetkých spustených aplikáciách, replikách a databázach. Serverová časť bola nasadená na adrese `https://glitter-weathered-54640.v2.vapor.cloud/api/`.

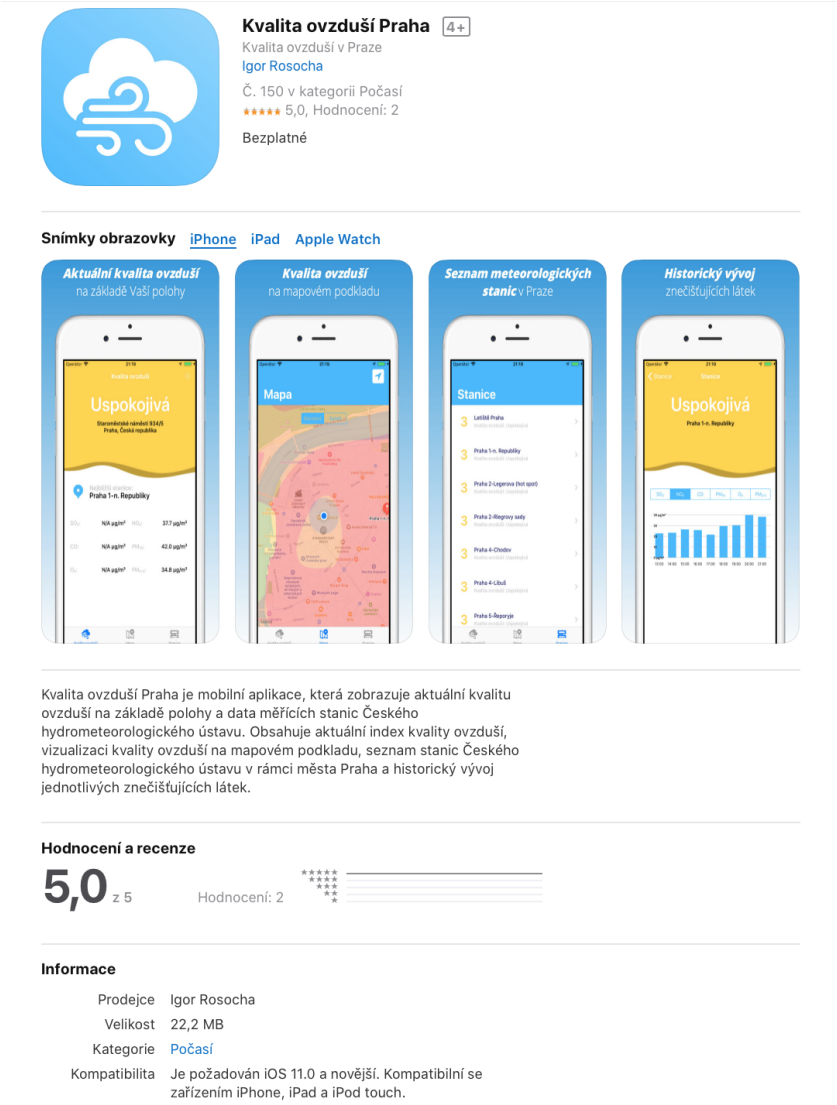
6.2.2 Mobilná aplikácia

Implementovaná a otestovaná aplikácia bola reálne nasadená a zverejnená zadarmo k stiahnutiu v obchode App Store. Podmienkou pre zverejnenie ap-

[40] `https://dashboard.v2.vapor.cloud`

likácie je vytvorenie vývojárského účtu za poplatok 99 dolárov ročne. Po vyplnení všetkých dôležitých údajov o aplikácii a jej nahratí musí prejsť procesom schvaľovania, ktorý trvá 2 až 3 pracovné dni. Pokiaľ aplikácia spĺňa všetky pravidlá a pokyny určené spoločnosťou Apple [63], je automaticky zverejnená a dostupná na obchode App Store.

Implementovaná aplikácia bola zverejnená pod názvom *Kvalita ovzduší Praha* (ang. *Air Quality Prague*). Primárnym jazkom aplikácie je čeština, no dostupná je aj v slovenskom a anglickom jazyku. Zaradená bola do kategórií *Počasie* a *Zdravie a fitnes*.



Obr. 6.1: Aplikácia nasadená na App Store

6.3 Budúcnosť vývoja

Server bol nasadený a možno predpokladať jeho dlhodobú a bezproblémovú dostupnosť. Server je však závislý na dostupnosti a zachovaní formátu dát, ktoré spracováva. Pokiaľ by tieto dáta zmenili formát, či neboli dlhodobo dostupné, bude potrebné vykonať menšie zmeny, resp. vyhľadať iný zdroj dát. Vzhľadom na konzistentnosť dátových zdrojov IPR Prahy a meracích staníc ČHMÚ však tento jav nie je v budúcnosti predpokladaný.

Počas vývoja mobilnej aplikácie bol veľký dôraz kladený na prípadnú rozšíriteľnosť o ďalšiu funkcionalitu. Možným rozšírením do budúcnosti by mohla byť možnosť vytvorenia zoznamu obľúbených lokalít, v ktorých používateľ trávi najviac času. Pri náhlom zhoršení kvality ovzdušia v niektorej z lokalít by bol následne používateľ informovaný o tejto skutočnosti prostredníctvom notifikácie. Ďalším rozšírením by mohlo byť vypracovanie a poskytovanie špecifických odporúčaní, aké činnosti je vhodné na základe aktuálneho stavu znečistenia ovzdušia vykonávať a ktorým činnostiam sa naopak vyhýbať, aby bol minimalizovaný čas vystaveniu znečistenému ovzdušiu. Vypracovanie týchto odporúčaní však vyžaduje odborné konzultácie so špecialistami v danej oblasti.

Záver

Na základe vykonanej analýzy bolo zistené, že aj napriek tomu, že existuje hneď niekoľko mobilných aplikácií, ktoré poskytujú informácie o kvalite a znečistení ovzdušia, žiadna z nich neposkytuje detailné informácie pre mesto Praha, či informácie založené na aktuálnej polohe používateľa. Existujú však 2 otvorené zdroje, ktoré dáta týkajúce sa týchto informácií poskytujú, konkrétne otvorené dáta Inštitútu planovania a rozvoja hlavného mesta Prahy a dáta meracích staníc Českého hydrometeorologického ústavu.

V rámci praće bola navrhnutá a implementovaná mobilná aplikácia pre operačný systém iOS a server založený na frameworku Vapor, ktorý pripravuje aktuálne dáta na základe spomínaných otvorených zdrojov a aplikácii ich poskytuje vo forme REST API. Aplikácia bola následne nasadená na obchod App Store a je dostupná všetkým používateľom zadarmo.

Aplikácia slúži pre všetkých ľudí žijúcich v Prahe na zistenie aktuálnej kvality ovzdušia a aktuálneho stavu znečistenia ovzdušia. Je dostupná pre všetky zariadenia s operačným systémom iOS 11 a vyššie spolu s rozšírením pre chytré hodinky Apple Watch. Výsledná aplikácia je dôkladne zdokumentovaná a otestovaná formou používateľského testovania s veľmi pozitívnou spätnou väzbou od používateľov.

Aplikácia tak môže aspoň malou mierou prispieť k informovanosti ľudí žijúcich v meste Praha o kvalite a znečistení ovzdušia v ich okolí a pomôcť im k lepšiemu plánovaniu každodenných aktivít a udržiavaniu svojho zdravia, či zdravia svojich blízkych.

Literatúra

[1] Kampa, M.; Castanas, E.: Human health effects of air pollution. *Environmental pollution*, ročník 151, č. 2, 2008: s. 362–367.

[2] Anderson, J. O.; Thundiyil, J. G.; Stolbach, A.: Clearing the air: a review of the effects of particulate matter air pollution on human health. *Journal of Medical Toxicology*, ročník 8, č. 2, 2012: s. 166–175.

[3] Hyrynsalmi, S.; Mäkilä, T.; Järvi, A.; aj.: App store, marketplace, play! an analysis of multi-homing in mobile software ecosystems. *Jansen, Slinger*, 2012: s. 59–72.

[4] Breezometer: Terms of Use - BreezoMeter Air Quality Data. [cit. 26.3.2019]. Dostupné z: `https://breezometer.com/terms-of-use`

[5] Institut plánování a rozvoje hlavního města Prahy: Bonita klimatu z hlediska znečištění ovzduší. [cit. 26.3.2019]. Dostupné z: `https://www.geoportalpraha.cz/cs/fulltext_geoportal/id/5BB4E2C5-9D4B-4B2B-BF0A-E0B98EE6013A`

[6] Český hydrometeorologický a ústav: Informace o kvalitě ovzduší v ČR. [cit. 26.3.2019]. Dostupné z: `http://portal.chmi.cz/files/portal/docs/uoco/web_generator/actual_hour_data_CZ.html`

[7] Institut plánování a rozvoje hlavního města Prahy: Pražská otevřená data. [cit. 26.3.2019]. Dostupné z: `https://www.geoportalpraha.cz/cs/clanek/271/prazska-otevrena-data`

[8] Janssen, M.; Charalabidis, Y.; Zuiderwijk, A.: Benefits, adoption barriers and myths of open data and open government. *Information systems management*, ročník 29, č. 4, 2012: s. 258–268.

[9] Petrisor, T.-R.: An empirical evaluation of using the Swift language as the underlying technology of RESTful APIs. 2016.

[10] Gouy, I.: Spectral-norm benchmarks. The Computer Language Benchmarks Game. [cit. 26.3.2019]. Dostupné z: https://benchmarksgame-team.pages.debian.net/benchmarksgame/performance/spectralnorm.html

[11] Yeung, A.: *Hands-On Server-Side Web Development with Swift: Build dynamic web apps by leveraging two popular Swift web frameworks: Vapor 3.0 and Kitura 2.5*. Packt Publishing Ltd, 2018.

[12] Evans, J.: IBM says: 'Swift is now ready for the enterprise'. [cit. 26.3.2019]. Dostupné z: https://www.computerworld.com/article/3122994/ibm-says-swift-is-now-ready-for-the-enterprise.html

[13] Evans, J.: WWDC 2016: Apple's Swift 3.0, star of the show. [cit. 26.3.2019]. Dostupné z: https://www.computerworld.com/article/3040196/wwdc-2016-apple-s-swift-3-0-star-of-the-show.html

[14] GitHub: Vapor stargazers. [cit. 26.3.2019]. Dostupné z: https://github.com/vapor/vapor/stargazers

[15] VAPOR: Vapor 3.0.0 released. [cit. 26.3.2019]. Dostupné z: https://medium.com/@codevapor/vapor-3-0-0-released-8356fa619a5d

[16] Wiegers, K. E.: *Požadavky na software*. Computer Press, Albatros Media as, 2017.

[17] Chung, L.; Nixon, B. A.; Yu, E.; aj.: *Non-functional requirements in software engineering*, ročník 5. Springer Science & Business Media, 2012.

[18] Apple, Inc.: About App Development with UIKit. [cit. 1.4.2019]. Dostupné z: https://developer.apple.com/documentation/uikit/about_app_development_with_uikit

[19] Plakalovic, D.; Simic, D.: Applying MVC and PAC patterns in mobile applications. *arXiv preprint arXiv:1001.3489*, 2010.

[20] Orlov, B.: iOS Architecture Patterns. [cit. 1.4.2019]. Dostupné z: https://medium.com/ios-os-x-development/ios-architecture-patterns-ecba4c38de52

[21] Nguyen, L.; Nguyen, K.: Application of protocol-oriented mvvm architecture in ios development. 2017.

[22] Arnowitz, J.; Arent, M.; Berger, N.: *Effective prototyping for software makers*. Elsevier, 2010.

[23] Apple, Inc.: Human Interface Guidelines. [cit. 1.4.2019]. Dostupné z: https://developer.apple.com/design/human-interface-guidelines/

[24] Apple, Inc.: Class UINavigationController. [cit. 1.4.2019]. Dostupné z: https://developer.apple.com/documentation/uikit/uinavigationcontroller

[25] Apple, Inc.: Class UITabBarController. [cit. 1.4.2019]. Dostupné z: https://developer.apple.com/documentation/uikit/uitabbarcontroller

[26] Apple, Inc.: Apple Watch Display Sizes. [cit. 1.4.2019]. Dostupné z: https://developer.apple.com/design/human-interface-guidelines/watchos/visual-design/display-sizes/

[27] Barnum, C. M.: *Usability testing essentials: ready, set... test!* Elsevier, 2010.

[28] VAPOR: Official Vapor 3 documentation, PostgreSQL. [cit. 4.4.2019]. Dostupné z: https://docs.vapor.codes/3.0/postgresql/getting-started/

[29] Smith, G.: *PostgreSQL 9.0: High Performance*. Packt Publishing Ltd, 2010.

[30] Fielding, R. T.; Taylor, R. N.: *Architectural styles and the design of network-based software architectures*, ročník 7. University of California, Irvine Doctoral dissertation, 2000.

[31] Bray, T.: The javascript object notation (json) data interchange format. Technická zpráva, 2017.

[32] Apple, Inc.: Xcode. [cit. 4.4.2019]. Dostupné z: https://developer.apple.com/xcode/

[33] Apple, Inc.: App Store distribution. [cit. 4.4.2019]. Dostupné z: https://developer.apple.com/support/app-store/

[34] Apple, Inc.: Codable. [cit. 4.4.2019]. Dostupné z: https://developer.apple.com/documentation/swift/codable

[35] Apple, Inc.: Encoding and Decoding Custom Types. [cit. 4.4.2019]. Dostupné z: https://developer.apple.com/documentation/foundation/archives_and_serialization/encoding_and_decoding_custom_types

[36] Apple, Inc.: Class JSONDecoder. [cit. 4.4.2019]. Dostupné z: https://developer.apple.com/documentation/foundation/jsondecoder

[37] Mishra, A.: Installing Quick. In *iOS Code Testing*, Springer, 2017, s. 329–349.

[38] Apple, Inc.: Interface Builder. [cit. 4.4.2019]. Dostupné z: https://developer.apple.com/xcode/interface-builder/

[39] Loeliger, J.; McCullough, M.: *Version Control with Git: Powerful tools and techniques for collaborative software development*. Ö'Reilly Media, Inc.", 2012.

[40] Apple, Inc.: Instance method loadView(). [cit. 5.4.2019]. Dostupné z: https://developer.apple.com/documentation/uikit/uiviewcontroller/1621454-loadview

[41] Horowitz, E.: *Fundamentals of programming languages*. Springer Science & Business Media, 2012.

[42] Blackheath, S.; Jones, A.: *Functional Reactive Programming*. Manning Publications Company, 2016.

[43] Costa, C.: *Reactive Programming with Swift*. Packt Publishing Ltd, 2016.

[44] GitHub: ReactiveCocoa stargazers. [cit. 1.4.2019]. Dostupné z: https://github.com/ReactiveCocoa/ReactiveCocoa/stargazers

[45] Apple, Inc.: Instance method viewDidLoad(). [cit. 5.4.2019]. Dostupné z: https://developer.apple.com/documentation/uikit/uiviewcontroller/1621495-viewdidload

[46] Prasanna, D. R.: *Dependency injection*. Manning, 2009.

[47] Zabłocki, K.: Leveraging protocol composition for more maintainable dependency injection. [cit. 1.4.2019]. Dostupné z: https://medium.com/@merowing_/leveraging-protocol-composition-for-more-maintainable-dependency-injection-640a7389239e

[48] Tamplin, J.: Firebase is Joining Google! [cit. 5.4.2019]. Dostupné z: https://firebase.googleblog.com/2014/10/firebase-is-joining-google.html

[49] Apple, Inc.: Framework MapKit. [cit. 5.4.2019]. Dostupné z: https://developer.apple.com/documentation/mapkit

[50] Apple, Inc.: Class UITableView. [cit. 5.4.2019]. Dostupné z: https://developer.apple.com/documentation/uikit/uitableview

[51] Apple, Inc.: Framework UIKit. [cit. 5.4.2019]. Dostupné z: https://developer.apple.com/documentation/uikit

[52] Apple, Inc.: Class UISegmentedControl. [cit. 5.4.2019]. Dostupné z: https://developer.apple.com/documentation/uikit/uisegmentedcontrol

[53] Apple, Inc.: watchOS. [cit. 15.4.2019]. Dostupné z: https://www.apple.com/lae/watchos/watchos-5/

[54] Galleso, M.: *Watch Os 3 for the Apple Watch: An Easy Guide to the Best Features*, ročník 1. First Rank Publishing, 2017.

[55] Apple, Inc.: Framework WatchKit. [cit. 15.4.2019]. Dostupné z: https://developer.apple.com/documentation/watchkit

[56] Apple, Inc.: Framework WatchConnectivity. [cit. 15.4.2019]. Dostupné z: https://developer.apple.com/documentation/watchconnectivity

[57] Pančur, M.; Ciglarič, M.: Impact of test-driven development on productivity, code and tests: A controlled experiment. *Information and Software Technology*, ročník 53, č. 6, 2011: s. 557–573.

[58] Apple, Inc.: Framework XCTest. [cit. 20.4.2019]. Dostupné z: https://developer.apple.com/documentation/xctest

[59] Apple, Inc.: TestFlight. [cit. 20.4.2019]. Dostupné z: https://developer.apple.com/testflight/

[60] Bangor, A.; Kortum, P. T.; Miller, J. T.: An empirical evaluation of the system usability scale. *Intl. Journal of Human–Computer Interaction*, ročník 24, č. 6, 2008: s. 574–594.

[61] Mellor, C.: DigitalOcean cuts cloud server pricing to stop rivals eating its lunch. [cit. 15.4.2019]. Dostupné z: https://www.theregister.co.uk/2018/01/18/digital_ocean_cuts_cloud_server_prices/

[62] Middleton, N.; Schneeman, R.: *Heroku: Up and Running: Effortless Application Deployment and Scaling*. Ö'Reilly Media, Inc.", 2013.

[63] Apple, Inc.: App Store Review Guidelines. [cit. 20.4.2019]. Dostupné z: https://developer.apple.com/app-store/review/guidelines/

Dodatok **A**

Zoznam použitých skratiek

API Application Programming Interface

BAQI Breezometer Air Quality

CO Oxid uhoľnatý

ČHMÚ Český hydrometeorologický ústav

DXF Drawing Exchange Format

FRP Funkcinonálne reaktívne programovanie

GeoJSON Geographic JavaScript Object Notation

GML Geography Markup Language

HTTP Hypertext Transfer Protocol

IPR Inštitút plánovania a rozvoja

JSON JavaScript Object Notation

MVC Model-View-Controller

MVVM Model-View-Viewmodel

NO_2 Oxid dusičitý

O_3 Ozón

$PM_{2.5}$ Suspedované jemné prachové častice

PM_{10} Suspedované prachové častice

REST Representational State Transfer

SO_2 Oxid siričitý

SUS System Usability Scale

SQL Structured Query Language

URL Uniform Resource Locator

WHO World Health Organization

XML eXtensible Markup Language

Dodatok B

Obsah priloženého CD

- `readme.txt` ..... stručný popis obsahu CD
- `ios-app` ..... zdrojový kód mobilnej aplikácie
- `vapor-server` ..... zdrojový kód serverovej časti
- `docs` ..... dokumentácia
  - `api` ..... dokumentácia API
  - `app` ..... dokumentácia mobilnej aplikácie
  - `server` ..... dokumentácia serverovej časti
  - `system-manual.pdf` ..... systémová príručka
  - `user-manual.pdf` ..... používateľská príručka
- `text` ..... text práce
  - `tex` ..... zdrojový kód práce vo formáte LaTeX
  - `thesis.pdf` ..... text práce vo formáte PDF
- `api-tests-report.html` ..... záznam z testovania API