FAKULTA INFORMAČNÍCH TECHNOLOGIÍ ČVUT V PRAZE

Zadání diplomové práce

Název:	Mobilní aplikace Tříděný odpad Praha
Student:	Bc. Lukáš Brhlík
Vedoucí:	Ing. Ondřej John
Studijní program:	Informatika
Obor / specializace:	Webové inženýrství
Katedra:	Katedra softwarového inženýrství
Platnost zadání:	do konce letního semestru 2022/2023

Pokyny pro vypracování

Cílem práce je návrh, implementace a otestování aplikace pro OS Android obsahující seznam kontejnerů na tříděný odpad na území hlavního města Prahy. Budou podporovány mobilní telefony, tablety a wearables (chytré hodinky). Aplikace bude obsahovat následující funkce:

- Zobrazení jednotlivých kontejnerů na interaktivní mapě.
- Zobrazení formou seznamu s možností filtrování, třídění a vyhledávání.

Součástí práce bude implementace back-end služby poskytující data pro mobilní aplikaci prostřednictvím REST API.

- Analyzujte a popište strukturu zdrojových dat poskytovaných hl. m. Praha.
- Specifikujte funkční a nefunkční požadavky na mobilní aplikaci a back-end API.
- Po dohodě s vedoucím práce navrhněte uživatelské rozhraní mobilní aplikace.
- Navrhněte rozhraní REST API, které bude poskytovat data pro mobilní aplikaci.
- Navrhněte datovou strukturu backend služby.
- Implementujte a otestujte API.
- Implementujte a otestujte mobilní aplikaci.
- Shrňte výsledek práce, popište její přínos.

Elektronicky schválil/a Ing. Jaroslav Kuchař, Ph.D. dne 19. října 2021 v Praze.

FAKULTA
INFORMAČNÍCH
TECHNOLOGIÍ
ČVUT V PRAZE

Diplomová práce

Mobilní aplikace Tříděný odpad Praha

Bc. Lukáš Brhlík

Katedra softwarového inženýrství
Vedoucí práce: Ing. Ondřej John

3. května 2022

Poděkování

Rád bych poděkoval svému vedoucímu, Ing. Ondřeji Johnovi za pomoc a rady při psaní a zpracování této diplomové práce. Poděkování také patří mé rodině za obrovskou podporu během celého studia.

Prohlášení

Prohlašuji, že jsem předloženou práci vypracoval samostatně a že jsem uvedl veškeré použité informační zdroje v souladu s Metodickým pokynem o dodržování etických principů při přípravě vysokoškolských závěrečných prací.

Beru na vědomí, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorského zákona, ve znění pozdějších předpisů. V souladu s ust. § 2373 odst. 2 zákona č. 89/2012 Sb., občanský zákoník, ve znění pozdějších předpisů, tímto uděluji nevýhradní oprávnění (licenci) k užití této mojí práce, a to včetně všech počítačových programů, jež jsou její součástí či přílohou a veškeré jejich dokumentace (dále souhrnně jen „Dílo"), a to všem osobám, které si přejí Dílo užít. Tyto osoby jsou oprávněny Dílo užít jakýmkoli způsobem, který nesnižuje hodnotu Díla a za jakýmkoli účelem (včetně užití k výdělečným účelům). Toto oprávnění je časově, teritoriálně i množstevně neomezené. Každá osoba, která využije výše uvedenou licenci, se však zavazuje udělit ke každému dílu, které vznikne (byť jen zčásti) na základě Díla, úpravou Díla, spojením Díla s jiným dílem, zařazením Díla do díla souborného či zpracováním Díla (včetně překladu) licenci alespoň ve výše uvedeném rozsahu a zároveň zpřístupnit zdrojový kód takového díla alespoň srovnatelným způsobem a ve srovnatelném rozsahu, jako je zpřístupněn zdrojový kód Díla.

V Praze dne 3. května 2022 . . . . . .

Odkaz na tuto práci

Brhlík, Lukáš. *Mobilní aplikace Tříděný odpad Praha.* Diplomová práce. Praha: České vysoké učení technické v Praze, Fakulta informačních technologií, 2022.

Abstrakt

Cílem této práce je implementace aplikace pro OS Android obsahující seznam kontejnerů na tříděný odpad na území hlavního města Prahy. Tato aplikace bude podporovat mobilní telefony, tablety a wearables (chytré hodinky). Součástí práce je i implementace serveru – aplikačního rozhraní, se kterým aplikace komunikuje a získává z něj data.

Práce se věnuje analýze otevřených dat hl. města Prahy, na kterých je aplikace postavena, rozebírá podobná konkurenční řešení a srovnává různé možnosti implementace aplikačního rozhraní v jazyce Ruby. Klíčovou částí je pak popis samotného návrhu a implementace mobilní aplikace i aplikačního rozhraní. Aplikace je napsána v jazyce Kotlin, pro implementaci API je zvolen jazyk Ruby.

Výsledné řešení je podrobeno testům a výstupem je funkční mobilní aplikace a API.

Klíčová slova Android, Ruby, Jetpack Compose, Mobilní aplikace, API, OpenData

Abstract

The aim of this thesis is to implement an application for Android OS containing a list of containers for recycling on the territory of Prague. This application will be compatible with smartphones, tablets and wearables (eg. smart watches). The thesis also includes the implementation of the application interface with which the application communicates and gets data from.

The thesis analyses the open data of the City of Prague, on which the application is built, analyses similar competing solutions and compares different options for implementing the application interface in Ruby. The key part of this thesis is a description of the actual design as well as implementation of the mobile application and the application interface. The application is written in Kotlin, and Ruby is chosen for the API implementation.

The final result is tested and the output is a functional mobile application and API.

Keywords Android, Ruby, Jetpack Compose, Mobile app, API, OpenData

Obsah

Seznam obrázků

Seznam tabulek

Úvod

Téma ekologie rezonuje dnešní společností čím dál tím víc. Lidé se zajímají o udržitelnost, snižování uhlíkové stopy, globální oteplování a nevymizelo ani téma třídění odpadu [2]. Počty kontejnerů na tříděný odpad rostou [1], v mnoha domácnostech jsou rozšířené kompostéry a existují i domácnosti, ve kterých úplně vymizely koše na směsný odpad, jelikož jsou schopny vše vytřídit nebo zrecyklovat. I přes to však může být v dnešní rychlé době problém najít místo, kam je možné vyhodit kus plastu, když jsem zrovna na místě, které neznám.

Se zvládnutím takových situací může pomoci aplikace, která je součástí této diplomové práce. Staví na datové sadě hlavního města Prahy, která popisuje rozmístění kontejnerů na tříděný odpad. Tato datová sada je spolu s dalšími zveřejňována jako součást otevřených dat. Aplikace pomůže najít místo pro vytřídění odpadu přesně dle zadaných parametrů.

Kapitola 1

Cíl práce

Cílem první části práce je analýza struktury zdrojových dat poskytovaných hl. m. Praha. Porovnány jsou také aktuálně fungující aplikace pro zobrazení míst, kde se dá třídit v Praze odpad. V analytické části je vybrán framework pro implementaci backendového API v jazyce Ruby a analyzovány jsou možnosti geokódování, které bude API využívat.

Druhá a třetí část se zabývají návrhem a implementací. Detailně popisují proces vzniku aplikace a API, mechanismy jejího fungování, použité nástroje a veškeré technologie, které byly při vývoji použity.

V sekci o testování je vysvětleno, jakým způsobem byla aplikace otestována z hlediska funkčnosti a uživatelské přívětivosti.

Výstupem práce je aplikace pro OS Android, Wear OS, které zpřístupňují pozice košů na tříděný odpad po Praze a API, které umožňuje tato data získávat.

Kapitola 2

Analýza

V rámci této kapitoly je provedena analýza struktury zdrojových dat poskytovaných hl. m. Praha. a datové sady STANOVIŠTĚ TŘÍDĚNÉHO ODPADU – POLOŽKY. Srovnány jsou existující aplikace, které se zabývají tříděním odpadu na území města Prahy. Následně je provedena analýza dostupných API pro geokódování, kde jsou srovnány jednotlivé možnosti, které je možno využít pro implementaci v Ruby. Současně jsou také porovnány frameworky pro implementaci backendového API. Z obou srovnání jsou vybrána řešení, následně využitá pro implementaci.

2.1 Otevřená data

Otevřená data jsou v [29] definována jako volně dostupná data bez omezení jejich dalšího využití zveřejněná ve strukturované, strojově zpracovatelné podobě. Jejich zveřejnění dle [29] vyžaduje pouze

- mít k dispozici potenciálně zajímavá data,
- zveřejnit je v úplné podobě,
- napsat, za jakých podmínek je možné jejich využití,
- alespoň základně data zdokumentovat.

Zveřejňována jsou v různých stupních kvality. Vyšší kvalita znamená lepší formát a usnadnění práce pro uživatele, který data zpracovává. Ideálním formátem pro následné strojové zpracování jsou Linked Open Data. Každý záznam v takovém formátu má svou URL adresu, která jej reprezentuje a jsou na ní dostupné další informace o tomto záznamu. Všech 5 stupňů hodnocení otevřených dat je popsáno v následujícím seznamu [28].

- **Jednohvězdičková** jsou data v nestrukturované podobě. Mohou to být naskenované dokumenty, dokumenty v digitální podobě např. ve formátu MS Word. Vhodné jsou pouze pro zpracování člověkem.

- **Dvouhvězdičková** data jsou zveřejněna ve strojově zpracovatelném formátu. Může to být například tabulka v MS Excel.
- **Tříhvězdičková** data jsou navíc oproti předchozímu stupni zveřejněna v otevřeném formátu. Místo proprietárního formátu bez otevřené specifikace je tedy využit formát otevřený, jako například CSV, což je text, kde jsou hodnoty oddělené např. středníkem.
- **Čtyřhvězdičková** označujeme data uložená v jazyce XML, nebo jazyce z něj odvozeném. Nejčastějšími formáty, které se využívají k definici jsou formáty RDF a SPARQL. Data jsou propojená v rámci sady odkazy.
- **Pětihvězdičková** data jsou navíc oproti čtyřhvězdičkovým napojena na další datové sady. To umožňuje tvorbu velkých propojených sítí dat a dodává obsahu sady kontext.

Přínosů otevřených dat je celá řada, od větší efektivity veřejné správy, přes hospodářský růst soukromého sektoru až po větší společenský blahobyt. Výhody pro ekonomiku spočívají ve snadnějším přístupu k různým informacím, přičemž příslušný obsah a znalosti zase přispívají k rozvoji inovativních služeb a tvorbě nových obchodních modelů. Nová pracovní místa vznikají díky stimulaci ekonomiky a větší poptávce po pracovnících, kteří jsou kvalifikovaní pro práci s daty. Díky otevřeným datům roste také efektivita, protože používání dat v reálném čase umožňuje snadný přístup k informacím, které mohou přispívat k snadnějšímu individuálnímu rozhodování [29].

2.1.1 Analýza struktury zdrojových dat poskytovaných hl. m. Praha.

Hlavní město Praha poskytuje otevřená data prostřednictvím svého webového portálu, dostupného na adrese *opendata.praha.eu*. Ten je provozován na open-source platformě CKAN. Ve strojově čitelných formátech je zde zveřejňují magistrát hlavního města, příspěvkové organizace, městské části a další související subjekty. Otevřená data definuje pražský portál jako data, která jsou:

- přístupná komukoliv zdarma a volně dostupná ke stažení na internetu,
- zveřejněna v úplné podobě bez odstraňování vybraných údajů,
- strojově čitelná,
- použitelná bez omezení (tj. pro komerční i nekomerční účely),
- katalogizovaná v Národním katalogu otevřených dat,
- zveřejněná dle Standardů publikace a katalogizace otevřených dat Ministerstva vnitra ČR dokumentovaných na tomto webu [5].

Datové sady, které splňují tato pravidla, lze na webu vyhledávat na základě organizace, která sadu vytvořila, skupin, tagů, formátů a licencí.

Mezi těmito sadami můžeme najít i tu, kterou se zabývá tato diplomová práce. Nese název „STANOVIŠTĚ TŘÍDĚNÉHO ODPADU – POLOŽKY“ a je poskytovaná ve formátech GEOJson, GML a jako Shapefile Institutem plánování a rozvoje hlavního města Prahy. Sada je dostupná pod licencí Creative Commons CCZero. Z dostupných informací je také možno vyčíst, že údržba není periodická, aktualizace probíhají pouze dle potřeby – poslední proběhla 8. září 2020[6]. Podrobnější dokumentaci lze najít na geoportálu hlavního města Prahy[7]. Tam jsou popsané atributy souboru, tvůrce datasetu, kvalita dat a další.

2.1.2 Struktura datasetu stanovišť tříděného odpadu

Pro potřeby této práce byla zvolena data ve formátu GeoJSON. Je to formát, který slouží pro reprezentaci jednoduchých prostorových geografických dat společně s jejich dalšími atributy. V datové sadě je nejprve uveden její typ - `FeatureCollection` a jméno – `ZPK_O_Kont_TOitem_b`. V následujícím poli features jsou již samotná stanoviště tříděného odpadu. Jako geometrické body se svojí zeměpisnou šířkou a délkou jsou uvedeny vždy kontejnery na jednotlivý typ odpadu. Máme tedy bod se souřadnicemi, kde se třídí např. plast a bod druhý s totožnými souřadnicemi, ale jiným typem odpadu a dalšími vlastnostmi. Zeměpisná šířka je uvedena před zeměpisnou délkou. Po geometrické klasifikaci následují další vlastnosti:

- `OBJECTID` – id objektu – očíslování záznamů v datasetu – 23455 v datasetu
- `STATIONID` – id stanoviště – očíslování stanovišť s kontejnery – 6287 v datasetu - tyto v aplikaci zobrazujeme a do nich data seskupujeme
- `TRASHTYPENAME` – typ odpadu, který se zde třídí (Papír, Plast, Nápojové kartony, Barevné sklo, Čiré sklo, Kovy, Elektroodpad)
- `CLEANINGFREQUENCYCODE` – Definice: četnost vývozu (1.hodnota = délka období, 2.hodnota = četnost vývozu, např.13 = 3x za 1 týden, 61 = 1x za 6 týdnů) – v aplikaci není využito
- `CONTAINERTYPE` – typ kontejneru (3350 Atomium Reflex SV, 2500 Reflex SV, 3000 Podzemní SV, 120 normální HV, 1100 mini H SV, 240 normální HV, 1100 normální HV, 3300 GFA DUO SV, 1500 M 1,5 SV, 2150 MEVA SV, 3200 Schaffer SV) – v aplikaci není využito
- `CONTAINERS` – počet kontejnerů – desetinné číslo udávající počet kontejnerů na daný druh odpadu na stanovišti

Data jsou zveřejněna pod licencí `Creative Commons CCZero`. Autor tedy souhlasil s vystavením tohoto díla jako díla volného a celosvětově se vzdal všech svých autorských práv k dílu, včetně všech práv souvisejících a práv příbuzných v rozsahu, který je povolen zákonem [14]. Vytvářená aplikace zdroj dat nemusí nikde uvádět, nemá však žádné záruky o správnosti.

Dle dříve definované kategorizace kvality otevřených dat, tato datová sada spadá mezi tříhvězdičkové.

2.2 Podobné aplikace

Pro stanovení funkčních a nefunkčních požadavků aplikace, analýzu problematiky a porovnání nabízených možností jsou zde uvedena některá řešení zabývající se tříděním odpadu. Koše na tříděný odpad na území hl. města Prahy ukazuje aplikace *Tříděný odpad (Barevné kontejnery) Praha* [15]. Na mapě zobrazuje jednotlivá stanoviště pro třídění, včetně typu kontejneru i frekvence vyvážení. Umožňuje offline stažení dat kontejnerů i mapových podkladů a filtraci zobrazovaných míst podle druhu odpadu.

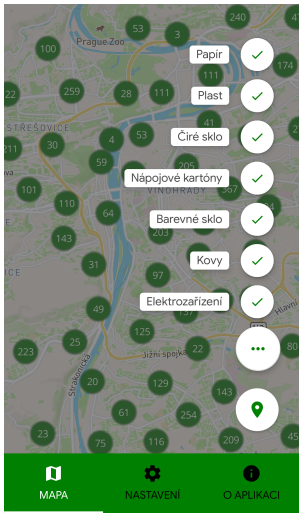
Další možností, která radí s tříděním odpadu nejen v Praze je aplikace *Kam s ním?* [16]. Nabízí pokročilé možnosti filtrování různých druhů odpadu od autovraků, přes léky až po obyčejné nápojové kartony. Je zde možnost spravovat svůj uživatelský účet a kontejnerům jsou přiřazeny i adresy. Aplikace není nativní, je to pouze zobrazení webové stránky kamsnim.cz a nepůsobí tedy na uživatele tak dobře, jako kdyby nativní byla – méně přirozené ovládání, nefunkční gesta.

Další aplikace se zaměřují na menší územní celky a často nabízejí i možnost nahlásit plný koš, či poskytují informace o aktuálně umístěných kontejnerech na svoz odpadu, tato data však pro větší územní celky nejsou veřejně ke zpracovávání zpřístupněna. Mezi tyto aplikace patří zejména *Třídění odpadu ve xxx*, kde je za xxx dosazeno jméno města, kterému slouží [3].

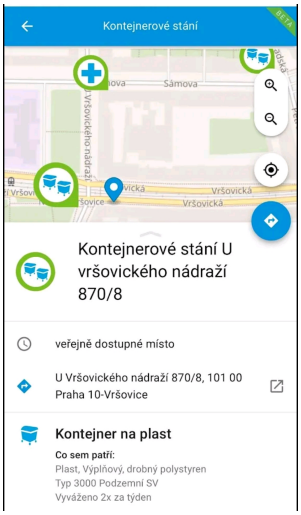
Část kontejnerů zobrazuje i široce rozšířená aplikace *Mapy.cz* [4]. Nenabízí však žádnou možnost jejich filtrování, vyhledávání podle adresy, ani jinou podobnou funkcionalitu. Srovnání funkcionality aplikací je znázorněno v tabulce 2.1.

	Vyhledávání podle adresy	Filtrování	Nativní
Tříděný odpad Praha	✗	✓	✓
Kam s ním?	✓	✓	✗
Mapy.cz	✓	✗	✓

Tabulka 2.1: Tabulka srovnávající funkcionalitu podobných aplikací



(a) Aplikace Tříděný odpad – Praha [15]



(b) Aplikace Kam s ním? [16]

Obrázek 2.1: Příklady podobných aplikací

2.3 Ruby frameworky pro implementaci API

2.3.1 Úvod

Pro implementaci API je zvolen jazyk Ruby pro jeho uváděnou jednoduchost použití a osobní chuť naučit se práci s novou technologií. Frameworků pro tento účel existuje v Ruby velké množství, výběr je tedy omezen našimi požadavky.

2.3.2 Požadavky na framework

- Aktuálnost
- Kompatibilita
- Jednoduchost řešení
- Rozšířenost

Na základě těchto požadavků byla vybrána 3 řešení splňující všechny stanovené požadavky. Každý z požadavků je následně bodově ohodnocen. Za aktuálnost jsou přiděleny body podle doby, kdy byla vydána poslední aktualizace, (měsíc-rok-2 roky-starší). V rámci kompatibility jsou body přidělovány podle počtu dostupných rozšíření, které by mohly být při tvorbě API využity (gem `ActiveRecord` – práce s databázovými modely, stránkování, práce s PostgreSQL databází...). U jednoduchosti řešení je brán ohled na počet souborů a složek obsažených v kostře projektu daného frameworku. U rozšířenosti

se body přidávají za jejich využití širokou veřejností a firmami. Čím větší rozšířenost, tím lépe. Ta zajistí větší dostupnost návodů a rozsáhlejší komunitu, která je schopna pomoci s řešením problémů.

2.3.3 Sinatra

Open-source framework pro tvorbu webových aplikací, či aplikačních rozhraní. Využívá serverového rozhraní Rack. Poslední aktualizace kódu v hlavní vývojové větvi jejich verzovacího systému je v čase psaní 3 měsíce stará, poslední vydání gemu pak proběhlo 12. února 2022. Projekt je tedy často aktualizován. Plně kompatibilní je s ActiveRecord, postgresql databází, dostupné je pokročilé parsování volaných parametrů, dostupný je i gem pro stránkování. Sinatra nevnucuje přístup model-view-controller, tím se redukuje množství kódu potřebného pro základní funkčnost malé aplikace. Jednoduchou Sinatra aplikaci můžeme napsat na 4 řádky [9]. Se 176 miliony stažení se řadí na 3. místo mezi gemy, využívanými pro tvorbu webových aplikací [8].

Parametr	Body
Aktuálnost	4
Kompatibilita	5
Jednoduchost řešení	5
Rozšířenost	5
Celkem	**16**

Tabulka 2.2: Bodové ohodnocení frameworku Sinatra

2.3.4 Roda

Framework, jehož cílem je stavět na dobrých základech vybudovaných dříve zmiňovanou Sinatrou. Vylepšení míří především na větší webové aplikace s lepší škálovatelností, mimo to nabízí větší rychlost nebo menší jádro. Aktivně je spravován pouze jedním člověkem, který se o něj však neustále stará a framework je stále vyvíjen. Poslední aktualizace hlavní vývojové větve je v čase psaní 9 dní stará, poslední vydání byl potom proběhlo před 2 týdny. Mezi pluginy rozšiřující jádro s omezenou funkcionalitou nenajdeme žádné napojení na ActiveRecord, pro práci s databází se používá Sequel, ten umí pracovat pouze s SQL dotazy. Najdeme zde však plugin na stránkování, práci s postgresql databází, nebo usnadnění vytváření REST API. Jednoduchou Roda aplikaci, dokážeme napsat do jednoho krátkého souboru[10]. Něco málo přes milion a půl stažení značí, že Roda není příliš rozšířená[8].

Parametr	Body
Aktuálnost	4
Kompatibilita	3
Jednoduchost řešení	5
Rozšířenost	1
Celkem	**13**

Tabulka 2.3: Bodové ohodnocení frameworku Roda

2.3.5 Grape

Framework pro tvorbu REST-like API, který se dá používat buď samostatně, nebo jako doplněk jiných frameworků jako jsou Rails, či Sinatra. Grape je částečně spravován týmem placených lidí, jelikož jsou na frameworku závislé některé další služby, proto je pravidelně aktualizován. Poslední vydání i aktualizaci najdeme v období uběhlých 3 týdnů. Bez dodatečných pluginů nabízí funkce jako pokročilou kontrolu parametrů nebo práci s verzemi a hlavičkami. Umí také pracovat s ActiveRecord a stránkováním. Se svými 33 miliony stažení je na tom s popularitou a rozšířením velmi dobře, v rámci projektů však vyžaduje rozsáhlejší strukturu i pro jednodušší aplikace.[11]

Parametr	Body
Aktuálnost	4
Kompatibilita	5
Jednoduchost řešení	2
Rozšířenost	5
Celkem	**16**

Tabulka 2.4: Bodové ohodnocení frameworku Grape

2.3.6 Vyhodnocení

Mezi hodnocené frameworky nebyly zařazeny ty nejrozšířenější z nich, jako jsou Ruby on Rails, či jbuilder. Hodí se totiž spíše pro rozsáhlejší aplikace. Z hodnocených aplikacích vyšel nejlépe framework Sinatra. Splňuje všechny naše požadavky. Přesto, že je Grape využívanější, rozšířenost Sinatry je dostatečná. Framework Sinatra byl tedy vybrán jako nástroj pro implementaci backendového API. Celkové bodové výsledky jsou shrnuty v tabulce 2.5.

Framework	Akt.	Kompatib.	Jednoduchost	Rozšířenost	**Celkem**
Sinatra	4	5	5	5	**19**
Roda	4	3	5	1	**13**
Grape	4	5	2	5	**16**

Tabulka 2.5: Výsledná tabulka hodnotící frameworky pro tvorbu jednoduchého API

2.4 Geokódovací API

2.4.1 Úvod

V aplikaci bude možné vyhledávat koše na základě jejich adresy. Ta se však v datové sadě nenachází. Jednou z možností, jak ji určit, je pomocí reverzního geokódování. Pomocí něj určujeme na základě zadané souřadnice adresu. Tuto funkcionalitu nám nabízí velké množství rozhraní pro programování aplikací.

2.4.2 Požadavky na API pro reverzní geokódování

- Jednoduchost použití v jazyce Ruby
- Cena
- Možnost kódování většího množství dat současně
- Dostupnost bez vlastního hostování

Z analýzy datové sady bylo zjištěno, že bude potřeba zakódovat 6287 míst. Pro toto množství budou zohledňovány ceny a limitace. Za jednoduchost použití v jazyce Ruby budou uděleny 0 až 3 body v případě dostupnosti skrze gem a jeho jednoduchost použití. U ceny je možné získat 0 až 3 body. Ideálem je řešení nabízené zdarma, bez omezení relevantních pro tento projekt. Za možnost kódování většího množství dat současně jsou uděleny 0 až 2 body. 2 body se udělí za nadstandardní rychlost. Za dostupnost bez nutnosti vlastního hostování se uděluje 0, nebo 1 bod.

2.4.3 Google Geocoding API

API od společnosti Google, jehož výhodou je zastřešení firmou, která má k dispozici velké množství dat. Google mapy využívá denně velký počet lidí lidí. Nabízí množství parametrů pro filtrování a detailní rozdělení výsledných parametrů pro potřeby dělení adresy na např. dům, ulici a orientační číslo. V případě, že chceme zobrazovat výsledky na mapě, musí to být specificky Google mapa [21]. Pro využívání je nutný API klíč, na který je vázán účet, u kterého je povoleno účtování poplatků. Ty jsou pro 0 – 100.000 dotazů

0.005 dolarů za každý. Je zakázáno výsledky ukládat, indexovat nebo cachovat. Povoleno je 50 dotazů za vteřinu a API není nutné ani možné hostovat samostatně. Využití v rámci Ruby je možné s využitím Geocoder gemu, je však potřeba nastavit příslušný API klíč.

Parametr	Body
Jednoduchost použití v jazyce Ruby	2
Cena	1
Možnost kódování většího množství dat naráz	2
Dostupnost bez vlastního hostování	1
Celkem	**6**

Tabulka 2.6: Bodové ohodnocení Google Geocoding API

2.4.4 MapTiler

Geokódovací API je nabízeno v rámci služby MapTiler Cloud. Zdarma, pro nekomerční účely nabízí až 100.000 dotazů za měsíc. Pro využívání je nutné vytvořit účet a k dotazům přikládat API klíč. Nabízí však také možnost pro vlastní hostování, kde nejsou limity žádné. MapTiler je nabízen jako jedna z možností Geocoder gemu pro jazyk Ruby. Nikde nejsou uvedeny limitace pro rychlost ani maximální objem pro jednorázové kódování většího množství dat současně [22].

Parametr	Body
Jednoduchost použití v jazyce Ruby	2
Cena	2
Možnost kódování většího množství dat naráz	2
Dostupnost bez vlastního hostování	1
Celkem	**7**

Tabulka 2.7: Bodové ohodnocení MapTiler API

2.4.5 Geoapify

Služba umožňující geokódování, reverzní geokódování a mnoho dalšího. Nabízí bezplatný plán, který pro potřeby projektu umožňuje až 3.000 dotazů za den nebo 90.000 dotazů za měsíc s limitací na 5 dotazů za sekundu. Je nutné připojit účet a vytvořit API klíč. Výsledky je možné ukládat, jelikož je Geoapify založené na otevřených datech. API je pro Ruby dostupné v rámci Geocoder gemu [23].

Parametr	Body
Jednoduchost použití v jazyce Ruby	2
Cena	2
Možnost kódování většího množství dat naráz	2
Dostupnost bez vlastního hostování	1
Celkem	**7**

Tabulka 2.8: Bodové ohodnocení Geoapify

2.4.6 Pelias

Vyhledávací open-source nástroj, který je postaven na využívání otevřených dat. Službu je nutné samostatně nasadit, následně však poskytuje všechny služby bez omezení a klíčů. Umožňuje sestavení omezit na lokace o menší velikosti, vhodný je tedy pro naše omezené využití na město Praha. Pro nasazení se doporučuje Docker kontejner a autoři uvádí, že při omezení na 1 město bude služba dostupná přibližně do půl hodiny [25]. Nabízí detailní nastavení parametrů, včetně volby zdroje dat. Lze jej využívat v rámci gemu Geocoder.

Parametr	Body
Jednoduchost použití v jazyce Ruby	3
Cena	3
Možnost kódování většího množství dat naráz	2
Dostupnost bez vlastního hostování	0
Celkem	**8**

Tabulka 2.9: Bodové ohodnocení Pelias API

2.4.7 Nominatim

Nominatim je Open-source nástroj, který využívá data z Open Street Map [17]. Nevyžaduje žádné vytváření uživatelských účtů ani klíčů. Zakládá se na férovém využívání stanoveném podmínkami použití [24]. Nominatim je základní možností, kterou využívá Geocoder gem pro Ruby. Při uvedení zdroje dat ve výsledné aplikaci, identifikaci při dotazech v hlavičce a dodržení kvóty na maximálně 1 dotaz za vteřinu, se naše požadavky bez problémů vejdou do pravidel využívání.

Parametr	Body
Jednoduchost použití v jazyce Ruby	3
Cena	3
Možnost kódování většího množství dat naráz	2
Dostupnost bez vlastního hostování	1
Celkem	**9**

Tabulka 2.10: Bodové ohodnocení API Nominatim

2.4.8 Vyhodnocení

Jako nejvhodnější se ukázalo API Nominatim. Nevyžaduje žádný API klíč a náš omezený záběr aplikace splňuje stanovené požadavky na využití. Není také nutné jej samostatně hostovat, což programátorovi značně ulehčuje práci. Výsledky analýzy (bez výsledků za možnost kódování většího množství dat současně, jelikož všechna API dosáhla stejných bodů) jsou shrnuty v tabulce 2.11.

API	Ruby	Cena	Dostupnost	**Celkem**
Google API	2	1	1	**4**
MapTiler	2	2	1	**5**
Geoapify	2	2	1	**5**
Pelias	3	3	0	**6**
Nominatim	3	3	1	**7**

Tabulka 2.11: Výsledná tabulka hodnotící API pro geokódování

Kapitola 3

Návrh

Kapitola se zabývá návrhem backendového aplikačního rozhraní a návrhem uživatelského rozhraní aplikace. Po specifikaci cílové skupiny jsou definovány funkční a nefunkční požadavky aplikace i API. V rámci této kapitoly jsou také určeny typické případy užití, rozkresleny wireframy pro návrh uživatelského rozhraní a navržena je i architektura.

3.1 Specifikace cílové skupiny

Pro potřeby aplikace je nezbytné vymezit si potenciální skupiny uživatelů, jelikož od nich se bude odvíjet požadovaná funkcionalita aplikace. Stanovíme tedy 3 persony, které vytvoříme na základě uživatelů již existujících analogických řešení. Představují fiktivní uživatele aplikace, kteří mají svoji vlastní osobnost, potřeby, vlastnosti, specifické přístupy a názory. Takový podrobný popis usnadňuje všem vcítit se do uživatelů a zajistit, aby práce odpovídala příběhu.

- Ekolog, stěhující se často po Praze – osoba, na kterou je aplikace cílená
- Průměrný Pražan – možný uživatel aplikace
- Starší člověk s bez zájmu o ekologii – někdo, na koho aplikace rozhodně necílí

3.1.1 Ekolog

- **Věk:** 24 let
- **Pohlaví:** Muž
- **Koníčky:** ekologie, sporty (běh, posilování), kryptoměny, příroda

- **Typický den:** Andrej vstává kolem 9 hodin ráno, pokud nutně nemusí dřív. Pokud ten den má školu, tak jede do školy, většinou jen na cvičení. Pokud školu nemá, jede do práce, kde dělá jako brigádník pro neziskovou organizaci v oblasti managementu. Ve svém volném čase se věnuje sportům, výletům do přírody a občas hraní PC her. Občas si s kamarády v pátek večer vyrazí na pivo nebo do klubu.

- **O jeho životě:** Andrej rád cestuje a navštěvuje své kamarády, většinou si vyhradí čas po zkouškách a odjede na týden nebo dva někam do ciziny. Vystudoval gymnázium a tento rok bude ukončovat Vysokou školu ekonomickou (VŠE) se zaměřením na podnikový management. Trochu se zajímá o technologie a je tedy zvyklý vše, co jde, řešit přes svůj chytrý telefon. Každé léto se vrací domů za rodinou na Moravu, aby si v Praze nemusel platit byt. Každé září si tedy hledá nový, vždy se na toto dobrodružství těší. Trvá na tom, že odpadky, které vyhazuje on, nebo lidé z jeho okolí jsou řádně vytříděné. Často ale některé druhy odpadu nosí u sebe, dokud nenarazí na vhodný koš.

3.1.2 Průměrný Pražan

- **Věk:** 30 let

- **Pohlaví:** Muž

- **Koníčky:** rodina, lukostřelba, turistika

- **Typický den:** Bořek je otcem 2 dcer, Anny a Barbory, které navštěvují základní školu. Vstává obvykle kolem půl sedmé. Dětem připraví snídani a odvede je do školy. Následně míří rovnou do zaměstnání. Pracuje jako recepční IT firmě a rád má ve všem pořádek. Spolu s rodinou žije v Praze a téměř všude chodí pěšky, má tedy rád pohodlné outdoorové oblečení. Většinu svého volného času věnuje rodině: manželce Blance a dětem. Jejich velkou společnou zálibou je turistika, a to především po českých končinách. Když Bořek zrovna není s rodinou, rád se potká se spolužáky z gymnázia, typicky u piva, případně v lukostřeleckém klubu, který si společně založili.

- **O jeho životě:** Vystudoval osmileté gymnázium se zaměřením na jazyky, kde se seznámil i se svou manželkou. Následně chtěl pokračovat ve studiu marketingu na UK, nicméně se mu nepodařilo uspět u přijímacích zkoušek. V práci je zvyklý používat standardní kancelářské nástroje jako je emailový klient a MS Office. Lpějící na pořádek a řád používá také Google kalendář, a to i ve svém chytrém telefonu. Jeho manželka je také velká ekoložka a Bořek je nucen pravidelně vynášet domácí koše, které se dělí na 6 druhů. Bez ní třídění odpadu neřeší.

3.1.3 Starší člověk bez zájmu o ekologii

- **Věk:** 62 let
- **Pohlaví:** Žena
- **Koníčky:** Křížovky, zahrádkaření, procházky
- **Typický den:** Cecilie vstává v šest hodin ráno, aby o hodinu později mohla být v Dukovanské elektrárně, kde pracuje posledních 20 let. Po konci směny se vrátí domů, kde si vychutná kávu a svou oblíbenou odpolední osmisměrku. Zbytek dne stráví sledování televizních hitparád z minulého století, ke kterým si vždy otevře alespoň 1 pivo. O víkendech ji často navštěvují vnoučata.
- **O jejím životě:** Cecilie je hodná babička, ale je velmi konzervativní. Ve svém životě už toho spoustu zažila a má ráda když má všechno pevný řád. Posledních 20 let pracuje jako sekretářka v elektrárně. Svoji práci vykonává pečlivě, ale moc si nerozumí s novými technologiemi. Odpady netřídí, nedělala to její maminka, ani babička, tak k tomu nevidí důvod. Když uklízí ve velkém, odpad vozí na cestu za les na kraji vesnice.

3.2 Funkční požadavky aplikace

- **F1 – Mapa kontejnerů**: Aplikace bude schopna ukázat uživateli na mapě všechny kontejnery na tříděný odpad na území hlavního města Prahy.
- **F2 – Filtrování**: V aplikaci bude možné filtrovat typy kontejnerů k zobrazení podle odpadu, který se do nich vyhazuje.
- **F3 – Nejbližší kontejner**: Aplikace bude umět ukázat nejbližší místo s kontejnery pro typ odpadu, který si vyberu.
- **F4 – Vyhledávání míst**: Aplikace dokáže vyhledat kontejnery na mapě podle adresy, na které se nachází.
- **F5 – Detail**: V aplikaci bude možné zobrazit detail místa s kontejnery, kde se zobrazí, které odpady lze na tomto místě třídit.
- **F6 – Vycentrování na uživatele**: Při pohybu na mapě bude možné vycentrování pozice na aktuální polohu uživatele.
- **F7 – Navigace**: Aplikace poskytne možnost navigovat uživatele k vybranému místu s kontejnery.
- **F8 – Hodinky**: Aplikace pro hodinky umožní zobrazit nejbližší stanoviště pro tříděný odpad dle vzdálenosti.

3.3 Nefunkční požadavky aplikace

- **NF1 – Podpora**: Aplikace bude podporovat telefony a tablety s Android os verze 7.1 a výše.
- **NF2 – Zařízení**: Aplikace bude podporovat chytré hodinky s operačním systémem Wear OS od verze Wear 2.0

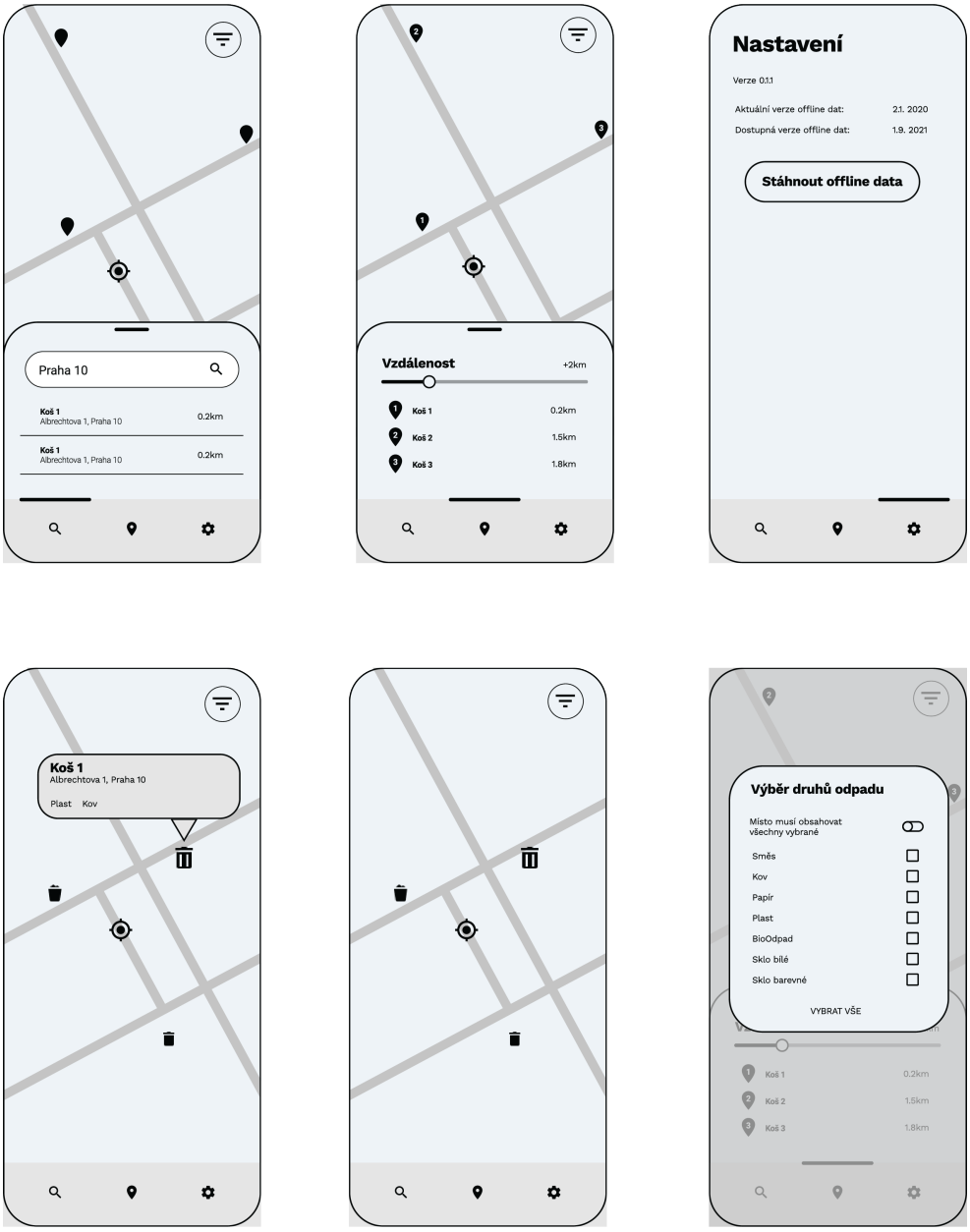
3.4 Typické případy užití

Případy užití (z angl. use-cases) popisují chování systému z pohledu uživatele. Usnadňují komunikaci mezi zadavatelem a vývojářem produktu. Nejsou zde zahrnuty technické a implementační detaily, definují pouze to, co má daná aplikace umět. Případy užití aplikace jsou popsány níže:

- **UC1** – Vyhledání nejbližšího místa pro více druhů tříděného odpadu
  - Uživateli se doma naplnily odpadkové koše s tříděným odpadem a chce zjistit, kde nejblíž může všechny uložit.
- **UC2** – Kontrola dostupnosti tříděného odpadu v oblasti destinace
  - Uživatel se chystá na vyklízení starého bytu, na místě, kde aktuálně nebydlí, a chce si zkontrolovat, kde se v okolí nachází koše na tříděný odpad.
- **UC3** – Ověření dostupnosti koše na konkrétní druh tříděného opadu na známé destinaci.
  - Uživatel pravidelně chodí okolo místa, kam by chtěl jít následující den vyhodit odpad, není si však jistý, zda je na daném místě i koš na konkrétní druh odpadu.
- **UC4** – Navigace k nalezenému koši
  - Uživatel si v aplikaci našel koš, kde chce vyhodit odpad, chce být k tomuto místu navigován.

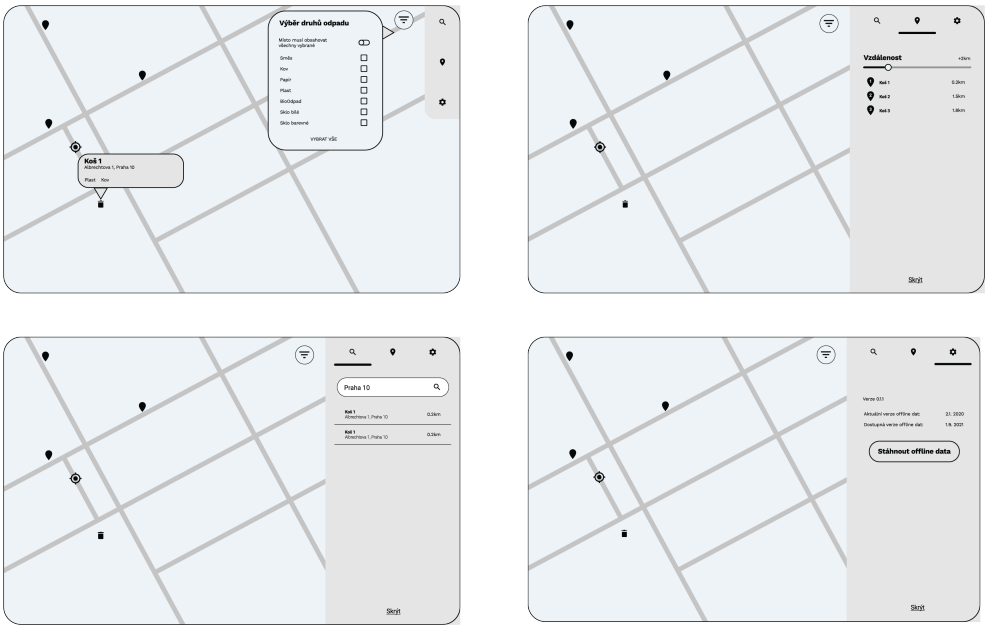
3.5 Wireframy

Na základě popisu uživatelů, funkčních a nefunkčních požadavků a typických případů užití byly vytvořeny wireframy, někdy také počeštěné na „drátové modely“. Ukazují 3 verze aplikace. Verzi pro telefon, tablet a chytré hodinky. Pro tento prvotní návrh byla využita webová aplikace Figma. Všechny 3 verze jsou vyobrazeny na obrázcích: 3.1, 3.2 a 3.3.

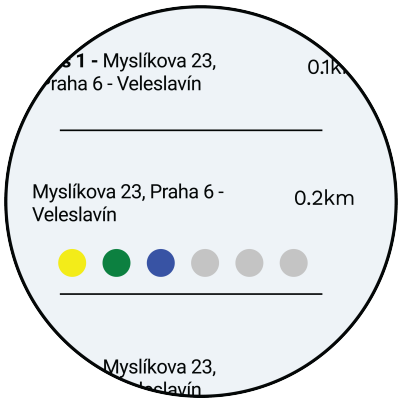


Obrázek 3.1: Návrh wireframů telefonu

Pro telefony a tablety se aplikace skládá z obrazovky mapy, kde může uživatel filtrovat a vyhledávat koše, vyhledávat nejbližší nebo se k nim nechat navigovat. Na hodinkách je zobrazen seznam nejbližších košů.



Obrázek 3.2: Návrh wireframů pro tablet



Obrázek 3.3: Návrh wireframů pro chytré hodinky

3.6 Funkční požadavky API

Funkční a nefunkční požadavky pro API jsou stanoveny na základě definovaných požadavků pro mobilní aplikaci.

- **FB1 – Vyhledávání dle jména**: API poskytne endpoint k vyhledání v databázi podle adresy.
- **FB2 – Všechny informace**: API poskytne endpoint k vrácení všech informací v databázi pro offline použití.
- **FB3 – Nejbližší kontejnery**: API poskytne endpoint k vyhledání nejbližších kontejnerů dle aktuální lokace.
- **FB4 – Filtrování**: Ve všech endpointech, které bude API poskytovat bude možné filtrovat výsledky dle typů kontejnerů.

3.7 Nefunkční požadavky API

- **NFB1 – Formát**: API bude vracet informace ve formátu JSON
- **NFB2 – Aktualizace dat**: API bude implementovat cachovací mechanismus ke kontrole aktuálnosti dat.

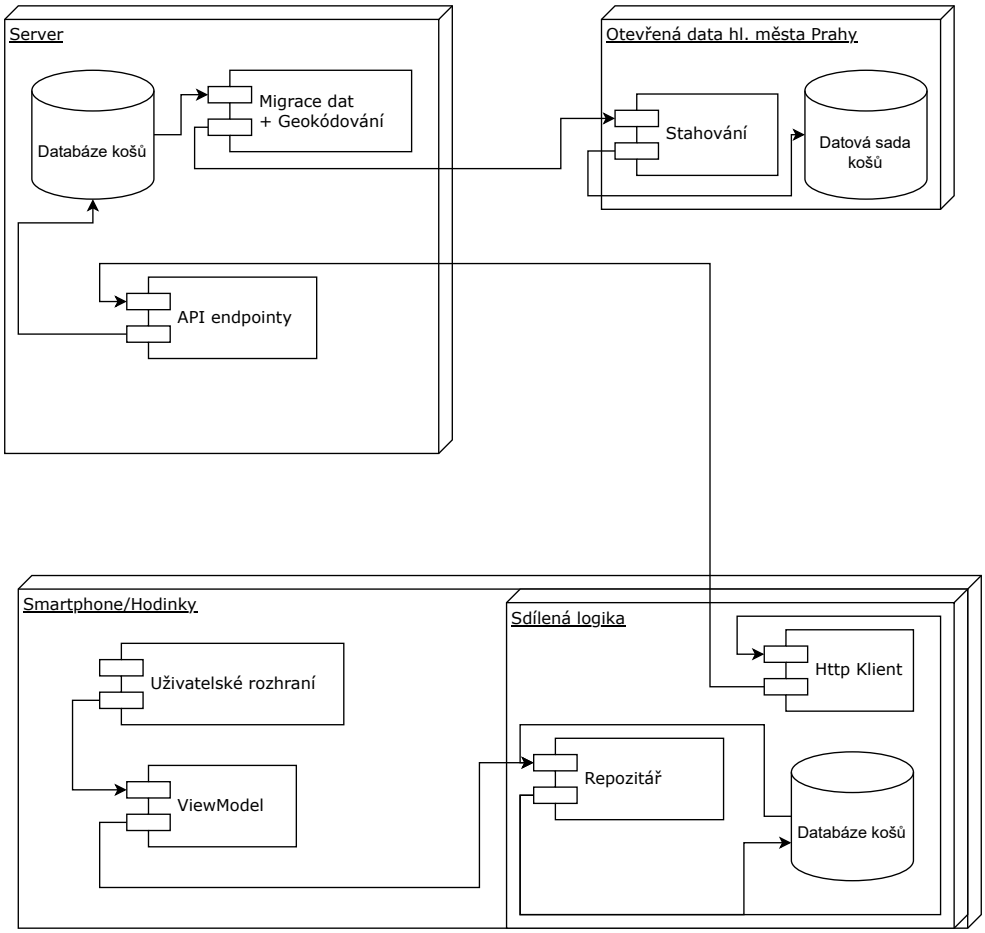
3.8 Návrh architektury

Mobilní aplikace, která bude instalována na uživatelských zařízeních, bude všechna data získávat od backendového API. Část dat si bude ukládat pro následné zpracování, další data získává postupným dotazováním na základě parametrů zvolených uživatelem. API získá svá data transformací datové sady získané ze stránek otevřených dat hl. města Prahy. Tok dat je znázorněn na obrázku 3.4.

3.8.1 Návrh API

Mobilní aplikace komunikují s backendem pomocí REST API. To se skládá pouze z několika endpointů, které jsou graficky znázorněny na diagramu 3.6. Všechny vrací výsledné hodnoty ve formátu JSON. Výsledek dotazu pak může vypadat jako v ukázce 1.

Databáze API obsahuje pouze jedinou tabulku reprezentující všechny koše. Je uložena s využitím PostgreSQL. Každý záznam obsahuje souřadnice, adresu a typy odpadů, které jsou možné v koši vytřídit. Adresa se získává reverzním geokódováním ze souřadnic, což je detailněji popsáno v kapitole 4.2.6. Typy odpadů k třídění jsou získávány slučováním jednotlivých záznamů košů se stejnými souřadnicemi. Mapovány jsou podle tabulky 3.1.



Obrázek 3.4: Tok dat v rámci celého projektu

Druh odpadu	**ID**
Papír	0
Plast	1
Nápojové kartóny	2
Barevné sklo	3
Čiré sklo	4
Kovy	5
Elektroodpad	6

Tabulka 3.1: Tabulka mapování typů odpadu

```
[
  {
    "id": 4851,
    "latitude": 50.07052485100007,
    "longitude": 14.414760675000025,
    "address": Myslíkova 23, Praha 6 - Veleslavín,
    "trash_types": [
      [
        0,
        1
      ]
    ],
    "distance": 0.06538944219147931,
    "bearing": 75.60619791467
  }
]
```


Výpis kódu 1: Ukázka odpovědi na dotaz z API

Tyto transformace dat se provádí nad získanou datovou sadou otevřených dat hl. města Prahy při nasazení aplikace.

Konkrétně rozepsaná podoba endpointů je rozepsána znázorněna na diagramu 3.5.

- `GET /bins` Vrátí seznam košů podle vzdálenosti od nejbližšího po nejvzdálenější
  - ***lat** Zeměpisná šířka uživatele.
  - ***lon** Zeměpisná délka uživatele.
  - **radius** V jakém okolí, chceme zobrazovat výsledky. Zadává se v kilometrech a výchozí hodnota je 5.
  - **filter** Filtr pro typy odpadů. V základu povoluje všechny.
  - **allRequired** Volba, jestli chceme, aby koš nabízel všechny druhy odpadů. Výchozí hodnota je nevyžadovat.
  - **page** Kolikátou stranu chceme.
  - **perPage** Kolik výsledků na stranu chceme.

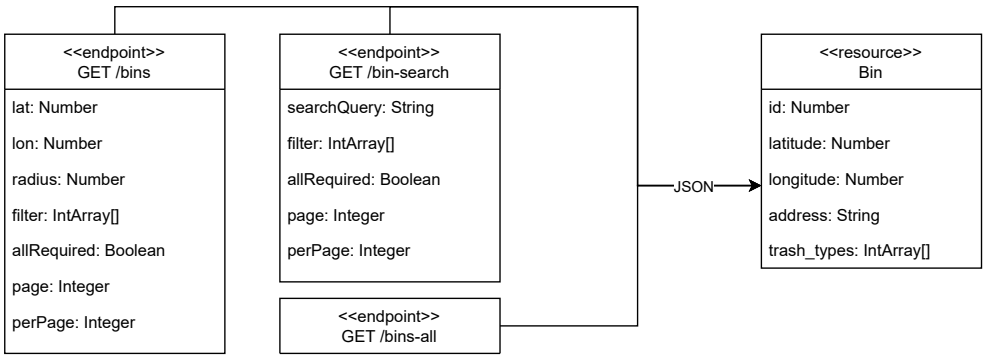
- `GET /bin-search` Vrátí seznam košů na základě podobnosti adresy
  - ***searchQuery** Dotaz pro vyhledávání.
  - **filter** Filtr pro typy odpadů.
  - **allRequired** Volba, jestli chceme, aby koš nabízel všechny druhy odpadů.
  - **page** Kolikátou stranu chceme.
  - **perPage** Kolik výsledků na stranu chceme.
- `GET /bins-all` Vrátí seznam všech košů

3.8.2 Návrh architektury mobilní aplikace

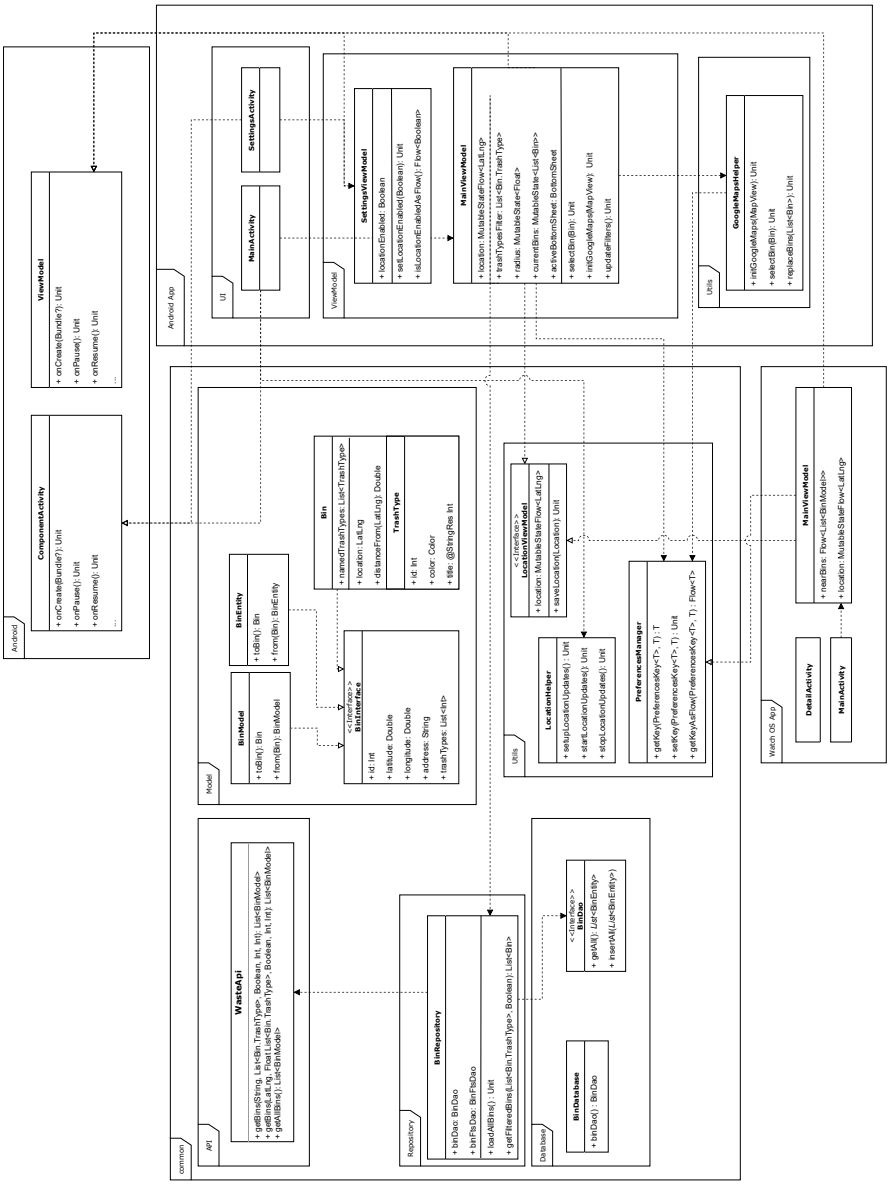
Pro tvorbu Android aplikací je společností Google doporučován architektonický vzor MVVM (Model-View-ViewModel) [31]. Podporuje jej jak množství externích knihoven, tak i základní architektonické prvky knihoven `androidx`. Architektura se skládá ze tří základních vrstev:

- **Model** – Vrstva pro práci s daty, zahrnuje také business logiku.
- **ViewModel** – Zprostředkovává data zpracovaná Modelem pro View.
- **View** – Prezentační vrstva, zahrnuje uživatelské rozhraní.

Jelikož je vzor MVVM běžným standardem a je podporován knihovnami přímo od společnosti Google, je při implementaci aplikace využit.



Obrázek 3.5: Diagram api endpointů



Obrázek 3.6: Návrh architektury mobilní aplikace

Kapitola 4

Implementace

Kapitola Implementace se soustředí na popis nástrojů použitých při vývoji a uvádí konkrétní příklady použitých technologií a způsoby jejich aplikace. Rozdělena je na implementaci backendového API a Android aplikace.

4.1 Nástroje a technologie použité pro vývoj

4.1.1 Kotlin

Nativní aplikace pro OS Android je dnes možné vyvíjet buď v programovacím jazyce Kotlin nebo Java. V roce 2019 Kotlin nahradil Javu jako oficiálně podporovaný a preferovaný jazyk pro vývoj [30]. Proto byl zvolen jako jazyk pro implementaci Android aplikace. Oproti Javě nebo multiplatformním řešením má následující vlastnosti:

Typový systém Kotlin je staticky typovaný jazyk, což umožňuje odhalit chybné použití typů již v době kompilace. Proměnné a konstanty definujeme klíčovými slovy `var` a `val`. Typ můžeme určit buď explicitně nebo se přiřadí automaticky. Typový systém je navíc rozšířen o nulovatelnost. Pokud nulovatelnost nepovolíme sami, proměnné nelze přiřadit hodnota `null`. Povolit takové přiřazení je možné doplněním otazníku `?` za datový typ.

Asynchronní kód Asynchronní kód se v Kotlinu píše pomocí tzv. *coroutines*. Je to koncept umožňující přerušení a opětovné spuštění aktuálně vykonávaného vlákna. Kód je díky tomu možné psát velmi podobně jako obyčejný kód synchronní. Blokující funkce které mohou trvat dlouhou dobu se označí klíčovým slovem `suspend` a následně mohou být spouštěny jen z *coroutines*. Jejich využíváním se vyhýbáme programováním za využití callbacků.

Rozšiřující funkce Kotlin dovoluje rozšířit funkcionalitu jednotlivých tříd bez nutnosti jejich dědění. Stačí vytvořit funkci nebo proměnnou definovanou

jménem třídy, kterou chceme rozšířit, následovanou tečkou a definicí funkce nebo proměnné. Např. jako `fun Int.timesTwo() = this * 2`. Rozšiřující funkce nám tedy umožňuje doplnit požadovanou funkcionalitu tříd, které dědit není možné. Vše se také obejde bez velkého množství nového kódu.

Datové třídy Pokud třídu definujeme jako datovou, klíčovým slovem `data`, jsou nám automaticky vygenerovány funkce pro porovnávání, gettery, settery nebo funkce pro vypsání takové třídy. Takový zápis eliminuje kód, který by se jinak často opakoval.

4.1.2 Android Studio

Oficiálním nástrojem pro vývoj Android aplikací je Android studio. Nástroj vyvíjí Google společně se společností JetBrains [32]. Umožňuje sestavování kódu, jeho balení do balíčků, pokročilé možnosti pro krokování a inspekci kódu nebo bezdrátové připojení k telefonu pro pohodlnější vývoj.

Integruje také Android Emulátor, který umožňuje spouštění aplikací na virtuálních zařízeních. Tato funkcionalita byla obzvlášť přínosná pro testování aplikace pro chytré hodinky a pro tablety, jelikož taková zařízení nemá pro vývoj dostupná každý.

4.1.3 Ruby

Ruby je interpretovaný, objektově orientovaný, skriptovací jazyk. Běžně se používá na webový vývoj nebo v rámci DevOps. Nejvíce rozšířený je společně s jeho frameworkem Ruby on Rails [33]. Ten je nejčastěji využíván pro psaní webů napojených na databáze. Mezi přednosti Ruby patří rychlost vývoje jednoduchých aplikací a open-source komunita. Jazyk se naopak nehodí pro komplikované výpočetní úkony nebo strojové učení.

4.1.4 RubyMine

Jako nástroj pro vývoj API v jazyce Ruby byl využit program RubyMine. Je vyvíjen společností JetBrains. Umožňuje pohodlné spouštění definovaných testů, spouštění kódu, nahlížení do připojené databáze nebo spouštění rake úkolů [34].

4.1.5 SwaggerHub

SwaggerHub byl využit pro dokumentaci API. Nástroj funguje jako webová aplikace, ze které je možné API sdílet jako URL adresu, či exportovat definici v jednom z mnoha formátů. Nabízí validaci definovaného API a také jeho vizualizaci.

4.2 Ruby API

4.2.1 Gem

V terminologii jazyka Ruby se většinou nepoužívá termín knihovna, nahrazuje se gemy. Gem je knihovnou, která se skládá z kódu napsaného v Ruby, je zde ale přibalena část dat navíc [35]. Většina je hostovaná na webu `rubygems.org`. Definice využitých gemů v projektu se píší do souboru `Gemfile`, ze kterého se při instalaci vygeneruje další soubor `Gemfile.lock`, kde jsou zadefinované verze a závislosti. Tento vygeneruje gem `bundler`, který ulehčuje správu jednotlivých pracovních prostředí. Pro využití gemů při implementaci stačí v souboru, kde s nimi pracujeme, dopsat `require 'požadovaný gem'`.

4.2.2 ActiveRecord

Active Record je architektonický návrhový vzor pro práci s datovými zdroji [36]. Reprezentuje vrstvu systému, která je zodpovědná za business a datovou logiku. Spravuje vytváření a využívání objektů, jejichž data vyžadují uložení v databázi. Implementací je objekt, který nese jak chování, tak data, která mají většinou perzistentní podobu a potřebují být uložena v databázi.

Pro implementaci tohoto návrhového vzoru slouží v jazyce Ruby gem `ActiveRecord`. Model ukládaný v tabulce definujeme v API jako v ukázce 2.

```
class Bin < ActiveRecord::Base
  extend Geocoder::Model::ActiveRecord
  include PgSearch::Model
  # Used for searching by nearest locations
  reverse_geocoded_by :latitude, :longitude

  validates :latitude, numericality: {...}
  validates :longitude, numericality: {...}
  validates :trash_types, presence: true
  validates :address, presence: true
end
```

Výpis kódu 2: Ukázka definice modelu v ActiveRecord

Nad takto definovaným modelem ActiveRecord následně vygeneruje definici tabulky a umožňuje nad modelem volat operace hledání, zápisu nebo jiných (i RAW) SQL operací.

4.2.3 Rake

Gem Rake slouží v jazyce Ruby pro spouštění úkolů. Úkolem se zde může rozumět spuštění testů, migrace dat do databáze či vytvoření statistik. V našem API slouží pro manuální aktualizaci dat v databázi, vytvoření databázových tabulek a migraci dat do tabulek (tzv. *seedování*). Ukázka základního úkolu je uvedena na výpisu kódu 3.

```
task :load_config do
    require './app'
  end
```

Výpis kódu 3: Ukázka definice rake úkolu

4.2.4 Databáze

Jako databáze je pro API využit PostgreSQL, což je open-source, objektově-relační databázový systém. Zajišťuje ACID vlastnosti a umožňuje fulltextové vyhledávání. Pro jazyk Ruby je dostupný skrze gem `pg`. Pro naše potřeby je rozšířen o gemy `pg_search` a `pg_trgm`. Oba jsou využity pro fulltextové vyhledávání. V modelu se definují sloupce, nad kterými bude fulltextové vyhledávání fungovat. Druhý z dříve jmenovaných gemů nám umožňuje vyhledávat pomocí trigramů. Takové vyhledávání funguje na základě počtu třípísmenných posloupností znaků původního slova/věty. Čím více shod těchto posloupností máme mezi vyhledávaným textem a záznamem v databázi, tím je výsledek hodnocen lépe. Toto vyhledávání je využito pro shodu mezi zadaným dotazem adresami v databázi.

4.2.5 Další využité gemy

`Kaminari` je gem, který usnadňuje implementaci stránkování. K modelu stačí před odesláním výsledků přidat metody `per` a `page`, které zajistí navrácenou konkrétní stránku při počtu výsledků na stránku.

Gem `json` umožňuje vracet výsledky jednoduše ve formátu JSON.

`Shotgun` byl využit při vývoji API. Díky němu je jednoduché za provozu nasazovat nové verze.

Gem `Geocoder` byl využit pro reverzní geokódování míst, na kterých se koše nacházejí. Nabízí velké množství API, ze kterých lze získávat výsledky a široké možnosti konfigurace.

Pro automatické testování aplikace byl využit gem `rspec`. Automaticky je testován návratový typ endpointů a jejich parametry (povinné a základně nastavené hodnoty). Příklad automatického testu je na ukázce 4.

```
it "search has required params" do
  get '/bins-search'
  expect(last_response).not_to be_ok
end
```

Výpis kódu 4: Ukázka definice rspec testu

4.2.6 Geokódování

Geokódování existují dva druhy. Dopředné, kde hledáme zeměpisnou pozici (zeměpisnou šířku a délku) dle zadaného řetězce s adresou a zpětné, které nám umožňuje na základě zadané souřadnice najít zeměpisné objekty s různou granularitou. Aplikace bude umožňovat vyhledávání stanovišť s kontejnery na základě adresy, v datasetu však máme dostupné pouze souřadnice. Využijeme tedy geokódování reverzní.

K zakódování souřadnic dochází při transformaci dat ze zdrojového souboru od hl. města Prahy do databáze. Využíván je pro to gem `geocoder` [12]. Ten má možnost využívat několik různých api, která samotné kódování provádí. Jediná neplacená možnost, kterou není potřeba samostatně hostovat je API `Nominatim` [17]. Omezení, která uvádí pro naše využití (jednorázové geokódování většího množství dat), jsou pouze:

- Nanejvýš 1 dotaz za sekundu
- Identifikace aplikace pomocí hlavičky **User-Agent**, nebo **HTTP Referer**
- Jasné uvedení atribuce
- Dotazy odesílané pouze jedním vláknem
- Dotazy musí provádět pouze jeden počítač
- Ukládání výsledků do mezipaměti pro zamezení stejných opakovaných dotazů

4.2.7 Nasazení aplikace na platformu Heroku

Heroku je služba nabízená jako platforma založená na kontejnerech. Umožňuje nasazování, správu a škálování aplikací [20]. Ty pak mají unikátní doménu, která je využívána ke směrování http dotazů správným kontejnerům. Každý aplikační kontejner, neboli dyno, je rozprostřen v dyno mřížce, která se skládá z několika serverů. Tato platforma je v rámci projektu využita pro nasazení API.

Podporuje programovací jazyky Ruby, Java, PHP, Python, Node, Go, Scala a Clojure. Nasazení aplikace spočívá v nahrání do Heroku pomocí Gitu.

Následně je aplikace sestavena a zveřejněna. Skrze administraci je možné monitorování využívaných prostředků, změny v nastavení, nebo dokoupení některých zpoplatněných služeb.

Pro nasazení je potřeba Heroku účet a v terminálu nainstalované rozhraní `Heroku Command Line Interface`. Aplikace nemůže jako databázi využívat Sqlite, musí mít kompatibilní verzi Ruby a definovaný způsob napojení na využitou databázi. Soubor 'Procfile' umístěný v kořenovém adresáři aplikace následně definuje, jak má Heroku aplikaci spustit.

Příkazy `heroku create` a `git push heroku main` je nasazena aplikace. Otevřít lze příkazem `heroku open`. V některých případech je ještě nutné spustit migrace či inicializace dat databáze. Aktualizace a vydávání nových verzí je jednoduše prováděno pomocí aktualizace zdrojových souborů skrze Git.

4.3 Android aplikace

Oproti návrhu z diagramu 3.6 se implementace neliší, je však důležité zmínit některé implementační detaily. Pro seznamy nejbližších košů a výsledky vyhledávání košů je využito rozhraní `PagingSource`, které usnadňuje načítání dat ze vzdálených stránkovaných zdrojů do seznamů v uživatelském rozhraní. Modely nemohly být spojeny skrze jednotné rozhraní pro konflikty s databázovou knihovnou. Funkce převádějící jeden model na druhý však zajišťují, že při změně jednoho z nich bude programátor upozorněn tím, že se aplikace nesestaví. Nový diagram 4.9 je doplněn o tato implementační specifika. Nachází se na konci této kapitoly.

4.3.1 Uživatelské rozhraní

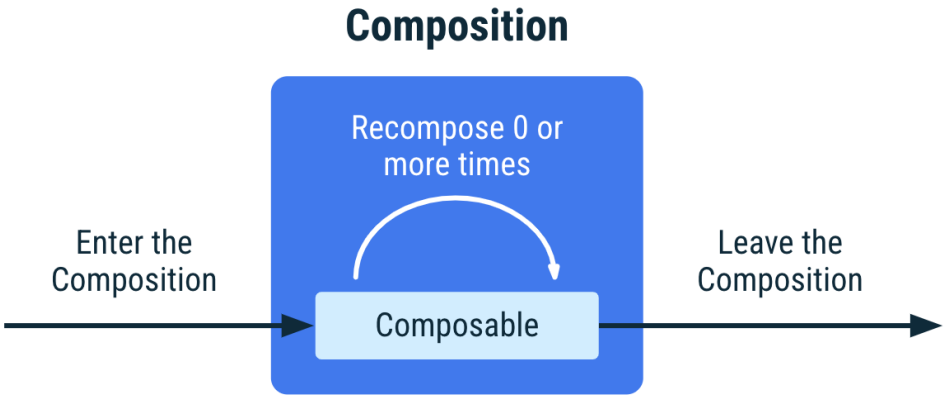
V době psaní této práce jsou dostupné a podporované 2 způsoby tvorby uživatelského rozhraní. Jetpack compose a tvorba rozhraní v XML. V této práci je využita první možnost.

Jetpack compose je sada nástrojů pro tvorbu nativního uživatelského rozhraní pro aplikace běžící na OS Android. První stabilní verze vyšla v květnu 2021 a je vytvořena jako náhrada pro dlouho využívaný přístup využívající UI obrazovek, psaných imperativně v jazyce XML [13]. Oproti tomu se veškerá rozvržení obrazovek v Compose píší deklarativně v Kotlinu. To dělá kód čitelnějším, je jej potřeba psát méně a výsledkem je velká redukce velikosti aplikace a rychlejší časy potřebné pro její sestavení.

Compose si udržuje vlastní životní cyklus kompozitů nad aktivitou, ve které je spuštěn. Skládá se z kompozice a následných rekompozic, což je znázorněno na obrázku 4.1. Kompozice proběhne pouze jednou při vykreslení kompozitu na základě změn stavů, které si drží následně probíhají rekompozice.

Jetpack Compose má i vlastní specifické balíčky pro tvorbu UI pro aplikace pro hodinky. Využity jsou i některé součásti, které jsou prozatím označené jako

experimentální. Znamená to, že v budoucích aktualizacích může být jejich funkcionalita změněna nebo odstraněna.



Obrázek 4.1: Životní cyklus kompozitů [13]

Pro realizaci jednotlivých komponent a sekcí aplikace byly použity některé zajímavé kompozity, které Jetpack Compose a na něj navázané knihovny poskytuje. Jsou rozepsány v následujícím odstavci:

- `Box`, `Column`, `Row` – základní rozložení udávající chování obsahu uvnitř. `Box` je ekvivalentem `FrameLayout`, všechen obsah se skládá přes sebe. `Column` funguje jako `LinearLayout` s vertikální orientací, `Row` potom s horizontální.

- `LazyColumn`, `LazyRow`, `ScalingLazyColumn` – jedno z velkých lákadel pro Jetpack Compose. Nahrazují `RecyclerView` – seznam, který ale nepotřebuje žádné zvláštní adaptéry. Snadno se nastaví chybové či načítací hlášky a funguje velmi dobře i s Android knihovnou Paging 3, která je v aplikaci využita. `ScalingLazyColumn` je verze určená pro hodinky. V momentě, kdy položka seznamu přestává být viditelná, zmizí animací pro uživatelské rozhraní hodinek přirozenou.

- `FlowRow` – rozšíření `LazyRow`, které při dosažení maximální šířky automaticky pokračuje na dalším řádku.

- `Card`, `Chip` – základní komponenty uživatelského rozhraní pro telefony a hodinky se zaoblenými rohy. Reprezentují jednotlivé koše.

- `AndroidView` – zpřístupnění implementací původních rozložení pro Compose. V mobilní aplikaci je využit pro implementaci Google mapy.

- `Scaffold` – definuje rozložení obrazovky, kde nabízí místa pro obsah, horní lištu aplikace, navigační lištu a další.

- `BottomSheetScaffold` – rozšíření `Scaffold` pro využití spodních vysouvacích listů.
- `NavigationBar`, `NavigationRail` – komponenty sloužící jako hostitelé pro jednotlivé části navigace v aplikaci. `NavigationBar` nabízí navigaci ve spodní části obrazovky, `NavigationRail` potom na boku, což je vhodnější pro větší displeje při zobrazení na šířku.

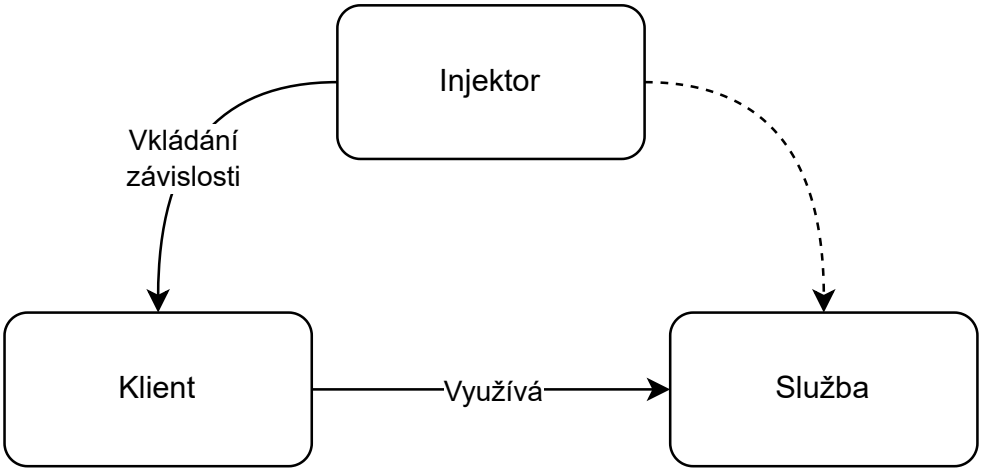
4.3.2 Mapy

Využití map Jetpack compose neusnadní. Knihovny, které přináší mapy, jako komponenty Jetpack compose, jsou v době psaní této práce teprve v rané fázi vývoje a nepřináší veškerou požadovanou funkcionalitu. Chybí například shlukování bodů na mapě do klastrů [37].

Compose přináší možnost využít klasické typy UI obrazovek používané původně v XML jako jeho komponenty. Funkční řešení, které s touto možností spolupracuje, jsou Google mapy. Objekt mapy je potřeba napojit na životní cyklus a následně dát pozor na zbytečné překreslování samotné mapy v rámci rekompozic.

4.3.3 Dependency injection

Dependency injection je architektonický vzor. V zásadě je to technika pro vkládání závislostí mezi jednotlivými komponentami a službami programu. Jedna komponenta tak může používat druhou, aniž by na ni měla v době sestavování programu referenci 4.2. Tato technika je pro programování Android aplikací široce rozšířená, proto její implementaci nabízí mnoho knihoven [18].



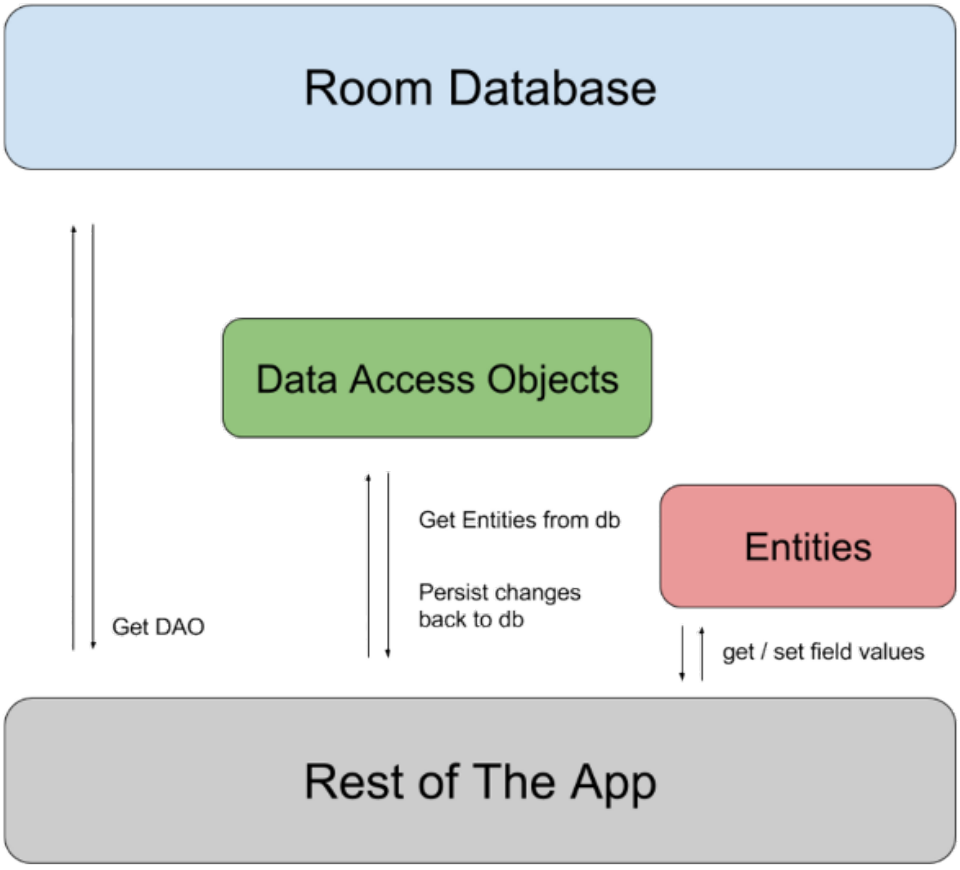
Obrázek 4.2: Diagram vkládání závislosti / dependency injection

Sám Google spravuje knihovny Hilt a Dagger, které tak nabízí jako doporučené. Ty fungují na principu generovaného kódu podle anotací. Alterna-

tivou může být Koin, což je doménově specifický jazyk napsaný kompletně v jazyce Kotlin. Obě možnosti podporují vkládání závislostí v komponentech Jetpack compose. Avšak v případě knihovny Hilt je zde tato funkcionalita omezena na ViewModely. Z tohoto důvodu je v projektu využita knihovna Koin.

4.3.4 Databáze

Android aplikace umí nativně pracovat s SQLite databází. Pro usnadnění práce s touto databází slouží knihovna Room, kterou vyvíjí sám Google. Poskytuje abstrakční vrstvu nad SQLite, která umožňuje plynulý přístup k databázi. U dotazů nad SQL databází je ověřena jejich správnost v čase kompilace, funguje na principu anotací, ze kterých se následně generuje kód a usnadňuje migrace mezi jednotlivými verzemi databází [19]. Architektura databáze je znázorně na na obrázku 4.3.



Obrázek 4.3: Diagram architektury knihovny Room [19]

Pro ukládání jednoduchých dat ve formátu klíč-hodnota je využíván Preferences DataStore. Umožňuje ukládat data asynchronně, konzistentně a pomocí transakcí s využitím kotlin coroutines a Flow. Pro ukládání složitějších objektu/datových typů by bylo možně využít Proto DataStore. Ten následně poskytuje typovou bezpečnost.

4.3.5 Síťová komunikace

Pro komunikaci s internetem nemá Google pod svou správou žádnou z knihoven/frameworků. Využijeme proto framework Ktor spravovaný vývojáři jazyka Kotlin. Usnadní nám parsování výsledků dotazů ve formátu JSON, loggování, nebo využívaní HTTPS protokolu. Tato rozšířená funkcionalita se do přidává instalováním pluginů. Těmi konfigurujeme základní `HttpClient`. Po nakonfigurování takového klienta je možné nad ním volat jednoduché dotazy k API. Příklad dotazu na ukázce kódu 5.

```
private val ktorClient = HttpClient(Android) {
    defaultRequest {
        host = "prague-waste-api.herokuapp.com"
        url {
            protocol = URLProtocol.HTTPS
        }
    }

...

suspend fun getBins(
    query: String,
    filter: List<Bin.TrashType>? = null,
    allRequired: Boolean? = null,
    page: Int? = null,
    perPage: Int? = null
) = ktorClient.get(path = "bins-search") {
    parameter("searchQuery", query)
    parameter("filter", filter?.map { it.id })
    parameter("allRequired", allRequired)
    parameter("page", page)
    parameter("perPage", perPage)
}
```

Výpis kódu 5: Ukázka dotazu na API pomocí knihovny Ktor

4.3.6 Modely

Jelikož aplikace pracuje jak s online daty z API, které přicházejí ve formátu JSON, tak s lokální databází založenou na SQLite, je vhodné oddělit modely, které s entity košů pracují. Mimo zmíněné 2 je ještě potřeba implementovat typ třetí pro použití v datové vrstvě. Tato se bere jako základní, na kterou je možné obě předchozí převádět.

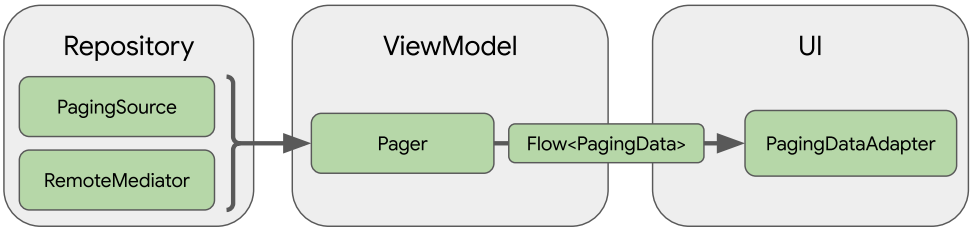
Toto oddělení sice vede k částečnému vzniku duplicitního kódu, získanou výhodou je následně separace závislostí jednotlivých zdrojů dat. Při změně jednoho modelu je programátor díky vzájemným konverzním funkcím vždy upozorněn na nutnost upravit i modely další.

Rozdíly mezi jednotlivými modely popisuje následující přehled:

- **Doménová entita** – Reprezentuje entitu z „lidského“ pohledu návrhu. Využívá se také jako entita, která je zobrazována v rámci mapy - implementuje rozhraní `ClusterItem`. Obsahuje také výčet typů, které reprezentují typy tříděného odpadu pro snadnější práci s nimi. Tyto ve zbývajících entitách nepotřebujeme.
- **API entita** – Reprezentuje data získávaná z API. Tato entita je serializovatelná, což umožňuje její kódovaní a dekódování na primitivní typy, které je možné předávat po síti nebo mezi objekty.
- **Databázová entita** – Slouží k práci s daty pocházejícími z lokální SQLite relační databáze. Reprezentuje její tabulku a sloupce. Obsahuje pouze datové typy, se kterými umí databáze pracovat.

4.3.7 Další knihovny

Mezi další zajímavé knihovny, které projekt využívá patří Accompanist, která řeší některé nedostatky Jetpack Compose, knihovna Androidx Paging, která usnadňuje načítání a cachování výsledků získávaných z dotazů z internetu, které se zobrazují v seznamech (`RecyclerView`, `LazyColumn`...) [27]. V rámci datové vrstvy se vytvoří tzv. `PagingSource`, který definuje zdroj dat a způsob, jakým je získávat. Ve ViewModelu se nachází komponenta, která napojuje zdroj dat s uživatelským rozhraním. Při využití Jetpack Compose pro tvorbu tohoto rozhraní, data z ViewModelu přijmeme jako `LazyPagingItems`, které následně můžeme načítat do seznamů. Napojení je znázorněno na obrázku 4.4.



Obrázek 4.4: Zavedení komponentů knihovny Paging do architektury aplikace [27]

4.3.8 Google maps API klíč

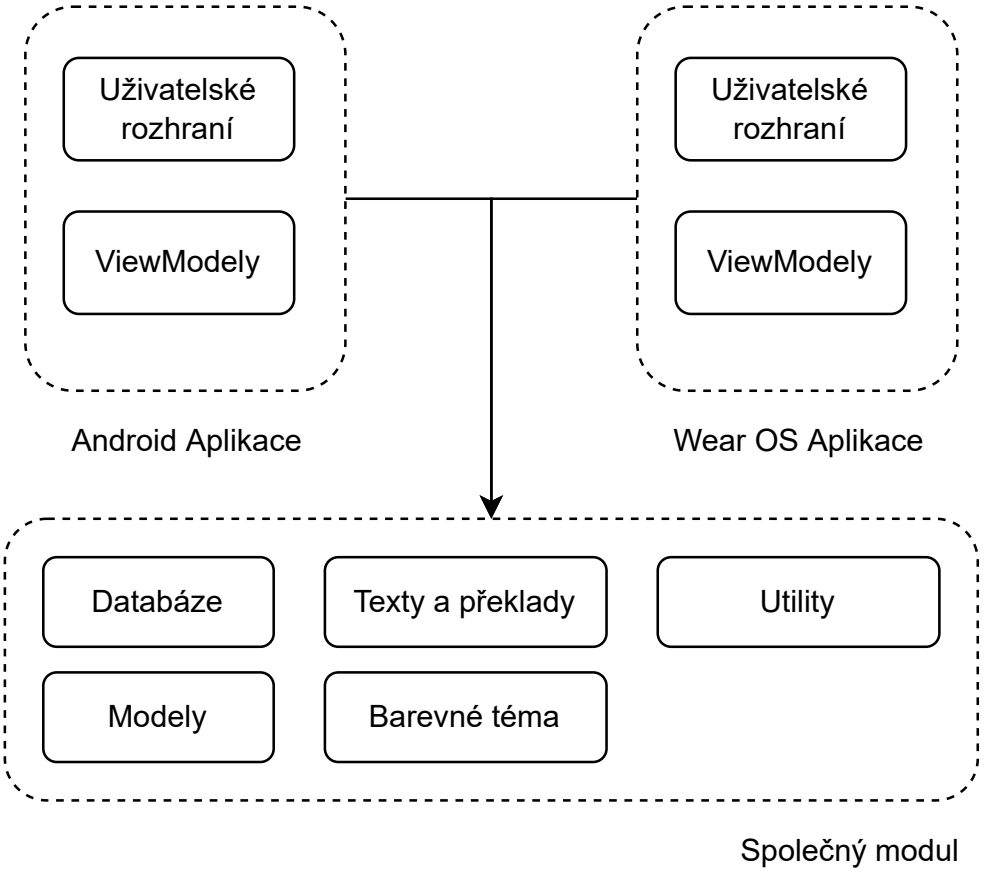
Pro implementaci Google map, které jsou v aplikaci použity je potřeba API klíč. Ten je možné získat pomocí libovolného Google účtu skrze Google maps platform, kde je třeba vytvořit nový projekt a přiřadit mu API klíč. Je důležité omezit jeho použití – v našem případě pouze na Android aplikaci. Po získání se klíč musí přidat do souboru `AndroidManifest.xml` jako meta-data, kde se načítá z `local.properties`, který se nenahrává do verzovacích systémů [38]. Pro lokální spuštění tohoto projektu je tedy potřeba přidat klíč vlastní.

4.3.9 Moduly

Aplikace je pro možnost sdílení kódu mezi aplikací pro telefony a hodinky rozdělena do modulů. Ty zde definujeme jako kolekci zdrojových souborů a možností nastavení, které nám umožňují dělit náš projekt do funkčních celků. Projekty mohou mít jeden nebo více modulů, které na sobě mohou být vzájemně závislé [39]. Pokud na sobě závislé nejsou, je možné je zvlášť testovat či spouštět.

Ve vývojovém nástroji pro tvorbu Android aplikací Android Studio má vývojář možnost vygenerovat několik druhů modulů. Relevantní pro náš projekt jsou moduly aplikací a knihoven. U modulu aplikace můžeme dále vybírat z přednastavených typů pro telefony a tablety, hodinky, chytré televize či chytré brýle. Každá z možností předpřipraví modul pro vývoj aplikace pro požadovaný systém. Modul pro knihovnu je zaměřen na znovu využitelný kód. Při sestavení se nevytváří spustitelný soubor aplikace, ale archiv kódu. Vytvářet můžeme buď Android nebo Java knihovnu. Při vybrání Android knihovny je následně jednoduché předat aplikaci závislost. Ta se pak v kódu uvede jako `implementation project(':Název_sdílené_knihovny')`.

Tento projekt se skládá ze 3 modulů. Je to aplikace pro operační systém Android, aplikace pro Wear OS a knihovna, kde se nachází sdílený kód pro obě aplikace. Na sdílené knihovně jsou obě aplikace závislé. Znázornění spolupráce modulů je zobrazeno na obrázku 4.5.

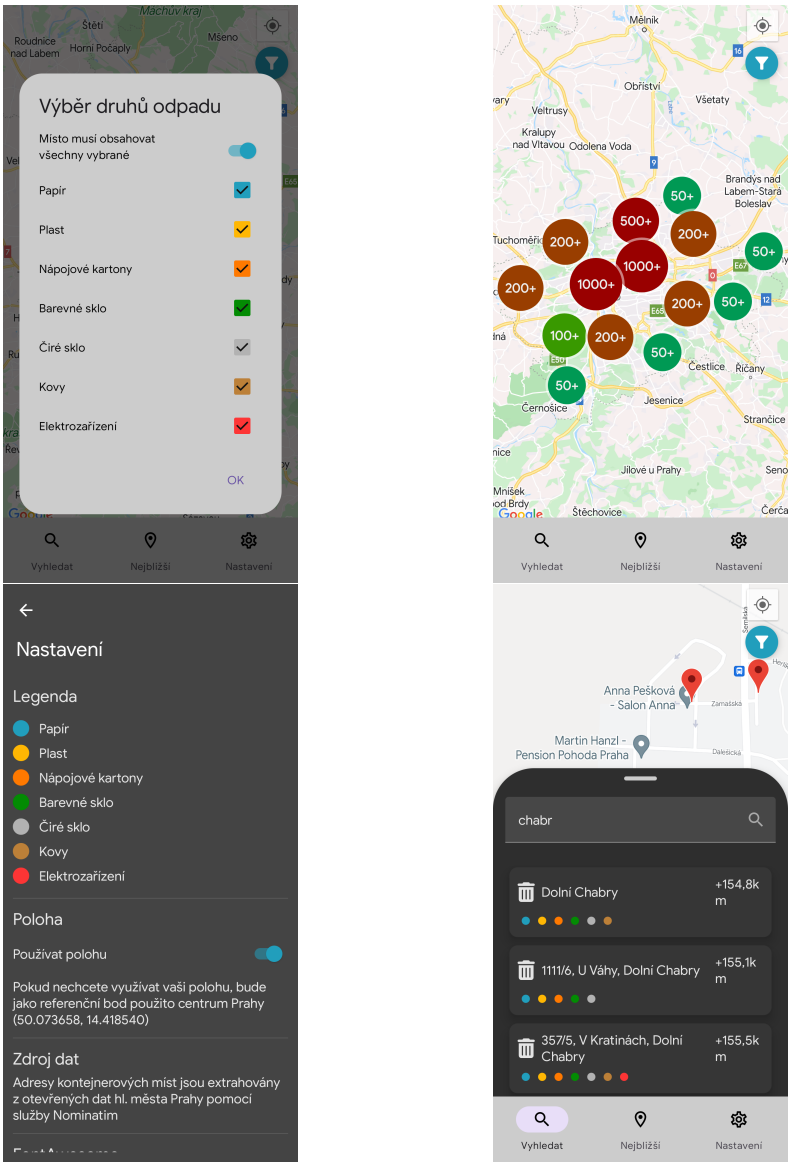


Obrázek 4.5: Znázornění spolupráce modulů v aplikaci

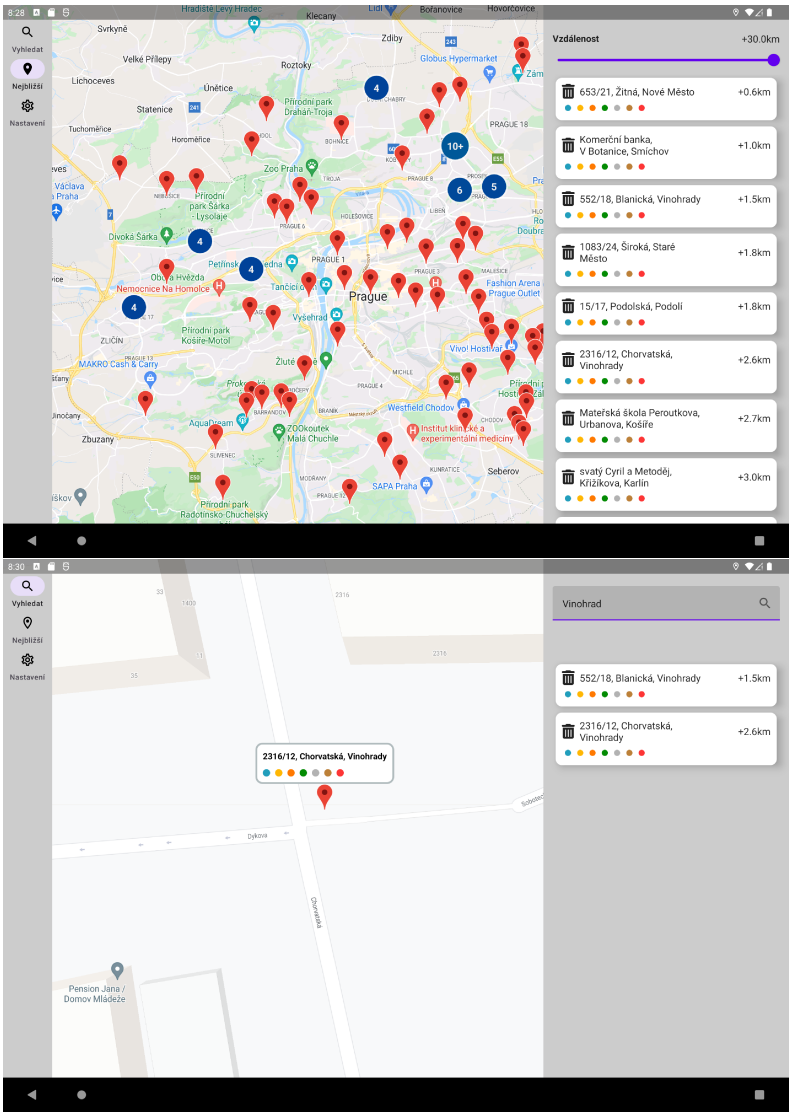
4.3.10 Implementace uživatelského rozhraní

Ve výsledném uživatelském rozhraní byly oproti wireframům na obrázcích 3.1, 3.2 a 3.3 provedeny změny. Nejvíce se změnila navigační lišta pro rozložení na obrazovkách tabletů. Byla přesunuta do levé části jako lišta a z nastavení se stala samostatná obrazovka. Pro hodinky přibyla obrazovka s detailem jednotlivých košů na které se nachází druhy odpadu vypsané slovně a tlačítko, díky kterému je možné spustit navigaci. Uživatelské rozhraní pro mobilní telefony zůstalo zachováno. Výsledné UI je zobrazeno na obrázcích: 4.6, 4.7 a 4.8.

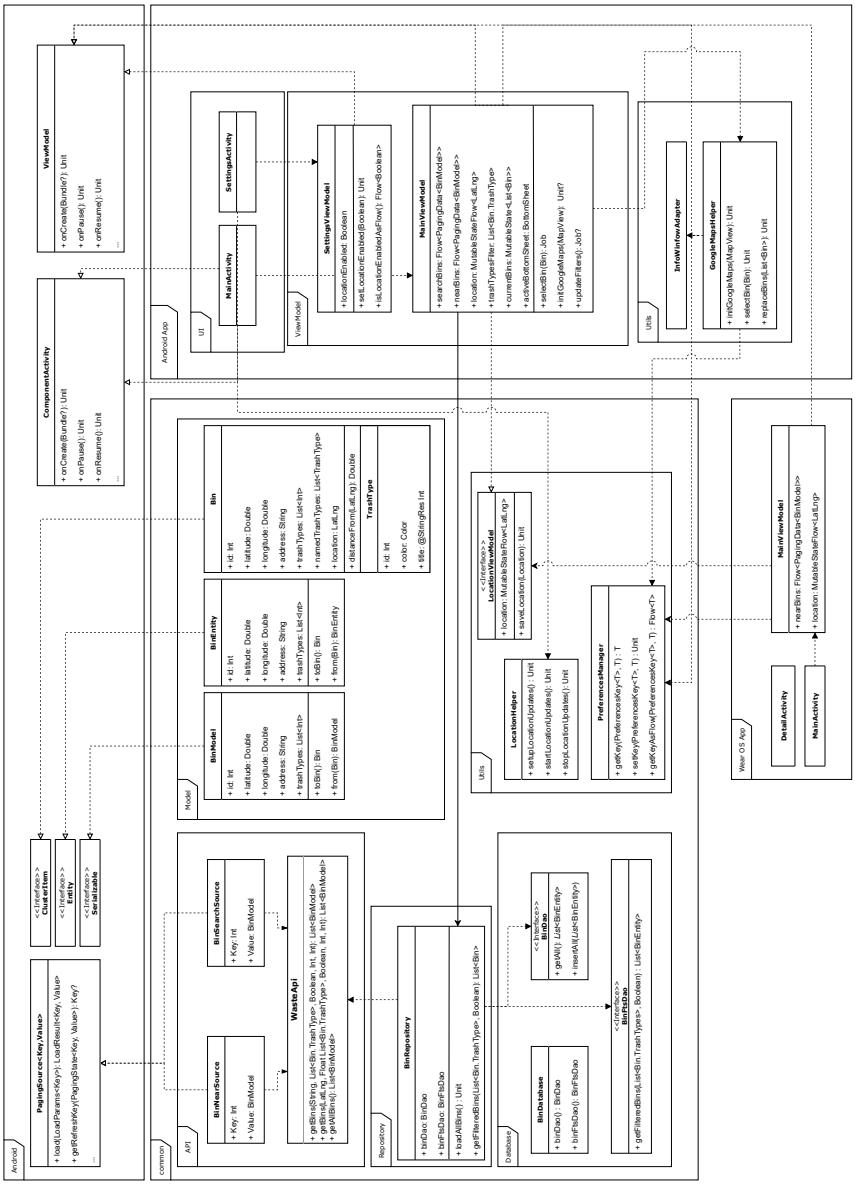
Obrázek 4.6: Finální uživatelské rozhraní pro hodinky



Obrázek 4.7: Finální uživatelské rozhraní pro aplikace na mobilní telefony



Obrázek 4.8: Finální uživatelské rozhraní pro aplikace na tablety



Obrázek 4.9: Finální diagram tříd mobilní aplikace

Kapitola 5

Testování

V rámci této kapitoly je popsána fáze testování. Nejprve je aplikace podrobena heuristické analýze uživatelského rozhraní, následně bylo provedeno uživatelské testování.

5.1 Heuristická analýza

Heuristická analýza patří mezi nejužívanější metody testování použitelnosti interaktivních rozhraní. Spočívá v odhalování chyb a slabých míst v rozhraní za pomocí porovnávání jeho současného stavu s pravidly (heuristikami), která jsou předem daná. Jednou z těchto heuristik je i *Nielsenova heuristická analýza* [26], která definuje deset základních pravidel pro návrh grafického rozhraní.

Cílem heuristické analýzy je co nejrychleji a nejefektivněji nalézt problémy webové stránky, programu či aplikace. Výsledky jsou dostupné téměř okamžitě, na rozdíl od testování s reálnými uživateli.

Pokud není uvedeno jinak, srovnání platí jak pro aplikaci na telefony, tak pro aplikaci pro Wear OS. Následuje porovnání se všemi 10 heuristikami:

1. Viditelnost stavu systému Uživatel by měl vždy být informován o tom, co se děje. Ať už jde o načítání, chybu, či úspěšnou akci, vše by mělo být srozumitelné. Pokud uživatel zná aktuální stav systému, dozví se výsledek svých předchozích interakcí a určí další kroky. Předvídatelné interakce vytvářejí důvěru v produkt i značku.

Realita Aplikace vždy při načítání dat z internetu zobrazí načítací indikátor. Po novém výběru filtrů se změna promítne okamžitě, což je znázorněno animací nad body na mapě. Všechny přepínače i zaškrtávací políčka indikují změnu stavu.

2. Shoda mezi systémem a skutečným světem Návrh by měl mluvit jazykem uživatelů. Měly by být využívány slova, fráze a pojmy, které uživatel zná, a ne interní žargon. Doporučuje se dodržování konvencí reálného světa a zobrazování informací v přirozeném a logickém pořadí.

Způsob návrhu, do značné míry závisí na konkrétních uživatelích. Pojmy, koncepty, ikony a obrázky, které se někomu mohou zdát naprosto jasné, mohou být pro uživatele neznámé nebo matoucí.

Pokud se ovládací prvky návrhu řídí konvencemi reálného světa a odpovídají požadovaným výsledkům (tzv. přirozené mapování), uživatelé se snáze naučí a zapamatují si, jak rozhraní funguje. To pomáhá vytvořit intuitivní prostředí.

Realita Barvy typů tříděného odpadu odpovídají košům a kontejnerům v reálném světě. Jejich názvy se také shodují.

3. Kontrola a svoboda uživatele Uživatel často provádí akce omylem. Potřebuje jasně označený „nouzový východ“, aby mohl nechtěnou akci opustit, aniž by musel procházet zdlouhavým procesem.

Když je pro lidi snadné z procesu vycouvat nebo akci vrátit zpět, podporuje to pocit svobody a důvěry. Východy umožňují uživatelům mít systém stále pod kontrolou a vyhnout se zaseknutí a pocitu frustrace.

Realita Na obrazovce nastavení je jasně viditelné tlačítko zpět na standardním očekávaném místě na horní liště vlevo. Při otevření filtru druhů odpadu, může uživatel kliknout mimo něj a jednoduše volbu opustit.

4. Konzistence a standardy Uživatel by neměly přemýšlet, zda různá slova, situace nebo akce znamenají totéž. Důležité je dodržovat zaběhlé standarty a konvence.

Jakobův zákon říká, že lidé tráví většinu času používáním jiných digitálních produktů, než je ten náš. Zkušenosti uživatelů s těmito jinými produkty určují jejich očekávání. Nedodržení konzistence může zvýšit kognitivní zátěž uživatelů tím, že je nutí učit se něco nového.

Realita Ikony jednotlivých bodů na mapě odpovídají klasickým mapovým. Všechny položky navigačního menu mají ikony ze standardní knihovny android ikon. Navigační menu se nachází vždy na místě, které je standardem. Tlačítko přesunu kamery mapy na místo, kde se aktuálně nacházím a tlačítko navigace se nachází na očekávatelných místech, která odpovídají klasickým mapovým aplikacím.

5. Prevence chyb Dobrá chybová hlášení jsou důležitá, ale nejlepší návrhy pečlivě předcházejí vzniku problémů. Buď eliminujte podmínky, které jsou náchylné k chybám, nebo je kontrolujte a nabídněte uživatelům možnost potvrzení předtím, než se k akci zaváží.

Uživatel by neměl mít možnost zadat nevalidní hodnotu, např. text do číselného pole. Povinné položky by měly být zvýrazněny, uživatel by neměl mít možnost pokračovat se špatnými hodnotami, aby byl následně nucen se vracet zpět kvůli jejich opravě.

Realita Zadání nevalidních možností zde není možné. Uživatel vybírá ze zaškrtávacích políček, kde jsou všechny kombinace validní. Při vyhledávání adresy pak není možné zadat špatnou hodnotu.

6. „Kouknu a vidím“ Minimalizujte zatížení paměti uživatele tím, že zviditelníte prvky, akce a možnosti. Uživatel by si neměl pamatovat informace z jedné části rozhraní do druhé. Informace potřebné k používání návrhu (např. popisky polí nebo položky nabídek) by měly být viditelné nebo v případě potřeby snadno vyhledatelné.

Lidé mají omezenou krátkodobou paměť. Rozhraní, která podporují rozpoznávání, snižují množství kognitivního úsilí vyžadovaného od uživatelů.

Realita Všechny možnosti navigace jsou označeny popiskem. Možnost výběru filtrů může nový uživatel přehlédnout, dlouhodobý uživatel ocení, že nepřekáží při prohlížení mapy.

7. Flexibilita a efektivita použití Zkratky – skryté před začínajícími uživateli – mohou urychlit interakci pro zkušeného uživatele, takže návrh může vyhovět jak nezkušeným, tak zkušeným uživatelům. Umožněte uživatelům přizpůsobit si časté akce.

Flexibilní procesy lze provádět různými způsoby, takže si lidé mohou vybrat ten způsob, který jim vyhovuje.

Realita Vše je v aplikaci dostupné na jedno, až dvě kliknutí. Jedinou výjimkou může být navigace k oblíbenému koši na tříděný odpad. Pro tuto akci však není předpoklad, jelikož uživatel si po prvním nalezení místa, tuto lokalitu často zapamatuje sám a navigaci tak není nutné opakovaně spouštět.

8. Estetický a minimalistický design Rozhraní by neměla obsahovat informace, které jsou nepodstatné nebo které jsou potřeba jen zřídka. Každá další jednotka informací v rozhraní konkuruje relevantním jednotkám informací a snižuje jejich relativní viditelnost.

Tato heuristika neznamená, že musíte používat plochý design – jde o to, abyste se ujistili, že se obsah a vizuální design soustředí na to podstatné. Zajistěte, aby vizuální prvky rozhraní podporovaly primární cíle uživatele.

Realita Na mapě se nevyskytují žádné přebytečné informace. Detail koše zobrazí nanejvýš vždy jen tříděné typy odpadů, vzdálenost a adresu. Všechno jsou to data nezbytná pro požadovanou interakci s aplikací. Detailní informace se skrývají až v nastavení.

9. Smysluplná chybová hlášení Chybová hlášení by měla být vyjádřena jednoduchým jazykem (bez chybových kódů), přesně označovat problém a konstruktivně navrhovat řešení.

Tato chybová hlášení by měla být rovněž prezentována s vizuálním zpracováním, které pomůže uživatelům si jich všimnout a rozpoznat je.

Realita Při načítání seznamů nejbližších košů nebo při vyhledávání se v případě chyby uživateli zobrazí možnost zkusit akci provést znovu. Pokud vyhledávání neposkytne žádné výsledky, je toto uživateli také sděleno.

10. Nápověda a dokumentace Nejlepší je, když systém nepotřebuje žádné další vysvětlení. Někdy však může být nutné poskytnout dokumentaci, která uživatelům pomůže pochopit, jak dokončit jejich úkoly.

Nápověda a dokumentace by měly být snadno vyhledatelné a zaměřené na úkoly uživatele. Buďte struční a uvádějte konkrétní kroky, které je třeba provést.

Realita Všechny potřebné vysvětlivky jsou poskytnuty v nastavení.

5.2 Uživatelské testování použitelnosti

Cílem uživatelského testování je ověřit, zda je uživatelské rozhraní dostatečně intuitivní a zda uživatelům nedělá používání aplikace problémy. Provádí se na základě vytvořeného scénáře s uživateli vybranými na základě person 3.1.

5.2.1 Scénář testování

Následující scénář popisuje pokyny, které zadává vedoucí testování uživateli. Instrukce se snaží simulovat budoucí využívání aplikace uživatelem reálným. Sestaven byl na základě definovaných typických případů užití 3.4.

Jste obyvatelem hl. města Prahy. Často se stěhujete a vždy vám při tom vznikne hromada odpadu. Takové stěhování právě proběhlo a vy máte v pokoji jako vždy velkou kouli plastové lepenky a štos papírů, kterými byly popsané krabice, které potřebujete vyhodit. Jste však v nastěhované lokalitě nový/á

a vůbec se tu nevyznáte. Nevíte tak, kde se nachází v nejbližším okolí koše na tříděný odpad. Padlo tedy rozhodnutí stáhnout si aplikaci, která vám úkol nalezení takového místa usnadní. Tu právě držíte v ruce.

1. Najděte nejbližší místo, kde nyní můžete vyhodit hromadu papírů a větší množství plastové lepenky.
2. Jste trochu paranoidní z toho, že vás někdo sleduje, proto zakažte aplikaci používat vaši polohu.
3. Do práce budete z nového bydliště dojíždět nedaleko na ulici Chorvatská a jelikož rádi třídíte, chcete si vyhledat, kde se na tomto místě dají vyhodit všechny možné druhy odpadu.
4. Se zjištěnými informacemi jste spokojen/a, rozhodnete se proto k nejbližšímu koši vyrazit na procházku. Spusťte na dané místo navigaci.

5.2.2 Testování s uživateli

Pro uživatelské testování byla zvolena aplikace pro telefony. Chytrý telefon má dnes téměř každý, což usnadní hledání vhodných testerů. Zbývající 2 platformy mohou výsledky testování také reflektovat, jelikož jsou postaveny na stejných základech. K testu byly vybráni uživatelé na základě předem definované cílové skupiny – viz. kapitola 3.1. Dva uživatelé uživatelé spadají pod osobu A, jeden potom pod osobu B.

Uživatel 1 je student přírodovědecké fakulty Univerzity Karlovy, je mu 21 let a bydlí v Praze na studentském bytě. **Uživatel 2** je studentka fakultu potravinářské a biochemické technologie na Vysoké škole chemicko-technologické, bydlí na kolejích a v Praze má i práci na půl úvazek. Je jí 23 let. **Uživatel 3** pracuje jako manažer ve větší Pražské firmě, kam každý den dojíždí. Je mu 33 let.

Testování proběhlo na zařízeních testovaných osob. Tato zařízení uvádí tabulka 5.1.

	Zařízení	Verze OS
Uživatel 1	Samsung Galaxy A52	12.0
Uživatel 2	Xiaomi Mi 9T	11.0
Uživatel 3	Samsung Galaxy J5	9.0

Tabulka 5.1: Použitá testovací zařízení

1. Nalezení místa pro vyhození papíru a plastu

Uživatel 1 Nejprve vycentroval mapu na svoji polohu a proklikal koše v okolí. Následně našel filtr, nechal jediný plast a papír mezi vybranými a na mapě našel nejbližší koš vyhovující parametrům.

Uživatel 2 Otevřel seznam nejbližších košů a z nabídky vybral první, který nabízel koš i papír.

Uživatel 3 Stejně jako uživatel 2 našel koš přes nabídku v menu „Nejbližší".

2. Vypnutí využívání polohy

Uživatel 1 Bez zaváhání přechod do nastavení a vypnutí využívání polohy.

Uživatel 2 V nastavení nalezena položka, vypnutí využívání polohy, nejprve přečten význam, následně vypnuta.

Uživatel 3 Vypnutí využívání polohy přes položku nastavení.

3. Odpad na ulici Chorvatská

Uživatel 1 Uživatel ihned nalezl na mapě ulici Chorvatskou pouhým pohybem po mapě (bydlí na vedlejší ulici). Na ní se nachází koš pouze jeden, vybral tedy ten.

Uživatel 2 Využil nabídku navigačního menu „Vyhledat", vyhledal „Chorvatská" a našel koš.

Uživatel 3 Našel skrze vyhledávací menu koš na ulici chorvatské, upravil filtr, aby odpovídal všem druhům odpadu. Po zjištění, že koš nezmizel potvrdil tento jako hledaný.

4. Navigace na místo

Uživatel 1 Po zvolení nejbližšího koše, jako v prvním kroku uživatel tápal, kde nalezne možnost navigace. Snažil se klikat na otevřené okno s detaily o místu, tlačítko nenalezl.

Uživatel 2 Nejbližší koš našel opět skrz jejich seznam, následně spustil navigaci skrze nabídnuté tlačítko v rohu mapy.

Uživatel 3 Stejně jako Uživatel 1 se pokoušel spustit navigaci klikáním na otevřené okno s detaily.

5.3 Závěry testování

V rámci Heuristické analýzy 5.1 bylo vyhodnoceno jako málo výrazné tlačítko pro možnost filtrování typů odpadu. Ikoně na něm byla přidána animace, která se ve smyčce přerušeně spouští, dokud na tlačítko uživatel poprvé neklikne.

Při testování s uživateli se při plnění 4. úkolu – *Navigace na místo* objevil problém s navigací k vybranému koši. Toto bylo reflektováno a po kliknutí na detail koše je nyní spuštěna navigace na dané místo.

Celkově se tedy při testování neprojevil žádný velký nedostatek a dopadlo úspěšně.

Závěr

Tato práce se zabývala návrhem a implementací aplikace Tříděný odpad Praha spolu s příslušným API napojeným na otevřená data hl. m. Prahy. Cílem bylo také analyzovat strukturu zdrojových dat. Součástí analýzy byl také výběr frameworku pro tvorbu API v jazyce Ruby a srovnání API pro geokódování.

V rámci návrhu byly definovány funkční a nefunkční požadavky. Spolu se stanovením specifikace cílové skupiny a typických případů užití byly navrhnuty i wireframy (drátové modely) s nízkým stupněm věrnosti. Vypracovány byly verze pro mobilní telefony, tablety i chytré hodinky.

Na základě návrhu bylo v jazyce Ruby implementováno API a v jazyce kotlin napsána mobilní aplikace pro OS Android a Wear OS. Popsány byly implementační specifika obou částí.

API bylo otestováno jednotkovými (unit) testy, aplikace potom byla podrobena testování uživateli a heuristické analýze. Zjištěné nedostatky byly opraveny.

Cíl práce se tedy podařilo splnit, výstupem je funkční mobilní aplikace pro systémy Android a Wear, stejně jako API nasazené na platformě Heroku, kde obě splňují všechny definované požadavky.

Obě aplikace budou dostupné ke stažení na Google Play.

Literatura

[1] *jaktridit.cz* [online]. [cit. 2022-04-12]. Dostupné z: https://www.jaktridit.cz/

[2] UNITED NATIONS. *#YouthStats: Environment and Climate Change* [online]. [cit. 2022-04-12]. Dostupné z: https://www.un.org/youthenvoy/environment-climate-change/

[3] *Města Třídí* [online]. [cit. 2022-04-12]. Dostupné z: https://play.google.com/store/apps/developer?id=M%C4%9Bsta+t%C5%99%C3%ADd%C3%AD

[4] SEZNAM.CZ, A.S. *Mapy.cz* [online]. [cit. 2022-04-12]. Dostupné z: https://mapy.cz/

[5] *opendata hlavního města Prahy* [online]. [cit. 2022-04-12]. Dostupné z: https://opendata.praha.eu/about

[6] *STANOVIŠTĚ TŘÍDĚNÉHO ODPADU - POLOŽKY* [online]. [cit. 2022-01-20]. Dostupné z: https://opendata.praha.eu/dataset/stanoviste-trideneho-odpadu-polozky

[7] *Geoportál hl. m. Prahy* [online]. [cit. 2022-01-20]. Dostupné z: https://www.geoportalpraha.cz/cs/data/metadata/8B2B3E58-1B3F-4870-B205-82453E90A2F8

[8] OLSZOWKA, Christoph. *Web App Frameworks* [online]. [cit. 2022-01-27]. Dostupné z: https://www.ruby-toolbox.com/categories/web_app_frameworks

[9] *Sinatra* [online]. [cit. 2022-01-27]. Dostupné z: http://sinatrarb.com/

[10] *Roda* [online]. [cit. 2022-01-27]. Dostupné z: https://roda.jeremyevans.net/

[11] *Ruby Grape* [online]. [cit. 2022-01-28]. Dostupné z: http://www.ruby-grape.org/

[12] REISNER, Alex. *Gem geocoder* [online]. [cit. 2022-02-17]. Dostupné z: https://github.com/alexreisner/geocoder

[13] GOOGLE LLC. *Compose lifecycle* [online]. [cit. 2022-02-28]. Dostupné z: https://developer.android.com/jetpack/compose/lifecycle

[14] *Creative Commons Zero* [online]. [cit. 2022-02-28]. Dostupné z: https://creativecommons.org/publicdomain/zero/1.0/deed.cs

[15] JAVŮREK, Marek. *Tříděný odpad (Barevné kontejnery) Praha* [online]. [cit. 2022-02-28]. Dostupné z: https://play.google.com/store/apps/details?id=com.barevnekontejnery

[16] SPOLEK UKLIĎME ČESKO. *Kam s ním?* [online]. [cit. 2022-02-28]. Dostupné z: https://www.kamsnim.cz/

[17] *Nominatim* [online]. [cit. 2022-03-04]. Dostupné z: https://nominatim.org/

[18] GOOGLE LLC. *Dependency injection* [online]. [cit. 2022-03-23]. Dostupné z: https://developer.android.com/training/dependency-injection

[19] GOOGLE LLC. *Knihvna Room* [online]. [cit. 2022-03-24]. Dostupné z: https://developer.android.com/training/data-storage/room

[20] HEROKU, INC. *Heroku* [online]. [cit. 2022-03-29]. Dostupné z: https://www.heroku.com/

[21] GOOGLE LLC. *Google Geocoding API* [online]. [cit. 2022-03-30]. Dostupné z: https://developers.google.com/maps/documentation/geocoding/requests-reverse-geocoding

[22] MapTiler AG. *MapTiler* [online]. [cit. 2022-03-30]. Dostupné z: https://www.maptiler.com/cloud/

[23] GEOAPIFY GMBH. *Geoapify* [online]. [cit. 2022-03-30]. Dostupné z: https://www.geoapify.com/reverse-geocoding-api

[24] *Nominatim Usage Policy* [online]. [cit. 2022-03-30]. Dostupné z: https://operations.osmfoundation.org/policies/nominatim/

[25] *Pelias* [online]. [cit. 2022-03-30]. Dostupné z: https://github.com/pelias/documentation/

[26] NIELSEN, Jakob. *10 Usability Heuristics for User Interface Design* [online]. [cit. 2022-03-31]. Dostupné z: https://www.nngroup.com/articles/ten-usability-heuristics/

[27] GOOGLE LLC. *Knihovna Paging 3* [online]. [cit. 2022-04-10]. Dostupné z: https://developer.android.com/topic/libraries/architecture/paging/v3-overview

[28] *What is Five-Star Linked Open Data?* [online]. [cit. 2022-04-12]. Dostupné z: https://www.ontotext.com/knowledgehub/fundamentals/five-star-linked-open-data/

[29] THE PUBLICATIONS OFFICE OF THE EUROPEAN UNION *Co jsou otevřená data?* [online]. [cit. 2022-04-12]. Dostupné z: https://data.europa.eu/cs/trening/what-open-data

[30] GOOGLE LLC. *Android's Kotlin-first approach* [online]. [cit. 2022-04-13]. Dostupné z: https://developer.android.com/kotlin/first

[31] GOOGLE LLC. *Guide to app architecture* [online]. [cit. 2022-04-13]. Dostupné z: https://developer.android.com/jetpack/guide#recommended-app-arch

[32] GOOGLE LLC. *Android studio* [online]. [cit. 2022-04-13]. Dostupné z: https://developer.android.com/studio

[33] *Ruby on Rails* [online]. [cit. 2022-04-13]. Dostupné z: https://rubyonrails.org/

[34] JETBRAINS S.R.O. *RubyMine* [online]. [cit. 2022-04-13]. Dostupné z: https://www.jetbrains.com/ruby/

[35] *What is a gem?* [online]. [cit. 2022-04-13]. Dostupné z: https://guides.rubygems.org/what-is-a-gem/

[36] *The Active Record Design Pattern* [online]. [cit. 2022-04-13]. Dostupné z: https://researchhubs.com/post/computing/web-application/the-active-record-design-pattern.html

[37] *Maps Compose* [online]. [cit. 2022-04-13]. Dostupné z: https://github.com/googlemaps/android-maps-compose

[38] GOOGLE LLC. *Using API Keys* [online]. [cit. 2022-04-16]. Dostupné z: https://developers.google.com/maps/documentation/android-sdk/get-api-key

[39] GOOGLE LLC. *Android Modules* [online]. [cit. 2022-04-16]. Dostupné z: https://developer.android.com/studio/projects#ApplicationModules

Příloha A

Slovník

API Application Programming Interface - rozhranní pro programování aplikací. vii, viii, 3, 5, 9, 10, 11, 12, 13, 15, 17, 23, 29, 30, 31, 32, 33, 38, 39, 40, 55

CKAN The Comprehensive Knowledge Archive Network. 6

GML Geography Markup Language. 7

JSON JavaScript Object Notation. 23, 32, 38, 39

OS Operační systém. vii, 3, 29

RDF Resource Description Framework - systém popisu zdrojů. 6

REST Representational State Transfer - architektura rozhraní, navržená pro distribuované prostředí. 10, 11, 23

SPARQL Simple Protocol and RDF Query Language. 6

UI User interface - uživatelské rozhraní. 34, 36, 41

XML Extensible markup language - rozšiřitelný značkovací jazyk. 6, 34, 36

Seznam výpisů kódu

Příloha **B**

Obsah přiloženého CD

- `readme.txt` ..... stručný popis obsahu CD
- `apk` ..... adresář se spustitelnou formou obou aplikací
- `src`
  - `impl` ..... zdrojové kódy implementace
  - `thesis` ..... zdrojová forma práce ve formátu LaTeX
- `text` ..... text práce
  - `thesis.pdf` ..... text práce ve formátu PDF