VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
FACULTY OF INFORMATION TECHNOLOGY

ÚSTAV INTELIGENTNÍCH SYSTÉMŮ
DEPARTMENT OF INTELLIGENT SYSTEMS

MODULÁRNÍ SYSTÉM PRO HLÁŠENÍ A SPRÁVU PROBLÉMŮ V OBCI

MODULAR SYSTEM FOR REPORTING AND MANAGING FAULTS IN TOWN

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE **JAN KOTAS**
AUTHOR

VEDOUCÍ PRÁCE **Ing. LUKÁŠ SEMERÁD**
SUPERVISOR

BRNO 2019

Vysoké učení technické v Brně
Fakulta informačních technologií

Ústav inteligentních systémů (UITS) Akademický rok 2018/2019

Zadání bakalářské práce

Student: **Kotas Jan**
Program: Informační technologie
Název: **Modulární systém pro hlášení a správu problémů v obci**
Modular System for Reporting and Managing Faults in Town
Kategorie: Uživatelská rozhraní

Zadání:

1. Prostudujte literaturu týkající se tvorby mobilních aplikací a existující aplikace pro hlášení problémů obcím.
2. Navrhněte mobilní uživatelskou a administrátorskou aplikaci pro hlášení a přijímání závad.
3. Navrhněte moduly, které mohou být spojeny s navrženým systémem.
4. Vyberte vhodné existující nástroje a navrženou aplikaci implementujte.
5. Do aplikace implementujte alespoň dva navržené moduly z bodu 3.
6. Proveďte experimenty na efektivitu a použitelnost aplikace a zhodnoťte dosažené výsledky.

Literatura:

- A. Grant, J. Mužík. *Android 4 - Průvodce programováním mobilních aplikací* - Computer Press, 2013
- ROGERS, R., LOMBARDO, J., MEDNIEKS, Z., MEIKE, B. *Android application development*. 1st ed. Sebastopol, Calif.: O'Reilly, 2009. ISBN 05-965-2147-2.
- Procházka, P.: *Mapování a modelování jako nástroj pro zlepšování procesů*. Brno: Ekonomicko správní fakulta Masarykovy univerzity, 2008.

Pro udělení zápočtu za první semestr je požadováno:

- První tři body zadání.

Podrobné závazné pokyny pro vypracování práce viz http://www.fit.vutbr.cz/info/szz/

Vedoucí práce: **Semerád Lukáš, Ing.**
Vedoucí ústavu: Hanáček Petr, doc. Dr. Ing.
Datum zadání: 1. listopadu 2018
Datum odevzdání: 15. května 2019
Datum schválení: 1. listopadu 2018

Abstrakt

Tato bakalářská práce se zabývá návrhem a tvorbou systému pro hlášení a správu problému v obcích. Tento systém demonstruje možnosti pro zefektivnění procesu hlášení problémů za pomoci moderních technologií a aktivního zapojení občanů obcí. Systém se skládá z mobilní aplikace, webové administrace a aplikačního serveru. Po seznámení se se systémem vývoje Android aplikací byl proveden průzkum existujících řešení a analýza mínění potenciálních uživatelů. Následně byla vytvořena specifikace systému, podle které byl systém navržen a implementován. Po dokončení implementace bylo provedeno testování na skupině uživatelů, kteří posléze systém hodnotili.

Abstract

The aim of this bachelor thesis is designing and creating system for reporting and managing issues in towns. The system demonstrates options how to make process of reporting and managing issues more effective thanks to the modern technologies and citizen involvement. The system consists of mobile application, web administration and application server. After getting familiar with the Android application development system, the author of this thesis conducted a research of existing solutions and also analysis of potential customers opinion. Afterwards, the system specification was created. The system was designed and consequently implemented according to the specificiation of the system. The implementation was followed by system testing on group of users who then made its evaluation.

Klíčová slova

Hlášení problémů, samospráva, e-government, mobilní aplikace, webová aplikace, Android, Kotlin, PHP, Laravel, Vue.js

Keywords

Reporting issues, self-government, e-government, mobile application, web application, Android, Kotlin, PHP, Laravel, Vue.js

Citace

KOTAS, Jan. *Modulární systém pro hlášení a správu problémů v obci.* Brno, 2019. Bakalářská práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce Ing. Lukáš Semerád

Modulární systém pro hlášení a správu problémů v obci

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením pana Ing. Lukáše Semeráda. Další informace mi poskytli Mgr. Martin Hudec a Mgr. Jan Zvara PhD. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

.....
Jan Kotas
24. července 2019

Poděkování

Rád bych poděkoval Ing. Lukášovi Semerádovi za ochotu, pomoc, podnětné připomínky i velkou dávku trpělivosti. Dále bych rád poděkoval panu Mgr. Martinu Hudcovi za poskytnutí cenných zkušeností s existujícím systémem z Uherského Hradiště a panu Mgr. Janu Zvarovi PhD. za cennou zpětnou vazbu při návrhu aplikace.

Obsah

Kapitola 1

Úvod

Cílem bakalářské práce je vytvořit systém pro hlášení a správu problémů v obcích pomocí dostupných digitálních technologií. Tento systém má sloužit k zefektivnění správy obcí a k většímu zapojení občanů do veřejného dění. Systém by také mohl přispět k lepší informovanosti obyvatel o kvalitě správy jejich obce a správcům by mohl poskytnout příležitost, jak své výsledky prezentovat.

Z technického hlediska by se systém měl skládat z mobilní aplikace, která má sloužit k hlášení problémů a webové administrace, jejímž prostřednictvím se budou hlášení spravovat. Součástí řešení by měla být i podpora přidávání modulů administrátory obce. Cílem těchto modulů je umožnit správcům získávat dodatečné informace o nahlášených podnětech. Tyto informace mohou posloužit k bližšímu specifikování problému nebo k jednoduší identifikaci místa, kde se problém nachází.

Obsah práce je rozdělen do několika kapitol. V první kapitole se věnuji mobilní platformě Android. Kromě popisu jeho součástí se zde zabývám i nástroji pro vývoj Android aplikací. V další kapitole analyzuji existující řešení a specifikuji obecné požadavky na systém. Následující kapitola se soustředí na návrh jednotlivých částí systému a volbě vhodných technologií pro jeho implementaci. Té se věnuji v další kapitole, kde popisuji celý její průběh u všech částí systému. V poslední kapitole rozebírám způsoby testování systému, které jsem při jeho tvorbě použil, hodnocení systému uživateli a možnosti dalšího vývoje.

Kapitola 2

Vývoj mobilních aplikací

Mobilní telefony – na rozdíl od stolních počítačů – nabízí úplně jiné možnosti využití, ale stejně tak představují i řadu omezení, které vývojáři aplikací musejí respektovat (omezená kapacita baterie, malý displej, nižší výpočetní výkon, různé velikosti zařízení atd.). Telefony ze své podstaty sází především na vysokou mobilitu a díky pokročilým technologiím poskytují i funkce, které dříve byly doménou výhradně osobních počítačů. Těmito funkcemi mohou být např. přístup k internetu, prohlížení dokumentů či fotografií, přehrávání videa nebo používání pokročilých pracovních nástrojů. Díky tomu se mobilní telefony staly během několika málo let téměř nepostradatelnou součástí našich životů a získávají si stále dominantnější pozici na trhu s elektronikou (v roce 2018 používalo mobilní telefon více než 5 miliard uživatelů [15]).

2.1 O Androidu

Operační systém Android je aktuálně nejpoužívanějším operačním systémem zejména v oblasti chytrých telefonů. V současné době je používán více než 2,3 miliardami chytrých telefonů po celém světě [22]. Kromě chytrých telefonů se mu podařilo prosadit také na poli chytrých hodinek, televizí, tabletů nebo automobilů.

Vývoj Androidu začal v roce 2003, kdy zakladatelé Andy Rubin, Rich Miner, Nick Sears a Chris White založili stejnojmennou společnost [9]. V roce 2005 Android koupila společnost Google za nejméně 50 milionů dolarů [14]. Klíčoví vývojáři se s Google dohodli a přidali se k němu jako součást akvizice. V říjnu roku 2007 vzniklo konsorcium Open Handset Alliance celkem 34 firem (mezi nimiž figurovaly globální společnosti jako např. Google, Motorola, Qualcomm, HTC nebo T-Mobile) s cílem vytvořit první opravdovou svobodnou platformu pro mobilní zařízení. Od té doby se na vývoji spolupodílí i tyto firmy [19].

Open Handset Alliance poskytla velké množství softwaru a jiného duševního vlastnictví svých členských organizací pod licencí Apache 2, tedy jako otevřený software [3]. Uplatnění licence Apache je naprosto kritické, protože to umožňuje výrobcům mobilních telefonů využívat zdrojové kódy Androidu ve svých zařízeních a v případě potřeby si jej i svobodně upravovat. Zejména díky tomu si Android oblíbilo velké množství společností. Dnes je naprosto běžné, že si výrobci telefonů vyvíjí vlastní modifikované verze (např. MIUI, YunOS, Touchwiz apod.) [18].

2.2 Architektura systému Android

Architektura operačního systému Android se skládá z komponent, z nichž nejdůležitější jsou znázorněny v diagramu na obrázku 2.1 Tyto komponenty jsou tvořeny kolekcemi knihoven, které pomáhají vytvářet robustní, testovatelné a snadno udržovatelné aplikace [5].

- **Linuxové jádro** – základním kamenem platformy Android je Linuxové jádro. Díky tomu může Android využívat všech klíčových bezpečnostních prvků, které jsou v Linuxu obsaženy. Zároveň se tím usnadní vývoj hardwarových ovladačů, jelikož jde o již dobře známé jádro.

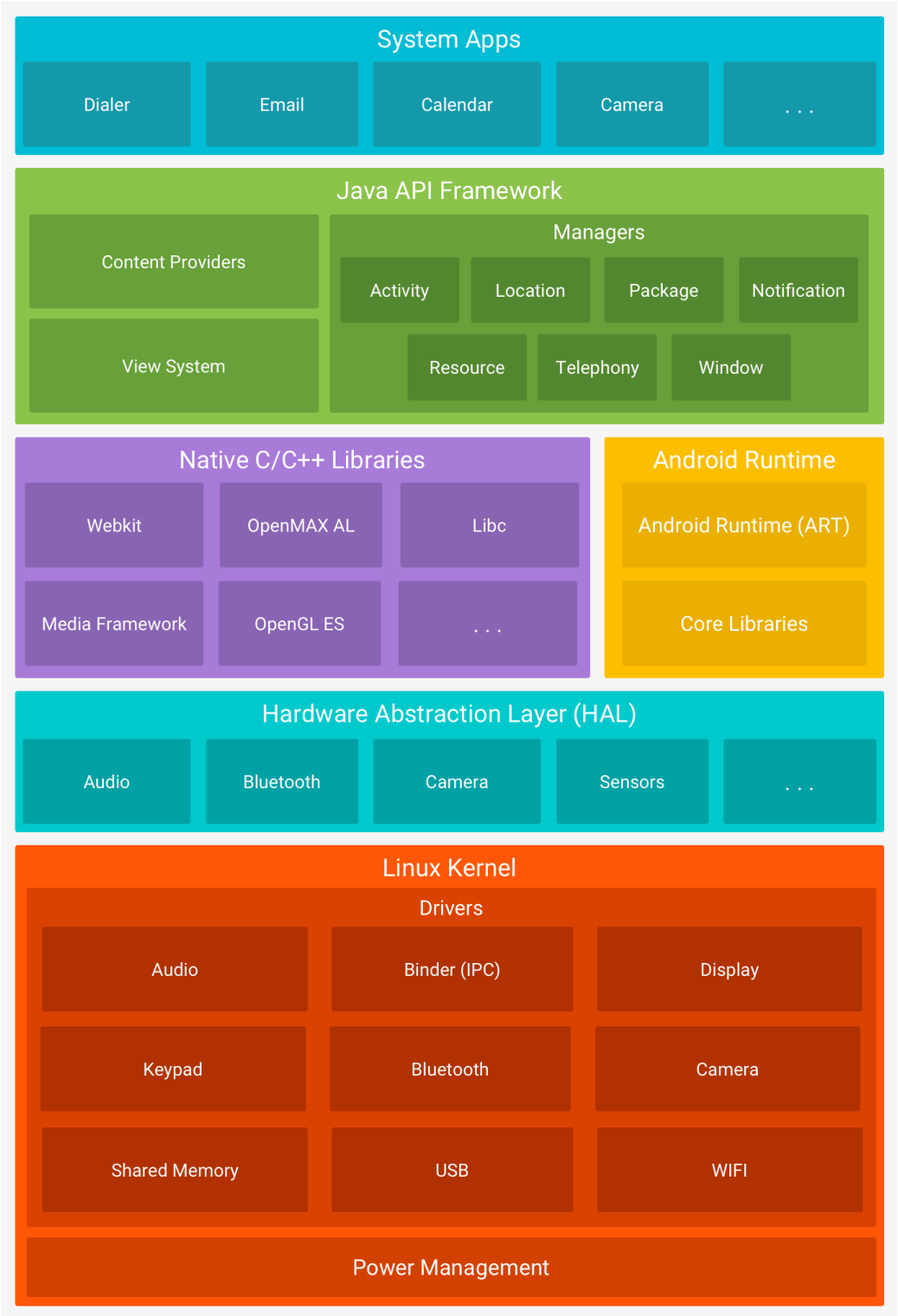
- **Hardwarová abstraktní vrstva (HAL)** – poskytuje standardní rozhraní, které umožňuje užívání hardwarových prostředků vyšším, abstraktnějším vrstvám. HAL se sestává z mnoha modulů, z nichž každý implementuje rozhraní pro specifickou hardwarovou komponentu (např. fotoaparát, Bluetooth apod.).

- **Android Runtime (ART)** – Všechny aplikace, běžící na zařízeních s Androidem verze vyšší než 5.0 (API 21), mají spuštěný samostatný proces instance Android Runtime (ART). ART slouží k spouštění virtuálních strojů na zařízeních s nízkou kapacitou operační paměti a díky tomu maximálně snižuje nároky aplikací na její užívání. Výsledkem je optimalizovaný chod aplikací s efektivnějším uvolňováním alokované paměti. Součástí ART jsou i různé ladící a diagnostické nástroje, které se hodí zejména při vývoji nových aplikací.

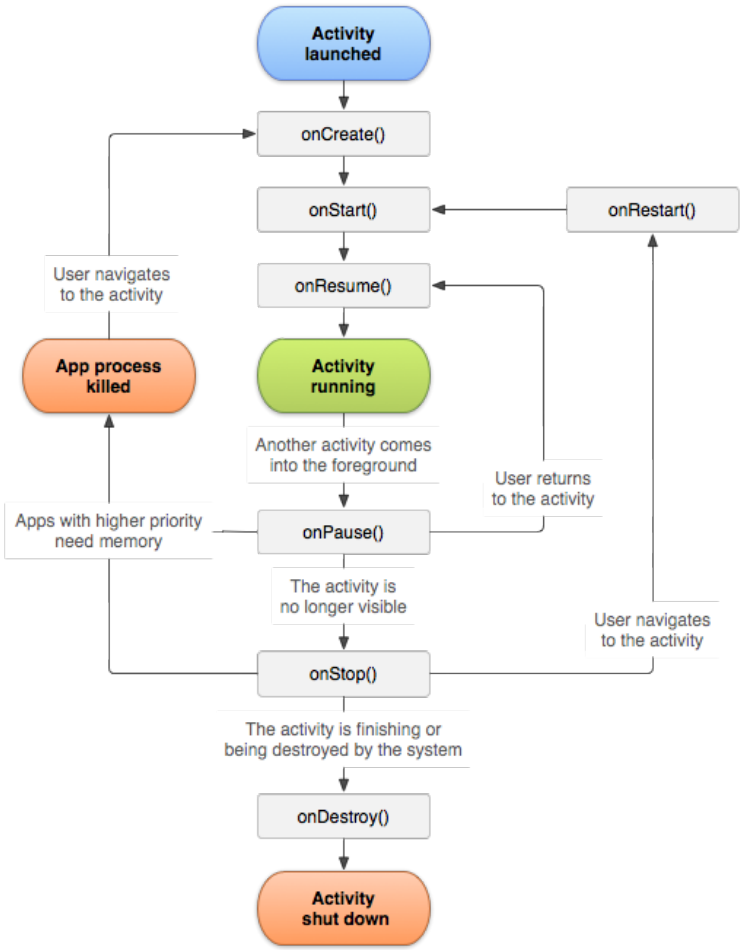
- **Nativní C/C++ knihovny** – mnoho systémových komponent, jako například ART nebo HAL, jsou postaveny na základě kódu, který vyžaduje podporu nativních knihoven napsaných v C a C++. Platforma Android poskytuje rozhraní pro přístup k některým těmto knihovnám. Příkladem může být přístup ke knihovně OpenGL pro podporu kreslení a manipulaci s 2D a 3D grafikou v aplikacích.

- **Java API Framework** – celá sada funkcí operačního systému Android je k dispozici prostřednictvím rozhraní napsaném v jazyce Java. Toto rozhraní vývojářům umožňuje používat již předpřipravené stavební bloky aplikací. Díky tomu je možné recyklovat design aplikací a soustředit se čistě jen na požadovanou funkční stránku aplikací, implementovanou na vyšším stupni abstrakce. Pro koncového uživatele aplikací to má pozitivní dopad zejména v tom, že již umí zacházet s použitými stavebními bloky. Pro vývojáře mobilních aplikací je tato vrstva naprosto stěžejní.

- **Systémové aplikace** – tato nejvyšší vrstva obsahuje systémové aplikace jako například email klienta, SMS klienta, kalendář, internetový prohlížeč apod. Tyto aplikace jsou základním softwarovým vybavením nově instalovaného systému a nemají oproti aplikacím třetích stran žádný speciální status. Pokud se tedy uživatel rozhodne zvolit jinou výchozí aplikaci namísto původní systémové, může tak učinit změnou v nastavení výchozích aplikací. Funkce systémových aplikací, stejně jako funkce aplikací třetích stran, obvykle poskytují prostředky, které vývojáři mohou využívat při tvorbě vlastních aplikací. Například pokud chce uživatel v aplikaci vyfotit fotografii, může tak učinit skrze zavolání systémové aplikace fotoaparát, která aplikaci výslednou fotografii předá.



Obrázek 2.1: Architektura operačního systému Android. Převzato z [5].



Obrázek 2.2: Ilustrace životního cyklu aktivit. Převzato z [6].

2.3 Součásti Android aplikací

Stejně jako jiné platformy i Android využívá jisté koncepce, které zaručují větší odolnost aplikací vůči chybám a které značně usnadňují vývojářům práci. Těmito koncepcemi jsou [1]:

- **Aktivity** – Stavebními bloky uživatelského rozhraní jsou aktivity. Aktivitu si lze představit jako entitu systému Android analogickou k oknu nebo dialogu klasické aplikace pro počítače. Každá aktivita se obvykle zaměřuje pouze na jednu konkrétní věc (např. přihlášení uživatele, registrace, fotografování apod.) a je nezávislá na ostatních aktivitách aplikace. Aktivity mají možnost spouštět jiné aktivity, ale pouze za předpokladu, že jim to tyto aktivity umožní.

  Jelikož jsou prostředky mobilního telefonu omezené, bylo nutné zajistit efektivní provoz aktivit. Každá aktivita se proto řídí tzv. životním cyklem aktivit, který spravuje systém. Ten se stará o to, aby v jeden moment na popředí běžela maximálně jedna aktivita. Ostatní aktivity jsou buď pozastavené nebo ukončené. Schéma životního cyklu aplikací je znázorněno na obrázku 2.2.

- **Služby** – Služby jsou – na rozdíl od aktivit – navrženy k neustálému provozu, nezávisle na aktivitách. Služby slouží k umožnění částečného běhu aplikace na pozadí. Lze je

proto použít například k přehrávání hudby na pozadí nebo k získávání aktualizovaných dat aplikací.

- **Záměry** – Záměry jsou systémové zprávy, které doručují aplikacím upozornění na různé události, a to i v případě, že aplikace v daný moment neběží. Těmito událostmi mohou být změny hardwaru (např. vložení karty SD), příchozí data (např. zprávy SMS) nebo události aplikací (např. aktivace budíku).
- **Poskytovatelé obsahu** – Poskytovatelé obsahu zprostředkovávají úroveň abstrakce pro jakákoliv data uložená v zařízení, ke kterým lze přistupovat z více aplikací. Příkladem může být poskytování informací o kontaktech uložených v zařízení.

2.4 Verze systému Android

Jako každý jiný systém, tak i Android se postupem času vyvíjí. Android s každou novou verzí přináší řadu novinek a vylepšení, které musejí vývojáři aplikací reflektovat. K určení verze API se využívá označení ve tvaru API a číslo verze (např. API 22). Je důležité podotknout, že číslování API se neshoduje s číslováním verzí systému (např. Android 4.4 KitKat používá API 19).

Jelikož jsou nové verze Androidu podporovány pouze nejnovějšími zařízeními na trhu, bylo nutné zajistit zpětnou kompatibilitu aplikací. K určení minimální kompatibilní verze slouží nastavení tzv. *minSdkVersion* v souboru *AndroidManifest.xml*. Aby byla kompatibilita zajištěna, je nutné v aplikaci využívat pouze prostředky dostupné v této nebo nižší verzi. Pokud by byly použity funkce z novějších verzí, aplikace by se nepřeložila. Čím nižší verze API se pro vývoj aplikace zvolí, tím více zařízení si ji bude moci nainstalovat.

Pro účely mé aplikace jsem zvolil verzi Android 6.0 Marshmallow s API 23. Díky tomu bude možné aplikaci instalovat na přibližně 75 %[1] všech aktuálně používaných zařízeních [2].

Android 6.0 Marshmallow

Verze Android 6.0 Marshmallow přišla s několika zásadními změnami. Tou nejzásadnější změnou je zavedení tzv. *Runtime Permissions*, čili udělování oprávnění aplikacím nejen při instalaci, ale i za běhu aplikace. Uživatel tak má lepší přehled o tom, jaké prostředky a data aplikace užívá a také může tato oprávnění v průběhu času měnit. Další výraznou změnou je zavedení automatického režimu spánku (tzv. *Doze*) a pohotovostního režimu u aplikací. Tyto změny přispěly k výrazným úsporám spotřeby energie, a tím pádem i k nárůstu výdrže zařízení na jedno nabití.

2.5 Vývojové prostředí a nástroje

K programování aplikací pro Android je k dispozici mnoho vývojových prostředí, z nichž nejrozšířenějšími jsou Android Studio a Eclipse. Až do roku 2014 bylo oficiálně podporováno pouze prostředí Eclipse s nainstalovaným pluginem ADT (Android Development Tools) a potřebným Android SDK (Software Development Kit). To bylo ovšem ještě v době, kdy neexistovalo žádné speciální vývojové prostředí pro Android, kterým je dnes již téměř výhradně používané Android Studio.

[1] Data pochází z května roku 2019. V době publikování práce bude tohle číslo pravděpodobně vyšší.

K programování logiky aplikací je možno zvolit jeden ze tří podporovaných jazyků. Těmito programovacími jazyky jsou Java, C++ a Kotlin. Nejběžnějším, a také nejdéle užívaným jazykem, je Java, která je s Androidem úzce spjata už od jeho samotného počátku. Poslední dobou se ovšem těší čím dál větší popularitě jazyk Kotlin. Ten z Javy vychází, ale zároveň nabízí i mnoho funkcí a možností, kterými Java nedisponuje. Jelikož jsem se rozhodl pro programování aplikace zvolit právě Kotlin, pokusím se jeho přednosti popsat v samotné podkapitole.

2.5.1 Android Studio

Android Studio je vývojové prostředí založené na IntelliJ IDEA. Představeno bylo poprvé firmou Google 16. května 2013 na konferenci Google I/O [12]. Od června téhož roku je k dispozici zdarma na platformách Windows, Mac OS a Linux. Mezi základní součásti, které Android Studio nabízí, patří:

- nástroje pro profiling
- pokročilé nástroje pro debugging a sledování výkonu
- nástroje pro grafickou (drag-and-drop) tvorbu uživatelského rozhraní
- testování kompatibility
- podepisování aplikací
- zabudovaný emulátor AVD (Android Virtual Device)
- předpřipravené schémata pro různé typy zařízení
- podpora jazyků Java, C++ a Kotlin

2.5.2 Kotlin

Kotlin je staticky typovaný jazyk běžící nad JVM (Java Virtual Machine), který byl vytvořen týmem programátorů společnosti JetBrains. Jeho vznik se datuje na rok 2011 s tím, že v roce 2012 byl uvolněn jako open-source[2] pod licencí Apache 2. V té době se však stále nejednalo o stabilní verzi. Oficiálně první stabilní verze Kotlinu (Kotlin v. 1.0) byla představena až 15. února 2016 [7].

Základní vlastnosti Kotlinu:

- *Kompatibilita* – Kotlin je plnohodnotně kompatibilní s JDK 6, což umožňuje bezproblémový běh jeho aplikací i na starších zařízeních. Programování v tomto jazyce je podporováno i ze strany vývojářů Androidu potažmo Android Studia.
- *Výkonnost* – Aplikace naprogramované v Kotlinu jsou – díky podobnosti tzv. byte kódu – stejně rychlé, jako aplikace v Javě.
- *Interoperabilita* – Kotlin je 100% interoperabilní s jazykem Java, což znamená, že lze stále využívat všech existujících Android knihoven, napsaných právě v tomto jazyce.

[2]Open-source software (česky otevřený software) je software s otevřeným zdrojovým kódem, který si může kdokoliv zobrazit, upravit či vylepšit[17].

- *Malý objem dat* – Kotlin má velmi kompaktní knihovnu, která ve výsledku přidá méně než 100 KB dat na celkové velikosti aplikace.
- *Kompilační čas* – Kotlin využívá efektivní inkrementální kompilaci, která je minimálně stejně rychlá a častokrát i rychlejší než standardní kompilace Javy.
- *Učící křivka* – Pro Java programátory je přechod na Kotlin velmi jednoduchý. Android Studio umožňuje jednoduchou konverzi Java kódu do Kotlinu. Na oficiálních stránkách je také k dispozici série cvičení v rámci Kotlin Koans, která demonstruje základní syntax a funkce Kotlinu [13].

2.6 Uživatelské rozhraní mobilních aplikací

S různými formami uživatelského rozhraní se lze v každodenním životě setkat nesčetněkrát. Cílem jakéhokoliv uživatelského rozhraní je umožnění efektivní interakce se zařízením nebo programem. V případě mobilních aplikací lze mluvit o podmnožině uživatelského rozhraní – tzv. grafickým uživatelským rozhraní (anglicky *Graphical User Interface*) – též známé pod zkratkou GUI.

Stejně jako u všech jiných zařízení i u mobilních aplikací se předpokládá, že budou s uživatelem interagovat způsobem, který je předvídatelný a především uživatelem požadovaný. A zde sehrává grafické uživatelské rozhraní zásadní roli. Grafické uživatelské rozhraní je totiž jedinou součástí aplikace, kterou má uživatel možnost vidět a v drtivé většině případů rozhoduje o tom, zda se uživatel rozhodne aplikaci dále používat, či nikoliv. Kdo by chtěl trávit svůj drahocenný čas používáním aplikace, která není uživatelsky přívětivá? Obzvlášť v případě, kdy si může stáhnout jinou, uživatelsky přívětivější aplikaci.

Vývojáři mobilních platforem si důležitost grafického uživatelského rozhraní dobře uvědomují. Proto se snaží poskytovat širokou škálu nástrojů a doporučení, jakým způsobem uživatelské rozhraní koncipovat. Příkladem sady doporučení může být tzv. *Material Design* od společnosti Google nebo obecnější *Human Interface Guidelines*, který doporučuje společnost Apple.

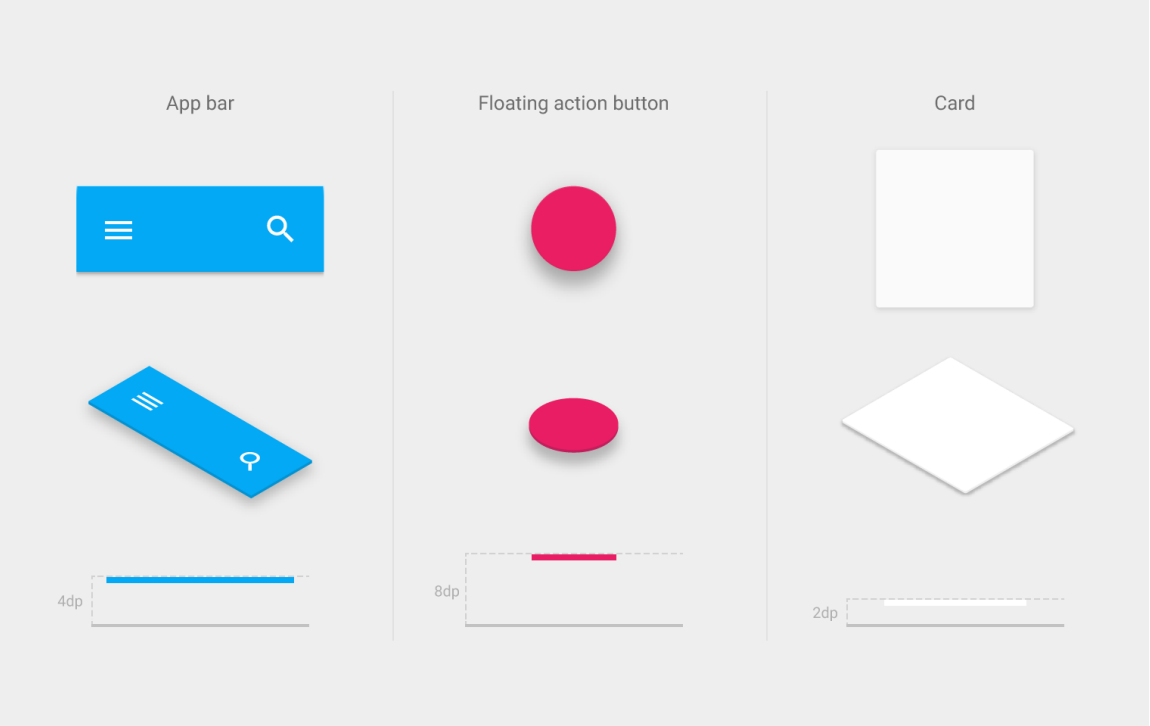
2.6.1 Material Design

Material Design je designový jazyk[3] vyvinutý společností Google v létě roku 2014 a představuje obrovský posun jak ve vzhledu, tak i chování moderního grafického uživatelského rozhraní. Uplatnění nachází zejména u mobilních a webových aplikací, ale postupem času si získává přízeň i vývojářů desktopových aplikací.

Material Design je z velké části založen na tradičních principech designu, ale přináší i mnoho změn, a to zejména v oblasti realistického, plynulého chování. Dalším posunem je i snaha o přiblížení k reálnému fyzickému designu, kterého se v Material Designu dosahuje zejména pomocí stínování. To umožňuje zvýraznit komponenty aplikace, které slouží k interakci (např. tlačítka nebo plovoucí karty), a dodávat jim iluzi třetího rozměru. Ukázka principu stínování je na obrázku 2.3.

Pod pojmem materiál si lze v kontextu Material Designu představit něco jako chytrý papír. Stejně jako papír má plochy a hrany, které odráží světlo a vytváří stíny, ale na rozdíl od klasického papíru má i schopnost se pohybovat, měnit svůj tvar a velikost. Díky

[3]Designový jazyk (anglicky design language) je soubor vzorů, konvencí a technik, které jsou uplatňovány k tomu, aby pomocí nich vytvořené návrhy poskytovaly konzistentní uživatelský dojem [21].



Obrázek 2.3: Ukázka používání stínů k navození iluze existence třetího rozměru. Převzato z [4].

těmto vlastnostem získává vytvořené rozhraní hloubku a strukturu, což pomáhá uživatelovi s orientací v aplikaci a zlepšuje tím uživatelský prožitek (anglicky User Experience) [16].

V případě Android aplikací je implementace Material Designu o dost jednoduší, než je tomu například u webových aplikací. Android Studio totiž nabízí již předpřipravené komponenty, které je třeba pouze drobně upravit a poté začlenit do připraveného layoutu.

Kapitola 3

Analýza situace a existující řešení

V této kapitole se zaměřuji na analyzování současné situace, existující řešení, nastínění ideálního stavu a jeho možné realizace. Nedílnou součástí kapitoly je i shrnutí motivace k tvorbě systému a možné problémy, které mohou vývoj provázet.

Analýza situace kolem existujících řešení a potřeb budoucích uživatelů je jednou ze stěžejních částí vývoje každého nového produktu. Ať už se jedná o nový informační systém nebo nový automobil, v obou případech je cílem vyvinout co nejlepší produkt, který budou zákazníci rádi používat a za který rádi zaplatí. K dosažení tohoto stavu je nejprve nutné analyzovat všechny důležité oblasti, které mohou mít vliv na výsledný produkt. Důležité je si také ověřit, zda vyvíjený produkt řeší reálný problém neboli naplňuje reálnou potřebu koncového uživatele. Nezřídka se totiž stává, že je problém pouhým důsledkem jiného problému nebo jej už někdo stihl vyřešit. Lze také dojít k závěru, že daný problém neexistuje nebo jej ostatní jako problém nevnímají. V takové situaci je nejlepší se zamyslet, zda se produkt vyvíjí jen pro vlastní potřebu nebo zda je určen širší skupině lidí. Pokud se jedná o druhý případ, je vhodné zvážit realizaci takového produktu.

Jelikož jsem chtěl od počátku aplikaci zpracovat tak, abych ji alespoň přiblížil možnosti uplatnění v reálném provozu, věnoval jsem právě analýze nemalé množství času.

3.1 Analýza současné situace

Abych mohl aktuální situaci dostatečně porozumět z pohledu všech zúčastněných stran, potřeboval jsem splnit následující body:

1. Získat od občanů informace, zda jsou ochotni problémy hlásit, jaké formy hlášení v současnosti využívají a jaké formě by do budoucna dali přednost. Tyto data poté vyhodnotit a získat představu o cílové skupině a její šíři. Výsledky průzkumu popisuji v podkapitole 3.2.

2. Udělat rešerši existujících řešení a vyhodnotit jejich použitelnost. V ideálním případě si zařídit schůzku s někým, kdo s nějakým existujícím řešením přímo pracuje a kdo je zároveň ochotný se o zkušenosti s jeho používáním podělit. Více v podkapitole 3.3.

3. Kontaktovat vedení alespoň jednoho města a zjistit bližší informace o tom, jakým způsobem řeší dané problémy a zda jsou otevřeni novému řešení. Tímto bodem za zabývám v podkapitole 3.4.

3.2 Průzkum veřejného mínění

K sestavení on-line veřejného průzkumu jsem se rozhodl zejména proto, že lze tímto způsobem oslovit poměrně velké množství lidí a výsledky jsou dobře a snadno měřitelné. Nevýhodou pro některé aplikace může být fakt, že zde nemusejí a zpravidla také nejsou zachovány demografické poměry jednotlivých skupin obyvatel.

V mém případě ale tento fakt nehraje příliš velkou roli. Lze totiž předpokládat, že případnou budoucí mobilní aplikaci budou používat pouze lidé, kteří umí ovládat moderní technologie – k nimž internet a sociální sítě bezesporu patří.

Ke zvýšení výpovědní hodnoty takového průzkumu lze přispět i kladením vhodných otázek (např. otázka na věk nebo pohlaví), pomocí kterých lze pak odpovědi filtrovat. Velmi důležité je také nastavit průzkum tak, aby dotyční odpovídali pouze na otázky, na které znají odpověď a nezkreslovali tak získaná data nesmyslnými odpověďmi. Toho lze docílit pomocí podmíněného výběru otázek, závislého na předchozích odpovědích.

Z průzkumu jsem potřeboval zjistit odpovědi na různé otázky v závislosti na předchozích zkušenostech respondentů s hlášením problémů, a také na jejich ochotě hlášení podávat. Proto zde rozčlením informace, které jsem potřeboval získat od jednotlivých skupin respondentů.

Informace nutné od všech respondentů:

- *pohlaví,*
- *věk,*
- *předchozí zkušenost s hlášením podnětů.*

Průzkumu se zúčastnilo celkem 144 respondentů, z nichž 51,4 % tvořily ženy a 48,6 % muži. Věkové kategorie byly následující – do 20 let 6,3 %, 20-29 let 43,3 %, 30-39 let 21,5 %, 40-49 let 16,7 %, 50-59 let 4,9 %, 60 a více let 6,9 %. Předchozí zkušenost s hlášením mělo 38,9 % respondentů.

Informace od respondentů, kteří zatím nenahlásili žádný podnět:

- *Povědomí o možnostech, jakými lze podněty hlásit.*
- *Ochota nahlásit podnět v případě, že by nějaký podnět ve svém okolí zaregistrovali.*

Této části průzkumu se účastnilo 88 respondentů. Povědomí o možnostech, jakými lze podněty hlásit má 48,2 % z nich. Ochotu hlásit podněty vyjádřilo 65,9 % zúčastněných.

Informace od respondentů, kteří již nějaký podnět nahlásili:

- *Kolik podnětů již v minulosti nahlásili.*
- *O jaké kategorie hlášení šlo.*
- *Jakou formu hlášeni zvolili.*
- *Zda dostali zpětnou vazbu na své hlášení.*
- *Zda došlo k vyřešení problému.*

Nějaký podnět nahlásilo celkem 56 dotázaných. Jeden podnět nahlásilo 32,1 % respondentů, 2-4 podněty pak 41,1 %, 5-9 podnětů 16,1 % a více než 10 podnětů nahlásilo 10,7 % dotázaných. Nejvíce hlášení byla v kategoriích: vodovody a kanalizace, veřejné osvětlení, zeleň, dopravní značení, vozovka, chodník, zvířata a černé skládky.

Nejčastějším prostředkem pro podání podnětu byl email, který zvolilo 31,6 % dotázaných. Následovala osobní žádost s 24,6 %, s 21 % telefonní hovor, 12,3 % kontaktní formulář na stránkách městského úřadu a na posledním místě s 10,5 % mobilní či webové aplikace. Zpětnou vazbu na své hlášení dostalo 59,6 % oslovených lidí s tím, že k vyřešení problému došlo v 65 % případů a ve 23 % nikoliv. Zbytek dotázaných (12 %) odpověděli, že tuto informaci neví.

Informace od respondentů, kteří jako formu hlášení zvolili mobilní nebo webovou aplikaci:

- *Jakou aplikaci pro hlášení použili a jak s ní byli spokojeni.*
- *Co by na dané aplikaci případně vylepšili.*

Jelikož mobilní či webovou aplikaci zvolilo pouze 10 dotázaných, nejsou výsledky příliš reprezentativní. Díky průzkumu jsem ale objevil dvě aplikace (Odkaz pre starostu a Brňáci pro Brno), o kterých jsem do té doby nevěděl. Mezi tipy na zlepšení těchto aplikací bylo například, aby ji používala i druhá strana (myšleno tím správa města) nebo zvýšení povědomí o těchto aplikacích i u ostatních obyvatel.

Informace od respondentů, kteří jsou ochotni v budoucnu hlásit podněty nebo již nějaký v minulosti hlásili:

- *Jakou formu hlášeni by v budoucnu nejraději zvolili.*
- *Zda by měli zájem sledovat i ostatní nahlášené problémy ve svém okolí.*

Na tuto otázku odpovídalo celkem 114 dotázaných. Email by zvolilo celkem 24,6 % oslovených, dalších 21,9 % by dalo přednost kontaktnímu formuláři, 20,2 % mobilní aplikaci, 17,5 % webové aplikaci, osobně nebo telefonicky pak 15,8 % oslovených. O sledování ostatních podnětů v okolí má zájem 76,3 % oslovených.

Informace od respondentů, kteří zatím žádný problém nenahlásili a nejsou ochotní problémy hlásit:

- *Z jakého důvodu problémy nechtějí hlásit.*

Nejčastějšími odpověďmi, proč lidé nechtějí problémy hlásit byly zejména tyto důvody:

- Neznalost způsobu, jakým problémy hlásit.
- Spoléhání na město, že má problémy zmapované.
- Očekávání, že to udělá někdo jiný.
- Nedůvěra v to, že by to k něčemu vedlo.

Informace od respondentů, kteří by nejraději v budoucnu použili mobilní nebo webovou aplikaci a kontaktní formulář:

- *Kolik času by byli maximálně ochotni věnovat hlášení.*

Tuhle část dotazníku zodpovědělo celkem 68 respondentů. Shodně 30,9 % dotázaných by hlášení věnovalo maximálně 5 respektive 10 minut, 26,5 % jednu minutu, 8,8 % více než 20 minut a 3 % 20 minut.

Shrnutí výsledků průzkumu

Jelikož je v mém průzkumu poměrně malý statistický soubor, nelze brát výsledná procenta příliš vážně. Cílem tohoto průzkumu ovšem nebylo získat co nejpřesnější data, ale spíše poodhalit postoje lidí k hlášení problémů ze svého okolí a získat představu o nástrojích, které v současné době k informování správy měst nejčastěji využívají.

Z výzkumu poměrně jasně vyplývá, že existuje velké množství lidí, kteří by byli ochotní problémy hlásit, ale nemají povědomí o možnostech, jak hlášení podat. Mnoho lidí také nevěří, že by se jimi nahlášené podněty opravdu řešily. Z toho vyplývá, že zcela zásadním faktorem pro úspěšnost jakéhokoliv systému hlášení je poskytnutí kvalitní zpětné vazby. Bez zpětné vazby nebudou lidé dostatečně motivovaní k tomu, aby problémy hlásili.

Z dotazníku také vyplynulo, že přibližně 3/4 lidí, kteří již nějaký problém nahlásili nebo jsou v budoucnu ochotni je hlásit, má zájem sledovat i ostatní podněty v jejich okolí. Zobrazení okolních podnětů ostatním uživatelům by mohlo aktivizovat i tu část obyvatel, kteří se o dění a o problémy ve svém okolí příliš nezajímají. Zároveň by se tím mohlo částečně zabránit vznikání duplicitních hlášení.

Z výsledků je také zřejmé, že by bylo v budoucnu vhodné poskytnout k hlášení podnětů i webové rozhraní, případně formulářový widget pro stránky městského úřadu. Mobilní aplikaci v průzkumu preferovala pouze pětina dotázaných, což je poměrně málo. V kombinaci s webovým rozhraním, případně i formulářovým widgetem by dosah mohl být až 60 %. Tato čísla je ovšem taky nutno brát s rezervou, jelikož mnoho lidí nemá dobrou představu o tom, jak by řešení v podobě mobilní či webové aplikace mohlo vypadat. Lze předpokládat, že pokud by se výsledné řešení uchytilo mezi tzv. early adopters[1], začali by ho postupně používat i lidé, kteří jej původně zavrhovali.

3.3 Existující řešení

Analýza existujících řešení může poskytnout mnoho informací o tom, které modely řešení fungují, a které naopak selhaly. Díky tomu se lze poučit z chyb ostatních produktů a minimalizovat tím neúspěch vlastního řešení.

Do výběru jsem zařadil jak mobilní aplikace, tak i webové aplikace, které řeší hlášení podnětů. V následujících podkapitolách stručně uvádím, jaké jsou klady a zápory při jejich používání.

[1] Early adopter je označení zákazníka, který je ochoten zkoušet a používat nové věci i přes riziko možného nepohodlí, jen aby mohl používat produkt s předstihem.

3.3.1 Hlášení závad a nedostatků

Hlášení závad a nedostatků je produkt společnosti T-MAPY s.r.o., která se zabývá především tvorbou geografických informačních systémů. Jedná se o čistě webovou aplikaci, která na mapě zobrazuje již přidané podněty a umožňuje komukoliv přidat podnět nový.

Díky spolupráci s panem Mgr. Martinem Hudcem jsem se mohl seznámit s fungujícím řešením v Uherském Hradišti, které se stalo pilotním městem tohoto projektu. Informace, které zde uvádím, pochází převážně z naší schůzky. Díky této schůzce jsem měl možnost vidět systém v chodu a vyslechnout si nabyté zkušenosti s jeho používáním.

Systém Hlášení závad a nedostatků je částečně integrován do geoinformačního systému, který T-MAPY s.r.o. nabízí a skládá se ze dvou oddělených částí. První část běží nezávisle na geoinformačním systému města a slouží k hlášení a správě aktuálních podnětů. Z důvodu optimalizace a přehlednosti nahlášených podnětů se v aplikaci zobrazuje pouze několik desítek aktuálně řešených hlášení. Hlášení, která jsou již delší dobu vyřešeny se pak archivují v druhé, analytické části, která je již součástí geoinformačního systému města a umožňuje podrobnější analýzu získaných dat.

Řešení od společnosti T-MAPY s.r.o. je oproti většině konkurenčních aplikací unikátní v tom, že do řešení problémů města přímo zapojuje a očekává od nich zpětnou vazbu pro uživatele. Uživatel je při každé změně stavu jím nahlášeného podnětu upozorněn v podobě emailu a může případně svůj podnět upřesnit.

Řešením hlášení je pověřeno obvykle několik osob, které mají různé role. Těmito rolemi jsou administrátor, schvalovatel, řešitel a supervizor. Administrátorem je obecně jeden člověk, který má pravomoc přidělovat role. Schvalovatelem může být jeden nebo i více lidí, kteří rozhodují o tom, zda se podnětem bude někdo zabývat či nikoliv. V případě schválení předá podnět řešitelovi, který se obecně stará o jednu či více kategorií hlášení. Poslední možnou rolí je supervizor, který dostává notifikace o nově nahlášených podnětech, ale nemá přímou pravomoc jakkoliv měnit jejich stav.

Obecně lze říci, že jde o velmi propracované a funkční řešení, které těží především z možnosti začlenění do geoinformačního systému města od stejnojmenné společnosti. Tomuto řešení bych vytkl především neexistenci mobilního klienta a špatnou zpětnou kontrolu výsledného řešení problému.

3.3.2 Zlepšeme Česko

Zlepšeme Česko je projekt společnosti Neogenia s.r.o. ve spolupráci se stávající službou ZmapujTo. Mobilní stejnojmenná aplikace nabízí kromě hlášení podnětů i další funkce jako hledání ztracených zvířat, informace o dění v obci nebo ztráty a nálezy.

Právě vzhledem k velkému množství funkcí působí mobilní aplikace velmi nepřehledně a obsahuje i velké množství chyb. Aplikace také neumožňuje zobrazit seznam aktuálně řešených podnětů a zobrazení na mapě je velmi pomalé a nepřehledné. Mnoho podnětů je v systému duplicitních a i několik let starých. Zobrazení detailu podnětu v aplikaci je taktéž problematické, protože pro větší množství informací se aplikace odkazuje na stránky ZmapujTo.cz. Z množství velmi starých a nevyřešených problémů v aplikaci lze také usuzovat, že města se o řešení problémů v této aplikaci příliš nezabývají.

Z výše uvedených neduhů lze usoudit, že je tohle řešení hlášení podnětů laciné. Řešení webové administrace jsem bohužel neměl možnost zhlédnout. Z toho důvodu nemohu posoudit, do jaké míry se s daty dále pracuje.

3.3.3 Odkaz pre starostu

Odkaz pre starostu je slovenský projekt neziskové organizace Institut pro dobře spravovanou společnost. Jde o velmi povedený projekt, který používá velké množství obcí po celém Slovensku. Od roku 2010, kdy byla aplikace oficiálně spuštěna, se díky ní podařilo vyřešit již přes 26 tisíc podnětů.

Systém obsahuje jak webovou, tak i mobilní verzi aplikace. Zpracování aplikace je precizní a v porovnání s ostatními aplikacemi má velmi propracovaný a přehledný design. Podněty je možno přidávat a zobrazovat jak v mobilní, tak i ve webové verzi. Obě verze také poskytují zobrazení podnětů na mapě a podrobné statistiky jednotlivých samospráv. Při procesu přidávání nového podnětu je možné detailně kategorizovat a popsat hlášený problém. To pak samosprávám umožní rychleji a lépe posoudit, o jaký problém se jedná a kdo za něj nese zodpovědnost. Za velký plus tohoto řešení také považuji zapojení městských i soukromých firem do řešení podnětů. Uživatelé aplikace tak mají dokonalý přehled o tom, v jakém stavu se nachází řešení podnětu, a kdo přesně za něj nese zodpovědnost.

Tohle řešení shledávám za velmi povedené a především funkční. Aplikace obsahuje všechny zásadní funkce a atributy k tomu, aby ji používali jak občané, tak i místní samosprávy.

3.3.4 Brňáci pro Brno

Brňáci pro Brno je projekt společnosti Brněnské komunikace a.s. (BKOM), která vznikla výhradně k monitorování města Brna. Podle informací, které jsem získal na Magistrátu města Brna (MMB) se aplikace již delší dobu nevyvíjí a používá ji pouze hrstka uživatel.

Výhodou tohoto řešení je, že umožňuje používání jak webové, tak i mobilní aplikace. Obě verze jsou jednoduché a přehledné, což umožňuje jejich snadné používání. Ovšem zásadním nedostatkem aplikace je nezačlenění některých důležitých kategorií (např. odpadkové koše nebo lavičky). Aplikace také umožňuje pouze minimální interakci s uživatelem a o nahlášených podnětech poskytuje pouze omezené množství informací. Chybí také implementace zodpovědnosti za řešení nahlášených podnětů a nejsou k dispozici ani žádné statistiky.

Tento projekt má pouze velmi omezenou funkcionalitu a jeho správci nemají ambici ji v budoucnu dále rozšiřovat. Další vývoj je, dle informací poskytnutých na MMB, nepravděpodobný. Jde totiž o poměrně nákladnou záležitost, která se v měřítku jednoho města nevyplácí.

3.3.5 Shrnutí

Za nejpropracovanější řešení považuji slovenský projekt Odkaz pre starostu, který problém hlášení a správy podnětů řeší komplexně. Pokud by získaná data byla dále využitelná například k analýze v některém z geoinformačních systémů, jednalo by se o naprosto dokonalé řešení s obrovským potenciálem do budoucna. Ostatní aplikace buď neřeší problém dostatečně obsáhle nebo obsahují velké nedostatky, které zásadně omezují jejich použitelnost.

3.4 Průzkum mínění vedení měst a obcí

Pro objasnění postoje obcí k vytvoření nového řešení pro hlášení a správu problémů jsem oslovil některé z nich. Celkově jsem absolvoval čtyři schůzky, z nichž tři byly na Magistrátě města Brna a s panem Mgr. Janem Zvarou Ph.D. a jedna se správcem geografického informačního systému Uherského Hradiště, panem Mgr. Martinem Hudcem. Tyto schůzky

mi pomohly objasnit postoje obcí k řešení hlášení problémů pomocí digitálních technologií a zjistit jejich požadavky na potenciální nový systém.

3.5 Specifikace obecných požadavků na systém

Specifikace obecných požadavků slouží k definování obecných náležitostí, které by finální systém měl splňovat. Tyto informace poté slouží k tvorbě návrhu aplikace a k výběru nástrojů, které se pro vývoj a chod aplikace využijí.

V průběhu schůzek na MMB jsem postupně představoval možné koncepty, jak by výsledný systém mohl fungovat a na příkladech jsem demonstroval možné funkcionality, které by do systému mohly být začleněny. Výsledky těchto schůzek a data z veřejného průzkumu přispěly k sepsání obecných požadavků na systém.

1. Uživatelé
    - Každý uživatel se bude moci registrovat a přihlásit do systému.
    - Každý uživatel bude spadat do některé ze tří kategorií (běžný uživatel, zaměstnanec samosprávy nebo administrátor aplikace).
    - O uživatelích se budou ukládat povinné informace jméno, email, telefonní číslo a heslo.
2. Zaměstnanci samospráv
    - Zaměstnanci samospráv jsou uživatelé, kteří mají navíc přístup do administrace jimi spravované obce.
    - Zaměstnanci budou moci nabývat celkem čtyř rolí, přičemž musejí nabývat alespoň jedné z nich, aby mohli být za zaměstnance považováni:
        - *Administrátor:* pouze jeden zaměstnanec v rámci jedné samosprávy, který přiřazuje role zbylým zaměstnancům. V rámci samosprávy má nejvíce pravomocí. Hlavní pravomocí, kterou se odlišuje od ostatních rolí je možnost přidávat nové zaměstnance a přidělovat jim role v systému. Administrátor může být také řešitelem podnětů.
        - *Schvalovatel:* pouze jeden zaměstnanec v rámci jedné samosprávy, který je zodpovědný za schvalování podnětů. V případě, že podnět schválí, pověří některého z řešitelů za konkrétní oblast zodpovědných. V případě zamítnutí podnětu může napsat krátké zdůvodnění, které se zobrazí uživateli v detailu hlášení. Schvalovatel, stejně jako administrátor, může být zároveň řešitelem podnětů.
        - *Řešitel:* řeší problémy z jedné nebo více konkrétních oblastí (např. doprava, zeleň, odpady apod.). Řešitel se stará o to, aby byl problém co nejdříve vyřešen a dává zpětnou vazbu uživateli, který podnět nahlásil. Řešitel může také měnit stav podnětů. Zaměstnanec může být v rámci jedné samosprávy mnohonásobným řešitelem (může zodpovídat za řešení podnětů z více kategorií).
        - *Supervizor:* dostává informace o všech nahlášených podnětech v obci (městské části), ale nemůže jakkoliv měnit jejich stav ani na ně reagovat.
3. Administrátoři aplikace

- Administrátoři aplikace budou mít absolutní moc nad celým systémem.

4. Podněty

   - Podněty budou vytvářeny uživateli a budou obsahovat ID uživatele, který jej zadal, titulek, polohu místa (zeměpisná šířka, zeměpisná délka), časovou známku přidání, časovou známku poslední úpravy, uživatelovu poznámku, poznámku schvalovatele/řešitele, stav, identifikátor kategorie, URL fotografie, název obce, identifikátor obce (městské části) a identifikátor zodpovědné osoby (řešitel/schvalovatel/administrátor samosprávy).
   - Podněty bude možné zobrazit v seznamu a jednotlivé podněty také na mapě.
   - Každý podnět bude mít celkem 4 stavy - čeká na schválení, schváleno, vyřešeno a zamítnuto:
     - *Čeká na schválení:* podnět je odeslaný uživatelem a čeká na schválení schvalovatelem nebo administrátorem obce.
     - *Schváleno:* schvalovatel nebo administrátor schválil uživatelův podnět a přiřadil mu jeho řešitele.
     - *Vyřešeno:* podnět byl úspěšně vyřešen.
     - *Zamítnuto:* schvalovatel nebo administrátor zamítl uživatelův podnět.
   - U každého podnětu bude odkaz do mapy na konkrétní místo, kde se problém nachází.
   - Každý podnět může spadat do jedné z kategorií s tím, že ve výsledném programu budou demonstrativně implementovány pouze některé z nich:
     - *Cesty a chodníky:* cesty, chodníky, cyklotrasy, schody, oplocení, zábradlí a jiné.
     - *Zeleň:* strom, keř, trávník. nepořádek a odpadky, znečištění a jiné.
     - *Odpady:* odpadky, kontejnery, černé skládky a jiné znečištění veřejného prostranství.
     - *Doprava:* značky, semafory, dopravní zrcadla, zpomalovací prahy, zábrany, přechody pro chodce, značení na cestě a jiné.
     - *Mobiliář:* lavička, památník / socha, zastávka MHD, stojan na kola a jiné.
     - *Automobily:* parkování, dlouhodobě odstavená vozidla a jiné.
     - *Veřejné služby:* kanalizace, osvětlení, MHD, webová stránka města, rozvodné sítě a jiné.
     - *Veřejný pořádek:* stavby a budovy, vandalismus, reklama a jiné.

5. Moduly

   - Administrátor samosprávy může v systému vytvářet a aktivovat různé moduly (např. moduly veřejného osvětlení, odpadkové koše, kontejnery s tříděným odpadem, lavičky apod.), díky nimž pak řešitelé získají více informací o nahlášených problémech (v případě osvětlení např. číslo lampy, typ poruchy atd.). Mobilní aplikace se serveru dotáže, jaké moduly jsou v dané samosprávě aktivovány a podle toho uživateli zobrazí vstupy.
   - Moduly vždy spadají do konkrétní kategorie a ke konkrétní samosprávě.

- Moduly budou vytvářeny dynamicky. Každý modul může vlastnit jeden a více různých vstupů. Pro každou kategorii může existovat vícero na sobě nezávislých modulů.

6. Hlášení chyb

   - Uživatelé mobilní i webové aplikace budou moci nahlásit případné chyby v systému pomocí jednoduchého formuláře.
   - Nahlášené chyby si budou moci zobrazovat pouze administrátoři aplikace.

Kapitola 4

Návrh systému

V této kapitole se zaměřuji na celkový návrh systému, popis jeho jednotlivých částí a jejich vzájemných závislostí. Kromě vysvětlení aplikačního rozhraní se v podkapitolách věnuji i návrhu uživatelského rozhraní, databázového modelu a volbě technologií pro implementaci aplikačního serveru a databáze.

Správný návrh systému je další z klíčových částí vývoje jakéhokoliv softwaru. Pokud není návrhu věnováno dostatečné úsilí, zvyšuje se tím riziko možných budoucích problémů v pozdějších částí vývoje a tím i riziko neúspěchu celého projektu.

I když je kvalita návrhu systému častokrát přímo úměrná zkušenostem člověka, který jej navrhuje, pokusil jsem se svůj nedostatek zkušeností alespoň zčásti vykompenzovat zvýšeným úsilím a opřením se o dobře specifikované požadavky na výsledný systém. Přesto musím přiznat, že jsem musel návrh systému několikrát měnit a drobné úpravy byly nutné i v samotné fázi implementace. V kapitole se ovšem nehodlám věnovat popisu, jak se návrh systému v průběhu času vyvíjel, nýbrž se zaměřím pouze na jeho finální verzi.

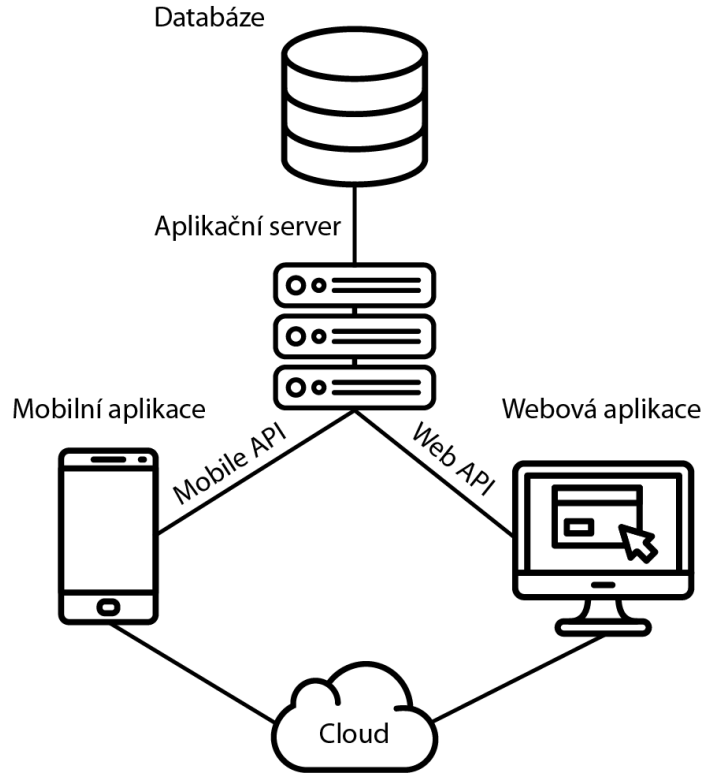
4.1 Architektura systému

Architektura mnou navrženého systému se skládá z prezentační, aplikační a datové vrstvy. Schéma architektury je znázorněno na obrázku 4.1. Zde jsou popsány základní části systému:

1. Aplikační server s API[1]
   - Aplikační server je jádrem celého systému. Slouží k obsluze požadavků od klientů (mobilní a webové aplikace) a komunikuje s databází, uchovávající stav systému.
   - Rohraní API serveru je založeno na architektuře REST[2]. Z důvodu rozdílných potřeb, jak mobilního klienta, tak i toho webového, jsem se rozhodl rozdělit API na dvě části. Cesty pro mobilní API proto začínají slovem „mobile".
2. Databáze a cloud
   - Databáze uchovává stav systému a přístup k ní má pouze aplikační server.
   - V databázi se ukládají data systému (např. uživatelé, podněty, moduly apod.).

[1] Application programming interface (API) je aplikační rozhraní, které umožňuje komunikaci mezi jinak oddělenými částmi aplikace (systému).

[2] Representational State Transfer (REST) je architektura rozhraní, která standardizuje základní serverové operace (zobrazení, čtení, zápis, editování a odstranění) pomocí jednoduchých HTTP volání.



Obrázek 4.1: Schéma architektury navrženého systému.

- Cloud slouží k ukládání fotografií pořízených uživateli mobilní aplikace.

3. Webová administrace
   - SPA[3] vytvořená pomocí front-endového frameworku.
   - Načítá svůj obsah dynamicky pomocí AJAX[4] volání.
4. Mobilní aplikace
   - Android aplikace vyvíjená v programovacím jazyce Kotlin.
   - Uživatelské prostředí je založeno na principech material designu.
   - Data aplikace jsou získávána pomocí asynchronních volání prostřednictvím knihovny Retrofit. Grafické komponenty, které zobrazují data uživatelovi, jsou založeny na principu MVVM[5].

4.2 Návrh uživatelského rozhraní

Jakmile jsem si ujasnil obecné požadavky uživatele na systém a osvojil si princip fungování aplikace, přesunul jsem se k návrhu grafického uživatelského rozhraní (GUI). Jelikož jsem

[3]Single-page application (SPA) je webová aplikace, která dynamicky mění svůj obsah bez nutnosti opětovného načítání všech jejích komponent.

[4]Asynchronous JavaScript and XML (AJAX) je obecné označení pro technologie a metody asynchronní komunikace se serverem ze strany klienta, sloužící k tvorbě interaktivních webových aplikací.

[5]Model-view-viewmodel (MVVM) je návrhový vzor, který slouží k oddělení grafického uživatelského rozhraní, od logiky programu.

již od počátku vývoje usiloval o maximální možnou použitelnost aplikace, snažil jsem se vycházet z již existujících a dobře fungujících schémat.

4.2.1 Výběr barev

Barvy hrají v každodenním životě lidí významnou roli. Některé barvy mají tendenci přitahovat naši pozornost a působí až agresivním dojmem (např. červená barva), jiné barvy na nás naopak působí uklidňujícím dojmem (např. modrá barva). Díky těmto vlastnostem je při tvorbě produktu vhodné používat barvy, které jsou v souladu s jeho vlastnostmi.

Krásným příkladem je značka Ferrari, která má patent na barvu Ferrari Red (#FF2800). Tato agresivní červená barva je v naprostém souladu s produkty firmy, kterými jsou rychlá, sportovní auta s dravým designem. Symbol koně ve značce vozů ještě podtrhuje důraz na rychlost a sílu jejich vozů, na kterou se i tímto způsobem snaží upozornit.

Z tohoto příkladu je zřejmé, že barvy mají nemalou roli v úspěšnosti firemní značky. Proto je třeba se při výběru barev zamyslet nad tím, co je hlavním cílem produktu a jaká je filosofie týmu, který na něm pracuje.

V případě tohoto projektu se aplikace zaměřuje na řešení problémů ve městech a jejím cílem je přispět k lepší organizaci a funkci měst. Filosofie projektu by se tedy dala shrnout do několika bodů:

- zefektivnit a stabilizovat správu měst
- zlepšit čistotu měst
- zapojit občany do veřejného dění
- podpořit komunikaci mezi správou měst a občany
- zvýšit spokojenost občanů žijících ve městech
- poskytnout občanům přehled o kvalitě správy jejich měst

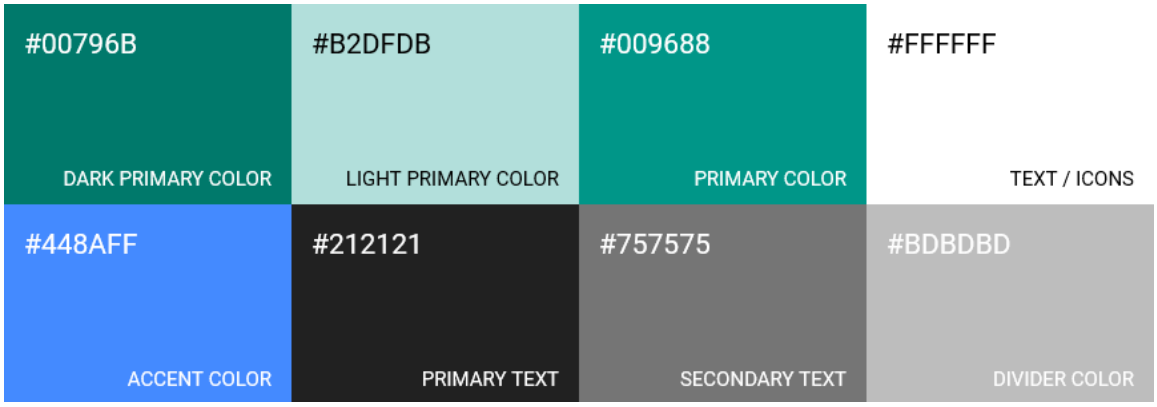
Z výše uvedených bodů lze vybrat klíčová slova, která se posléze použijí při výběru základních barev. Těmito slovy jsou *efektivita, stabilita, čistota, komunikace, spokojenost, kvalita*. Podle teorie psychologie barev[8] jsem vybral barvy, které jsou v souladu s výše uvedenými klíčovými slovy. Těmito barvami jsou:

- *Modrá* – Celosvětově nejoblíbenější barva, která evokuje pocit bezpečí, důvěru, logiku, ale i produktivitu.
- *Zelená* – Barva spojená se zdravím, prosperitou, přírodou.
- *Bílá* – Barva představující pocit čistoty a jednoduchosti.

V další fázi bylo nutné se rozhodnout, jaké barevné schéma pro aplikaci zvolit a přitom vycházet s již zmíněných barev. Při výběru palety barev jsem postupoval podle zásad material designu a k usnadnění práce jsem použil aplikaci Material Design Palette[6]. Výsledná paleta barev je znázorněna na obrázku 4.2.

Po dlouhém zvažování různorodých kombinací jsem se rozhodl pro barvu teal (#009688), která vzniká spojením modré a zelené s vyšším podílem zelené. Jako zvýrazňující barvu jsem podle analogického systému výběru barev zvolil modrou (#448AFF). Bílá barva se skvěle hodí pro pozadí aplikace, a také pro nadpisy na tmavém podkladu.

[6] https://www.materialpalette.com/teal/blue



Obrázek 4.2: Paleta barev aplikace založená na principech material designu.

4.2.2 Mobilní aplikace

Pro Android aplikace se přímo nabízí použití Material Designu, který popisuji v kapitole 2.6.1. Ten je mezi Android vývojáři velmi rozšířen a společnost Google, která Material Design vytvořila, přímo vybízí programátory, aby se jeho principy řídili. Rozhodujícím faktorem pro použití Material Designu je zejména fakt, že se zásadním způsobem zkracuje doba, po kterou se uživatel s aplikací učí pracovat. Tím pádem se zvyšuje pravděpodobnost, že se bude aplikace uživatelovi líbit, a že bude spokojený s jejím rozhraním.

Návrhy aktivit jsem kvůli úspoře času nejdříve načrtl na papír a později překreslil pomocí bezplatného grafického editoru Figma. Tento program je přímo ideální pro tvorbu tzv. mockupů[7], které mohou být provázány pomocí předem definovaných akcí, a poté použity k prototypování.

Po překreslení prvních mockupů jsem se je pomocí prototypovacího nástroje rozhodl provázat a následně otestovat v rámci UX testování[8]. Pro testování jsem oslovil pět dobrovolníků, na kterých jsem testování prováděl. Pět lidí je totiž podle výzkumu Jeffa Saura, Ph.D. schopno odhalit až 85% chyb v uživatelském rozhraní [20].

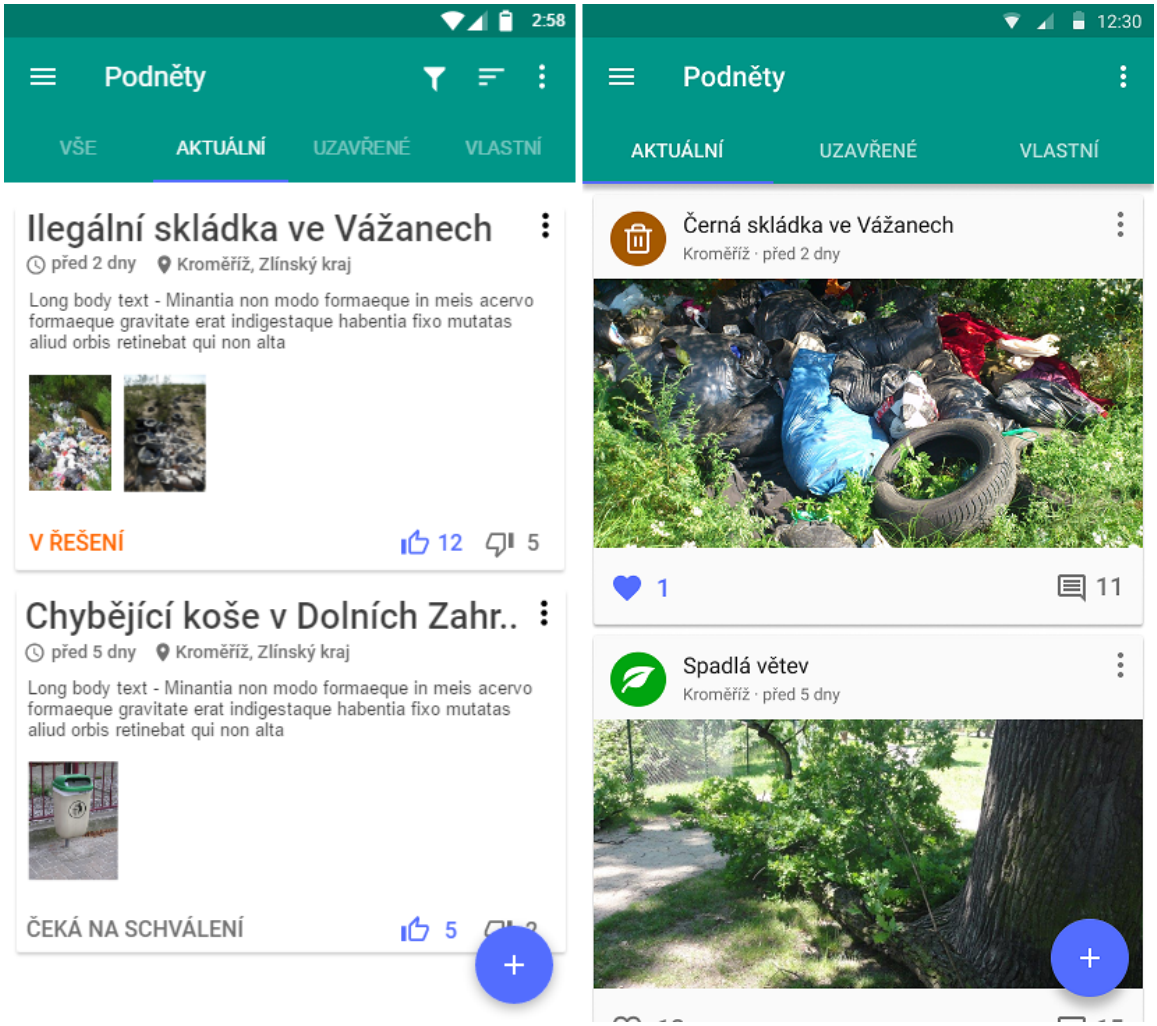
Každý zúčastněný uživatel nejdříve dostal několik jednoduchých úkolů (např. z hlavní aktivity se proklikat až na zobrazení podnětu na mapě), které měli přede mnou v aplikaci splnit. Tito uživatelé nebyli předem seznámeni s tím, jak aplikace funguje, aby výsledky testování nebyly zkresleny. Důležité je také podotknout, že každý z testovaných subjektů dostal úkoly v jiném pořadí, aby výsledky úkolů nebyly pokaždé ovlivněny již předchozí zkušeností. V průběhu testu jsem si dělal poznámky o jeho průběhu.

Po dokončení testování jsem se uživatele vždy zeptal na pár dodatečných otázek, které měly odhalit, co si o aplikaci myslí a jak ji vnímá. Jelikož mě všechny testovací subjekty znaly, požádal jsem je ještě před zodpovězením otázek, aby byli ve svých odpovědích maximálně upřímní a nebáli se mi dát kritickou zpětnou vazbu.

Poznatky, které jsem díky testování získal jsem zapracoval do nového návrhu. Srovnání obou návrhů ilustruji na obrázku 4.3. Zde uvádím několik příkladů chyb z úvodní aktivity, kterých jsem se v prvotním návrhu dopustil:

[7] Mockup je v informatice grafický návrh aplikace, který bývá používán zejména ve fázi prototypování k otestování použitelnosti uživatelského rozhraní.

[8] User experience (UX) můžeme chápat jako dojem, který uživatelovi zůstane z řady interakcí se zařízením, aplikací či jiným produktem. UX testování je pak jednou z metod, jak odhalit chyby plynoucí ze špatného návrhu.



Obrázek 4.3: Srovnání obou návrhů. Před (vlevo) a po zapracování poznatků (vpravo).

- Karta podnětu obsahuje příliš mnoho detailních informací, přičemž jsou velmi nahuštěné a nepřehledné. Popis podnětu je v původním návrhu naprosto nadbytečný, a proto byl přesunut do aktivity s detaily hlášení. Stejně tomu je i s obrázky. Pro ilustraci je daleko lepší jeden větší obrázek než více menších obrázků.
- Velikost textu nadpisu karty podnětu je zbytečně moc velká. Delší nadpisy tak přetékají přes vymezenou šířku textového pole.
- Karta objektu by mohla obsahovat informaci o kategorií, do které podnět spadá. Uživatelé by si tak rychleji osvojili, jaké hlášení spadají do určitých kategorií.
- Záložek je mnoho a jsou natěsnané na sebe. Záložka se všemi podněty byla podle většiny testovacích subjektů nadbytečná. Většinu uživatelů zajímaly především aktuální hlášení a přijde jim dostačující, aby vyřešené podněty byly ve vlastní sekci. Další argumentem se ukázalo to, že po určitém čase by uzavřené podněty jasně převažovaly nad těmi aktuálními a sekce by tak s postupem času přestala dávat smysl.

- Negativní zpětná vazba (palec dolů) by mohla mnoho uživatelů sítě odradit od hlášení dalších podnětů. Je proto lepší použít pouze pozitivní zpětnou vazbu a negativní – ale zato konstruktivní – přenechat správcům obce.
- Lišta obsahuje nadbytek ikon, což podle principů material designu není správně.
- Ikona tří teček je moc kontrastní.

Některé z uvedených chyb se mohou zdát jako drobnosti, ale ve výsledku může každá drobnost rozhodovat o tom, jestli uživatel aplikaci bude nebo nebude používat. Proto jsem si především u hlavní aktivity dal velmi záležet, aby byla uživatelská přívětivost na vysoké úrovni.

4.2.3 Webová administrace

Stejně jako u mobilní aplikace jsem postupoval i v případě webové administrace. Na rozdíl od mobilní aplikace jsem ale daleko větší důraz kladl na funkčnost a její celkovou použitelnost. Uživatelské rozhraní je samozřejmě důležité i u webové aplikace, ale oproti mobilní aplikaci mají uživatelé webové administrace (čili správci obce) daleko větší motivaci aplikaci používat a zároveň mají i více času a prostoru se s aplikací naučit pracovat. To jaký má aplikace obal je proto druhořadé.

U webové administrace také není třeba řešit mnoho věcí, které jsou pro jiné webové aplikace naopak klíčové. Jednou z těchto věcí je například responzivita[9]. Předpokládá se, že zaměstnanci samosprávy budou pracovat na počítači, tudíž není nutné stránku přizpůsobovat například zobrazení na telefonu, hodinkám či chytré televizi.

Jelikož webová aplikace společně s tou mobilní tvoří jeden systém, měl by být design obou aplikací postaven na společných principech. Použil jsem proto stejnou paletu barev jako v případě mobilní aplikace a taktéž jsem se snažil alespoň do určité míry vycházet z Material Designu.

4.3 Návrh databáze

Kvalitní návrh databáze je naprosto klíčová věc, protože jakékoliv pozdější změny v databázovém modelu mají přímý dopad na již existující implementaci. Je tedy třeba dobře zvážit, jaké funkcionality a vlastnosti má systém obsahovat.

V první fázi návrhu databáze jsem vytvořil ER diagram[10] podle specifikace systému. Základní entity systému ilustruji na obrázku 4.4.

Entita uživatele byla zřejmá již od počátku tvorby modelu. Problém ovšem nastal ve chvíli, kdy jsem chtěl začít uvažovat i různé role, kterých uživatelé mohou nabývat. Pro běžného uživatele, který používá pouze mobilní aplikaci bohatě stačí jedna entita. Jak se ale vypořádat s ostatními rolemi jako administrátor, schvalovatel, řešitel a supervizor samosprávy? Situaci navíc komplikuje i to, že jsem se rozhodl systém modelovat tak, aby v něm bylo možné v budoucnu přidávat další samosprávy. No a aby toho nebylo málo, počítal jsem taky s možnou situací, kde by obce mohly zaměstnance navzájem sdílet. To znamená, že by například supervizor obce A byl zároveň řešitelem obce B.

[9] Responzivní web je schopen svůj obsah přizpůsobit různým rozlišením a zařízením.

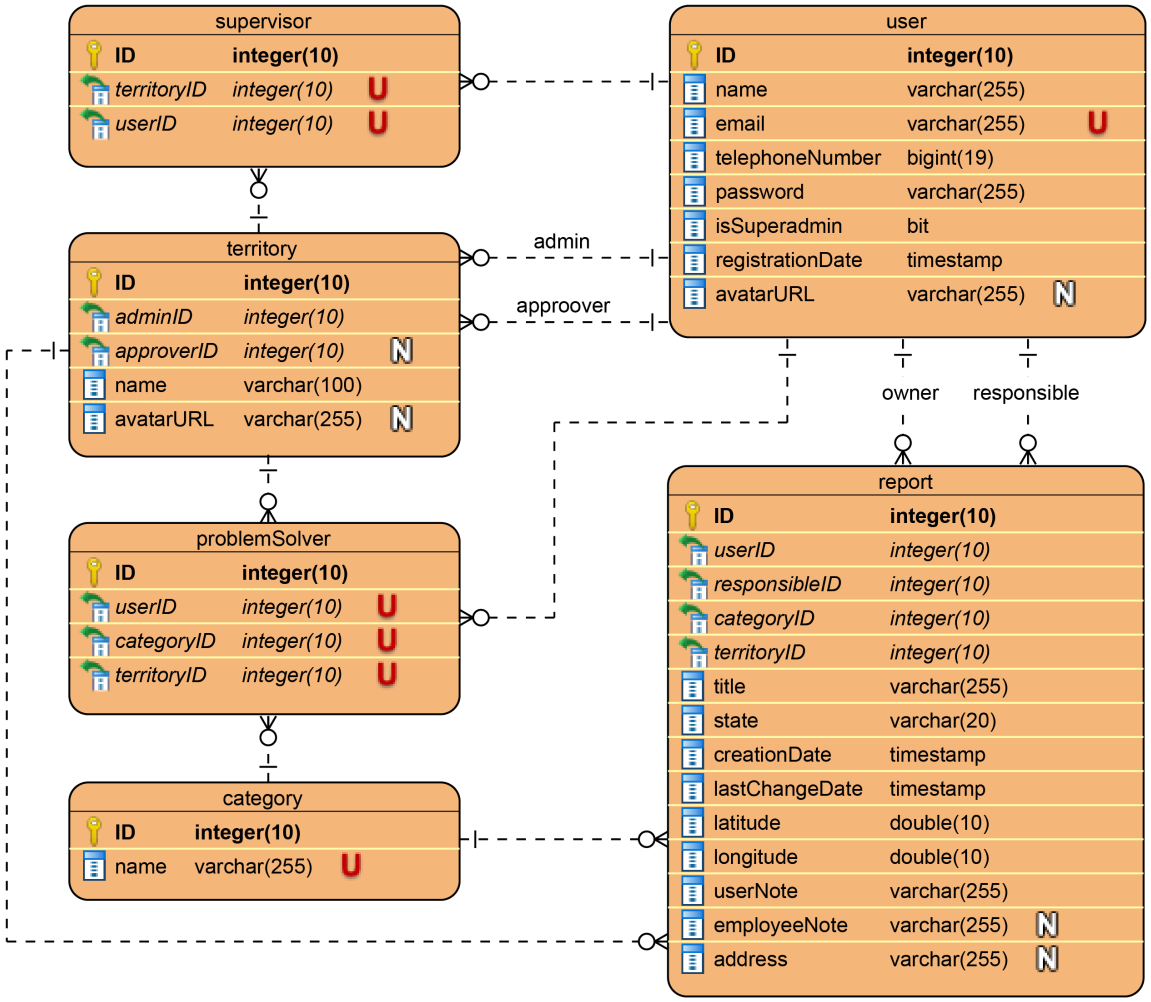
[10] Entity-relationship diagram (ER diagram) slouží k tvorbě abstraktních datových modelů zejména pro účely modelování relačních databází.

V případě administrátora a schvalovatele je řešení poměrně jednoduché. Každá samospráva musí mít právě jednoho administrátora a maximálně jednoho schvalovatele. Stačí tedy přidat do její entity po jednom cizím klíči entity uživatele pro jednu či obě tyto role.

Pro supervizora a řešitele je třeba vytvořit vlastní entity, jelikož každá samospráva jich může mít více. V opačném případě by vznikla vazba M:N, která v relačním databázovém systému není řešitelná.

Z důvodů vytvoření nových entit, které jsou ve vztahu se samosprávami a uživateli (v případě řešitelů i s kategoriemi), je nutné také zajistit, že nebudou vznikat duplicitní záznamy. To lze zajistit pomocí unikátního klíče, který je v případě supervizora složen z identifikátorů uživatele a samosprávy. V případě řešitele je tento klíč doplněn ještě o identifikátor kategorie. Díky unikátnímu klíči nelze do relace přidávat záznamy, u nichž se hodnota tohoto klíče shoduje již s jiným existujícím záznamem.

V případě podnětu opět vzniká problém s uživatelskými rolemi. Jelikož je v zadání specifikováno, že hlášení mohou spravovat za určitých podmínek všichni kromě supervizorů, není možné vytvořit vztah s jednou z těchto entit. Opět se tak musí využít entita uživatele k identifikaci zadavatele a také za řešení zodpovědné osoby. O ošetření rolí se tak v tomto případě stará server v rámci svého aplikačního rozhraní.



Obrázek 4.4: Stěžejní entity systému zakreslené pomocí ER diagramu.

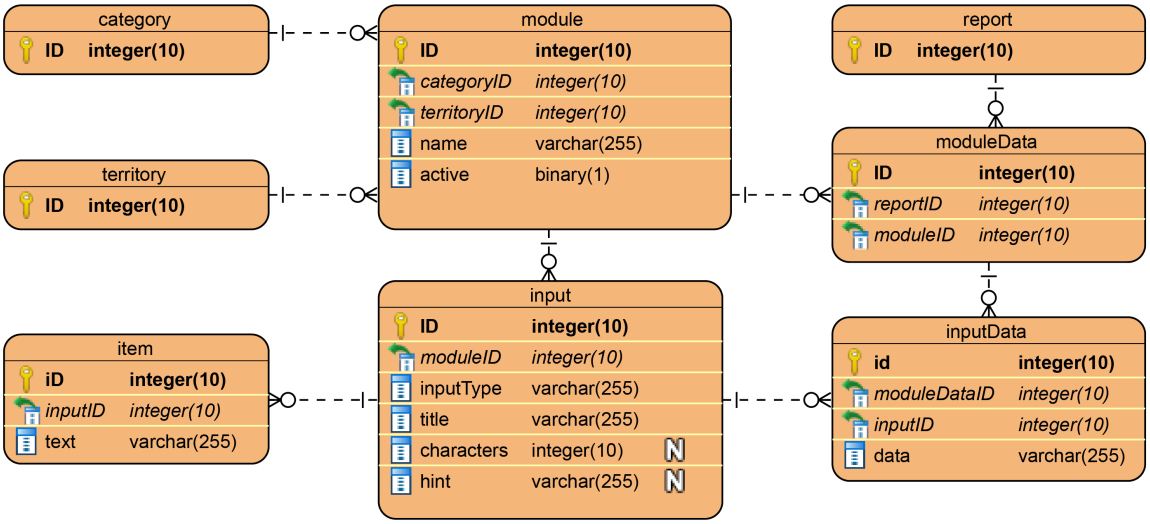
4.4 Návrh modulů

Jedním z hlavním požadavků zadání práce na výsledný systém bylo umožnit administrátorům obcí přidávat, odebírat i aktivovat moduly.

Původní myšlenkou, jak by se dal daný požadavek realizovat bylo vytvoření samostatných a na systému nezávislých skriptů, které by systému umožnily zpracovávat dodatečné informace o nahlášených podnětech. Tohle řešení by ale bylo značně komplikované a obsahovalo by spoustu nedostatků, které mě nakonec přiměly vymyslet řešení nové a lepší.

Nutno také dodat, že jsem po konzultování myšlenky se správci měst dospěl k závěru, že každá obec funguje trochu jinak a není tak zrovna jednoduché vytvořit univerzální řešení modulů, které by vyhovovalo co nejširšímu okruhu obcí. Mým cílem tedy bylo nalézt pokud možno co nejflexibilnější řešení, které by nebylo příliš komplikované a bylo použitelné pro všechny části systému.

Jako ideální a maximálně flexibilní řešení se ukázalo dynamické vytváření modulů přímo administrátory obce. K tomu, aby bylo tohle řešení proveditelné, musel jsem zajistit ukládání modulů i jejich dat v databázi. Proto jsem ER diagram doplnil o další nezbytné entity, které můžete vidět na obrázku 4.5.



Obrázek 4.5: Model ukládání modulů a jejich dat znázorněný v ER diagramu.

Každý modul je vázán k obci, pro kterou je vytvořen a ke kategorii, pod kterou spadá. Každá kategorie může mít až N modulů, proto je zde vazba 1:N. To stejné platí i pro obec. Ta může mít v systému vytvořených až N různých modulů.

Modul je ale jen schránka, která musí obsahovat zvolené vstupy. Opět platí vztah 1:N mezi modulem a jeho vstupy. Vstupy jsou pak dynamicky generovány v mobilní i webové aplikaci a mohou být různých typů. V rámci této práce jsem uvažoval pouze tři vstupy, a to textový řetězec, číslo nebo seznam položek. V případě seznamu položek je pak nutné ještě doplnit model o entitu položky seznamu. Každý vstup musí kromě typu obsahovat informaci o jeho názvu a popřípadě nápovědy. Pokud je typ vstupu textový řetězec, může být specifikována jeho délka. Maximální a výchozí hodnotou je 255 znaků.

Pro potřeby ukládání dat modulů jsou nutné další dvě entity. První entita slouží k identifikaci, ke kterému modulu a podnětu data patří. Opět zde jsou vazby 1:N, protože každý modul může mít až N vlastních dat a zároveň každé hlášení může obsahovat vícero

dat modulů. U hodnot vstupů je situace totožná. Data vstupu je nutné ukládat jako textový řetězec, který je univerzální. Při zpracování dat se musí vždy zjistit, jakého typu data jsou a poté jsou překonvertovány.

4.5 Návrh API

V případě větších projektů, kde je zapotřebí větší množství programátorů, je kvalitní návrh aplikačního rozhraní naprosto zásadní. Pokud je aplikační rozhraní dobře navrženo, je možné jeho implementaci rozvrhnout mezi více programátorů, kteří mohou pracovat na sobě naprosto nezávisle. V situaci, kdy je rozhraní aplikace navrženo špatně a musí se v průběhu implementace měnit, je potřeba počítat s vyššími náklady na projekt a jeho výrazným zpožděním.

V případě mého projektu byla ale situace značně jednoduší. Tím, že jsem na projektu pracoval sám, nemusel jsem návrhu API věnovat takovou pozornost a mohl jsem si jej libovolně přizpůsobovat podle potřeby v průběhu implementace. V praxi to znamenalo, že jsem si určil jasná pravidla, na kterých chci API stavět a definoval si alespoň základní URL cesty rozhraní. Zde uvádím seznam pravidel, které jsem se snažil dodržovat:

- *Pro cesty používat pouze anglické názvy* – Angličtina je ve světě informačních technologií mnohem přirozenějším jazykem než čeština. Mezi její hlavní výhody patří nepoužívání diakritiky a její rozšířenost ve světě informačních technologií. Z hlediska budoucnosti jakéhokoliv IT projektu je vhodné zvážit její užití. Zejména pro případ, když se do projektu zapojí i zahraniční spolupracovníci.

- *Tvořit cesty v závislosti na objektovém modelu* – Adresy cest by měly korespondovat s objektovým modelem. To znamená, že pokud bude jeden objekt náležet druhému objektu, mělo by být z URL adresy zřejmé, ke kterému objektu náleží. Pro lepší pochopení uvedu příklad na objektech samosprávy a podnětu (v modelu označeny jako *territory* a *report*). Podněty spadají pod konkrétní samosprávu, proto je nutné vytvořit tyto cesty:

  - **GET** /territories/{id}/reports – získání seznamu podnětů
  - **POST** /territories/{id}/reports – uložení nového podnětu
  - **GET** /territories/{id}/reports/{id} – získání jednoho konkrétního podnětu
  - **PUT** /territories/{id}/reports/{id} – upravení podnětu
  - **DELETE** /territories/{id}/reports/{id} – odstranění podnětu

  V případě moderních webových frameworků je tvorba dynamických cest jednoduchá a často bývá oddělena od zbylé aplikační logiky. To vývojářům umožňuje budovat rozsáhlá a velmi pokročilá aplikační rozhraní.

- *Použití plurálu* – Jak je z příkladu výše zřejmé, použil jsem u cest anglický plurál. Ten je oproti singuláru mnohem logičtější, a to hned z několika důvodů:

  - Singulár a plurál se u některých anglických podstatných jmen liší (např. person → people).

  - Singulár předpokládá požadavek na jeden konkrétní objekt. Tento objekt se ale stejně musí určit pomocí jedinečného identifikátoru. Použití adresy jako např. */territories/territory/{id}* je zbytečně dlouhé a nedává tak velký smysl. Stejně by tomu bylo i s adresou */territory/{id}*, kde by se samotné *territory* v singuláru odkazovalo na seznam samospráv.
  - Používání plurálu je mezi IT komunitou ustálené a je považováno za vhodné řešení.

- *Logicky pojmenovávat akce, které neodpovídají CRUD* [11] *operacím* – Čas od času se vyskytne situace, kdy je každý programátor aplikačního rozhraní nucen vytvořit cestu, která přímo neodpovídá základním CRUD operacím. V případě mého systému tato situace nastala u modulů. U modulů totiž přibyla operace aktivovat modul, která má upravovat pouze jednu konkrétní položku záznamu. Tento problém jsem vyřešil použitím cesty */territories/territory/{id}/modules/{id}/activate*, která logicky poukazuje na činnost, která je na ní navázána.

- *Používat pouze HTTPS* – V systémech, kde se pracuje s uživatelskými a jinými citlivými daty je šifrovaná komunikace naprosto nezbytná. Proto je nutné všechny cesty zabezpečit pomocí protokolu HTTPS s šifrováním SSL/TLS. Z toho důvodu jsem v souboru *.htaccess* nastavil automatické přesměrovávání z adres s HTTP na adresy s HTTPS.

- *Na chyby reagovat odpovídajícími HTTP stavovými kódy* – pokud v průběhu zpracování požadavku nastane chyba, je třeba o tom uživatele informovat pomocí chybového hlášení. K signalizaci chyb se u HTTP protokolu používají 4XX a 5XX status kódy, které uživateli poskytují informací, o jaký typ chyby se jedná. Pro správnou funkci API je důležité, aby stavové kódy odpovídaly typům chyb, které při zpracování požadavku vznikly.

4.6 Volba technologií

Při vývoji jakéhokoliv softwaru se projekt po čase dostane do fáze, kdy je třeba rozhodnout, jaké technologie budou při jeho vývoji použity. Pro mobilní aplikaci jsem již od počátku kalkuloval s použitím Androidu, a to z důvodu předchozích zkušeností s vývojem aplikací pro tuto platformu a velkému množství zařízení, které ji využívá.

V případě webové aplikace jsem ale jasno neměl. Technologie v oblasti webových technologií jsou velmi rozmanité a každá z nich se hodí na jinou aplikaci. Rozhodnutí, kterou z nich použít obvykle nebývá jednoduché, a je při něm nutné brát v potaz mnoho faktorů. Zde uvádím základní faktory, se kterými se lze při rozhodování nejčastěji setkat, a podle kterých jsem se i já rozhodoval:

- *Velikost projektu* – Je zcela zásadním faktorem. Pro projekty menšího rozsahu lze použít agilnější technologie, které jsou jednoduší a rychlejší z hlediska jejich vývoje. Pro větší projekty je lepší vybrat robustnější technologie, které zajistí dostatečný výkon a nabídnou větší množství funkcí. Výběr zbytečně komplexní technologie pro malý projekt může vyústit v neúspěch celého projektu, jelikož s narůstající komplexitou vzrůstají i náklady na vývoj a jeho doba se výrazně prodlužuje.

[11] CRUD (Create, Read, Update, Delete) jsou základní operace při práci s daty.

- *Staří technologie* – Informační technologie jsou extrémně rychle se vyvíjející obor. Každý rok na trh přicházejí nové nadějné technologie, které mají potenciál ovlivnit trh s vývojem webových aplikací. Ne vždy se tak ale stane. Velmi často se stává, že se kolem nové technologie vytvoří haló efekt, který po čase rychle vyprchá a nikdo si pak na danou technologii ani nevzpomene. U nových technologií se je proto třeba mít na pozoru a slepě nenásledovat první vlnu nadšených uživatelů. Na druhou stranu příliš staré technologie mohou mít problémy s údržbou a z hlediska budoucnosti projektu je zde riziko, že se přestanou používat. Je proto důležité sledovat trendy vývoje aplikací a odhadovat budoucí vývoj.

- *Komunita vývojářů* – Komunita vývojářů je naprosto zásadní zejména v prostředí open-source softwaru. Početná komunita obvykle předznamenává, že se jedná o rozšířenou a do budoucna perspektivní technologii. Výhodou komunity je i znalostní báze a snadná dohledatelnost řešení chyb, s kterými se již někdo v minulosti setkal.

- *Testovatelnost* – Testování je jednou ze stěžejních fází vývoje jakékoliv aplikace. Možnost vytvářet automatické testy je velmi žádoucí pro uchování stability aplikace i při její narůstající komplexitě.

- *Údržba* – Snadná údržba aplikace je důležitá a přímo ovlivňuje její budoucí cenu. Ta obvykle představuje jeden z nejvyšších nákladů projektu a s postupem času roste.

- *Škálovatelnost* – Škálovatelnost v oblasti webových aplikací dělíme do dvou oblastí:

  - *Horizontální* – Schopnost aplikace se přizpůsobit vzrůstajícímu počtu uživatelů.
  - *Vertikální* – Aplikace může být rozšířena o další komponenty bez ovlivnění těch stávajících a bez výrazného vlivu na její výkon.

- *Bezpečnost* – Každá webová aplikace by měla být schopna odolat kybernetickým útoků, a to zejména v oblasti ochrany citlivých dat uživatelů. Proto je důležité vybírat takové technologie, u nichž je bezpečnost prioritou, a které neposkytují mnoho prostory pro bezpečnostní chyby.

- *Dokumentace* – Kvalitní dokumentace je základem každé kvalitní technologie a je určující pro její rychlou adopci vývojáři.

- *Licence* – Při vývoji nové aplikace je také důležité dodržovat licenční podmínky použitých technologií. V případě porušení licence může časem dojít až k softwarovému auditu, jehož výsledkem může být vysoká pokuta.

4.6.1 Laravel

Po zvážení všech výše uvedených bodů jsem se pro implementaci back-endové části systému rozhodl využít PHP frameworku Laravel. Zde jsou důvody, které mě k tomuto rozhodnutí přiměly:

- *Široká komunita programátorů* – Laravel se v posledních letech stal nejpoužívanějším PHP frameworkem a vyrostla kolem něj velká komunita vývojářů. Oblíbenosti se těší zejména kvůli strmé učící křivce a kvalitní architektuře.

- *Kvalitní dokumentace a tutoriály* – Laravel si zakládá na kvalitní dokumentaci a poskytuje vlastní série tutoriálů Laracast. V rámci Laracastu je natočeno již více než 1500 videí a nové stále přibývají. Je ovšem nutné dodat, že zadarmo jsou pouze základní tutoriály a za ostatní se platí. Cena tutoriálů ale není nikterak vysoká a vzhledem k jejich kvalitě je odpovídající.

- *Podpora MVC* – Jednou z hlavních předností Laravelu je struktura projektu a separování jeho logické části od té prezentační. Tohle rozdělení odpovídá modelu MVC, a proto se i méně zkušený programátor v projektu rychle zorientuje. Díky kvalitní architektuře, která přímo nutí programátora dodržovat osvědčené programovací postupy, je možné dosáhnout vysokého výkonu, a to i u větších projektů.

- *Bezpečnost* – U mnoha frameworků se bezpečnost aplikace musí řešit zvlášť a často tak dochází k bezpečnostním chybám. Laravel ovšem sám o sobě obsahuje bezpečnostní opatření (např. hašované hesla pomocí algoritmu Bcrypt, validace zabraňující SQL injekci apod.), které usnadňují programátorům práci a zvyšují zabezpečení aplikace.

- *Artisan* – Laravel obsahuje zabudovaný nástroj nazvaný Artisan. Tento nástroj poskytuje velké množství nástrojů, které usnadňují častokrát se opakující operace. Například lze pomocí tohoto nástroje vytvářet nové soubory, optimalizovat cache, zobrazovat nastavené cesty, generovat záznamy v databázi atd.

- *Integrace Vue.js* – U novějších verzí Laravel automaticky instaluje balíčky front-endového frameworku Vue.js. Ne, že by nebylo možné Laravel spojit s jinými front-endovými frameworky, ale vývojáři se zkrátka rozhodli, že podpoří perspektivní projekt, který je rozvíjen čistě a jen open-source komunitou.

4.6.2 Vue.js

Jelikož jsem zvolil Laravel jako back-endový framework, bylo poměrně logické zvolit Vue.js pro front-endovou část. Díky podpoře od Laravelu existuje v Laracastu série tutoriálů, které se zaměřují právě na Vue.js. Existuje ovšem mnoho dalších důvodů, proč zvolit právě tento progresivní framework:

- *Minimální velikost* - Úspěch Javascriptového frameworku závisí na jeho finální velikosti. Toho si jsou vývojáři Vue.js vědomi, a proto jej navrhli tak, aby bylo množství přenášených dat co nejnižší. Ve výsledku se velikost pohybuje řádově níže než u konkurenčního Reactu či Angularu.

- *Jednoduchost* - Jednou z hlavních výhod Vue.js je jeho jednoduchost. I to je zřejmě důvod, proč si jej v poslední době oblíbilo tolik vývojářů.

- *Flexibilita* - Vue.js poskytuje spoustu prostoru pro uzpůsobení projektu pro jeho individuální potřeby. Díky tomu je skvělou volbou pro vývoj multiplatformních aplikací.

- *Dokumentace* - Na oficiálních stránkách frameworku lze nalézt velké množství podrobných návodů, které usnadňují začínajícím programátorům rychle překonat znalostní bariéru.

4.6.3 MySQL

Pro ukládání a správu dat jsem se rozhodl zvolit MySQL. MySQL je systém řízení báze dat, vlastněný společností Oracle. Jedná se o open-source, vysoce výkonný software, který je vhodný pro různorodé databázové aplikace.

Původně jsem se rozhodl použít MySQL verze 8.0.11, ale po problémech, které nastaly během nasazování systému na produkční server, jsem byl nucen systém přizpůsobit verzi 5.6.12. Tyto problémy více popisuji v kapitole 5.1.7.

Pro MySQL 8 jsem se nerozhodl náhodou. K jeho volbě mě vedlo hned několik důvodů, z nichž ty hlavní popisuji zde:

- *Podpora prostorových funkcí* – Vzhledem k tomu, že má systém pracovat s lokalizací a do budoucna by mohl podporovat i řazení a vyhledávání podnětů podle lokality, podpora prostorových funkcí je naprosto klíčová. MySQL 8 přichází hned s celou řadou nových funkcí a značně usnadňuje práci s prostorovými datovými typy. Také přidává podporu SRID[12], které v předchozích verzích chybělo a podepisovalo se to na přesnosti výpočtu prostorových funkcí.

- *Podpora a vylepšení InnoDB* – InnoDB sice MySQL podporuje již od verze 5.5, ale až ve verzi 5.7 se přidala podpora prostorového indexování, která má značný pozitivní vliv na rychlost vyhledávání prostorových dat. Tyto data také nejsou ukládány ve formátu BLOB[13], ale v samostatném typu DATA_GEOMETRY.

- *Výkonnost InnoDB* – U MySQL verze 8 došlo k signifikantnímu nárůstu výkonu zápisu i čtení při vysoké zátěži. Při vysoké zátěži dosahuje MySQL 8 až dvojnásobného výkonu oproti předchozí verzi MySQL 5.7[11].

[12] Spatial Reference System Identifier (SRID) je unikátní číslo, které jasně identifikuje referenční souřadnicový systém (spatial reference system).

[13] Binary large object (BLOB) je datový typ blíže nespecifikovaných binárních dat v databázi.

Kapitola 5

Implementace

V této kapitole se věnuji implementaci systému, a to jak mobilní aplikace, webové administrace i aplikačního rozhraní. V kapitole také podrobněji rozebírám detaily použitých technologií a na příkladech vysvětluji jejich princip.

Implementace je po specifikaci a návrhu dalším krokem ve vývoji softwaru. Kvalitní návrh softwaru je naprosto stěžejní pro hladký proces implementace a kvalitu výsledného softwaru. Špatný návrh aplikace téměř vždy vyústí v neúspěch projektu anebo přinejmenším v jeho kompletní reimplementaci.

S vlastním návrhem systému jsem byl poměrně spokojen, ale i přesto jsem v průběhu implementace narazil na množství problémů, které zbrzdily vývoj systému. I těm se budu věnovat v kapitole implementace.

5.1 Aplikační rozhraní a databáze

Aplikační rozhraní (API) neboli tzv. back-end je jádrem systému, které vytváří prostředníka mezi prezentační vrstvou (front-endem) a datovou vrstvou (databází). Tato část systému běží na HTTP serveru (Apache, IIS, NGINX) často společně s databází. Jakožto prostředník má několik úkolů:

- Přijímat, zpracovávat a odpovídat na klientovy požadavky
- CRUD operace nad databází
- Zabezpečovat přístup k datům pomocí autorizace
- Určovat cesty rozhraní
- Validovat data získaná z front-endu

Jak už bylo zmíněno v kapitole 4.6.1, rozhodl jsem se k tvorbě aplikačního rozhraní použít PHP framework Laravel, který běží na serveru Apache.

5.1.1 Instalace a inicializace prostředí

Jako první jsem musel zprovoznit lokální HTTP server s MySQL databází a nainstalovaným PHP verze 7.2. V systému Microsoft Windows stačí nainstalovat aplikaci Laragon[1], která

[1] https://laragon.org/

výše uvedený software obsahuje a stará se o jeho spouštění. Laragon také umožňuje vytvořit nový Laravel projekt, a to bez nutnosti použití příkazové řádky. Poté jsem stáhl a v aplikaci změnil MySQL z verze 5.7 na 8.0 a nastavil přesměrování adresy *http://localhost* do */public* adresáře projektu. Následně bylo třeba v prohlížeči otestovat, zda server běží a zda je správně nastaveno přesměrování.

Po zprovoznění základního nastavení bylo třeba upravit soubor *.env*, který obsahuje hodnoty globálních proměnných. V tomto souboru lze nastavit např. jméno aplikace, režim aplikace, připojení k databázi nebo přístupové klíče, které mají být skryty okolí aplikace.

V případě používání verzovacího systému (Git, CVS apod.) je třeba zajistit, aby tento a další konfigurační soubory, data apod. byly při verzování ignorovány. V případě, že by tyto soubory by ignorovány nebyly a soubory byly ukládány veřejně, klíče i data by mohly být odcizeny a zneužity. Zejména v případě přístupových údajů k databázi, které jsou v souboru také uloženy, spočívá velké riziko odcizení či modifikaci citlivých dat.

V mém případě jsem využil verzovacího systému Git, u kterého jsem v *.gitignore* zadal tyto cesty:

```
/node_modules
/public/hot
/public/storage
/storage/*.key
/vendor
/.idea
.env
.phpunit.result.cache
Homestead.json
Homestead.yaml
npm-debug.log
yarn-error.log
```

5.1.2 Směrování

V rámci směrování HTTP požadavků jsou v Laravelu důležité dva soubory, a to *web.php* a *api.php*. Tyto soubory se nacházejí ve složce */routes*. V prvním zmíněném souboru se obvykle nachází cesty k prezentační části webové aplikace (jako odpověď se předává HTML stránka) a v druhém zmíněném se nacházejí cesty aplikačního (datového) rozhraní aplikace (odpověď ve formátu JSON).

Jelikož jsem se rozhodl implementovat webovou administraci jako SPA, která funguje na bázi asynchronního volání, a která přijímá pouze data ve formátu JSON, všechny cesty jsou uvedeny v souboru *api.php*. Do souboru *web.php* bylo ovšem nutné umístit přesměrování všech URL adres na soubor *main.php*, který obsahuje hlavní komponentu front-endového frameworku Vue.js. Přesměrování jsem provedl pomocí následujícího kódu:

```
Route::get('{any}', function () {
    return view('main');
})->where('any', '.*');
```

Z výše uvedeného příkladu je zjevné, jak je v Laravelu snadné cesty vytvářet. Laravel obsahuje třídu *Route*, která abstrahuje směrování do té míry, že lze cestu vytvořit pomocí jediného příkazu. Cesty se definují podle architektury REST (GET, POST, PUT, PATCH, DELETE, OPTIONS). Na kódu níže demonstruji implementaci jednoduché cesty:

```
Route::post('bugs', 'BugController@store');
```

Tato cesta má adresu *http://domena-serveru/bugs* a pro zpětné volání (callback) určuje metodu *store* v kontoléru *BugController*. Tomu, jakým způsobem kontroléry fungují, se věnuji v kapitole 5.1.3.

Kromě základních funkcí, které vychází z REST operací, Laravel přidal i další operace, které usnadňují tvorbu cest. Pro potřeby mé aplikace jsem využíval zejména funkci *apiResource*. Tato funkce vytvoří cesty ke všem základním operacím v kontroléru (index, show, store, update a delete). Výhodou této funkce je zejména zjednodušení a zpřehlední kódu. Ukázka jak definování cest tohoto typu vypadá:

```
Route::apiResource('/territories', 'TerritoryController');
Route::apiResource('/territories/{territory}/reports', 'ReportController');
Route::apiResource('/territories/{territory}/modules', 'ModuleController');
```

Middleware a autentizace

Middleware je další velmi důležitou částí aplikace. Slouží k filtrování HTTP požadavků, které přichází do serverové aplikace. Middleware se například skvěle hodí pro filtrování požadavků od nepřihlášených uživatelů. Díky tomu lze jednoduše zabezpečit ty cesty aplikace, ke kterým nepřihlášení uživatelé nemají přístup. V takovém případě aplikace odpovídá stavovým kódem 401, popřípadě uživatele přesměruje na stránku s přihlašováním.

V případě autentizace je doporučeno používat middleware, který je součástí frameworku a stejně tomu je i s autentizačními metodami a nastavování jejich cest. Ke zprovoznění autentizace jsem nainstalovat balíček *laravel/passport*, který obsahuje potřebné soubory a nastavení.

Po instalaci tohoto balíčku jsem musel nahrát migrace (popisuji v kapitole 5.1.5), které byly frameworkem automaticky vytvořeny. Následně jsem vygeneroval přístupové klíče aplikace, které slouží k bezpečnému generování přístupových tokenů (access_token). Ty jsou posléze předávány v hlavičce každého HTTP požadavku, aby identifikovaly majitele každého příchozího požadavku.

5.1.3 Kontroléry

Kontroléry slouží k oddělení logiky zpracování požadavků od definic cest. To umožňuje lepší škálovatelnost aplikace a především to zlepšuje její přehlednost. Kontroléry aplikačního rozhraní standardně obsahují (ale nemusí nutně) tyto základní funkce:

- *index* (GET) – vrací seznam objektů
- *show* (GET) – vrací jeden konkrétní objekt
- *store* (POST) – uložení zaslaného objektu
- *update* (PUT) – úprava již existujícího objektu
- *delete* (DELETE) – smazání objektu

Velmi užitečnou funkcí Laravelu, kterou jsem hojně využíval, je automatické vázání modelů na parametry cest a následná možnost je získat z parametru funkce. Pro ukázku uvádím příklad na funkci *show* v kontroleru *ReportController.php*.

```
public function show(Report $report){
    return response()->json([
        'report' => $report
    ]);
}
```

Z ukázky kódu výše je zřejmé, že funkce *show* přijímá jako parametr objekt třídy *Report*. Tento objekt framework dodá díky mapování identifikátorů objektů z cest ke konkrétním modelům. O získání objektu z databáze se framework stará na pozadí a provádí ho tzv. Eloquent object-relational mapper (ORM). Ve výsledku se tím zjednoduší a především zefektivní kód kontroléru.

5.1.4 Modely

Modely neboli třídy objektů jsou spravovány Eloquent ORM a jsou uloženy ve složce */app* hlavního adresáře projektu. Modely primárně slouží k definování vazeb s ostatními modely. Tyto vazby umožňují jednoduché mezi-modelové dotazování, o které se Eloquent stará.

Typy vazeb modelů odpovídají relačním vazbám v databázového modelu. V příkladu kódu níže z modelu *Report* je implementován vztah podnětu k uživateli. Platí, že každý podnět náleží právě jednomu uživateli:

```
public function user(){
    return $this->belongsTo(User::class);
}
```

Aby byla vazba obousměrná, musel jsem také v modelu *User* přidat vazbu vůči modelu *Report*. Tímto způsobem jsem implementoval typický relační vztah mezi dvěma relacemi. V tomto případě se jedná o vztah 1:N (User:Report):

```
public function report(){
    return $this->hasMany(Report::class);
}
```

Výše uvedené funkce lze poté díky Eloquent ORM v kontroléru použít k vyhledání objektů modelu, které jsou ve vztahu s modelem objektu, z něhož je funkce volána. Níže je příklad Eloquent dotazu, který jsem použil k získání vlastníka podnětu:

```
$report->user()->first();
```

5.1.5 Migrace

Migrace slouží k jednoduchému verzování databáze. Pomocí nich lze vytvářet a upravovat tabulky v databázi způsobem, který umožňuje sdílet změny v databázovém modelu s ostatními členy vývojářského týmu. Jako příklad uvedu vytvoření nového databázového schématu (tabulky) samosprávy v rámci jedné migrace:

```
class CreateTerritoriesTable extends Migration {
    public function up() {
        Schema::create('territories', function (Blueprint $table) {
            $table->increments('id');
            $table->timestamps();
            $table->string('name');
```

```
            $table->string('avatarURL');
            $table->unsignedInteger('approver_id')->nullable();
            $table->unsignedInteger('admin_id');

            $table->foreign('approver_id')->references('id')->on('users');
            $table->foreign('admin_id')->references('id')->on('users');
        });
    }

    public function down() {
        Schema::dropIfExists('territories');
    }
}
```

Jak je z příkladu zřejmé, každá migrace obsahuje dvě funkce *up()* a *down()*. První zmíněná funkce slouží k migrování změn do databáze, druhá slouží k navrácení změn do stavu před migrováním.

5.1.6 Seed databáze

Seed databáze neboli automatické vložení určitého množství záznamů do tabulek databáze je užitečné zejména pro účely testování v průběhu implementace. Generování záznamů mi při vývoji ušetřilo mnoho času a umožnilo mi to lépe otestovat funkčnost systému. Občas se totiž stává, že je pro otestování aplikace potřeba větší množství dat, jejichž manuální vložení by trvalo velmi dlouhou dobu.

K vygenerování objektů modelu a jejich vložení do databáze jsem použil tovární funkci, která generuje atributy objektu a seeder, který tovární funkci použije k vygenerování určitého počtu záznamů. Zde je ukázka implementace generování uživatelů:

```
public function run(){
    factory('App\User', 10)->create();
}

$factory->define(App\User::class, function (Faker $faker) {
    return [
        'name' => $faker->name,
        'email' => $faker->unique()->safeEmail,
        'email_verified_at' => now(),
        'avatarURL' => "https://res.cloudinary.com/../default-profile.png",
        'telephone' => mt_rand(100000000,999999999),
        'password' => bcrypt('password'),
        'remember_token' => str_random(10),
    ];
});
```

5.1.7 Nasazení na produkční server

Jak už bylo zmíněno v kapitole 4.6.3, potýkal jsem se při nasazení back-endové části systému na produkční server s několika problémy, z nichž jeden zapříčinil částečné omezení funkcionality systému.

Tímto problémem bylo nepodporování verze MySQL 8 produkčním hostingem. Ten podporoval pouze MySQL 5.6.12, který neobsahuje množství prostorových funkcí, které jsem v aplikaci používal. Důsledkem tohoto problému byla nutnost odstranit některé části implementace a zdržení celého projektu.

Tou nejzásadnější změnou byla nemožnost identifikování území, na kterém se nahlášený podnět nachází. Kvůli tomu jsem musel omezit množství samospráv na pouhou jednu samosprávu a všechny nahlášené podněty jí automaticky přiřazovat.

Další změnou bylo odebrání řazení podnětů v hlavní aktivitě podle jejich vzdálenosti od polohy uživatele. Tato změna se na první pohled nezdá být až tolik závažná. Pro koncového uživatele to ale znamená, že se mu budou zobrazovat méně relevantní podněty místo těch, které má ve svém nejbližším okolí.

5.2 Mobilní aplikace

Android mobilní aplikace slouží v systému jako prostředník pro interakci s uživateli, občany měst. Při jejím vývoji jsem proto musel dbát obzvlášť důraz na její intuitivnost a praktické používání. To je velmi důležité hned z několika důvodů:

- *Množství uživatelů rozhoduje o úspěchu aplikace* – u crowdsourcing aplikací[2], kterou tato aplikace bezesporu je, je obzvlášť důležitá vysoká aktivita uživatelů. Bez dostatečné aktivity uživatelů v aplikaci nebudou mít další uživatelé ani správci obcí motivaci aplikaci používat.
- *Uživatel není programátor* – Běžný uživatel má mnohem omezenější intuici, co se používání softwaru týče, než programátor. Ti mají ze zkušenosti lepší představu o tom, jak daný software funguje a dokáží tak lépe předvídat jeho akce. Je proto důležité, aby programátoři dostávali zpětnou vazbu od reálných uživatelů aplikace a software maximálně přizpůsobovali jejich potřebám.
- *Hodnocení aplikace* – Uživatelé mají na Google Play možnost přidat aplikacím jejich hodnocení. Nepraktičnost aplikace se odrazí v negativním hodnocení, které je u mobilních aplikací obzvlášť důležité.

Při programování jsem se proto snažil řídit doporučeními, které jsou dostupné v dokumentaci Androidu a po dokončení aplikace jsem ji nechal běžnými uživateli otestovat. O průběhu testování a jejich výsledcích pojednávám v kapitole 6.2.

5.2.1 Síťová komunikace

Pro komunikaci s aplikačním rozhraním serveru jsem zvolil knihovnu *Retrofit*. Ta umožňuje vytvoření HTTP klienta s REST rozhraním a usnadňuje zpracování JSON a XML dat. Abstrakce a ryze objektový přístup, které Retrofit poskytuje, umožňují programátorům budovat rozsáhlá rozhraní, a to bez ztráty přehlednosti a funkcionality kódu.

5.2.2 Architektura MVVM

Model-view-viewmodel (MVVM) je doporučovaný návrhový vzor, který slouží k oddělení prezentační vrstvy aplikace od té logické. Tohle rozdělení je důležité zejména kvůli asynchronnímu zpracování dat. U systému Android platí, že jedno vlákno aplikace je výhradně

[2]Crowdsourcing aplikace využívají uživatele ke generování jejich obsahu.

používáno pro zobrazování UI. Proto musí být některé operace přesunuty na jiná vlákna, aby neblokovaly grafické rozhraní systému. Jedná se zejména o síťovou komunikaci, nebo jiné služby na pozadí.

Zde ovšem nastává problém. Pokud běží operace asynchronně, musí existovat způsob, jakým upozorní UI vlákno na nutnost aktualizace zobrazení komponent. Právě k tomuhle účelu vznikna architektura MVVM, která výše uvedený problém řeší a je nastíněna na obrázku 5.1.

Obrázek 5.1: Schéma architektury MVVM.

V aplikaci jsem architekturu použil zejména při tvorbě seznamu hlášení, kde se využívá síťové komunikace k získání dat podnětů. Ty jsou zobrazovány v hlavní aktivitě aplikace v komponentě *RecyclerView*. Ta má schopnost dynamicky zobrazovat seznamy totožných komponent (dochází k jejich recyklaci pouhou změnou dat, které zobrazují).

Pro plynulý chod aplikace a také nízký objem přenášených dat je zásadní, aby se podněty načítaly po částech, nikoliv najednou. Přizpůsobit tomu aplikační rozhraní nebyl problém, ovšem situace ze strany Android aplikace byla mnohem složitější.

Řešení jsem našel zkombinováním ViewModelu a knihovny *Paging*[3]. Tato knihovna řadí získané objekty do stránek po určitém počtu (v případě mé aplikace je to 5 podnětů) s tím, že se tyto stránky vytváří postupně podle potřeby. Pokud se uživatel při procházení seznamu dostane do mezního bodu, aplikace požádá server o další data a ty poté použije při vytváření nové stránky. Objekt obsahující stránky podnětů se nachází ve Viewmodelu, aby mohlo UI reagovat na změny jeho stavu.

Proto, aby mohlo být UI aktualizováno, musí jeho aktivita obsahovat tzv. *observer*. Ten naslouchá změnám dat jemu přiřazeného ViewModelu a provádí činnosti spojené s aktualizací UI komponent.

5.2.3 Lokalizace a mapy

K lokalizaci uživatele jsem použil knihovnu *AirLocation*, která dokáže rychle a efektivně získat přesnou pozici zařízení fúzí všech dostupných lokalizačních prostředků. Výhodou knihovny je, že po získání přesné pozice své služby přeruší a zbytečně tak nevytěžuje hardwarové prostředky telefonu. To má pozitivní vliv zejména na výdrž baterie.

Zobrazení podnětu na mapě jsem implementoval pomocí *Google Maps API*, které společnost Google umožňuje v omezené míře používat zdarma. K zprovoznění služby byl nutný unikátní klíč, který identifikuje všechny požadavky aplikace.

[3] https://developer.android.com/topic/libraries/architecture/paging

5.2.4 Databáze

Jedním z předpokladů kvalitní Android aplikace je automatické přihlášení uživatele v případě, že už se v minulosti jednou přihlásil. Žádného uživatele nebaví se při každém spuštění aplikace znovu přihlašovat. Proto jsem potřeboval najít způsob, jak uložit uživatelská data ve vnitřní paměti aplikace, abych je při každém dalším startu aplikace mohl použít k přihlášení.

Jako ideální řešení se ukázalo použití knihovny *Room*, která vytváří abstrakci nad *SQLite*. Po přihlášení uživatele a obdržení jeho podrobnějších dat je vytvořen objekt User, který je skrze databázové rozhraní uložen. Vzhledem k tomu, že některá data uživatele v aplikaci zobrazuji, vytvořil jsem ViewModel k umožnění dynamického zobrazování jeho dat. V případě, že by se v budoucnu implementovala aktivita pro upravování těchto informací, aplikace by na to byla připravena.

5.2.5 Moduly

Jelikož jsou moduly vytvářeny dynamicky, musel jsem vymyslet způsob, jak je v aplikaci zobrazit. Každá samospráva může mít jiné typy modulů, tudíž není možné vytvářet XML šablony jako v případě ostatních aktivit. Rozložení je tedy nutné generovat dynamicky. To sice znamená větší zátěž pro hlavní vlákno aplikace, ale v případě malého množství komponent to nepředstavuje velký problém. Pokud by se jednalo o generování složitého rozložení, bylo by nutné použít jiný, efektivnější přístup, který by zajistil hladký chod aplikace.

Informace o modulech aplikace jsou od serveru vyžádána až po vybrání kategorie hlášení uživatelem. Poté, co aplikace informace obdrží, vygeneruje rozložení modulů na určené místo uvnitř aktivity. Každé pole je označeno identifikátorem, aby z nich bylo možné data získat.

Jakmile uživatel vyplní pole a rozhodne se pokračovat v procesu vytváření nového podnětu, spustí se druhá fáze, a to fáze zpracování dat ze vstupních polí. Pomocí cyklů, identifikátorů modulů a identifikátorů vstupů se data postupně vyberou a uloží do zvláštních objektů. Pro každý modul je vytvořen objekt *ModuleData*, který obsahuje seznam objektů *InputData*. V těchto objektech jsou uložena data, které uživatel zadal.

5.3 Webová administrace

Webová administrace je druhá klientská aplikace, která slouží ke správě hlášení, zaměstnanců a tvorbě nových modulů. Jak již bylo zmíněno v kapitole 4.6.2, rozhodl jsem se ji implementovat pomocí frameworku Vue.js.

5.3.1 Směrování

Pro budování SPA je nutné v rámci front-endu zajistit směrování aplikace respektive namapovat URL cesty na patřičné stránky. Toho jsem docílil přidáním oficiálního Vue balíčku *Vue-router*. Mapování cest na Vue komponenty jsem poté nastavoval v souboru *router.js* .

Vue-router sám o sobě neřeší autorizaci, a proto jsem musel najít způsob, jak zabránit neautorizovaným uživatelům v přístupu na stránky, na které mohou pouze autorizovaní uživatelé. Jako první jsem ke každé cestě přidal informaci, zda je pro zobrazení jejího obsahu nutné být přihlášen. Při každé změně cesty jsem zajistil, aby se jako první volala

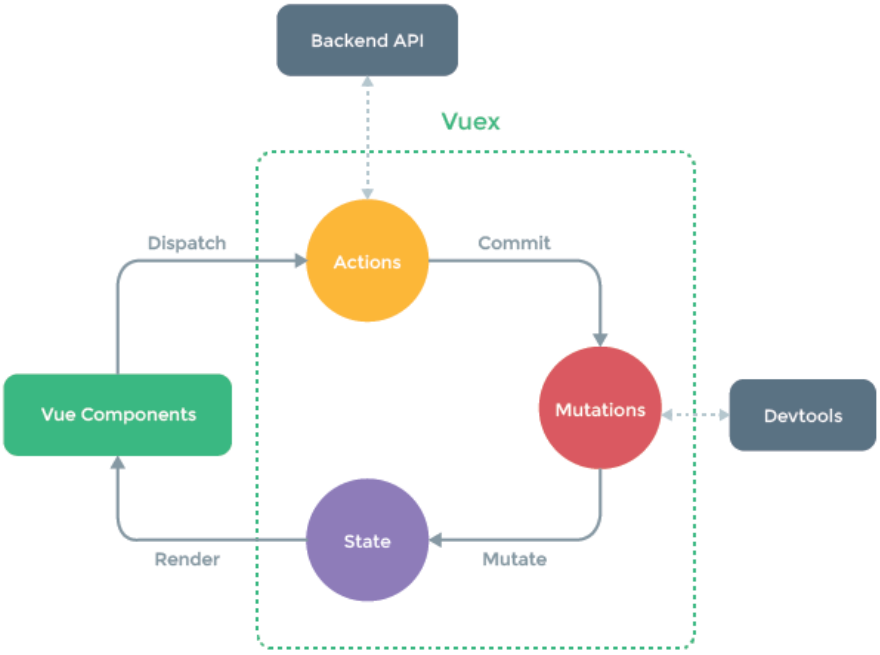
funkce, která kontroluje stav přihlášení. V případě, že uživatel chce použít cestu, ke které je nutné být přihlášen, funkce ho přesměruje na stránku s přihlašováním.

V případě, že se uživatel snaží přistoupit k datům, ke kterým nemá přístup, objeví se chybové upozornění a uživatel je přesměrován na výchozí stránku.

5.3.2 Data aplikace

Jedním ze specifických rysů každé SPA je nutnost správy vnitřních dat aplikace. Některé data se nemusejí opakovaně získávat ze serveru a je daleko výhodnější je v rámci aplikace uložit a později v případě potřeby z paměti načíst. V některých případech je dokonce nutné, aby byly data uložena a nebyla při každém opětovném načtení ztracena. Takovým případem jsou například informace o přihlášeném uživateli a jeho přístupovém tokenu, který je předáván při každém síťovém volání. Pokud by byly tyto data ztraceny, uživatel by se ocitl odhlášen po každém načtení stránky. Další problém představuje sdílení globálního stavu aplikace mezi všechny jeho komponenty a zajištění dynamické aktualizace na sobě nezávislých komponent.

Vue.js tyto problémy řeší pomocí dalšího oficiálního balíčku *Vuex*. Vuex přináší způsob, jakým lze uchovávat globální stav aplikace a jak jej sdílet napříč nezávislými komponentami. Schéma principu je znázorněno na obrázku 5.2.

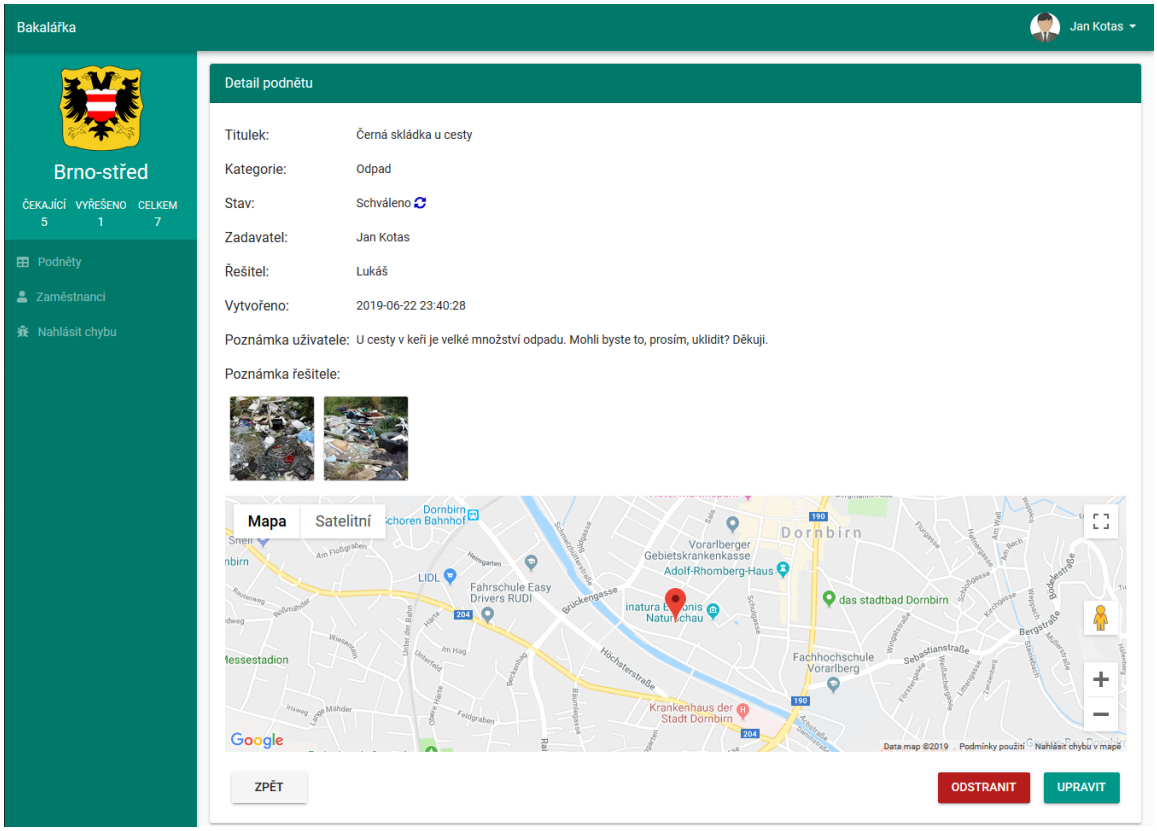


Obrázek 5.2: Schéma řízení globálního stavu aplikace pomocí Vuex. Komponenty Vue mohou iniciovat akce, které komunikují s rozhraním serveru a přijímají od něj požadovaná data. Tyto akce probíhají asynchronně na pozadí. Akce volají mutace, které jsou synchronní, a které kontrolují změny globálního stavu. Převzato z [10].

K uchování stavu webové administrace jsem použil jeden soubor *store.js*. V tomto souboru jsou deklarovány všechny globální proměnné a všechny důležité funkce spadající pod *mutace* a *akce*.

5.3.3 Design

Design webové administrace je sice podřízen její funkčnosti, ale i přesto jsem se snažil, aby byl uživatelsky přívětivý a v souladu s designem mobilní aplikace. Z toho důvodu jsem při implementaci hojně využíval komponent z balíčku *Vuetify*, které uplatňují principy Material Designu. Výsledný design aplikace ilustruje obrázek 5.3.



Obrázek 5.3: Stránka detailu hlášení z webové administrace. Kromě základních informacích o podnětu jsou k dispozici i fotografie a přesná poloha místa.

Kapitola 6

Testování

Tato kapitola se zabývá testováním systému, a to jak aplikačního rozhraní, tak i mobilní a webové aplikace. Testování probíhalo v průběhu vývoje systému i po jeho dokončení. Po dokončení vývoje proběhlo testování i na reálných uživatelích, kteří mi poskytli zpětnou vazbu v rámci dotazníků.

Testování je nedílnou součástí vývoje jakéhokoliv produktu, ať už se jedná o počítačový software nebo prášek na praní. Cílem každého testování je odhalit potenciální chyby produktu ještě předtím, než jej začnou používat zákazníci. Kritika, která by se v takovém případě na produkt snesla by mohla mít drastický dopad na jeho oblíbenost a možná i pověst firmy.

V případě softwaru je řešení chyb v porovnání s hmotnými produkty obvykle jednoduší. Existují ale i případy, kdy selhání programátora vedlo k fatálním škodám. Asi nejznámějším takovým příkladem je softwarová chyba vedoucí k destrukci rakety Ariane 5 včetně jejího nákladu, kde se výše škod pohybovala v řádu stovek milionů eur.

6.1 Aplikační rozhraní

Správná funkčnost API je stěžejní pro chod celého systému. Proto jsem jejímu testování věnoval nejvíce pozornosti, aby případné chyby neovlivňovali implementaci klientských částí systému.

6.1.1 Postman

Hlavním programem, který jsem po celou dobu vývoje rozhraní používal byl *Postman*. Tato aplikace umožňuje vytvářet kolekce dotazů, sdílet mezi nimi informaci o přístupovém tokenu přihlášeného uživatele, testovat výstupy pomocí skriptů a další užitečné funkce.

6.1.2 Telescope

Druhým důležitým nástrojem, který jsem využíval zejména při testování komunikace mezi klientskými aplikacemi a aplikačním rozhraním, byl *Telescope*. Tento nástroj z dílny vývojářů frameworku Laravel sleduje síťovou komunikaci přicházející do aplikačního serveru a její detaily poskytuje programátorovi v separátní webové aplikaci.

Telescope ovšem neobsahuje pouze informace o síťové komunikaci, ale i o celkové vitálnosti serveru. K dispozici jsou zde logy aplikace, výjimky, seznam databázových dotazů a doba trvání jejich exekuce a mnoho dalších užitečných funkcí.

V průběhu implementace mi tento nástroj ušetřil mnoho práce s laděním aplikace a častokrát mi velmi rychle pomohl odhalit vzniklé chyby.

6.2 Mobilní aplikace

Pro testování Android aplikací poskytuje Google nástroje v rámci své webové konzole pro správu aplikací. Z konzole je možné vytvářet testovací skupiny pro uzavřené testování, ale i alfa testování na širší skupině uživatelů. Výhodou je zejména možnost rychlého nasazování nových opravených verzí, které se testerům automaticky stáhnou po připojení k internetu.

Z počátku je vhodné testovat na menší skupině uživatelů, protože ta odhalí většinu závažných chyb. Jakmile je verze aplikace stabilní, je vhodné ji otestovat co nejširším počtem uživatelů, kteří odhalí zbylé a do té doby skryté nedostatky.

V mém případě jsem aplikaci testoval na skupině deseti uživatelů, které jsem přihlásil do interního testování. Testování nemělo žádný pevný scénář, pouze jsem testerům zadal obecné úkoly, které měli během testování splnit:

- Registrovat a přihlásit se do aplikace
- Zobrazit si jakýkoliv nahlášený podnět
- Zobrazit si podnět na mapě
- Přidat nové hlášení

Během plnění úkolů si každý z testerů měl zapisovat postřehy, jaké změny by bylo možné v aplikaci provést a případně hlásit chyby, které při testu nastaly. Po dokončení testování jsem nechal všechny zúčastněné vyplnit formulář, jehož cílem bylo změřit jejich celkovou spokojenost s používáním aplikace.

Ve formuláři jsem nejdříve pokládal obecné otázky ohledně spokojenosti uživatele s aplikací. Předpokládal jsem, že kdybych tyto otázky položil až na konci formuláře, hodnocení by mohlo být ovlivněno odpověďmi na předchozí otázky. V dalších částech hodnocení jsem se zaměřoval na konkrétní sekce aplikace.

Výsledky hodnocení aplikace uvádím v tabulkách 6.1, 6.2, 6.3 a 6.4.

Oblast	Hodnocení
Záměr aplikace	4.5/5
Provedení aplikace	4.1/5
Intuitivnost používání	4.0/5
Design aplikace	4.2/5

Tabulka 6.1: Obecné hodnocení aplikace

Oblast	Hodnocení
Ovládání sekce	4.5/5
Design sekce	4.1/5

Tabulka 6.2: Hodnocení hlavní aktivity se seznamem podnětů.

Oblast	Hodnocení
Ovládání aplikace	4.2/5
Design aktivit	4.5/5
Pořadí/rozložení aktivit	4.6/5

Tabulka 6.3: Hodnocení aktivit spojených v rámci procesu přidávání nového hlášení.

Oblast	Hodnocení
Ovládání sekce	4.6/5
Design sekce	3.9/5

Tabulka 6.4: Hodnocení detailu hlášení.

Jedním z bodů průzkumu, kterému jsem se věnoval v kapitole 3.2, zkoumal maximální možnou dobu, kterou jsou uživatelé ochotni přidání nového hlášení věnovat. Proto jsem se rozhodl v průzkumu uživatelů zeptat, jak dlouho jim přidání podnětu trvalo, abych získaná data mohl porovnat s daty z průzkumu.

Z výsledků vyplynulo, že všichni uživatelé strávili přidáním podnětu maximálně 5 minut. V porovnání s výsledky průzkumy se ukázalo, že doba přidání podnětu je přijatelná pro přibližně 3/4 potenciálních uživatelů.

6.3 Webová administrace

Testování webové aplikace probíhalo zejména v průběhu vývoje a zaměřovalo se především na funkčnost aplikace, nikoliv na její design či ovládání. Abych si byl jist, že všechny implementované komponenty správně fungují, vytvořil jsem si kontrolní seznam, který jsem při testování postupně procházel. V případě, že jsem narazil na chybu, poznačil jsem si ji a pokračoval v testování. Po dokončení testování jsem odhalené chyby opravil a testování provedl znovu. Tento cyklus jsem opakoval až do chvíle, kdy jsem prošel všemi body bez nalezení chyby.

Iterativní testování jsem zvolil z důvodu četných závislostí Vue komponent na stav systému. V případě některých komponent tak existovalo riziko, že by změny způsobené opravou starých chyb mohly vytvořit chyby nové. Tohle riziko se sice nepotvrdilo, ale díky opakování testů jsem alespoň narazil na chyby, kterých jsem si předtím nevšiml.

6.4 Návrh na další vývoj

Již během vývoje projektu jsem se setkával s velmi pozitivní zpětnou vazbou jak ze strany obcí, tak i ze strany občanů. Ve veřejném průzkumu i v průzkumu existujících řešení se ukázalo, že v současné době na trhu neexistuje žádný silný hráč, který by podobné řešení nabízel. Byla by proto škoda ve vývoji dále nepokračovat a nepokusit se aplikaci dovést do fáze, kdy by mohla být použita v ostrém provozu. Proto jsem se rozhodl shrnout základní body, které by v budoucnu mohla aplikace nabízet.

6.4.1 Prostorové funkce

Jak již bylo zmíněno v předchozích kapitolách, využití prostorových funkcí je pro úspěch aplikace stěžejní. Systém by měl být schopen vyhledávat a řadit podněty podle jejich lokace,

přiřazovat podněty území, na kterém se nachází a podle lokace upozorňovat na možné duplicitní podněty. Vzhledem k tomu, že jsou tyto operace poměrně náročné na výkon databázového serveru, je třeba zvážit možnosti MySQL a najít vhodné řešení pro škálování systému.

V rámci mobilní i webové aplikace by mohla být přidána mapa, která by zobrazovala okolní podněty. Ta by uživatelům poskytla lepší přehled o tom, kolik hlášení v obci je a v kterých jejích částech se nacházejí.

6.4.2 Uživatelská sekce

Webová i mobilní aplikace aktuálně postrádají uživatelskou sekci s informacemi o přihlášeném uživateli a možností upravovat osobní údaje. V produkční aplikace musí mít uživatel možnost si změnit základní údaje jako např. heslo nebo emailovou adresu. V případě registrace i přihlášení by měl být uživatelovi odeslán potvrzovací email, který by jeho emailovou adresu ověřil.

6.4.3 Kategorie

Aplikace nyní obsahuje pouze pět hlavních kategorií. Kromě doplnění hlavních kategorií by systém mohl podporovat víceúrovňové kategorie, které by měly stromovou strukturu. To by umožnilo blíže specifikovat typ problému. Administrátoři obcí by to jistě ocenili a pomohlo by jim to i v případě cílení modulů. Ty v případě obecných kategorií mohou být nadbytečné a mást tím uživatele.

6.4.4 Přehled a statistiky

Webová administrace by měla mít na úvodní stránce přehled s důležitými informacemi o aktuálním stavu samosprávy. Tento přehled by mohl obsahovat i statistiky, jak byly podněty přidávány v čase a jak se v průběhu času měnil jejich stav. Další možnou funkcionalitou je vytváření tepelných map, z kterých by správci obcí mohli vyhodnocovat, které části měst jsou nejvíce problematické. Tyto informace by pomohly predikovat možné budoucí problémy, což by vedlo k lepší připravenosti měst na jejich řešení, případně jejich úplnému předcházení.

6.4.5 Interakce uživatelů

Už v návrhu aplikace jsem kalkuloval s přidáním prvků, které jsou typické zejména pro sociální sítě, a které zvyšují interakci uživatelů. Těmito prvky jsou komentáře a tlačítka hodnocení obsahu. Díky těmto prvkům by ostatní uživatelé měli možnost přidávat vlastní postřehy k nahlášeným podnětům, což by administrátorům mohlo poskytnout více doplňujících informací. Dalším přínosem by byla větší motivace uživatelů přidávat kvalitní podněty. Komentáře by také přispěly k lepší zpětné vazbě od správců měst vůči uživatelům mobilní aplikace, což by přidalo uživatelům na pocitu, že má hlášení podnětů smysl.

Kapitola 7

Závěr

Cílem této bakalářské práce bylo navrhnout a implementovat systém pro správu problémů v obci. Systém se měl skládat z mobilní uživatelské a webové administrátorské aplikace. Součástí práce byl i návrh a implementace dynamického vytváření modulů pro sběr dodatečných informací o nahlášených problémech.

Na počátku byl proveden uživatelský průzkum, který měl odhalit, jaké formy hlášení uživatelé nejčastěji preferují a za jakých podmínek by byli ochotni k hlášení používat mobilní aplikaci. Poté byly analyzovány existující řešení a zkušenosti od lidí, kteří se danou problematikou zabývají. Tomu předcházelo několik konzultací na Magistrátě města Brna a na Odboru organizační správy a informatiky v Uherském Hradišti. Získané informace posloužily k vytvoření specifikace požadavků na systém.

Podle specifikace byl sestaven návrh systému, a to jak aplikačního rozhraní, tak i mobilní a webové aplikace. Navržen byl také systém přidávání a správy modulů. Po dokončení návrhu byly vybrány technologie pro implementaci aplikačního serveru (Laravel) a webové administrace (Vue.js). Pro účely uchování a správy dat byla zvolena technologie MySQL.

Po dokončení návrhu a zvolení stěžejních technologií byla zahájena implementace systému. Nejprve došlo k implementaci serverové části, která s klienty komunikuje prostřednictvím REST API, a která spravuje stav systému. Poté byla implementována Android aplikace, která slouží k hlášení problémů v obcích. V rámci ní bylo nutné vyřešit mnoho problémů jako například generování modulů, lokalizace, dynamické zobrazování seznamu podnětů nebo asynchronní komunikace se serverem. Poslední přišla na řadu implementace webové administrace. Ta byla implementována pomocí frameworku Vue.js a byla koncipována jako jednostránková aplikace (SPA). To umožnilo výrazně zrychlit odezvu aplikace a zlepšit tak uživatelskou zkušenost.

Posledním bodem práce bylo otestování mobilní aplikace na vzorku běžných uživatelů a vyhodnocení jejich uživatelské zkušenosti. Všichni testeři proto vyplnili formulář, v kterém hodnotili jak jejich celkovou zkušenost s aplikací, tak i provedení jejích jednotlivých částí. Po vyhodnocení výsledků byly sepsány možné rozšíření systému, a to zejména ty, které by byly pro jeho úspěšnost v budoucnu stěžejní.

Vzhledem k velmi pozitivní odezvě i samotnému potenciálu projektu hodlám v projektu dále pokračovat. Vzhledem k vysoké komplexnosti systému, která je dána tím, že je složen hned z několika rozsáhlých částí, je ale nutné zvážit zapojení dalších osob do procesu vývoje.

Literatura

[1] Allen, G.: *Android 4: průvodce programováním mobilních aplikací*. Brno: Computer Press, 2013, ISBN 978-80-251-3782-6.

[2] Android Open Source Project: *Android versions distribution*. [Online; navštíveno 30.06.2019].
URL https://developer.android.com/about/dashboards

[3] Android Open Source Project: *Content License*. [Online; navštíveno 22.12.2018].
URL https://source.android.com/setup/start/licenses

[4] Android Open Source Project: *Light and shadows*. [Online; navštíveno 29.12.2018].
URL https://material.io/design/environment/light-shadows.html

[5] Android Open Source Project: *Platform Architecture*. Září 2018, [Online; navštíveno 22.12.2018].
URL https://developer.android.com/guide/platform

[6] Android Open Source Project: *Understand the Activity Lifecycle*. Prosinec 2018, [Online; navštíveno 22.12.2018].
URL https://developer.android.com/guide/components/activities/activity-lifecycle

[7] Breslav, A.: *Kotlin 1.0 Released: Pragmatic Language for JVM and Android*. Únor 2016, [Online; navštíveno 23.12.2018].
URL https://blog.jetbrains.com/kotlin/2016/02/kotlin-1-0-released-pragmatic-language-for-jvm-and-android/

[8] Brian Lischer: *The Psychology of Color in Branding* . [Online; navštíveno 04.04.2019].
URL http://www.ignytebrands.com/the-psychology-of-color-in-branding

[9] Elgin, B.: *Google Buys Android for Its Mobile Arsenal*. Srpen 2005, [Online; navštíveno 21.12.2018].
URL https://web.archive.org/web/20110205190729/http://www.businessweek.com/technology/content/aug2005/tc20050817_0949_tc024.htm

[10] Evan You: *What is Vuex?* . [Online; navštíveno 25.06.2019].
URL https://vuex.vuejs.org/

[11] Geir Hoydalsvik: *What's New in MySQL 8.0?* . [Online; navštíveno 20.05.2019].
URL https://mysqlserverteam.com/whats-new-in-mysql-8-0-generally-available/

[12] Honig, Z.: *Google intros Android Studio, an IDE for building apps*. Květen 2013, [Online; navštíveno 22.12.2018].
URL https://www.engadget.com/2013/05/15/google-android-studio/

[13] JetBrains: *Kotlin Language Documentation*. [Online; navštíveno 23.12.2018].
URL https://kotlinlang.org/docs/kotlin-docs.pdf#__WKANCHOR_1hc

[14] Manjoo, F.: *A Murky Road Ahead for Android, Despite Market Dominance*. Květen 2015, [Online; navštíveno 21.12.2018].
URL https://www.nytimes.com/2015/05/28/technology/personaltech/a-murky-road-ahead-for-android-despite-market-dominance.html

[15] McDonald, N.: *Digital in 2018: World's internet users pass the 4 billion mark*. Leden 2018, [Online; navštíveno 20.12.2018].
URL https://wearesocial.com/us/blog/2018/01/global-digital-report-2018

[16] Mew, K.: *Learning Material Design*. Birmingham: Packt Publishing Ltd., 2015, ISBN 978-1-78528-981-1.

[17] Red Hat, Inc.: *What is open source?* . [Online; navštíveno 10.01.2019].
URL https://opensource.com/resources/what-open-source

[18] Rogers, R.; Lombardo, J.; Mednieks, Z.; aj.: *Android Application Development: Programming with the Google SDK*. Beijing: O'Reilly, 2009, ISBN 978-0-596-52147-9.

[19] Rubin, A.: *Where's my Gphone?* Listopad 2007, [Online; navštíveno 21.12.2018].
URL https://googleblog.blogspot.com/2007/11/wheres-my-gphone.html

[20] Sauro, J.; Lewis, J. R.: *Quantifying the user experience*. Cambridge, MA: Morgan Kaufmann, druhé vydání, 2016, ISBN 978-0128023082.

[21] Spacey, J.: *What is a Design Language?* Srpen 2016, [Online; navštíveno 29.12.2018].
URL https://simplicable.com/new/design-language

[22] van der Wielen, B.: *Insights into the 2.3 Billion Android Smartphones in Use Around the World*. Leden 2017, [Online; navštíveno 20.12.2018].
URL https://newzoo.com/insights/articles/insights-into-the-2-3-billion-android-smartphones-in-use-around-the-world

Příloha A

Obsah CD

- **android-app** – Zdrojové soubory Android aplikace a instalační soubor APK.
- **web-app** – Zdrojové soubory webové aplikace.
- **doc** – Technická zpráva
  - **src** – Zdrojové soubory technické zprávy pro LaTeX.
  - **xkotas07.pdf** – Technická zpráva ve formátu PDF.

Příloha B

Návod na instalaci

Webová aplikace

- Zkopírujte všechny soubory ze složky **web-app**.
- Otevřete soubor **.env** v kořenovém adresáři a upravte přístupové údaje k MySQL databázi.
- Nainstalujte všechny závislosti pomocí příkazu **composer install**. Tento příkaz musíte spouštět v kořenovém adresáři.
- Nyní je třeba spustit migrace, které vytvoří tabulky v databázi. To se provede pomocí příkazu **php artisan migrate:fresh --seed**. Přepínač **--seed** vygeneruje záznam samosprávy a jejího administrátora.
- K tomu, aby aplikace správně fungovala je ještě třeba vygenerovat unikátní klíče aplikace. Toho se dosáhne spuštěním příkazu **php artisan passport:install --force**.
- Nyní se lze do systému přihlásit pomocí emailu **test@test.cz** a hesla **12345678**.

Android aplikace

- Zkopírujte soubor **app.apk** ze složky se zdrojovými soubory Android aplikace do mobilního zařízení se systémem Android verze 6 (Marshmallow) a výše.
- V nastavení zabezpečení systému povolte instalaci aplikací z neznámých zdrojů.
- Spusťte soubor **app.apk**, čímž zahájíte instalaci aplikace v telefonu.
- Po instalaci můžete aplikaci spustit kliknutím na její ikonu v menu. Důležité je ale dodat, že aplikace komunikuje pouze s produkčním serverem, který běží na adrese `https://jankotas.cz`. Pokud se tedy pomocí mobilní aplikace registrujete, vaše registrace bude provedena na produkčním, a nikoliv lokálním serveru.