VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
FACULTY OF INFORMATION TECHNOLOGY

ÚSTAV INFORMAČNÍCH SYSTÉMŮ
DEPARTMENT OF INFORMATION SYSTEMS

MOBILNÍ APLIKACE PRO SPRÁVU SERVISNÍCH POŽADAVKŮ CHYTRÝCH MĚST

MOBILE APPLICATION FOR MANAGEMENT OF SMART CITY SERVICE REQUESTS

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE — **Bc. KLÁRA FORMÁNKOVÁ**
AUTHOR

VEDOUCÍ PRÁCE — **Ing. JIŘÍ HYNEK, Ph.D.**
SUPERVISOR

BRNO 2023

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
FAKULTA INFORMAČNÍCH TECHNOLOGIÍ

Zadání diplomové práce

146395

Ústav:	Ústav informačních systémů (UIFS)
Studentka:	**Formánková Klára, Bc.**
Program:	Informační technologie a umělá inteligence
Specializace:	Softwarové inženýrství
Název:	**Mobilní aplikace pro správu servisních požadavků chytrých měst**
Kategorie:	Mobilní aplikace
Akademický rok:	2022/23

Zadání:

1. Prostudujte koncept chytrých měst (*Smart City*). Zaměřte se na problematiku komunikace samospráv s obyvateli, způsoby hlášení závad ve městech, tvorbu a řešení servisních požadavků.
2. Prostudujte principy systémů řízení požadavků (*issue tracking systems*).
3. Seznamte se s principem tvorby multiplatformních aplikací pro mobilní zařízení. Prostudujte dostupné technologie a proveďte srovnání.
4. Proveďte analýzu současného stavu podpory pro hlášení závad ve městech, tvorbu a řešení servisních požadavků na platformě firmy Logimic. Uvažujte náklady na budoucí údržbu a rozšiřitelnost aplikace.
5. Navrhněte mobilní aplikaci na platformě firmy Logimic řešící požadavky plynoucí z analýzy v bodě 3.
6. Navrženou aplikaci implementujte.
7. Ve spolupráci s firmou Logimic otestujte použitelnost mobilní aplikace. Navrhněte možná rozšíření.

Literatura:

- Greengard, S. (2015). *The Internet of Things*. MIT Press.
- Kirimtat, A., Krejcar, O., Kertesz, A., & Tasgetiren, M. F. (2020). Future trends and current state of smart city concepts: A survey. *IEEE access*, *8*.
- Janák, J. (2009). *Issue Tracking Systems*. Online: https://is.muni.cz/th/pbpa1/. Diplomová práce. Masarykova univerzita, Fakulta informatiky. Brno.
- Bertram, D. (2009). The social nature of issue tracking in software engineering. *Calgary, Alberta, Canada*.
- Falessi, D., Hernandez, F., & Khosmood, F. (2018). Issue Tracking Systems: What Developers Want and Use. In *ICSOFT* (pp. 577-582).
- Johnson, J. (2010). *Designing with the Mind in Mind: Simple Guide to Understanding User Interface Design Guidelines*. Morgan Kaufmann Publishers/Elsevier.
- Interní dokumentace firmy Logimic.

Při obhajobě semestrální části projektu je požadováno:
Body 1 až 5.

Podrobné závazné pokyny pro vypracování práce viz https://www.fit.vut.cz/study/theses/

Vedoucí práce:	**Hynek Jiří, Ing., Ph.D.**
Vedoucí ústavu:	Kolář Dušan, doc. Dr. Ing.
Datum zadání:	1.11.2022
Termín pro odevzdání:	17.5.2023
Datum schválení:	18.10.2022

Abstrakt

Cílem této práce je ve spolupráci s firmou Logimic vytvořit mobilní aplikaci pro správu servisních požadavků v chytrých městech. Servisní požadavky reprezentují činnosti, které je ve městě třeba provést, přičemž za jejich správu jsou zodpovědní manažeři, kteří požadavky vytváří a řídí, a techničtí pracovníci, kteří požadované činnosti provádí. Záměrem této práce je zanalyzovat současný stav podpory pro správu servisních požadavků na platformě pro chytrá města a navrhnout a implementovat řešení pro efektivní správu servisních požadavků ve formě mobilní aplikace pro platformy Android i iOS. Výsledkem práce je multiplatformní aplikace vytvořená pomocí nástrojů Ionic a Capacitor, která je současně integrována jako modul platformy pro chytrá města. Pro efektivnější práci se servisními požadavky bylo umožněno např. jejich řazení do grafové struktury, slučování duplicitních požadavků a v mobilní aplikaci byly zavedeny některé typické mobilní prvky. Důležitým přínosem této práce je ověření toho, že pomocí nástrojů Ionic a Capacitor lze webovou aplikaci snadno transformovat do multiplatformní.

Abstract

The aim of this thesis is to develop a mobile application for managing service requests in smart cities and it is done in collaboration with Logimic company. Service requests represent activities that need to be performed in a city, where managers are responsible for managing them and technicians are responsible for executing them. This thesis intends to analyze the current state of support for service request management on a smart city platform and to design and implement a solution for effective service request management in the form of a mobile application for both Android and iOS platforms. The result of the work is a multiplatform application developed using Ionic and Capacitor frameworks, which allows more efficient work with service requests by organizing them into a graph structure, merging duplicate requests and introducing some typical mobile elements. The resulting application is also integrated as a module of the smart city platform. An important contribution of this thesis is the verification that using frameworks Ionic and Capacitor, a web application can be easily transformed into a multiplatform one.

Klíčová slova

multiplatformní vývoj aplikací, mobilní aplikace, hybridní aplikace, Ionic, Flutter, React Native, Angular, iOS, Android, IoT, chytrá města, Logimic

Keywords

multiplatform app development, mobile app, hybrid app, Ionic, Flutter, React Native, Angular, iOS, Android, IoT, smart city, Logimic

Citace

FORMÁNKOVÁ, Klára. *Mobilní aplikace pro správu servisních požadavků chytrých měst.* Brno, 2023. Diplomová práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce Ing. Jiří Hynek, Ph.D.

Mobilní aplikace pro správu servisních požadavků chytrých měst

Prohlášení

Prohlašuji, že jsem tuto diplomovou práci vypracovala samostatně pod vedením pana Ing. Jiřího Hynka, Ph.D. Další informace mi poskytly Ing. Michal Valný, Ph.D., Ing. František Mikulů a Ing. Petr John. Uvedla jsem všechny literární prameny, publikace a další zdroje, ze kterých jsem čerpala.

.....
Klára Formánková
27. července 2023

Poděkování

Moje poděkování patří panu Ing. Jiřímu Hynkovi, Ph.D. za odborné vedení, ochotu a spoustu užitečných rad po celou dobu práce. Dále bych ráda poděkovala firmě Logimic za možnost podílet se na řešení tak zajímavého problému a stát se alespoň dočasně součástí jejich týmu.

Obsah

Kapitola 1

Úvod

Servisní požadavky jsou důležitou součástí webové platformy dodávené firmou Logimic do chytrých měst. Každý servisní požadavek reprezentuje žádost o provedení nějaké činnosti ve městě, jako např. oprava nesvítící lampy nebo vybudování nové lavičky v parku. Správa servisních požadavků je komplexní problém a ve městě může zaměstnávat řadu lidí. Z nich někteří zaujímají roli technického pracovníka, tedy roli osoby, která dokáže požadované činnosti provádět, a jiní roli manažera, tedy roli osoby zodpovědné za tvorbu a přidělování požadavků. Manažer také kontroluje, jak probíhá řešení činností, a rozhoduje o tom, jestli byla činnost provedena správně. Většinu své pracovní doby tak manažer i technický pracovník tráví správou servisních požadavků.

Firma Logimic v současné chvíli nabízí webový modul pro práci se servisními požadavky jako součást dodávané platformy. Mezi zákazníky se ale čím dál častěji objevuje poptávka po mobilní aplikaci, která by zjednodušila práci se servisními požadavky na mobilních telefonech. Vytvoření takové aplikace je cílem této diplomové práce. Současně je však záměrem zanalyzovat současný stav podpory pro správu servisních požadavků a navrhnout a implementovat aplikaci tak, aby odpovídala požadavkům uživatelů a vylepšovala existující webové řešení. Mobilní aplikace by měla být zaměřena na servisní požadavky, měla by být dostupná uživatelům operačních systémů Android i iOS a měla by přinést výhody typické pro mobilní aplikace, jako je přímý přístup k fotoaparátu mobilního zařízení, offline dostupnost nebo možnost zasílání oznámení.

K naplnění cíle této diplomové práce je třeba provést několik kroků, kterým odpovídá struktura této práce. První tři kapitoly popisují teoretický základ potřebný k řešení. Postupně se věnují oblasti chytrých měst, systémům pro řízení požadavků na změny a možnostem vývoje multiplatformních aplikací pro mobilní zařízení, kde jsou představeny multiplatformní technologie Flutter, React Native a Ionic. Následuje kapitola 5 analyzující současný stav podpory pro práci se servisními požadavky, ve které jsou zároveň definovány požadavky na vznikající mobilní aplikaci. Kapitola 6 se zabývá návrhem mobilní aplikace, včetně návrhu změn ve způsobu práce se servisními požadavky. Kapitola 7 pak popisuje samotnou implementaci mobilní aplikace a kapitola 8 shrnuje, jak byla výsledná aplikace testována.

Kapitola 2

Chytrá města

Koncept chytrých měst (*Smart city*) je v Evropě znám při nejmenším od 90. let 20. století, kdy se začaly informační a komunikační technologie dostávat do povědomí širokého publika [10]. Vznik konceptu souvisel s rozvojem internetu, který přinesl řadu zajímavých možností pro využití i ve městech. Vedení inovativních měst se snažila najít způsoby, jak internet, a s ním přicházející technologie, využít ke zlepšení kvality života ve městě. V roce 2006 pak podle průzkumu Urban Audit[1] [37] Evropa čítala stovky měst s označením „chytrá“ a jejich počet během tří let výzkumu neustále narůstal. Rostoucí trend vykazují chytrá města i po roce 2006. Co je však podstatnější, dále se rozvíjí používané technologie, koncept je stále populárnější, život v chytrých městech nabývá větší kvality a řízení takových měst je čím dál více efektivní [10].

Podle Evropské komise[2] [30] je chytré město místo, kde jsou služby a infrastruktura zefektivněny pomocí digitálních řešení ve prospěch jeho obyvatel a podniků. Digitální technologie se v prostředí chytrého města podílí na zajištění lepšího využívání zdrojů a snížení emisí. To zahrnuje chytré dopravní sítě, modernizaci zásobování vodou, efektivnější likvidaci odpadu, účinnější osvětlení a vytápění budov, interaktivní městskou správu, bezpečnější veřejná prostranství apod. Další zdroje nabízí i jiné definice pojmu „chytré město“. Např. v rámci Středoevropské konference v regionální vědě (*Central European Conference in Regional Science*) v roce 2009 přinesli Andrea Caragliu a spol. [10] definici, která poukazuje na důležitost lidského faktoru v rámci chytrého města:

> *Věříme, že město je chytré, když investice do lidského a sociálního kapitálu a tradiční (dopravní) a moderní komunikační infrastruktury podporují udržitelný hospodářský růst a vysokou kvalitu života s moudrým hospodařením s přírodními zdroji prostřednictvím participativního vládnutí.*

Jiná definice [25] říká, že vývoj chytrého města je řízen tak, že město vyniká v klíčových oblastech, jako je ekonomika, mobilita, životní prostředí, lidé, bydlení a vedení. Na tyto oblasti uplatnění se zaměřují i další zmíněné zdroje a jejich popisu je věnována sekce 2.1.

Chytrá města ke svému fungování využívají mobilní počítačové systémy, které jsou síťově propojeny pro správu dat mezi všemi složkami a vrstvami samotného města. Propojení a správu dat v dnešních chytrých městech zajišťují technologie: internet věcí (IoT, *Internet of Things*), *big data* a cloudové výpočetní technologie (*Cloud computing*) [25], které jsou popsány ve druhé sekci této kapitoly.

[1] Urban Audit – název databáze Evropského statistického úřadu, která si vzala za cíl srovnávat kvalitu života ve městech Evropské unie

[2] https://commission.europa.eu/index_cs

2.1 Oblasti uplatnění

Chytré město stojí na několika klíčových oblastech, jež jsou řízeny jako „chytré“. Jde o oblasti běžného města, které jsou však řízeny v souladu s principy chytrého města. To zahrnuje především zapojení technologií pro síťové propojení a správu dat a zefektivnění služeb a infrastruktury za pomocí digitálních řešení. Lze se tak setkat s pojmy jako chytrá ekonomika nebo chytré životní prostředí. Klíčové oblasti s uplatněním principů chytrého města se mohou napříč městy lišit v závislosti na požadavcích a prioritách. Ve většině chytrých měst, kde je snahou využít technologie komplexně, je ale součástí klíčových složek těchto šest oblastí: lidé, ekonomika, vedení, mobilita, životní prostředí a bydlení [25].

2.1.1 Lidé

Významným cílem chytrých měst je zvýšit kvalitu života, především kvalitu života obyvatel města, jinak označovaných také jako „chytří lidé“. Oblast „lidé“ se zaměřuje přímo na obyvatele, a to nejen jako na uživatele chytrých zařízení a služeb města, ale také na obyvatele jako účastníky řízení města. Kvalita života obyvatel závisí na různých faktorech, které může zavedení a rozvoj principů chytrého města v oblasti „lidé“ ovlivnit. Mezi takové aspekty života patří např. kvalita poskytovaných informací a systémů, soukromí, bezpečnost nebo vzdělávání lidí o systémech a službách chytrého města [25].

Počátky zavádění principů chytrých měst byly zaměřeny pouze na obyvatele v roli uživatelů a nepřipouštěly jejich aktivní účast na řízení města. Hlavní snahou bylo poskytnout lidem možnosti pro rozvíjení svého potenciálu a pro zapojení se do společenského dění. Byly proto využívány technologie pro podporu komunikace mezi obyvateli a nabízeny příležitosti pro osobní vzdělávání a rozvoj. To znamenalo např. začátek využívání sociálních sítí nebo pořádání online kurzů a konzultací [19]. Chytří lidé by však měli nabízené systémy a služby nejen využívat, ale měli by pro ně také poskytovat data. Díky tomu pak mohou vznikat aplikace získávající znalosti z dat a předávající tyto znalosti zpět obyvatelům pro jejich užitek. Příkladem může být aplikace, která ze senzorů v chytrých telefonech a ze zadaných informací od uživatelů sbírá data o počasí. Ta následně vyhodnocuje a prezentuje, a poskytuje tak přesnější údaje o stávajícím a budoucím počasí. Podobně může fungovat aplikace zprostředkovávající aktuální informace o dopravě ve městě. I po zavedení sbírání dat pro potřeby vyhodnocování znalostí, jsou obyvatelé stále pouze v roli uživatelů chytrých systémů a služeb města [25].

Tradičně byl přístup k inovacím ve městech řízen shora dolů [34]. Tedy způsobem, kdy za rozhodováním stojí vedení města a obyvatelé přicházející inovace pouze přijímají a používají. Koncept chytrých měst přinesl model, v němž jsou od obyvatel aktivně získávány podněty a nápady, které jsou následně využity vedením při řízení města. Díky tomu lze strategii města lépe přizpůsobit očekáváním obyvatel a označit tento princip jako řízení zdola nahoru. Způsobů, jak získávat názory obyvatel, existuje velké množství, z nichž většinu je možné zařadit do jedné ze tří základních kategorií: *přímá interakce*, *živá laboratoř* a *online platformy* [34].

- **Přímá interakce** sjednocuje tradiční techniky sbírání názorů, které vyžadují fyzickou účast obyvatel. Může jít o rozhovory s odborníky, skupinové diskuze, setkání na radnicích nebo testování použitelnosti a funkčnosti nových systémů.
- **Živá laboratoř** označuje otevřený inovační ekosystém založený na partnerství podniků, obyvatel a vedení města, který umožňuje obyvatelům aktivně se podílet na výzkumu a vývoji inovací. Účastníci živé laboratoře jsou zapojeni do analýzy požadavků,

diskuzí o nápadech, konkrétního vývoje nápadů a nakonec i do testování vzniklých prototypů.

- **Online platformy** pomáhají oslovit větší počet obyvatel, než by bylo možné prostřednictvím metod přímé interakce a živé laboratoře. online platformy jsou nejméně časově a prostorově náročným způsobem získávání názorů a nabízí mnoho různých podob použití. Běžným příkladem je využití sociálních sítí, nástrojů pro dotazování, nástrojů pro spolupráci nebo centralizovaných platforem, které umožňují shromažďovat názory a zkušenosti obyvatel k určitým veřejným záležitostem.

Jakkoliv jsou názory obyvatel pro vedení přínosné a potřebné, je i nadále nutné uvažovat podněty odborníků a zkušených pracovníků [34].

Pro chytré město je zásadní, aby obyvatelé dokázali využít jeho potenciál. Proto je potřeba, aby byli otevření a snadno se přizpůsobovali měnícímu se prostředí. Aby byli vzděláváni, a to jak v oblasti používání systémů a služeb města, tak v oblasti zákonů a politik prostředí. A v neposlední řadě, aby byli kreativní, podíleli se na proměnách města a aby město budovali tak, jak si sami představují [25].

2.1.2 Ekonomika

Další důležitou oblastí chytrého města je chytrá ekonomika. Cílem řízení ekonomických složek jako „chytrých“ je zvýšit konkurenceschopnost města, což může být v různých městech naplňováno různými způsoby. Jedním ze způsobů je vznik tzv. inovačních ekosystémů, uvnitř kterých mohou začínající i zavedené podniky a další zájmové skupiny spolupracovat nebo soutěžit, aby efektivně rozvíjely inovační řešení [19]. Jiný způsob se snaží zvýšit konkurenceschopnost města využitím elektronického obchodu, který obvykle slouží jako podpora pro maloobchodníky, aby byli schopni získat větší zájem zákazníků. Takto může být ve městě implementován např. mobilní nákupní systém s nabídkou produktů a služeb i s jejich hodnoceními od předchozích zákazníků. Další součást chytré ekonomiky může být věnována chytrému podnikání. Podporu podnikům může město vyjadřovat prováděním a prezentací různých analýz obchodních dat nebo např. vytvářením prediktivních modelů vztahujících se k rizikům a řízení měnící se situace na trhu. Kromě zmíněného zvýšení konkurenceschopnosti lze chytrou ekonomiku představit také jako oblast se snahou o aplikaci nových technologických řešení se záměrem posílit motivaci k inovacím, podnikavost, produktivitu a flexibilitu trhu práce [25].

2.1.3 Vedení

Vedení chytrého města, neboli *chytré vedení*, někdy také *chytrá vláda*, má dva hlavní cíle [25]:

- usnadnit široké veřejnosti komunikaci s vedením,
- zjednodušit administrativní postupy.

Oba zmíněné cíle různými způsoby využívají principů přinášených konceptem chytrých měst. V rámci komunikace veřejnosti s vedením jsou významným prostředkem sociální média. Jejich používání může znamenat jednoduché předávání informací od vedení veřejnosti, ale také v opačném směru. Lze tak zapojit občany do diskuze na témata týkající se města, získávat odpovědi v anketách ohledně specifických problémů nebo čerpat inspiraci z nápadů veřejnosti. Spolupráce s lidmi z veřejnosti může být následně i prohloubena, a lidé tak mohou být přímými účastníky procesu inovací, jak bylo popsáno v podsekci 2.1.1. Při

komunikaci vedení směrem k veřejnosti umožňují sociální média předávat informace téměř odkudkoli, zpřístupňovat je rychleji, poskytovat více informací a zaujmout větší procento veřejnosti než u tradičních metod. Výsledkem pak může být větší motivace veřejnosti k účasti a spolupráci v chytrých městech [25].

Snahou vedení města by mělo být také zlepšování a zjednodušování administrativních postupů a rozvíjení elektronické veřejné správy tak, aby lidé mohli vyřizovat svoje záležitosti online bez ohledu na otevírací dobou nebo umístění úřadu. To zahrnuje především poskytnutí možnosti k získání formuláře z internetových stránek města, případně možnost formulář elektronicky vyplnit a podat. Elektronicky pak mohou být takové formuláře také zpracovány, vyřízeny a o výsledku může být podávající osoba informována opět elektronicky. Elektronizace jakékoli části tohoto procesu obvykle městu i široké veřejnosti přináší užitek [10].

2.1.4 Doprava

Chytrá doprava, jinak také *chytrá mobilita*, nabízí hned několik přínosů pro města. Mezi zásadní patří dle [2] např.:

- **Snížení zátěže dopravy** a s ní spojená minimalizace dopravních zácp může být výsledkem využívání chytrých přístrojů a pokročilé analýzy získaných dat. Řešení snižování zátěže může zahrnovat synchronizaci semaforů pro optimální tok dopravy, systémy pro předcházení dopravním nehodám, systémy pro oznamování nehod nebo systémy usnadňující parkování.
- **Zkrácení doby jízdy** souvisí s předchozím bodem. Zavedení zmiňovaných systémů umožňuje cestujícím plánovat trasy podle aktuálních informací o dopravě. Chytré městské dopravní sítě navíc dokáží lidi nasměrovat, kdy a kde je např. potřeba přestoupit, aby se do cíle dostali s nejnižšími náklady nebo v nejkratším čase.
- **Zlepšení veřejné bezpečnosti** usnadňuje práci bezpečnostním složkám města. Díky aktuálním dopravním informacím se mohou dostat na potřebná místa rychleji, než by bylo jinak možné. Optimalizace světelné signalizace pak může jejich cestu ještě urychlit.
- **Snížení znečištění z dopravy** může být způsobeno chytrým řízením silniční dopravy, které zkracuje dobu cestování automobily a tím i snižuje emise. Za snížením znečištění může stát také jednodušší a pohodlnější cestování veřejnou dopravou, čímž dochází k nižší závislosti na automobilech. Dále může být také podporováno používání elektromobilů, např. výstavbou dobíjecích stanic nebo volbou elektromobilů do vlastního vozového parku města.
- **Zlepšení rozpočtu na dopravu** je dalším důvodem k využití konceptů chytré dopravy. Stojí za ním především informace získané ze zavedených systémů a jejich důkladná analýza, která může vést k efektivnímu využití stávající dopravní infrastruktury města, ale také ke zjištění priorit pro její budoucí rozvoj.

2.1.5 Životní prostředí

Kvalita ovzduší, emise, zeleň, vodní plochy, nakládání s odpady a energetická efektivnost jsou hlavní témata životního prostředí v chytrém městě, neboli *chytrého životního prostředí*.

Koncept chytrého životního prostředí se zaměřuje na minimalizaci ekologického dopadu města, zároveň však bere v úvahu kvalitu života ve městě a snaží se ji neomezovat. Role chytrých zařízení a systémů jsou v této oblasti velmi rozmanité. Populární jsou systémy pro monitorování a predikci kvality ovzduší pomocí senzorů nebo systémy monitorující městské stromy, aby nedocházelo k poškozování infrastruktury. Složitým problémem především ve velkých městech je správná organizace vodního hospodářství a obměna stárnoucí vodohospodářské infrastruktury. V některých městech pomáhají chytré technologie zlepšit kvalitu pitné vody. V jiných městech je kladen důraz na ochranu a rozšiřování zeleně. Zvětšující se populace přináší problémy také s otázkou, jak nakládat s odpady. Chytré systémy mohou pomoci např. s tříděním odpadů nebo s dynamickým sběrem odpadu podle aktuálních potřeb [25].

2.1.6 Živobytí

Oblast *chytré živobytí*, někdy označovaná také jako *chytré bydlení*, se věnuje především službám poskytovaným ve městě a zahrnuje tak např. oblasti jako kultura, zdravotnictví, bezpečnost, vzdělání, sociální služby nebo cestovní ruch. V různých oblastech přináší chytré živobytí různá vylepšení. V oblasti cestovního ruchu mohou být např. nabízeny virtuální prohlídky města a jeho okolí [2]. V rámci bezpečnosti může jít o snadnější hlášení incidentů bezpečnostním složkám. Ve zdravotnictví může docházet k monitorování nemocných nebo postižených lidí v reálném čase. Ve vzdělání mohou chytré technologie znamenat vyšší kvalitu výukových materiálů i procesů. V jakékoli ze zmíněných oblastí mohou být od veřejnosti sbírána data za účelem analýzy a následného získávání nových znalostí v konkrétních otázkách. Taková data mohou být použita např. jako podklad pro marketing nebo přizpůsobení služby podle poptávky ve městě. Provozovatelé služeb mohou díky chytrým technologiím také získávat zpětnou vazbu, informovat veřejnost o důležitých událostech nebo ji vzdělávat vlastními poznatky [25].

2.2 Technologie

Pro vystavění chytrého města je nezbytné použití chytrých technologií, bez kterých by celý koncept nemohl fungovat. Naplňování všech vizí popisovaných v sekci 2.1 stojí na třech základních pilířích, přesněji třech základních technologiích chytrých měst. Těmi jsou internet věcí (IoT, *Internet of Things*), big data a cloudové výpočetní technologie (*Cloud computing*). Jejich vzájemná spolupráce začíná u chytrých systémů a zařízení, která jsou součástí internetu věcí. Ta shromažďují velké množství dat (*big data*), o jejichž správu se starají cloudové výpočetní služby. Mimo tyto tři základní technologie bývají často zapojeny také metody umělé inteligence, které se snaží fungování chytrého města dále vylepšovat, např. předpovídáním znečištění ovzduší nebo pokročilejším získáváním znalostí z dat [25].

2.2.1 Internet věcí

Internet věcí je pojem označující síť fyzických zařízení propojených pomocí internetu. Jednotlivá zařízení si lze představit jako chytré mobilní telefony, vozidla nebo budovy, které jsou opatřeny senzory, softwarovým vybavením a možností připojení k síti. Taková zařízení pak mohou díky senzorům sbírat data, díky síťovému připojení sdílet data a díky softwarovému vybavení pracovat podle očekávání [20]. Internet věcí je tak ve skutečnosti metodou, která z mnoha tradičních fyzických zařízení dělá „chytrá“ pomocí síťového připojení [25].

Velký vliv v rozvoji internetu věcí mají chytré mobilní telefony, které tento koncept dostávají do povědomí široké veřejnosti. Začátkem používání chytrých telefonů, jejich připojením k internetu a umožněním vzájemného sdílení dat došlo nejen k propojení fyzických zařízení, ale především k propojení lidí. Samotné propojení chytrých telefonů již naplňuje definici pojmu internet věcí, neustále populárnější je však i propojování chytrých telefonů s jinými fyzickými zařízeními, jako jsou automobily, pračky, lednice, osvětlení nebo elektronické hračky pro děti. Lidé tak mohou jejich chytrá zařízení ovládat nebo sledovat, a to odkudkoliv, kde mají síťové připojení pro svůj chytrý telefon. Čím dál častěji se tak lze setkat např. s pojmem *chytrá domácnost*, což označuje právě ovládání a sledování chytrých zařízení v domácnosti. Takové možnosti s sebou kromě výhod přináší i řadu nevýhod, příkladem může být narušení soukromí, možný únik citlivých dat nebo nehody vedoucí k fyzickým škodám [20].

Obecně lze za součást internetu věcí považovat jakékoliv zařízení se síťovým připojením. Konkrétních příkladů existuje nespočet a jejich využití přináší nejen v chytrých městech řadu možných vylepšení. Široká škála chytrých systémů a zařízení sahá ve městech od teplotních čidel, přes mobilní aplikace pro sdílení událostí ve městě, po systémy řízení dopravy. Z globálního pohledu pak může být každé chytré město označeno jako jeden chytrý systém, který je součástí obrovské sítě internetu věcí [2].

2.2.2 Big data

Pojem *big data* poukazuje na to, že za využíváním chytrých technologií stojí velké množství dat. Součástí chytrých systémů a zařízení, tedy i chytrých měst, je práce s daty, která obvykle zahrnuje shromažďování dat následované předáváním dat a zakončené analýzou dat. Analýza pak může mít za cíl prezentaci dat, zdokonalování nebo předpověď a bývá vztahována k pojmu big data. Oblast big data se snaží, aby data získávaná z chytrých systémů a zařízení byla dobře zpracovatelná a nesla podstatné informace pro analýzu. Podstatná je struktura dat, frekvence jejich získávání, ale také otázka zabezpečení dat. Zpracování kvalitních dat mění informace ve znalosti a pomáhá lidem i strojům lépe jednat a rozhodovat se. Tím vzniká koloběh, kdy lidé využívají data k lepšímu rozhodování a chování, což vede ke shromažďování většího objemu lepších dat, a tím se dále zlepšuje rozhodování a chování [2]. Velké množství dat získané z chytrých systémů a zařízení velmi úzce souvisí s cloudovými výpočetními technologiemi, které slouží k jeho uchovávání a zpracovávání. Obvykle je proto zapotřebí, aby tyto dvě technologie byly ve vzájemném souladu [25].

2.2.3 Cloudové výpočetní technologie

Cloudové výpočetní technologie představují vhodné řešení pro zpracování velkých objemů dat získávaných v rámci chytrých měst. Jde o metodu poskytování přístupu ke společné platformě výpočetních zdrojů, které může využívat více uživatelů. Výhodou použití takového řešení je především úspora nákladů, výkonné výpočetní technologie si tak mohou dovolit i menší města s nižším rozpočtem. Zahrnutím cloudových služeb do fungování chytrého města získává spotřebitel okamžitý přístup k hardwarovým zdrojům s nákladově efektivní údržbou, a to za pouze omezené počáteční investice. Výhodou využití cloudových služeb je také bezpečnost, která sice velmi záleží na konkrétním poskytovateli služby, ale obecně je cloudové řešení vnímáno jako bezpečné. Hlavním účelem využití cloudových výpočetních technologií je snaha o efektivní zpracování získaných dat, konkrétní metody zpracování jsou však závislé na typu dat a požadovaném druhu výsledků analýzy [25].

Kapitola 3

Systémy řízení požadavků

Za vznikem systémů řízení požadavků (ITS, *Issue Tracking Systems*) stojí potřeba řídit změny, resp. řídit požadavky na změny (*Issues, Modification Requests*). Změna je základním rysem vývoje, ale také provozu systému a znamená jeho modifikaci. Požadavek na změnu vyzývá osoby odpovědné za systém, aby tento systém upravili [26]. Změna může spočívat v modernizaci, opravě, vytvoření nebo jiné modifikaci systému či jeho části s cílem vylepšit funkčnost, usnadnit použití, snížit provozní náklady apod. Aby změna proběhla správně a nedošlo naopak k poškození systému, je vhodné její zavedení kontrolovat. Právě pro účely kontroly a řízení změn vznikly systémy řízení požadavků [24].

Systémy řízení požadavků jsou softwarové nástroje určené pro správu artefaktů, které přechází z počátečního stavu přes libovolné množství stavů do koncového stavu a během přechodů mezi těmito stavy shromažďují informace. Technicky jsou takové systémy navrženy jako databáze uchovávající informace o požadavcích na změny [3]. Jinými autory jsou ITS definovány např. jako softwarové nástroje, jejichž účelem je shromažďování, správa a sledování pokroku požadavků [24]. V oblasti vývoje softwaru pak mohou být ITS chápány jako nástroje umožňující vývojářům sledovat, upřednostňovat a přiřazovat chyby, požadavky na funkce a další úkoly ve vývoji, jako je testování [16]. Všechny tyto výklady pojmu ITS jsou si velmi podobné a autoři se tak shodují na důvodech používání ITS. Těmi jsou usnadnění komunikace v týmu, přehled o stavu systému a přístup k historii změn.

V této kapitole jsou nejprve popsány požadavky na změny, jejich struktura a typy. Tyto informace jsou dále využity ke správnému pochopení, jak jsou ITS používány, čímž se zabývá druhá sekce kapitoly. V této sekci jsou nejprve představeny funkce, které běžné ITS uživateli nabízí, dále pak typy rolí uživatelů a způsob, jakým používání ITS probíhá. Poslední bod této sekce je věnován existujícím nástrojům a jejich odlišnostem.

3.1 Požadavky na změny

Pod pojmem „požadavek na změnu“ se skrývá strukturovaný dokument, který popisuje výzvu k úpravě systému a je určen osobám odpovědným za systém. Za vznikem takové výzvy mohou stát různé okolnosti, např. na začátku vývoje systému je zdrojem požadavků specifikace, v pozdějších fázích vývoje mohou vznikat požadavky na základě chyb nalezených při testování a při provozu systému pak mohou požadavky reagovat na výzvy k rozšíření funkcionality systému. Podle důvodu vzniku výzvy jsou rozlišovány různé typy požadavků na změny, přičemž požadavek je z výzvy vytvořen uspořádáním získaných informací do

podoby strukturovaného dokumentu [26]. Struktuře a typům požadavků je věnován zbytek této sekce.

3.1.1 Struktura požadavku

Požadavek na změnu je deklarativní, tj. udává, co je třeba provést, ale neuvádí způsob, jakým má být změna provedena. Co je třeba provést, ale také kým nebo v jakém termínu, uchovává požadavek v různých bodech svojí struktury tak, aby byly informace přehledné a snadno k nalezení. Konkrétní prvky struktury, neboli atributy se liší podle potřeb daného ITS. V různých oblastech použití musí požadavky obsahovat různé informace, aby mohla být změna provedena. Některé informace jsou však klíčové a lze je ve struktuře požadavku nalézt ve většině případů. Mezi takové informace patří:

- **Jméno osoby**, která požadavek na změnu reportovala. V případě nejasností pak může být tato osoba kontaktována, aby poskytla upřesnění.
- **Typ požadavku**, např. chyba nebo žádost o přidání nové funkce.
- **Popis** stručně vysvětlující, jaká změna je vyžadována.
- **Označení systému**, resp. jeho části, které se změna týká.
- **Detailní informace**, např. k požadavku na opravu chyby je třeba znát přesné nastavení systému, prostředí běhu systému, kroky, které chybě předcházely a další podrobnosti, aby mohla být chyba nalezena a replikována.
- **Závažnost požadavku**, na jejímž základě lze některé požadavky upřednostňovat nad jinými. Požadavky na opravu chyb mohou mít např. větší prioritu pro zpracování než požadavky na rozšíření funkcionality systému.
- **Přiřazená osoba**, tedy osoba odpovědná za vykonání uvedené změny [24].
- **Termín**, do kterého by změna měla být provedena.
- **Stav** ukazující, v jaké fázi se nachází vykonávání změny [26].

Data uchovávaná uvnitř požadavků umožňují sestavovat pracovní plány a zároveň sledovat, zda jsou plány dodržovány. Mimo zmíněné prvky struktury může požadavek obsahovat jakékoliv další atributy, které jsou při řízení změn potřeba [26]. Atributy požadavku jsou ukládány v databázi ITS tak, aby bylo následně možné podle nich požadavky vyhledávat a seskupovat [3].

3.1.2 Typy požadavků

Jak bylo zmíněno v úvodu této sekce, typ požadavku souvisí s důvodem vzniku výzvy ke změně. Nejčastějšími zdroji požadavků jsou zprávy identifikující chyby v systému, návrhy na vylepšení systému, události ve vývoji dalších systémů, požadavky od nejvyššího vedení a změny v základní struktuře, standardech nebo legislativě. Podle zdrojů lze rozdělit požadavky na změny do pěti základních typů: problém, chyba, vylepšení, požadavek a úkol.

- **Problém** (*issue*) popisuje nestandardní chování systému. Je používán jako označení výzev, kdy není jisté, zda je chování systému takto zamýšleno, či nikoliv. Problém může být po prozkoumání prohlášen za chybu, vylepšení nebo může být uzavřen s informací, že jde o správné chování systému.

- **Chyba** (*bug*) sdružuje různé vady systému. Většina chyb je zaviněna člověkem, a to jak ze strany osob podílejících se na vývoji a provozu systému, tak ze strany uživatelů systému. Někdy může být chyba ale také způsobena selháním technologií použitých v systému.
- **Vylepšení** (*enhancement*) je typ požadavku, který říká, že by existující systém mohl být nějakým způsobem upraven k lepšímu. Může jít o úpravu celého systému nebo pouze jeho části, např. některé z jeho funkcí nebo vlastností.
- **Žádost** (*requirement*) typicky reprezentuje výzvu k přidání nové funkce nebo vlastnosti systému, kterou určí buď přímo zákazník, nebo osoba odpovědná za systém.
- **Úkol** (*task*) je hojně používaná kategorie požadavků, která říká, že je něco potřeba udělat. Příkladem může být požadavek na aktualizaci prostředí nebo výměnu hardwarové části systému.

Kromě zmíněných existuje celá řada dalších typů požadavků na změny, které se odvíjí od způsobu používání ITS [24].

3.2 Funkce systémů řízení požadavků

V procesu řízení změn nabízí ITS množství funkcí, které proces usnadňují a zpřehledňují. Představením těchto funkcí lze získat ponětí o rozsahu ITS a způsobu jejich používání. Analýzou známých ITS bylo nalezeno deset základních funkcí, jež jsou atomické, užitečné a nezávislé na konkrétních ITS a implementuje je většina běžně používaných nástrojů. Získané funkce jsou následující [16]:

- **Pokročilé vyhledávání** (*advanced search*) – umožňuje filtrovat požadavky podle zavedených kritérií a jejich kombinace. Např. získat všechny požadavky, pro které platí, že jejich priorita je větší než 1 a vznikly před datem 1. 1. 2020, lze pomocí filtru „priorita $>$ 1, datum vzniku $<$ 1. 1. 2020“.
- **Grafické zprávy** o projektových datech (*graphical reporting*) – nabízí uživatelům ITS jednoduchý přehled o vybrané doméně informací. Příkladem je srovnání počtu otevřených a uzavřených požadavků v určitém období.
- **Flexibilní termíny** (*flexible issue deadlines*) – dovolují při vytváření požadavku nastavit datum jeho uzávěrky.
- **Flexibilní oznámení** (*flexible notifications*) – přináší uživateli ITS možnost konfigurovat spouštěče oznámení. Příkladem spouštěče může být vytvoření požadavku nebo úprava konkrétního atributu požadavku (např. uzavření požadavku, změna priority apod.).
- **Integrace správy verzí** (*version control integration*) – umožňuje uživateli získat znalosti sloučením informací přicházejících z ITS a ze systému správy verzí.
- **Seskupování** (*grouping issues*) – poskytuje možnost získání seznamů s požadavky odpovídajícími různým filtrům. Dochází tak např. k zobrazení seznamu všech požadavků, seznamů podle typů požadavků, seznamů podle hodnoty konkrétního atributu požadavku nebo k vytvoření seznamu po kombinaci zmíněných kritérií.

- **Vytvoření externího požadavku** (*external issue creation*) – dává možnost externím osobám podílet se na vytvoření požadavku. Svoji výzvu ke změně může externí osoba zaslat pomocí e-mailu na konkrétní adresu s konkrétním předmětem, případně využít nástroje určeného pro zasílání výzev, pokud takový nástroj existuje.
- **Automatizace pracovních postupů** (*workflow automation*) – přináší možnost automatizovat vlastní zavedené postupy v ITS. Může jít např. o automatickou aktualizaci atributů požadavku po změně hodnoty jiného atributu.
- **Vynucování pracovního postupu** (*workflow enforcement*) – přispívá k dodržování pravidel nastavených vlastním pracovním postupem. Např. hodnoty atributů požadavku lze upravit nebo přiřadit pouze v případě, že jsou splněny určité předpoklady založené na hodnotách jiných atributů.
- **Vyhledávání podobné SQL** (*SQL-like search*) – umožňuje uživateli ITS dotazovat se na požadavky pomocí syntaxe podobné SQL.

Podle průzkumu, kterého se v roce 2018 účastnilo 78 vývojářů z USA a Evropy, jsou nejpoužívanějšími funkcemi pokročilé vyhledávání a flexibilní oznámení. Naopak nejméně používanou funkcí je vyhledávání podobné SQL [16].

Pro vysvětlení, jak ITS fungují a jak jsou používány, jsou v následující podsekci popsány role, v nichž osoby s ITS pracují. Dále je představeno, jak jsou tyto osoby zahrnuty do životního cyklu požadavku, a poslední podsekce je věnována existujícím nástrojům spadajícím pod ITS.

3.3 Uživatelské role

V rámci ITS lze rozlišit několik typů rolí, které s nástrojem buď přímo pracují, nebo se jiným způsobem podílí na vzniku a zpracování požadavků. Základními typy rolí jsou *uživatel*, *manažer* a *správce*. Uživatel označuje osobu používající systém, manažer a správce jsou osoby odpovědné za systém. Uživatel tak typicky přistupuje do ITS pouze jako čtenář, příp. může tvořit nové požadavky přímo v ITS, jiné úpravy ale neprovádí. Provádění úprav požadavků je pak starostí osob v roli manažer a správce. Pro osoby v roli uživatel je požadavek formální způsob, jakým mohou poskytovat zpětnou vazbu manažerům a správcům systému. Uživatel, který nahlásí chybu, nedostatek, možné vylepšení systému apod., má jistotu, že jeho požadavek bude vyslechnut a ověřen, a měl by mít možnost zjistit, jak byl požadavek vyřešen, případně i v jakém stavu se aktuálně nachází jeho řešení. Mimo to může ITS také poskytovat informace o všech zpracovávaných, případně vyřešených požadavcích, aby mohli uživatelé získat představu o nadcházejících změnách v systému [26].

Role manažera je přiřazena osobě, jejímž úkolem není přímo vykonávat změny v systému, ale pouze vykonávání změn řídit. To obvykle zahrnuje řízení osob v roli správce, ale také komunikaci s uživateli systému [3]. Hlavním cílem ITS je pomoci manažerům v řízení systému. Toho je dosaženo především prezentováním úkolů, které je třeba udělat. K prezentování dochází formou souhrnných zpráv včetně týdenních a měsíčních souhrnných zpráv a denních upozornění. Upozornění jsou generována na základě požadavků, jejichž termín již uplynul nebo se k uplynutí blíží, nebo které z jiných důvodů vyžadují akci v blízké době. Kromě toho by měl ITS umožnit manažerům dotazovat se na informace, které v souhrnných zprávách nejsou zahrnuty. Všechny informace získané z ITS pomáhají manažerovi při sestavování harmonogramů a určování, kam je třeba soustředit úsilí, aby mohl být harmonogram

splněn, nebo provést případné úpravy harmonogramu. Důležitým cílem souhrnných zpráv je také ukázat, kdo na čem pracuje. Díky tomu je manažer schopen přidělovat nové požadavky nejméně vytíženým pracovníkům a případně přerozdělovat již přiřazenou práci [26].

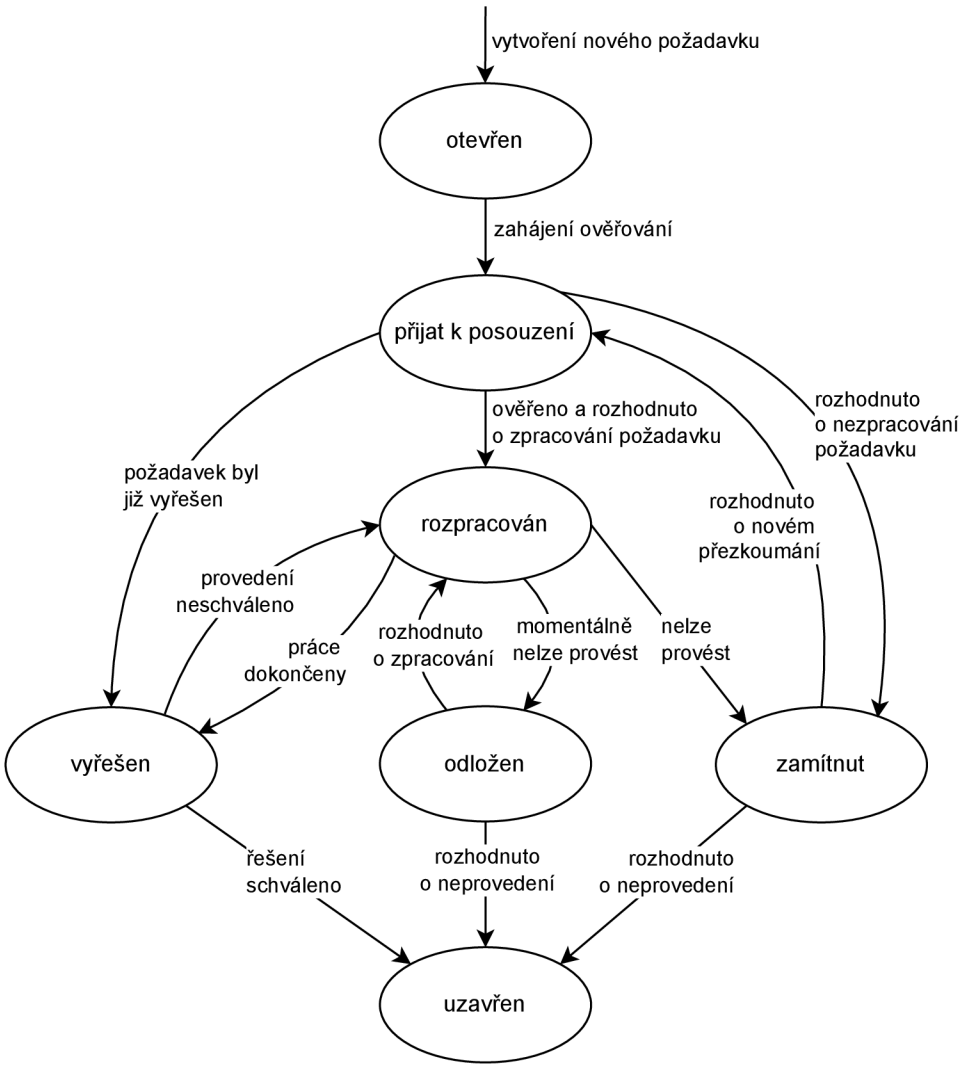
V roli správce si lze představit libovolnou osobu, která může v systému vykonat nějakou změnu. Např. správcem softwarového systému může být programátor nebo dokumentarista. Správce využívá funkce ITS v první řadě k dotazování, tedy k získávání informací o tom, za co je aktuálně odpovědný a kdy by měl být každý jemu přiřazený úkol dokončen. Některé ITS dokonce umožňují z těchto informací vygenerovat aktuální osobní rozvrh a případně upozornit správce na blížící se termín uzávěrky. Dále by měl ITS sloužit správci pro zaznamenávání pokroku, problémů a poznámek týkajících se přidělených úkolů. To usnadňuje sledovat vývoj úkolu jak správci, tak manažerům [26].

3.4 Životní cyklus požadavku

Průběh použití ITS koresponduje s životním cyklem požadavku. Životní cyklus (*life cycle*) označuje posloupnost stavů, ve kterých se požadavek postupně nachází od svého prvního otevření až po konečné uzavření. Životní cyklus tak označuje cestu požadavku napříč stavy konkrétního ITS. Sada možný cest požadavku v rámci ITS je často nazývána jako „pracovní postup“ (*workflow*). Pracovní postupy se mohou lišit od velmi jednoduchých až po velmi podrobné a složité. Minimální sada stavů vyžadovaná pro existenci pracovního postupu v ITS obsahuje stavy *otevřen* a *uzavřen* a přechody mezi nimi. Nový požadavek se od vytvoření nachází ve stavu *otevřen* a po dokončení vyžadovaných změn je označen jako *uzavřen*. Pokud je později zjištěno, že požadavek není vyřešen podle očekávání, např. znovu se objeví chyba nebo se nová funkce chová jinak, než bylo vyžadováno, může následovat přechod zpět do stavu *otevřen* [3]. Jakkoli je minimální sada stavů výmluvná a přehledná, v běžně používaných ITS takto zjednodušený pohled na sledování požadavků nestačí a stavů je definováno podstatně více [24]. Příklad sady stavů a přechodů mezi nimi je popsán schématem na obrázku 3.1. Z tohoto schématu vychází následující demonstrace možného průběhu použití ITS.

Aby mohl být životní cyklus požadavku odstartován, musí nejdříve požadavek vzniknout. Proces vzniku požadavku začíná ve chvíli, kdy osoba, ať už v roli uživatele, manažera nebo správce, požaduje nějakou změnu v systému. Podle způsobu použití systému může být na základě takové výzvy požadavek automaticky vytvořen nebo musí dojít k jeho vytvoření manuálně. Automatizované řešení je využíváno především v systémech se značným množstvím požadavků, kde by manuální vytváření stálo pověřeného člověka spoustu času. V takovém případě je ale nutné řešit, jak velkou část procesu automatizovat, a proces automatické tvorby požadavků nastavit tak, aby lidem ušetřil práci, ne naopak. Nově vzniklý požadavek začíná svůj životní cyklus ve stavu *otevřen* [26].

Po vzniku nového požadavku je obvykle prvním krokem ověření, jehož cílem je zkontrolovat, zda požadavek obsahuje všechny potřebné informace, zda je požadovaná změna proveditelná a zda je v zájmu osob odpovědných za systém změnu provést. Zároveň může dojít k ověření, že se nejedná o duplicitní požadavek. Součástí fáze ověření může být také přiřazení osoby odpovědné za vykonání požadavku, nebo může proces ověření provádět již přiřazená osoba. V systému tak přibývá stav *přijat k posouzení* označující fázi ověření. Výsledkem ověření může být přechod do stavu *rozpracován* oznamujícího, že požadavek byl ověřen, přiřazená osoba byla o požadavku informována a začne na něm pracovat, popř. již začala. V jiném případě může být výsledkem rozhodnutí o nezpracování požadavku a pře-



Obrázek 3.1: Schéma stavů požadavku a přechodů mezi nimi

chod do stavu *zamítnut*, nebo může jít o již zpracovaný požadavek a pak následuje přechod do stavu *vyřešen*. Fázi ověření mají na starost obvykle manažeři a správci [24].

Požadavek ve stavu *rozpracován* je řešen podle možností přiřazené osoby. Přiřazená osoba, což je typicky osoba v roli správce, může v rámci zpracovávání doplňovat atributy požadavku, o které mohou být opřena rozhodnutí v pozdějších fázích vývoje nebo které mohou být užitečné v dalších krocích zpracovávání. V průběhu prací na požadavku totiž může dojít i ke změně přiřazené osoby a doposud získané informace uchované jako atributy požadavku mohou další vývoj výrazně zjednodušit [26]. Zpracovávání požadavku probíhá až do události, která vynutí změnu stavu. V případě, že momentálně nemohou být práce na požadavku z technických důvodů, časového omezení nebo jiných důvodů dále prováděny, ale v budoucnu by být opět mohly, přechází požadavek do stavu *odložen*. Pokud nemůže být požadavek proveden, a to ani v budoucnu, je jeho stav nastaven na *zamítnut*, což značí, že přiřazená osoba má důvody k tomu, aby vyžadovanou změnu neprovedla. Takovými důvody jsou např. nedostatek informací, změna podmínek, nemožnost replikovat situaci

apod. Třetí možnou událostí vedoucí ke změně stavu *rozpracován* je dokončení prací na požadavku. Nově se takový požadavek nachází ve stavu *vyřešen* [24].

Posledním krokem při zpracovávání požadavku uvnitř ITS je jeho uzavření, o které se obvykle stará manažer. Aby mohl být požadavek uzavřen, je nejprve potřeba zkontrolovat, zda byla požadovaná změna vykonána podle očekávání, nebo zda existuje vhodný důvod k jejímu nevykonání. Např. v případě typu požadavku chyba může jít o ověření, že se již v systému chyba nenachází. V případě přidání nové funkce systému může být kontrola zaměřena na její správné chování. Při nevyhovění je požadavek vrácen zpět do jednoho z předcházejících stavů a znovu zpracováván nebo ověřován. Naopak při úspěšné kontrole je požadavek uzavřen a osoba stojící za jeho vznikem je informována o výsledku [26].

3.5 Existující nástroje

Systém řízení požadavků je v praxi známý i pod jinými názvy, jako např. *systém sledování problémů* nebo *tiketovací systém*, kde *tiket* je alternativním označením požadavku. V oblasti vývoje softwarových produktů, s níž jsou ITS spojovány nejčastěji, je známý pod pojmem *bug tracking system* (systém sledování softwarových chyb). ITS jsou ve většině případů používány v prostředích pro spolupráci, tedy v prostředích založených na práci v týmu. Výhody ale přináší také při využívání jednotlivci, nejčastěji jako nástroj pro řízení osobního času nebo produktivity. Ve firmách běžně ITS slouží jako součást zákaznické podpory k vytváření, aktualizaci a řešení problémů hlášených zákazníky nebo ostatními zaměstnanci dané organizace [3]. V praxi je používána řada nástrojů spadajících pod ITS, z nichž mezi šest nejpoužívanějších patří JIRA[1], Redmine[2], Pivotal Tracker[3], ZenHub[4], GitHub Issues[5] a Bugzilla[6] [16], jejichž stručnému představení jsou věnovány následující odstavce.

JIRA je nástroj vyvíjený společností Atlassian Software Systems. Jeho výrazným rysem je komplexnost, kterou nabízí při zachování přehledného a snadno použitelného uživatelského rozhraní. Nástroj je také známý svou komunitou, jež stojí za vznikem mnoha rozšíření, širokou škálou výukových materiálů a dokumentace. JIRA je nástroj s otevřeným zdrojovým kódem (*open source*), což umožňuje vznik právě zmíněných rozšíření. Nástroj je dostupný zdarma pro neziskové, nevládní, neakademické, nekomerční a nepolitické organizace a pro jednotlivce při nekomerčním použití. V jiných případech je používání nástroje zpoplatněno částkou ve výši určené zvolenou edicí [24].

Redmine je nástroj s otevřeným zdrojovým kódem, jehož používání je zdarma, až na výjimky ve formě placených rozšiřujících modulů. Redmine ale může být plnohodnotně využíván k řízení požadavků i bez těchto rozšíření [33]. Z deseti funkcí popsaných v úvodu této sekce nabízí Redmine pouze šest. Těmi jsou pokročilé vyhledávání, grafické zprávy, flexibilní termíny, flexibilní oznámení, vytvoření externího požadavku a vynucování pracovního postupu, i přes to jde však o jeden z nejpoužívanějších nástrojů pro řízení požadavků [16].

Další tři nástroje, ZenHub, Pivotal Tracker a GitHub Issues, nabízí podobné prostředí pro řízení požadavků a je tak na preferencích uživatele ITS, jaký z nich si vybere. Každý z těchto nástrojů implementuje osm či devět ze sledovaných deseti funkcí. Tyto funkce se ale různí, což může uživateli výběr usnadnit [16]. Pro všechny tři nástroje platí, že jsou zdarma

[1] https://www.atlassian.com/software/jira
[2] https://www.redmine.org
[3] https://www.pivotaltracker.com
[4] https://www.zenhub.com
[5] https://github.com/features/issues
[6] https://www.bugzilla.org

pouze v základní, omezené verzi. Existují pak i jejich placené verze, které přináší určité výhody, např. možnost spolupráce ve větším týmu nebo práce na více projektech zároveň. Pro nástroje je společná také možnost propojení s populárním verzovacím nástrojem GitHub[7], díky čemuž je možné přesunout správu požadavků velmi blízko správě zdrojového kódu. Všechny nástroje dále poskytují možnost částečné automatizace pracovních postupů, což může v různé míře uživatelům ITS pomoci ušetřit čas při řízení i zpracovávání požadavků.

Poslední z nástrojů, Bugzilla, je dostupný již od roku 1998 a jeho používání je zcela zdarma. Jeho nevýhodou je nemožnost integrace se systémem pro správu verzí. Nemůže tak dojít k propojení s nástrojem GitHub jako u předchozích nástrojů. I nástroj Bugzilla však umožňuje částečnou automatizaci procesu správy požadavků, nabízí např. automatické rozpoznávání duplicitních požadavků nebo automatické vyplňování některých atributů požadavku. Nástroj Bugzilla byl dříve známý pro svůj důraz na funkčnost a menší zaměření na uživatelské prostředí. V posledních letech však jeho vzhled prošel proměnami a je tak plně konkurenceschopným ITS [7].

[7] https://github.com

Kapitola 4

Vývoj multiplatformních aplikací pro mobilní zařízení

Tradiční vývoj aplikací pro mobilní zařízení probíhá specificky pro jednotlivé mobilní platformy. Jsou při něm používány nástroje a programovací jazyky specifické pro konkrétní platformu a výsledná aplikace může být nasazena pouze na platformě, pro kterou byla určena. Takový přístup k tvorbě aplikací je označován jako *nativní*. Pokud je pro vývoj zvolen nativní přístup a cílem je, aby výslednou aplikaci používali lidé napříč platformami, je třeba vyvinout nativní aplikaci pro každou platformu. Podle výzkumu [27] jsou aktuálně nejrozšířenějšími mobilními platformami Android a iOS, jejichž podíl na celosvětovém trhu mobilních operačních systémů činil ve třetím čtvrtletí roku 2022 přibližně 99 % (Android 71 %, iOS 28 %). Tato data jsou získána na základě více něž 1,7 miliard zobrazení webových stránek každý měsíc po celém světě. Vývoj aplikací pro jednotlivé platformy se vzájemně natolik odlišuje, že typicky existuje pro každou platformu jiný tým vývojářů. Vývojáři pro platformu Android používají vývojové prostředí Android Studio a programovací jazyk Java, Kotlin nebo C++, zatímco tým vývojářů pro iOS pracuje v prostředí Xcode s jazykem Objective-C nebo Swift. Výsledkem nativního vývoje jsou potom obvykle dvě samostatné aplikace — jedna pro Android, druhá pro iOS. Takové aplikace vyžadují nejen oddělený vývoj, ale také oddělenou údržbu po jejich nasazení. Oddělení obou aplikací však přináší možnost plně využít potenciál každé platformy a přizpůsobit aplikaci uživatelům konkrétní platformy [21].

Alternativou k nativní tvorbě aplikací je multiplatformní vývoj. Tento pojem zahrnuje velké množství různých řešení, jejichž hlavním cílem je zjednodušit tvorbu aplikací oproti nativnímu vývoji. Obecně jde o snahu vyvíjet a následně udržovat jediný zdrojový kód aplikace, který bude možné nasadit na různé platformy, a to pouze s malými nebo žádnými úpravami specifickými pro konkrétní platformu [4]. Podíl zdrojového kódu aplikace, který je možné sdílet napříč platformami, se liší podle zvoleného multiplatformního přístupu i konkrétního aplikačního rámce (*framework*). Každý přístup nabízí jiné technické řešení a s ním spojené vlastnosti pro vývoj i běh aplikace. Detailně se jednotlivým přístupům věnuje sekce 4.1. Kromě značných odlišností mezi přístupy jsou často zásadní rozdíly i mezi jednotlivými frameworky stejného přístupu. Těm nejpoužívanějším frameworkům se věnuje sekce 4.2.

Motivací pro zvolení multiplatformního vývoje oproti nativnímu je především nižší cena a další faktory, které s ní souvisí, jako je menší časová náročnost, nižší požadavky na zkušenosti vývojářů nebo jednodušší údržba aplikace [13]. Některé studie poukazují na

problémy multiplatformních aplikací v oblasti UX (*User experience*, uživatelská zkušenost), a to zejména na platformě iOS, kde se ani aplikace vyvíjené s důrazem na uživatelské prostředí nedokáží vyrovnat nativním aplikacím [1]. Další oblastí, ve které multiplatformní aplikace většinou zaostávají za nativními, je výkon [4]. Vždy však záleží na požadavcích na výslednou aplikaci a vlastnostech použitého multiplatformního řešení.

V multiplatformním vývoji nemusí jít pouze o tvorbu aplikací pro mobilní platformy. Multiplatformní vývoj se v posledních letech zaměřuje kromě spouštění jednoho zdrojového kódu na více mobilních platformách také na možnost nasazení tohoto kódu ve webovém i desktopovém prostředí [15]. Vzhledem k tématu této diplomové práce se však dále budu věnovat především multiplatformnímu vývoji pro mobilní zařízení.

4.1 Možnosti tvorby multiplatformních aplikací

Jak dobře se lze přiblížit k výkonu a vzhledu nativních aplikací, záleží na zvoleném přístupu k vývoji. Autoři článku *A Survey and Taxonomy of Core Concepts and Research Challenges in Cross-Platform Mobile Development* [4], z něhož tato sekce vychází, rozdělují multiplatformní přístupy do pěti kategorií podle základních konceptů, které umožňují běh aplikace na více platformách: *webový, hybridní, interpretovaný, kompilovaný* a *modelem řízený přístup*. Tato klasifikace rozlišuje přístupy pouze podle hlavních charakteristik, některé klasifikace však přináší podrobnější přehled a definují i další kategorie, např. *přístup založený na komponentách, přístup využívající virtuální stroj, cloudový přístup* nebo *kombinovaný přístup* [15]. Ačkoliv se způsoby klasifikace různí, konkrétní technické možnosti zůstávají stejné. Rozdělení do pěti, často i méně[1], základních kategorií proto převládá.

Každá z kategorií je reprezentována množstvím aplikačních rámců (*framework*), které mají sice společné prvky typické pro kategorii, ale mohou se od sebe velmi lišit, např. svojí popularitou, velikostí komunity, nabízenými funkcemi nebo způsobem vykreslování uživatelského rozhraní. Rozhodnutí, který framework použít pro vývoj aplikace, pak závisí na konkrétní situaci. Kromě požadavků na výslednou aplikaci je při volbě nutné zohlednit také možnosti vývoje a vývojového týmu. Neexistuje jeden přístup nebo framework, který by byl nejlepším řešením pro všechny, nebo alespoň většinu situací [4].

4.1.1 Webový přístup

Jak název naznačuje, mobilní aplikace tohoto přístupu jsou vytvářeny pomocí webových technologií, typicky pomocí HTML (Hypertext Markup Language), CSS (Cascading Style Sheets) a programovacího jazyka JavaScript. Výsledkem implementace podle webového přístupu je tzv. progresivní webová aplikace (PWA, *Progressive Web App*). Ve skutečnosti jde však o téměř běžnou webovou aplikaci, tedy o aplikaci běžící ve webovém prohlížeči [13]. PWA se odlišuje několika vlastnostmi — je progresivní, responzivní, instalovatelná, nezávislá na internetovém připojení, vypadá jako mobilní aplikace, je dohledatelná, aktuální a bezpečná. Pojem *progresivní* říká, že se PWA dokáže přizpůsobit vlastnostem prohlížeče a zařízení uživatele. Aplikaci lze proto použít např. i na zařízeních se slabším hardwarovým vybavením. Naopak na lépe vybaveném zařízení může být chování PWA pokročilejší. *Responzivní* je označení pro aplikace, které umí optimalizovat zobrazení svého obsahu na základě aktuálního rozlišení a typu zařízení. *Instalovatelná* je taková aplikace, kterou lze nainstalovat. PWA je ale možné nainstalovat pouze z webové adresy, odkud je dostupná,

[1] S. Xanthopoulos, S. Xinogalos. A Comparative Analysis of Cross-platform Development Approaches for Mobile Applications. ACM International Conference Proceeding Series. 2013

nikoliv z typických obchodů s mobilními aplikacemi, jako je např. Google Play[2] nebo App Store[3]. *Nezávislost na internetovém připojení* je jednou z nejzásadnějších odlišností. Běžné webové aplikace nebo stránky často bez připojení nevykreslují žádný obsah. To by ale pro PWA v konkurenci ostatních přístupů bylo velkou nevýhodou. Webový přístup proto zajišťuje zobrazení obsahu i v offline režimu, a to pomocí tzv. Service Workers. Snaha o vzhled *jako mobilní aplikace* cílí na to, aby si uživatel myslel, že pracuje s nativní mobilní aplikací. To mimo jiné znamená, že uživatel v progresivní webové aplikaci obvykle neuvidí ovládací prvky webového prohlížeče, jako např. adresní řádek nebo záložky. PWA je také *dohledatelná*, neboli identifikovatelná, a může tak být nalezena vyhledávači. *Aktuální* je obsah aplikace díky Service Workers a jejich získávání dat vždy, když jsou potřeba. Pro zajištění *bezpečnosti* vyžaduje PWA pro obsluhu použití protokolu HTTPS[4]. Tyto vlastnosti je zároveň možné označit za základní koncepty webového přístupu [6].

Technickými pilíři progresivních webových aplikací jsou Service Workers, Manifest a App Shell. Většinu vlastností, kterými se PWA liší od běžných webových aplikací, přináší Service Workers. Service Worker je program v jazyce JavaScript běžící na pozadí aplikace, jehož chování lze přirovnat k aplikačnímu proxy serveru[5] mezi webovým prohlížečem a serverem. Umožňuje tedy reagovat na síťové požadavky, ukládat data do vyrovnávacích pamětí pro offline dostupnost, provádět datové synchronizace se serverem nebo zpracovávat push notifikace[6] aplikace. Service Workers tak zajišťují spolehlivost aplikace a podstatně zvyšují její výkon. Proto a i pro další jejich funkcionalitu jsou programy Service Workers považovány za nejvýznamnější části progresivních webových aplikací [6]. Pod pojmem Manifest je v rámci PWA označován konfigurační soubor ve formátu JSON (*JavaScript Object Notation*), kde jsou zadána metadata aplikace. V Manifestu může vývojář nastavit např. jméno aplikace, popis aplikace, ikony, barvy, ale také chování aplikace po nainstalování nebo webovou adresu, odkud je aplikace spouštěna [32]. App Shell je část aplikace zásadní pro zajištění dobré uživatelské zkušenosti (UX, *User experience*). App Shell plní roli tzv. rámce uživatelského rozhraní, který obsahuje pouze statické prvky, např. úvodní obazovku, panely nástrojů nebo navigační panely. Takový rámec je načten okamžitě, což umožňuje rychlé zobrazení nějakého obsahu, aby uživatel neměl pocit, že je aplikace pomalá. Vlastní obsah aplikace je potom načítán dynamicky podle možností internetového připojení a dostupnosti dat ve vyrovnávacích pamětích [6].

Progresivní webovou aplikaci lze tvořit cíleně již od začátku vývoje nebo pouze transformovat existující webovou aplikaci do podoby PWA. V obou případech je dostačující využití technologií HTML, CSS a JavaScript. Usnadnění při vývoji PWA, stejně jako při vývoji webové aplikace, přináší běžně používané webové frameworky, jako např. Angular[7], React[8] nebo Vue[9]. Mimo to lze ale využít i frameworky specializované na tvorbu PWA, např. PWABuilder[10], který dokáže ulehčit proces transformace existující aplikace na PWA [32]. Důležitými záminkami pro výběr webového přístupu jsou rychlý, snadný a levný vývoj,

[2] https://play.google.com/store/

[3] https://www.apple.com/cz/app-store/

[4] HTTPS (Hypertext Transfer Protocol Secure) – protokol umožňující zabezpečenou síťovou komunikaci

[5] aplikační proxy server – počítačový program určený pro konkrétní aplikaci, který vystupuje v síťové komunikaci jako prostředník mezi klientem a cílovým serverem, slouží k předávání požadavků mezi klientem a jinými síťovými službami a také k ukládání získaných dat do vyrovnávacích pamětí

[6] push notifikace – oznámení zaslané aplikací uživateli, i když aplikace není otevřená

[7] https://angular.io

[8] https://reactjs.org

[9] https://vuejs.org

[10] https://www.pwabuilder.com

funkčnost na všech platformách a zařízeních a automatické aktualizace aplikace. Nutné je však zvážit i nevýhody přístupu: výsledkem není nativní aplikace, PWA vykazují slabší výkon a použití nativních funkcí je limitováno možnostmi prohlížeče. PWA jsou proto často vyčleňovány mimo multiplatformní aplikace jako samostatná kategorie [13].

4.1.2 Hybridní přístup

Tento přístup umožňuje, stejně jako webový přístup, vytvářet mobilní aplikace pomocí webových technologií, např. pomocí HTML, CSS a programovacího jazyka Javascript. Na rozdíl od webového přístupu je zde ale výsledkem opravdu nativní mobilní aplikace, resp. dvě nativní aplikace — jedna pro Android, jedna pro iOS. Obě vzniklé aplikace fungují na stejném principu, a to tak, že obsahují nativní komponentu WebView a také kód pro komunikaci mezi WebView a nativním kódem aplikace [5]. Komponenta WebView funguje jako webový prohlížeč vestavěný do nativní aplikace, jehož typické ovládací prvky, např. nastavení, záložky a adresní řádek, jsou však skryty. Takový prohlížeč se jinak chová běžným způsobem, dokáže tedy spouštět a zobrazovat jakýkoliv webový obsah. Druhý zásadní prvek vzniklé nativní aplikace, kód pro komunikaci mezi WebView a nativním kódem aplikace, umožňuje z webového prostředí (např. z kontextu jazyka JavaScript) volat nativní kód dané platformy (např. kód v jazyce Java nebo Swift). Tento způsob bývá označován jako *přemostění* a lze díky němu nativnímu kódu přenechat zpracování úloh, které jsou pro JavaScript drahé. Častěji se ale přemostění využívá pro přístup k nativním funkcím platformy, jako např. GPS, Bluetooth nebo síťové připojení [4].

Vývoj hybridních aplikací probíhá podle stejných principů jako vývoj webových aplikací. Programátor může tvořit frontend i logiku aplikace ve webových technologiích, následně vzniklý kód zabalit do nativní aplikace a zobrazit jej prostřednictvím komponenty WebView. Hybridní přístup nabízí řadu frameworků, které mohou programátorovi vývoj usnadnit. Různé frameworky se zaměřují na usnadnění různých fází hybridního vývoje. Nejčastěji jde o fázi vytvoření nativních aplikací, inicializaci komponenty WebView a její komunikace s nativním kódem a také zabalení webového kódu do nativní aplikace. Některé frameworky poskytují navíc prostředky pro efektivní vývoj samotné aplikace, především pak knihovny pro ovládání nativních funkcí platformy (Bluetooth, GPS, fotoaparát apod.). Hybridní frameworky je dále možné kombinovat s běžnými webovými frameworky (např. Angular, React nebo Vue), a tím udělat vývoj aplikace ještě více efektivní [4].

Frameworků pro hybridní vývoj existuje velké množství, většina z nich ale staví na nástroji Apache Cordova[11]. Apache Cordova je sám o sobě plnohodnotným hybridním frameworkem s propracovaným systémem pluginů. To umožňuje navázat na osvědčené hybridní funkce nástroje Cordova a vystavět nad ní nové hybridní frameworky. Takové frameworky se obvykle zaměřují na efektivní vývoj uživatelských rozhraní se snahou co nejvíce se přiblížit vzhledu a chování nativních aplikací. Příkladem jsou frameworky Ionic[12], Onsen UI[13], Framework7[14] nebo AppGyver[15] [5]. Nejznámějším nástrojem mimo Apache Cordova je Capacitor[16]. Ve skutečnosti jde o novější alternativu Apache Cordova, která přináší několik nových funkcí, např. podporu pro PWA, lepší přizpůsobení vzhledu aplikace podle platformy nebo možnost jednoduše vyvíjet část aplikace nativně a část pomocí webových

[11] https://cordova.apache.org
[12] https://ionicframework.com
[13] https://onsen.io
[14] https://framework7.io
[15] https://www.appgyver.com
[16] https://capacitorjs.com

technologií. Capacitor může být stejně jako nástroj Cordova využit jinými frameworky, které nad jeho základem vystaví vlastní pokročilé funkce [23].

Aplikace vzniklé hybridním přístupem lze vzhledem k použití webových technologií obvykle spustit v jakémkoliv webovém prohlížeči. Zároveň je možné výsledné nativní aplikace distribuovat v typických obchodech s mobilními aplikacemi, což je velkou výhodou oproti webovému přístupu [4].

4.1.3 Interpretovaný přístup

Interpretovaný přístup na rozdíl od hybridního není vázán na webové technologie. Potenciálně je možné implementovat interpretovanou multiplatformní aplikaci v jakémkoliv programovacím jazyce, který má pro tento přístup podporu ve formě frameworku. Ve skutečnosti je ale i zde nejčastěji využíván jazyk JavaScript a všechny populární frameworky interpretovaného přístupu pracují právě s ním, případně s jeho nadstavbami, např. s jazykem TypeScript. Na rozdíl od hybridního přístupu však výsledný vzhled interpretované multiplatformní aplikace není založený na HTML a CSS a pro zobrazování obsahu není využívána komponenta WebView. Místo toho dochází k vykreslování nativních prvků uživatelského rozhraní každé platformy. Zdrojový kód aplikace v jazyce JavaScript je transformován do nativních prvků pomocí interpretů vestavěných v mobilních zařízeních. Odtud také pochází název přístupu. Na zařízeních platformy iOS je výchozím interpretem JavaScriptCore. Na zařízeních Android je interpret závislý na použitém multiplatformním frameworku, nejpopulárnějším je V8 engine[17] [4]. Základ interpretované aplikace tvoří dvě části. První z nich poskytuje rozhraní pro vývojáře aplikace, nabízí nástroje potřebné k vývoji a umožňuje implementaci aplikace typicky v jazyce JavaScript. Druhá část má na starost nativní prvky aplikace a interpretaci zdrojového kódu z části první. Komunikace mezi vrstvou jazyka JavaScript a vrstvou nativního kódu, za kterého lze přistupovat k nativním funkcím mobilního zařízení, jako např. Bluetooth nebo GPS, probíhá na principu přemostění stejně jako u hybridního přístupu. Pokud interpretovaná aplikace požádá o přístup k nativní funkci, požadavek je přenesen přes „most“ k nativnímu kódu, který jej umí obsloužit. Nativní kód požadavek vykoná a následně výsledek požadavku vrátí přes stejný most zpět interpretované aplikaci [5].

Pod interpretovaný přístup spadá velké množství frameworků. Často jsou však do této kategorie nesprávně řazeny i frameworky kompilovaného přístupu, např. Titanium Appcelerator[18], a naopak [14]. Oba přístupy totiž fungují na principu generování nativních prvků uživatelského rozhraní, každý ale pomocí jiného technického řešení. Rozdíl spočívá v tom, že interpretovaný přístup nekompiluje, netranspiluje ani jinak nepřevádí zdrojový kód do byte kódu[19], což je způsob, jakým pracuje kompilovaný přístup. Interpretovaný přístup pro svoji správnou funkčnost vyžaduje interpret jazyka JavaScript, který slouží jako vrstva abstrakce mezi zdrojovým kódem v jazyce JavaScript a nativním kódem [4].

Pro interpretovaný přístup neexistuje nástroj, který by plnil roli nástroje Cordova v hybridním přístupu. Každý interpretovaný framework je implementován od základu technicky jinak, včetně způsobu implementace přemostění. To znamená, že interpretované frameworky nemají společný základ, na němž by stavěly, a nemohou tak vzájemně využívat svoje moduly. A to i přes skutečnost, že je většina z nich určena pro jazyk JavaScript. Takto striktní

[17] https://v8.dev

[18] https://titaniumsdk.com

[19] byte kód – programový kód, který byl získán ze zdrojového kódu a může být spuštěn pomocí virtuálního stroje na jakékoliv platformě, kde virtuální stroj funguje

vymezení způsobuje, že přechod mezi jednotlivými frameworky interpretovaného přístupu je stejně obtížný jako přechod na framework jiného přístupu. Dalším důsledkem je také značná fragmentace vývojářské komunity. To jsou ovšem pouze drobné nevýhody, frameworky interpretovaného přístupu neztrácí na oblíbenosti za jinými přístupy. Mezi nejznámější patří React Native[20], NativeScript[21] a již zmíněný Titanium Appcelerator [4].

4.1.4 Kompilovaný přístup

Název přístupu je odvozen od zásadního technického nástroje tohoto přístupu, kterým je kompilátor. Kompilátor je program, jehož úkolem je transformovat zdrojový kód (napsaný v některém z vysokoúrovňových programovacích jazyků) do cílového kódu, neboli do kódu nižší úrovně. Příkladem může být kód v programovacím jazyce C# přeložený do nativního byte kódu spustitelného na cílové platformě [13]. Tento způsob je alternativou k tzv. přemostění zmiňovanému v předchozích přístupech. Přemostění zde není využíváno ani pro přistupování k nativním funkcím zařízení, jako je GPS nebo Bluetooth. Nativní funkce jsou v kompilovaném přístupu vývojářům zprostředkovány pomocí SDK (*Software Development Kit*, sada vývojových nástrojů) zvoleného frameworku, který tuto funkcionalitu následně mapuje do SDK odpovídajících platforem. Ovládání nativních funkcí přes SDK platformy je způsob řešení i ve vývoji nativních mobilních aplikací, a je proto vnímán jako výhoda kompilovaného přístupu. Další výhodou překladu do nativního byte kódu jsou možnosti generování uživatelského rozhraní. Komponenty uživatelského rozhraní mohou totiž v kompilovaném přístupu být vykreslovány jako nativní komponenty [4].

Výsledkem implementace podle kompilovaného přístupu jsou nativní aplikace pro jednotlivé platformy. Kompilovaná aplikace může být vyvíjena v libovolném programovacím jazyce, pro nějž existuje kompilovaný framework. Základním a zároveň limitujícím prvkem takového frameworku je kompilátor. Úkol kompilátoru, tedy překlad zdrojového kódu na kód nižší úrovně, je totiž velmi komplexní a složitý problém, který omezuje využití potenciálu platformy [13]. Zároveň je ale překlad kódu hlavní funkcí frameworku a současně s programovacím jazykem také zásadním kritériem při volbě frameworku. Nejznámějšími zástupci kompilovaného přístupu jsou Xamarin[22] s podporou pro programovací jazyk C# a Flutter[23] s programovacím jazykem Dart. Dalšími, ale již méně významnými zástupci jsou např. Codename One[24], Xojo Mobile[25] nebo Qt Mobile[26] [4].

4.1.5 Modelem řízený přístup

Tento přístup vychází z konceptu vývoje řízeného modely (MDD, *Model-Driven Development*). Základní myšlenkou je tedy vygenerování aplikace specifické pro platformu z modelu společného napříč platformami. Generování aplikace z modelu, přesněji generování nativního kódu aplikace z modelu, má za cíl nejen zvýšení produktivity, ale také kvality výsledné implementace. Aby mohl být zmíněný model transformován do nativního kódu, musí být definován v některém z doménově specifických jazyků (DSL, *Domain-Specific Language*). A to konkrétně v takovém typu DSL, který je vyžadován zvoleným modelem řízeným apli-

[20] https://reactnative.dev
[21] https://nativescript.org
[22] https://dotnet.microsoft.com/en-us/apps/xamarin
[23] https://flutter.dev
[24] https://www.codenameone.com
[25] https://www.xojo.com/products/mobile.php
[26] https://www.qt.io/product/mobile-app-development/

kačním rámcem. DSL dále slouží vývojáři jako efektivní nástroj k zachycení a definování požadavků na výslednou aplikaci. Zároveň pomáhá zaměřit se při vývoji na funkce aplikace místo konkrétního technického řešení. Další výhodou používání DSL je fakt, že výsledek popisu aplikace má grafickou nebo textovou podobu, které snadno porozumí i člověk neorientující se v technickém či implementačním prostředí. Za nevýhodu DSL je považováno používání různých dialektů napříč aplikačními rámci. Pro nově vznikající frameworky modelem řízeného přístupu je navíc typické definování nového typu DSL, který bude frameworkem vyžadován. Současně je třeba zmínit, že frameworky negarantují vygenerování nativního kódu pro celou aplikaci a často záleží na zvoleném frameworku a s ním spojeným DSL, jaká část aplikace může být automaticky vygenerována [13].

I když je MDD obecně používaným konceptem, technické implementace podle modelem řízeného přístupu jsou mezi vývojáři multiplatformních aplikací vzácné. Autoři Umuhoza a Brambilla [36] rozdělují frameworky tohoto přístupu do dvou kategorií: výzkumná a komerční řešení. Ačkoliv je druhá z kategorií označena jako „komerční“, i její zástupci jsou používáni při vývoji jen zřídka. Do komerční kategorie se řadí nástroje Mendix App Platform[27], IBM Rational Rhapsody[28], WebRatio Mobile Platform[29] a Appian Mobile[30]. Kategorie výzkumných řešení čítá podstatně více zástupců, nejvýznamnějším z nich je framework MD2[31] a dále pak např. Applause[32] nebo JUSE4Android[33] [36].

4.2 Technologie pro vývoj multiplatformních mobilních aplikací

Tato sekce se věnuje konkrétním technologiím používaným ve vývoji multiplatformních aplikací pro mobilní zařízení. Popsány jsou zde tři nejpoužívanější technologie roku 2021. V celosvětovém průzkumu [39], jehož se v letech 2019 až 2021 zúčastnilo 31 743 vývojářů softwaru, uvedlo nejvíce dotázaných používání frameworku Flutter[34], konkrétně šlo o 42 % respondentů. Pro Flutter je navíc pozitivní, že jeho obliba od roku 2019, kdy framework získal ve stejném průzkumu 30 %, stále stoupá. Opačný trend zaznamenává druhá nejpoužívanější technologie — React Native[35]. Ten byl v letech 2019 a 2020 nejpoužívanějším frameworkem s 42 %, ale v roce 2021 dosáhl pouze na 38 %. I přes to je ale na druhém místě se značným náskokem před technologiemi Apache Cordova[36] a Ionic[37], jejichž používání uvedlo 16 % respondentů. I jejich popularita ale od roku 2019 stále klesá. V případě Apache Cordova je však výsledek zkreslen tím, že na jeho základu staví většina hybridních nástrojů, jak bylo popsáno v sekci 4.1. Proto jsou v této sekci popsány pouze technologie Flutter, React Native a Ionic. Všechny tyto nástroje jsou bezplatné a mají otevřený zdrojový kód (*open source*).

[27] https://www.mendix.com
[28] https://www.ibm.com/products/systems-design-rhapsody
[29] http://www.webratio.com
[30] http://www.appian.com
[31] https://wwu-pi.github.io/md2-web/
[32] https://github.com/applause/applause
[33] https://code.google.com/archive/p/juse4android/
[34] https://flutter.dev
[35] https://reactnative.dev
[36] https://cordova.apache.org
[37] https://ionicframework.com

4.2.1 Flutter

Flutter je multiplatformní framework kompilovaného přístupu pro vytváření mobilních, webových a desktopových aplikací. Za jeho vznikem stojí společnost Google, stejně jako za vznikem programovacího jazyka Dart, ve kterém je vedena implementace multiplatformní aplikace při použití frameworku [18]. Podle dokumentace [12] je Dart rychle se rozvíjející moderní programovací jazyk, který je optimalizován pro tvorbu uživatelských rozhraní. Zároveň je to jazyk dodržující objektově orientované paradigma a jeho syntaxe je přirovnávána k jazykům Java nebo C. Dart vznikl přímo pro účely frameworku Flutter, což s sebou přináší výhody i nevýhody. Výhodou je jeho optimalizace přímo pro účely použití, nevýhodou pak skutečnost, že je nutné se pro využití Flutteru naučit nový programovací jazyk a nelze tedy pracovat s jiným, více rozšířeným jazykem, případně s webovými technologiemi. Na rozdíl od multiplatformních nástrojů spoléhajících na webové technologie však poskytuje prostředí jazyka Dart nástroje pro snadné testování a odhalování chyb, což zvyšuje produktivitu a efektivitu vývoje [12].

Architektura frameworku Flutter se skládá ze tří vrstev. Tou nejnižší, tedy tou nejbližší platformě, je tzv. *Embedder*. Ten je implementován specificky pro každou platformu. Embedder zajišťuje integraci Flutteru do nativní aplikace, která je nově vytvořena pro vývoj ve frameworku, nebo může jít o již existující aplikaci a zdrojový kód s využitím frameworku Flutter do ní může být pouze přidán. Embedder zároveň umožňuje přístup ke službám operačního systému, jako je vykreslování povrchů, správa smyčky událostí nebo konfigurace vláken. Nad vrstvou Embedder se nachází vrstva Flutter Engine, která je již společná pro všechny platformy. Tato vrstva je jádrem frameworku a je implementována převážně v jazyce C++. Flutter Engine obsahuje základní funkce pro podporu všech aplikací frameworku Flutter. Zodpovědný je např. za vykreslování nových pohledů uživatelského rozhraní, architekturu modulů, překlad jazyka Dart nebo za síťovou komunikaci. Engine má na starost také komunikaci s nativními funkcemi, kterou řeší pomocí tzv. *Platform Channels*, což je způsob podstatně rychlejší než přemostění používané u hybridního a interpretovaného přístupu. Nejvyšší vrstva je nazvána přímo „framework Flutter“. Je implementována v jazyce Dart a je určena pro interakci s vývojáři aplikací. Tato vrstva zahrnuje několik vrstev knihoven, které může vývojář používat pro vytváření uživatelských rozhraní. Čím vyšší vrstva, tím větší míra abstrakce. Pro vývojáře jsou proto nejzajímavější dvě nejvyšší: vrstva *Widgetů* a nad ní vrstva sestávající z knihoven *Material* a *Cupertino*. Vrstva Widgetů umožňuje definovat vykreslované prvky a jejich kombinace, které lze znovu použít. Framework sám poskytuje řadu těchto znovupoužitelných vykreslovaných prvků (widgetů). Vývoj aplikace pak probíhá skládáním widgetů podle potřeb vzhledu a chování aplikace, přičemž každý widget může být složen z několika jiných widgetů, čímž vzniká hierarchie založená na kompozici. Knihovny Material a Cupertino potom na základě skládání widgetů tvoří komplexní sadu ovládacích prvků, jež je přizpůsobena konkrétní platformě. Knihovna Material nabízí ovládací prvky pro platformu Android, Cupertino pro iOS. [18].

4.2.2 React Native

React Native je framework interpretovaného přístupu pro vytváření aplikací pro platformy Android a iOS. Při vývoji multiplatformní aplikace je využíván jazyk JavaScript pro přístup k nativnímu rozhraní každé platformy a knihovna React[38] pro definování vzhledu a chování uživatelského rozhraní (UI, *User Interface*). Za vznikem frameworku React Na-

[38] https://reactjs.org

tive i knihovny React stojí společnost Facebook, obě technologie tak mají dobrou podporu a potenciál pro budoucí vývoj. React nabízí řadu komponent, tedy znovupoužitelných a vnořitelných částí kódu, které je možné při tvorbě UI ihned používat nebo upravovat, kombinovat a skládat tak, jak je pro potřeby aplikace nutné. Případně lze i definovat vlastní nové komponenty a použít je pro získání požadovaného vzhledu aplikace. Dostupné komponenty jsou např. prvek zobrazující text: *Text*, prvek zobrazující obrázek: *Image* nebo prvek obsahující další komponenty: *View*. Ke každé použité komponentě dokáže framework React Native za běhu vytvořit odpovídající komponentu konkrétní platformy, neboli nativní komponentu. To je umožněno pomocí přemostění v jazyce JavaScript, díky kterému lze i ovládat nativní funkce, jako je fotoaparát nebo Bluetooth [22].

Kromě komponent sdílených mezi platformami je možné, někdy i nevyhnutelné, psát zdrojový kód zvlášť pro každou platformu. Dochází k tomu např. v situacích, kdy je potřeba některou komponentu vizuálně přizpůsobit dané platformě. Zároveň existují komponenty poskytující některé vlastnosti pouze pro jednu z cílových platforem, což může být další motivací k psaní zdrojového kódu odděleně. Snahou však typicky zůstává sdílet mezi platformami co největší část kódu. I v případě, kdy je nutné vyvíjet vysoké procentu kódu zvlášť, zůstává výhodou, že zdrojový kód pro obě platformy je ve stejném programovacím jazyce. Implementace se tak, na rozdíl od nativního vývoje, stále odehrává v rámci jednoho vývojového týmu a snazší je proto i údržba kódu aplikace [22].

4.2.3 Ionic

Ionic je framework hybridního přístupu pro tvorbu uživatelského rozhraní (UI, *User Interface*) mobilních a desktopových aplikací pomocí webových technologií (HMTL, CSS, JavaScript). Pro efektivní vývoj UI nabízí Ionic řadu předem navržených komponent, které může vývojář dále upravit podle požadavků na aplikaci nebo umožnit automatické přizpůsobení jejich vzhledu a chování podle platformy běhu aplikace. Mimo to lze vývoj v Ionicu spojit s jedním ze tří podporovaných webových frameworků (Angular, React, Vue), místo implementace pomocí pouze základních technologií HMTL, CSS a JavaScript, a docílit tak snadnějšího a rychlejšího vývoje. Implementace UI aplikace tak může být vedena převážně ve zvoleném aplikačním rámci a z Ionicu může být využito pouze několik základních komponent, které zajišťují správné zobrazení obsahu ve výsledné mobilní aplikaci [23].

Jedna z hlavních myšlenek projektu Ionic — *„One codebase, running everywhere“* — jednoduše vystihuje cíl autorů. Tedy snahu umožnit vývojářům napsat jediný zdrojový kód a výsledek spouštět na různých platformách. Takovou možnost při použití frameworku Ionic zajišťuje Capacitor. Samotný Ionic lze tedy chápat jako framework zaměřený na tvorbu UI mobilních a desktopových aplikací a až po jeho spojení s nástrojem Capacitor jej lze označit za technologii pro tvorbu multiplatformních aplikací hybridního přístupu. Zatímco samostatné použití Ionicu nedává smysl, Capacitor je možné použít odděleně např. pro získání mobilní aplikace z existující webové aplikace. Vývojem ve frameworku Ionic je proto často označováno současné použití obou technologií, a to i z důvodu jejich úzkého provázání a společného vzniku [23].

Capacitor je open source projekt, který dokáže z webové aplikace vytvořit nativní mobilní aplikaci se stejným obsahem. Takto vzniklá aplikace je opravdu standardní nativní iOS nebo Android aplikací, která má stejnou strukturu a stejné prvky UI jako běžná nativní aplikace. Do popředí aplikace je ale vložena komponenta WebView, neboli webový prohlížeč vestavěný do nativní aplikace, který slouží pro zobrazení webového obsahu. Ve vzniklé nativní aplikaci je potom WebView jedinou komponentou celé aplikace a zobra-

zuje původní webovou aplikaci. Capacitor rozšiřuje běžnou funkčnost WebView mimo jiné o možnost přístupu k nativním funkcím operačního systému a nativnímu kódu aplikace. Z toho vychází dvě pro vývojáře zásadní vlastnosti. Za prvé lze přistupovat k nativním prvkům platformy, např. k fotoaparátu, z webového JavaScriptu a za druhé je možné vyvíjet části multiplatformní aplikace v nativním programovacím jazyce. Výhodou je také možnost integrace Capacitoru do již existující webové aplikace, jejíž výsledkem je poměrně snadné a rychlé získání mobilní aplikace. Alternativou k frameworku Capacitor je Apache Cordova, který funguje na podobném principu. Autoři Ionicu však preferují Capacitor a označují jej za moderního nástupce frameworku Cordova [29].

4.3 Shrnutí

První část této kapitoly byla věnována možnostem tvorby multiplatformních aplikací, kde byly postupně popsány přístupy webový, hybridní, interpretovaný, kompilovaný a modelem řízený.

- **Webový přístup** umožňuje vývoj aplikace ve webových technologiích. Multiplatformní běh zajišťuje spouštěním aplikace ve webovém prohlížeči a přístup k nativním funkcím platformy je tak zprostředkován webovým prohlížečem. Nejznámějším nástrojem pro tvorbu ve webovém přístupu je PWABuilder.
- **Hybridní přístup** rovněž nabízí možnost vývoje aplikace ve webových technologiích. Multiplatformní běh je zajištěn zobrazením aplikace v nativní komponentě WebView a přístup k nativním funkcí je získán díky metodě přemostění. Nejznámějšími nástroji hybridního přístupu jsou Cordova, Capacitor a Ionic.
- **Interpretovaný přístup** umožňuje vývoj aplikace v různých programovacích jazycích, pro které existuje framework přístupu, nejčastěji je používán jazyk JavaScript. Multiplatformní běh je zajištěn transformací zdrojového kódu aplikace do nativních prvků uživatelského rozhraní pomocí vestavěných interpretů. Přístup k nativním funkcím je opět získán díky metodě přemostění. Nejznámějšími nástroji interpretovaného přístupu jsou React Native, NativeScript a Titanium Appcelerator.
- **Kompilovaný přístup** nabízí vývoj aplikace v různých programovacích jazycích, pro které existuje framework přístupu. Multiplatformní běh je zajištěn transformací zdrojového kódu aplikace do kódu spustitelného na cílové platformě pomocí kompilátoru. Přístup k nativním funkcím platformy poskytuje SDK použitého frameworku. Nejznámějšími nástroji kompilovaného přístupu jsou Flutter a Xamarin.
- **Modelem řízený přístup** umožňuje vývoj aplikace v doménově specifických jazycích. Multiplatformní běh je zajištěn generováním nativního kódu z modelu a přístup k nativním funkcím je poskytnut přímo z vzniklého nativního kódu, ale může se lišit podle použitého frameworku. Nejznámějšími nástroji modelem řízeného přístupu jsou MD2, Mendix a App Platform.

Druhá část kapitoly popisovala technologie pro vývoj multiplatformních mobilních aplikací, konkrétně tři nejpoužívanější technologie roku 2021 — Flutter, React Native a Ionic.

- **Flutter** spadá pod kompilovaný přístup a za jeho vývojem stojí společnost Google. Flutter nabízí vývoj aplikací v programovacím jazyce Dart a pro tvorbu uživatelských

rozhraní používá princip skládání připravených nebo vlastních widgetů. Jeho výhody jsou snadné testování a odhalování chyb ve zdrojovém kódu, a lepší výkon výsledné aplikace. Nevýhodou je nutnost použití jazyka Dart.

- **React Native** se řadí do interpretovaného přístupu a za jeho vývojem stojí společnost Facebook. React Native umožňuje vývoj aplikací v jazyce JavaScript s použitím knihovny React a pro tvorbu uživatelských rozhraní používá princip skládání připravených nebo vlastních komponent. Jeho výhodou je používání webových technologií pro vývoj, nevýhodou pak častá nutnost psát zdrojový kód specifický pro platformu.
- **Ionic** je nástroj hybridního přístupu, za kterým stojí společnost Drifty. Ionic umožňuje vyvíjet aplikace ve webových technologiích za použití nástroje Capacitor. Princip tvorby uživatelského rozhraní zajišťuje skládáním komponent nebo využitím možností webových frameworků. Výhodou je opět použití webových technologií při vývoji. Dále pak možnost integrace do již existující aplikace a také možnost využít pouze základní komponenty frameworku, a přesto získat multiplatformní aplikaci.

Kapitola 5

Analýza současného stavu

Cílem této diplomové práce je ve spolupráci s firmou Logimic vytvořit mobilní aplikaci pro správu servisních požadavků chytrých měst. Logimic [28] je společnost zabývající se vývojem a dodávkou softwarových řešení pro oblast internetu věcí (IoT, *Internet of Things*). Jejími typickými zákazníky jsou průmyslové firmy a města, která chtějí do svého provozu zapojit chytrá zařízení a pomocí softwarového vybavení umožnit jejich správu a řízení. Zákazníkem může být firma, která požaduje komplexní řešení, tedy dodávku řady chytrých zařízení včetně softwarového vybavení, ale také firma, která již nějaká chytrá zařízení využívá a chce svůj provoz pouze rozšířit o nová a přidat softwarový produkt pro řízení všech zařízení jednotně. Společnost Logimic nabízí řešení v obou případech. Takové řešení bývá označováno jako platforma a zahrnuje, kromě dodávky chytrých zařízení a jejich uvedení do provozu, také přístup ke cloudovým službám uchovávajícím a zpracovávajícím data z chytrých zařízení, k webové aplikaci prezentující data podle potřeb zákazníka a případně k mobilním aplikacím specifickým pro konkrétní oblast.

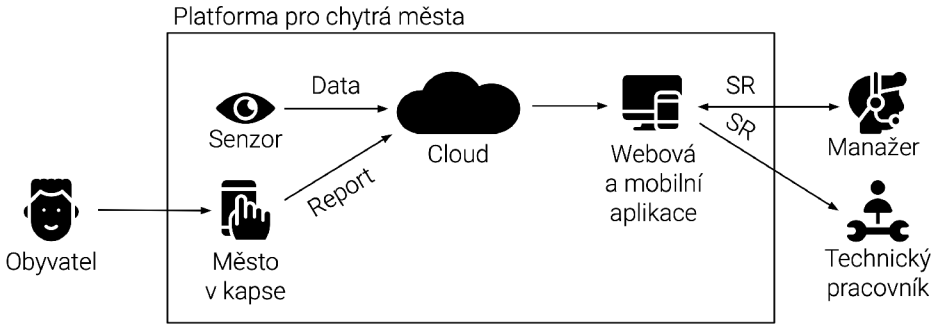
Proces provádění analýzy měl tři fáze. Cílem první z nich bylo seznámí se s platformou pro chytrá města, především s podporou pro hlášení závad a se způsobem tvorby a řešení servisních požadavků. Pro tento účel mi byl poskytnut přístup k používané webové i mobilní aplikaci. Obě jsem mohla prozkoumat jak z pohledu uživatele, tak z pohledu vývojáře. Dostala jsem přístup k dokumentaci, zdrojovému kódu i k databázi. Další podstatné informace jsem se dozvěděla z diskuzí se zaměstnanci firmy Logimic a také díky zapojení se do pravidelných schůzek týkajících se aktuálního dění ve firmě. Přehled takto získaných informací sumarizuje sekce 5.1. Druhá fáze spočívala v nalezení cílových uživatelů vznikající mobilní aplikace a popisu jejich potřeb v sekci 5.2. Ve třetí fázi jsem definovala požadavky na aplikaci na základě informací získaných v předchozích fázích a výsledek pak shrnula v sekci 5.3.

5.1 Současný stav

V oblasti chytrých měst nabízí firma Logimic řešení založená na bezdrátových technologiích, cloudových službách, webových a mobilních aplikacích. Celý systém spolupracujících technologií je označován jako platforma pro chytrá města a jeho základem jsou chytrá zařízení jako senzory pro monitoring ovzduší, odpadů, veřejného osvětlení, parkování apod. Tyto senzory sbírají data a odesílají je cloudovým službám, což je pojem označující výpočetní kapacitu na vzdáleném serveru. Data z cloudových služeb jsou pak prezentována uživatelům

pomocí webové aplikace, odkud lze některá zařízení také ovládat, např. formou nastavení režimu pro veřejné osvětlení ve městě.

Mimo právě popsanou základní strukturu platformy může město od firmy Logimic získat také mobilní aplikaci pro hlášení závad ve městě pojmenovanou jako *Město v kapse*. Taková aplikace umožňuje veřejnosti hlásit jakékoliv nedostatky, které ve městě objeví. To může být např. přeplněný odpadkový koš, překážející koloběžka, poškozený sloup či chodník a další. Hlášené nedostatky se pak dostávají k odpovědným osobám, které mohou problém vyřešit a o výsledku následně informovat autora hlášení, pokud o to požádá. Účelem vzniku Města v kapse je snaha propojovat vedení města, dodavatele technických služeb a obyvatele, což může vést k lepšímu chodu města.

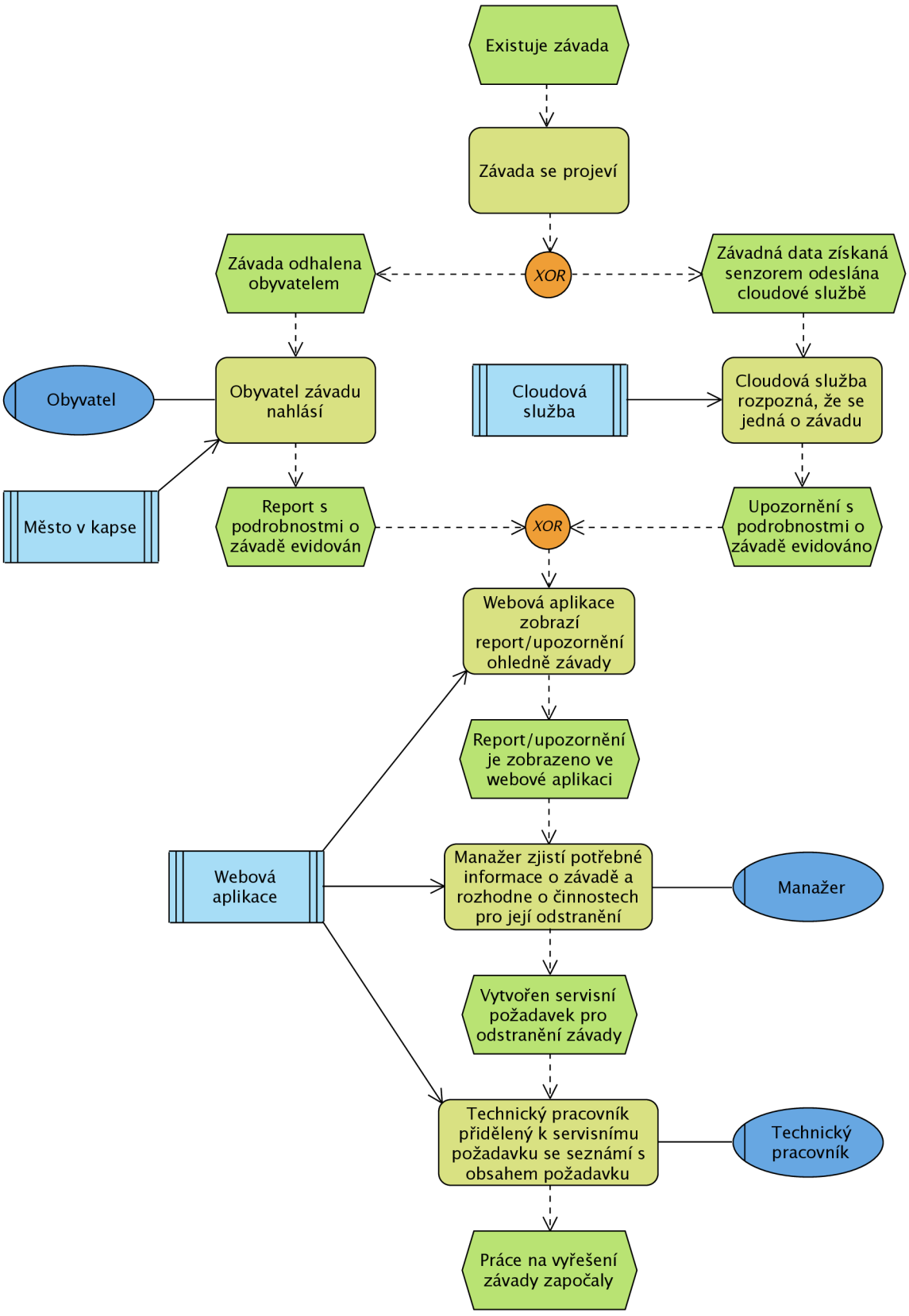


Obrázek 5.1: Schéma zapojení platformy pro chytrá města do řízení města

Schéma na obrázku 5.1 popisuje způsob zapojení platformy pro chytrá města do řízení města. Ve schématu vystupují tři skupiny uživatelů: *obyvatel města*, *manažer* a *technický pracovník*. Platforma pro chytrá města je vyznačena obdélníkem obklopujícím jednotlivé technologie platformy. Konkrétně se schéma věnuje situaci od vzniku a odhalení nějaké závady po začátek prací na jejím vyřešení. Šipky naznačují směr toku dat v této situaci a detailně tuto situaci popisuje *Event-driven process chain* (EPC) diagram na obrázku 5.2. K odhalení závady může dojít dvěma způsoby. Za prvé může závadu nahlásit obyvatel pomocí aplikace Město v kapse. Takové hlášení je označováno jako *report* a je předáno do cloudové služby. Za druhé může k odhalení dojít na základě vyhodnocení dat ze senzorů přímo v cloudové službě. V obou případech následuje zobrazení upozornění na závadu ve webové aplikaci, kde se o závadě může dozvědět manažer. Ten vytvoří tzv. *servisní požadavek* (SR, *service request*) s informacemi o závadě a přiřadí k němu odpovědnou osobu, obvykle technického pracovníka města. Technický pracovník se seznámí s obsahem servisního požadavku a začne provádět práce vedoucí k vyřešení závady.

5.1.1 Servisní požadavky

Platforma pro chytrá města aktuálně umožňuje tvorbu a řešení servisních požadavků v rámci webové aplikace. Za tvorbou servisních požadavků stojí manažer, který od vytvoření požadavku a přiřazení odpovědné osoby může sledovat vývoj prací na tomto požadavku. Ve chvíli, kdy je práce podle uvážení technického pracovníka hotová, požádá manažera o uzavření servisního požadavku. Této žádosti může nebo nemusí manažer vyhovět na základě dodaného vyjádření a fotografie dokumentující odvedenou práci. Pokud jde o závadu nahlášenou obyvatelem a dojde k dokončení všech potřebných činností k vyřešení závady, může být obyvatel o výsledku informován komunikačním kanálem, který zvolí dané město.



Obrázek 5.2: Event-driven process chain (EPC) popisující proces zpracování závady pomocí platformy pro chytrá města od vzniku závady po začátek prací na jejím vyřešení.

Správa servisních požadavků zaplňuje značnou část práce manažera. Snahou proto je tuto práci manažerovi co nejvíce usnadnit. Aktuálně je možné každý servisní požadavek spojit s reportem od obyvatele nebo s konkrétním zařízením, resp. senzorem. To manažerovi umožňuje sledovat, jak se práce směřující k vyřešení závady vyvíjí, a může díky tomu snadno rozeznat, zda již byla konkrétní závada vyřešena, či nikoliv. V současném řešení může být ke každému reportu vytvořen pouze jeden požadavek, který může být v jednu chvíli přiřazen nejvýše jedné odpovědné osobě. V průběhu zpracování požadavku však může dojít ke změně odpovědné osoby, což poskytuje možnost předání práce např. pracovníkovi s jiným technickým vybavením.

V souvislosti s tvorbou servisních požadavků musí manažer také řešit duplicity na úrovni reportů a upozornění. Může se totiž stát, že více obyvatel nahlásí jednu závadu nebo že je postupně generováno více upozornění na závadu na základě dat ze senzorů. V případě duplicitních reportů pak vzhledem ke způsobu propojování reportů se servisními požadavky nastává problém buď s informováním všech obyvatel, kteří závadu nahlásili, nebo se vznikem duplicitních servisních požadavků. Zdrojem duplicit může být ale také bezpečnostní incident, kdy útočník způsobí vznik mnoha reportů nebo upozornění, s nimiž se musí manažer vypořádat.

5.2 Cíloví uživatelé mobilní aplikace

Cílem této diplomové práce je vytvořit mobilní aplikaci pro správu servisních požadavků v chytrých městech, která bude vyhovovat potřebám cílových uživatelů firmy Logimic. Za cílového uživatele vznikající mobilní aplikace lze obecně označit město. Konkrétní uživatele ve městě je pak možné rozdělit do dvou skupin: *manažeři* a *technické služby*, přičemž pro každého z nich má aplikace jiný způsob využití a přínos.

- **Manažer** je cílovým uživatelem aplikace, jehož úkolem je v různé míře a podobě řídit chod města, případně jeho části. V oblasti servisních požadavků potřebuje manažer zajistit efektivní řízení servisních požadavků a s nimi související řízení technických služeb. To zahrnuje potřebu vytváření nových servisních požadavků, organizaci technických pracovníků a činností vedoucích k vyřešení servisních požadavků a možnost sledování vývoje zadaných prací. Pro manažera je proto důležité, aby mohl na základě reportů a upozornění od senzorů snadno vytvářet servisní požadavky pro potřebné činnosti a přiřazovat k nim odpovědné osoby. Také, aby měl přehled o aktuálně prováděných činnostech a aby mohl mezi existujícími servisními požadavky jednoduše vyhledávat. Dále pak, aby mohl schvalovat servisní požadavky, které jsou z pohledu odpovědné osoby dokončeny, a aby měl možnost vypořádat se s duplicitními reporty a upozorněními od senzorů. Manažer může mít navíc administrátorské oprávnění. V takovém případě je vhodné, aby měl v aplikaci přístup do administrátorského modulu, kde může provádět akce jako přidání dalšího uživatele nebo změna role existujícího uživatele.
- **Technický pracovník** je osoba, která je schopna provést práci definovanou v servisním požadavku. Pro uživatele z technických služeb je proto zásadní, aby se o servisním požadavku dozvěděl. Dále také, aby mu struktura požadavku usnadnila zorientovat se v informacích o činnosti, jejíž provedení je požadováno, a aby měl možnost obrátit se na vhodnou osobu v případě nejasností. Dále pak, aby mohl vyhledávat mezi požadavky, které mu byly přiřazeny, a aby měl po provedení činnosti možnost informovat

manažera o jejím dokončení. Mimo to je pro technického pracovníka podstatný rychlý přístup k poloze a fotoaparátu mobilního telefonu, aby věděl, kde má danou činnost provést, a aby mohl fotografií zdokumentovat výsledek svojí činnosti.

Mezi uživatele mobilní aplikace pro správu servisních požadavků nebudou patřit obyvatelé, kteří ani v současně používané verzi platformy pro chytrá města nemají možnost spravovat servisní požadavky. Je však zásadní, aby byla vznikající mobilní aplikace snadno propojitelná s modulem platformy, který umožní informovat obyvatele o stavu zpracování servisních požadavků.

Pro města jako uživatele mobilní aplikace je zásadní, aby byla aplikace dostupná všem zaměstnancům, kteří s ní budou pracovat. Aplikace cílí především na města v USA, kde se podíl na trhu mobilních operačních systémů mezi lety 2018 a 2022 pohybuje okolo 45 % pro Android a 54 % pro iOS [35].

5.3 Požadavky na mobilní aplikaci

Aktuálně umožňuje platforma pro chytrá města správu servisních požadavků uvnitř webové aplikace. Mezi zákazníky se ale čím dál častěji objevuje poptávka po mobilní aplikaci, která by zjednodušila práci se servisními požadavky na mobilních telefonech. Mobilní aplikace oproti webové přináší určité výhody. Mezi ty nejvýznamnější v oblasti správy servisních požadavků patří přímý přístup k fotoaparátu, offline dostupnost nebo přívětivé uživatelské rozhraní.

Nová mobilní aplikace by měla být dobře dostupná pro mobilní zařízení operačních systémů Android i iOS, měla by zapadat do běžného používání platformy pro chytrá města a měla by být pro firmu Logimic do budoucna dobře provozovatelná a udržovatelná. Aplikace by měla nabízet funkčnost v rozsahu aktuálně používaného modulu pro práci se servisními požadavky v současné webové aplikaci. Pro manažera by tak měla nabízet funkce:

- sledování statistik týkajících se servisních požadavků,
- zobrazování seznamů servisních požadavků podle potřebných kritérií (např. pouze dokončené požadavky nebo požadavky přiřazené konkrétní osobě),
- zobrazování mapy s vyznačenými polohami servisních požadavků,
- zobrazování detailů vybraného servisního požadavku,
- vytváření nového požadavku,
- správa existujících požadavků (např. změna přiřazené osoby nebo uzavření požadavku).

Přístup do mobilní aplikace by dále měl mít také technický pracovník, jemuž by mělo být umožněno:

- zobrazování seznamů servisních požadavků jemu přiřazených podle potřebných kritérií (např. pouze nedokončené požadavky),
- zobrazování mapy s vyznačenými polohami servisních požadavků jemu přiřazených,
- zobrazování detailů vybraného servisního požadavku,

- správa požadavků jemu přiřazených.

Pro manažera s administrátorskými právy by navíc měla aplikace umožňovat přístup k administrátorskému modulu. Pro všechny uživatele by měl být dostupný modul pro správu vlastního uživatelského účtu.

5.3.1 Požadavky na úpravy

Mimo specifikovanou funkčnost byly definovány požadavky na úpravu modulu servisních požadavků, které vychází ze zpětné vazby uživatelů a ze zkušeností zaměstnanců firmy z běžného používání platformy. Požadavky tak poukazují na různé nedostatky, mají různou úroveň podrobnosti popisu, některé mohou být náročnější na zpracování než jiné a mohou přinášet různou míru vylepšení. Požadavky na úpravy říkají, co by měla aplikace uživateli umožnit, a jsou definovány následovně:

P.1 Vytvářet více servisních požadavků pro jeden report/upozornění.

P.2 Jednoduše zobrazit servisní požadavky čekající na uzavření z role manažera.

P.3 Automaticky informovat technického pracovníka o novém servisním požadavku, ke kterému byl přiřazen.

P.4 Automaticky vytvářet nové servisní požadavky z reportů.

P.5 Jednoduše spravovat přiřazené servisní požadavky z role technického pracovníka.

P.6 Efektivně řešit duplicity na úrovni servisních požadavků.

P.7 Uzavírat servisní požadavky ihned po jejich vzniku.

Důležitým obecným požadavkem na aplikaci je také nutnost udržet modul servisních požadavků co nejjednodušší. Snahou je proces správy servisních požadavků zjednodušit a zefektivnit při zachování, případně rozšíření současného rozsahu funkčnosti.

Kapitola 6

Návrh mobilní aplikace

Hlavním důvodem pro vznik aplikace je narůstající poptávka zákazníků firmy Logimic po mobilní aplikaci, která by byla zaměřena na správu servisních požadavků. Výhody mobilní aplikace ocení uživatelé především při práci „v terénu“, kde je použití mobilního telefonu pro správu servisních požadavků běžné. V takových situacích může být pro uživatele přínosný přímý přístup mobilní aplikace k poloze a fotoaparátu telefonu, což jsou nativní funkce mobilního telefonu často využívané při řešení servisních požadavků. I z webové aplikace má uživatel k těmto funkcím přístup, oprávnění k přístupu však musí uživatel udělit celému webovému prohlížeči, nikoliv přímo konkrétní aplikaci. Další odlišností od webové aplikace je možnost použití mobilní aplikace alespoň v omezeném režimu bez připojení k internetu. Mobilní aplikace také obvykle přináší větší uživatelský komfort při používání. V souvislosti s implementací nové mobilní aplikace je dále vhodné provést úpravu modulu servisních požadavků, jejíž výsledkem by mělo být splnění některých z požadavků definovaných v podsekci 5.3.1.

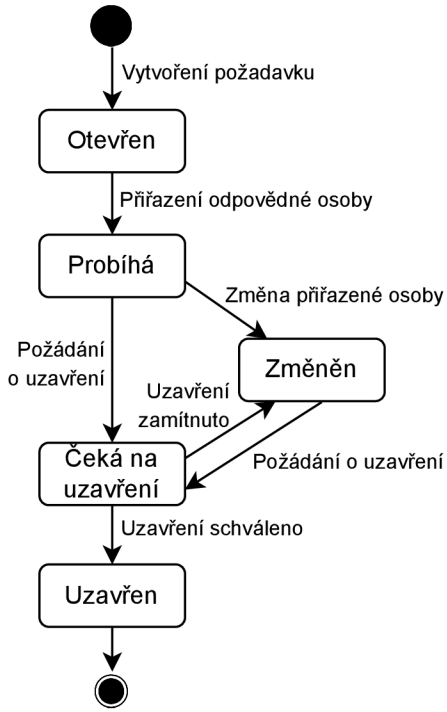
Tato kapitola se věnuje návrhu mobilní aplikace, který vychází z analýzy současného stavu, potřeb cílových uživatelů a požadavků uvedených v kapitole 5. Nejprve jsou v sekci 6.1 popsány změny ve způsobu práce se servisními požadavky. Dále jsou v sekci 6.2 definovány role a akce budoucích uživatelů aplikace. Následuje sekce 6.3 zabývající se architekturou výsledné mobilní aplikace a kapitola je zakončena představením návrhu uživatelského rozhraní 6.4.

6.1 Servisní požadavky

Aby mohly být splněny některé požadavky definované v podsekci 5.3.1, musí dojít k úpravám ve způsobu práce se servisními požadavky. Úpravy se v první řadě týkají zavedení uspořádání servisních požadavků do struktury grafu, s čímž souvisí i další úpravy. Grafová struktura určuje, jak mohou být servisní požadavky vzájemně nově propojovány. Každý servisní požadavek může mít několik potomků, tedy jiných servisních požadavků, které popisují dílčí činnosti vedoucí ke splnění rodičovského servisního požadavku. Tím bude uspokojena potřeba z požadavku **P.1**: *Vytvářet více servisních požadavků pro jeden report/upozornění.* I přes uspořádání servisních požadavků do grafové struktury nebude docházet k jejich automatickému uzavírání po dokončení všech dílčích servisních požadavků. Každý servisní požadavek bude i nadále přiřazen odpovědné osobě, která o jeho uzavření musí požádat. Tento přístup vychází z nutnosti kontrolovat, zda je splnění dílčích servisních požadavků po-

stačující ke splnění rodičovského požadavku, tedy zda je rozdělení rodičovského požadavku do dílčích činností kompletní.

Ke splnění požadavku **P.2**: *Jednoduše zobrazit servisní požadavky čekající na uzavření z role manažera* dojde díky rozšíření aktuálně používaných stavů servisního požadavku o stav „Čeká na uzavření“, tedy o stav označující čekání na uzavření. Na základě tohoto stavu pak bude možné servisní požadavky filtrovat, a zobrazit tak pouze požadavky čekající na uzavření. Způsob, jakým se mění stavy servisního požadavku v průběhu jeho zpracování, znázorňuje stavový diagram na obrázku 6.1.



Obrázek 6.1: Stavový diagram znázorňující přechody mezi stavy servisního požadavku

Další úprava spočívá v automatické tvorbě servisního požadavku z reportu od obyvatele. V takovém servisním požadavku budou automaticky uchovány všechny významné informace z reportu. Další informace, jež nelze získat automaticky, budou doplněny manažerem nebo odpovědnou osobou po jejím přiřazení. Takto vzniklé servisní požadavky budou označovány jako *servisní požadavky nejvyšší úrovně*. Tímto způsobem dojde ke splnění požadavku **P.4**: *Automaticky vytvářet nové servisní požadavky z reportů*. Požadavky nejvyšší úrovně mohou vzniknout také z upozornění od zařízení. Jejich automatické vytváření však vzhledem ke způsobu jejich zpracování na backendu platformy pro chytrá města není možné. Proto bude i nadále nutné servisní požadavky na základě upozornění od zařízení vytvářet manuálně.

Dále je třeba umožnit řešení duplicit. Z předchozího odstavce vyplývá, že při nahlášení více reportů ohledně jedné závady budou vznikat duplicity na nejvyšší úrovni servisních požadavků. V takovém případě bude možné duplicitní požadavek spojit s již existujícím požadavkem nejvyšší úrovně. Duplicitní požadavek tak bude uchovávat odkaz na originální požadavek ve formě jeho unikátního identifikátoru. Tím by mělo dojít ke splnění požadavku **P.6**: *Efektivně řešit duplicity na úrovni servisních požadavků.*

Požadavek **P.3**: *Automaticky informovat technického pracovníka o novém servisním požadavku, ke kterému byl přiřazen* umožní mobilní aplikace zapojením notifikací. Pokud bude

mít technický pracovník aplikaci ve svém mobilní telefonu a povolí zobrazování notifikací, je tímto způsobem požadavek efektivně řešitelný. Pro práci technického pracovníka je zásadní také požadavek **P.5**: *Jednoduše spravovat přiřazené servisní požadavky z role technického pracovníka.* Vznikající aplikace by měla pracovníkovi usnadnit správu přiřazených servisních požadavků jednak tím, že jako mobilní aplikace by měla přinášet přívětivé uživatelské rozhraní, a pak tím, že pracovník získá v aplikaci přístup pouze k nejnutnějším datům a operacím, tedy pouze k servisním požadavkům jemu přiřazeným a k operacím s nimi spojeným.

Požadavek **P.7**: *Uzavírat reporty a upozornění bez vytvoření servisního požadavku* nebude naplněn vzhledem k zavedení automatické tvorby servisních požadavků ze všech reportů. Report tak nemůže být přímo uzavřen, aplikace ale umožní uzavírání servisních požadavků nejvyšší úrovně, aniž by musely vzniknout dílčí servisní požadavky, což sémanticky odpovídá účelu tohoto požadavku.

6.2 Role a akce uživatelů

S výslednou mobilní aplikací budou pracovat uživatelé dvou rolí: *manažer* a *technický pracovník*. Na základě role pak bude uživateli umožněno přistupovat k různým datům a provádět různé akce v aplikaci. Pro obě role musí aplikace nabízet funkce definované požadavky v sekci 5.3 a zároveň musí být přidány akce umožňující práci se servisními požadavky tak, jak bylo popsáno v předchozí sekci 6.1. Následují proto seznamy akcí, které mohou uživatelé jednotlivých rolí v aplikaci provádět.

Manažer může:

- zobrazit statistiky zahrnující přehled servisních požadavků podle statusů, priority, přiřazené osoby a podle kategorie,
- zobrazit všechny servisní požadavky čekající na uzavření,
- zobrazit seznam servisních požadavků podle zvolených kritérií,
- zobrazit mapu s vyznačenými polohami servisních požadavků,
- zobrazit detail zvoleného servisního požadavku,
- vytvořit nový dílčí servisní požadavek k již existujícímu,
- přidat existující servisní požadavek jako dílčí k již existujícímu,
- odebrat dílčí servisní požadavek,
- modifikovat existující servisní požadavky,
- sloučit duplicitní servisní požadavek nejvyšší úrovně s již existujícím požadavkem nejvyšší úrovně,
- přidat informace ohledně řešení servisního požadavku,
- požádat o uzavření servisního požadavku,
- schvalovat, případně odmítat žádosti o uzavření servisních požadavků.

Manažer s administrátorskými právy může navíc:

- přistupovat do administrátorskému modulu.

Technický pracovník může:

- zobrazit seznam jemu přiřazených servisních požadavků podle zvolených kritérií,
- zobrazit mapu s vyznačenými polohami jemu přiřazených servisních požadavků,
- zobrazit detail zvoleného servisního požadavku,
- přidat informace ohledně řešení jemu přiřazeného servisního požadavku,
- požádat o uzavření jemu přiřazeného servisního požadavku.

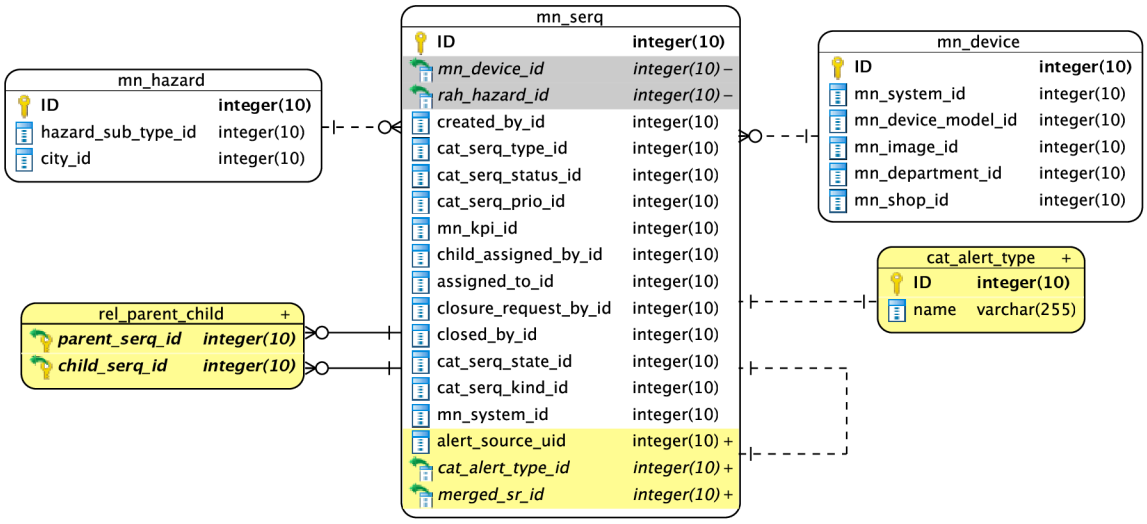
6.3 Architektura

Mobilní aplikace bude vytvořena jako multiplatformní. Vývoj nativní mobilní aplikace pro každý z požadovaných operačních systémů (Android, iOS) by byl totiž podstatně náročnější a výsledek by byl pro firmu do budoucna špatně udržovatelný. Z multiplatformních přístupů k tvorbě aplikací popsaných v kapitole 4 byl zvolen hybridní přístup. Hlavním kritériem při výběru byl požadavek na snadný budoucí provoz a údržbu výsledné aplikace. Vzhledem k faktu, že firma Logimic aktuálně vyvíjí softwarové produkty převážně v jazyce TypeScript za použití frameworku Angular, nabízí se použití některého z přístupů, které umožňují vývoj ve webových technologiích. Těmi jsou webový, hybridní a interpretovaný přístup, z nichž hybridní přístup má pro firmu největší potenciál. Na rozdíl od webového přístupu jsou hybridní aplikace dostupné v typických obchodech s mobilními aplikacemi (Google Play, App Store) a poskytují přímý přístup k nativním funkcím mobilního zařízení. Ve srovnání s interpretovaným přístupem může být hybridní aplikace spuštěna nejen na mobilním zařízení, ale také ve webovém prohlížeči. Do budoucna tak může být výsledná hybridní aplikace součástí platformy pro chytrá města také jako modul existující webové aplikace. Hybridní způsob vývoje zároveň nabízí možnost využít existující backendové řešení webové aplikace, které zpřístupňuje svoje data přes rozhraní REST (*Representational State Transfer*). Pro vznikající mobilní aplikaci tak stačí navrhnout změny v datovém modelu a definovat nové přístupové body (*endpoints*) rozhraní REST.

6.3.1 Datový model

Datový model vychází z aktuálně využívaného datového modelu na platformě pro chytrá města. Kvůli úpravám modulu servisních požadavků však aktuální datový model musí projít několika změnami. Návrh datového modelu je zachycen na obrázku 6.2, kde je znázorněno celkem pět tabulek a vazby mezi nimi. Pro jednoduchost jsou v modelu zachyceny pouze klíčové tabulky a z jejich atributů jen ty, které odkazují na jiné tabulky. V zobrazených tabulkách jsou vyznačeny změny oproti stávajícímu datovému modelu. Šedou barvou a symbolem − jsou zvýrazněny atributy, které již nebude třeba uchovávat. Žlutou barvou a symbolem + jsou označeny nově přidané tabulky a atributy.

Tabulka `mn_hazard` uchovává data reportů, `mn_device` shromažďuje chytrá zařízení a `mn_serq` je tabulka servisních požadavků. Tyto tři tabulky jsou aktuálně využívány tak, že servisní požadavek má uložen odkaz na report (`mn_hazard`) nebo na chytré zařízení (`mn_device`), na jehož základě byl vytvořen. Pro zachování konzistence je pak nutné zajistit, aby byl vyplněn nejvýše jeden z těchto atributů. Nově je potřeba přidat tabulku



Obrázek 6.2: Datový model aplikace s vyznačenými změnami

`rel_parent_child` jako spojovací tabulku pro dva servisní požadavky ve vztahu rodičovský požadavek – dílčí požadavek, aby mohlo dojít k propojování uzlů (servisních požadavků) do grafové struktury. Další novou tabulkou bude `cat_alert_type`, což je tabulka uchovávající výčet typů zdrojů servisních požadavků. Pro nynější potřeby tak výčet bude zahrnovat dva typy: report a chytré zařízení. Tato tabulka souvisí s odebráním atributů `mn_device_id` a `mn_hazard_id` a přidáním atributů `alert_source_uid` a `cat_alert_type_id` tabulky servisního požadavku (`mn_serq`). V atributu `alert_source_uid` bude uchován odkaz buď na chytré zařízení (`mn_device`), nebo odkaz na report (`mn_hazard`). Aby mohlo dojít ke zjištění, zda se jedná o odkaz na zařízení nebo report, je v atributu `cat_alert_type_id` uložen odkaz na tabulku `cat_alert_type` s typem zdroje servisního požadavku. Požadavek nejvyšší úrovně uchovává v atributech `alert_source_uid` a `cat_alert_type_id` odkaz na zdroj, z něhož přímo vznikl. Požadavek, který je dílčí, dědí data těchto atributů ze svého rodičovského požadavku. Zavedením takové struktury dat může v budoucnu dojít k rozšíření typů zdrojů servisních požadavků bez nutnosti provádět změny v datovém modelu. Pro možnost řešení duplicit bude navíc do tabulky `mn_serq` nutné zavést atribut `merged_sr_id`, který bude pro duplicitní požadavek uchovávat odkaz na jiný servisní požadavek, k němuž je duplicitní. Pro ostatní požadavky bude atribut ponechán prázdný.

6.3.2 Přístupové body

Přístupové body rozhraní REST slouží k předávání dat uvnitř aplikace aplikace (komunikace backend – frontend), čímž umožňují reagovat na akce uživatele v uživatelském rozhraní aplikace. Pro frontend vznikající mobilní aplikace je nezbytné definovat přístupové body pro následující komunikaci dat:

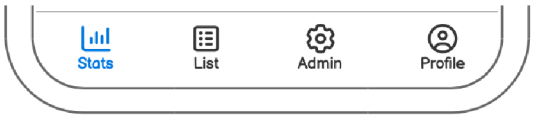
- získání statistických dat týkajících se servisních požadavků (podle priority, stavu a přiřazené osoby),
- získání seznamu servisních požadavků na základě volitelných filtrů, včetně získání seznamu servisních požadavků nejvyšší úrovně na základě volitelných filtrů,

- získání rodičovských servisních požadavků na základě identifikátoru servisního požadavku,
- získání dílčích servisních požadavků na základě identifikátoru servisního požadavku,
- získání detailních informací o servisním požadavku na základě jeho identifikátoru,
- úprava dat existujícího servisního požadavku na základě jeho identifikátoru, včetně úpravy přiřazených dílčích a rodičovských servisních požadavků,
- přidání, úprava a odebrání identifikátoru servisního požadavku nejvyšší úrovně k duplicitnímu požadavku nejvyšší úrovně,
- vytvoření nového servisního požadavku.

6.4 Uživatelské rozhraní

Návrh uživatelského rozhraní vychází z grafického vzhledu aktuálně používané webové platformy, přidává nově definované funkce a zavádí grafické prvky a rozložení, které jsou typické pro mobilní aplikace. Takovým prvkem je např. dolní navigační menu, které bude uživateli dostupné po celou dobu používání aplikace a bude nabízet rychlý přístup k jednotlivým pohledům. Pro manažera s administrátorskými právy může menu vypadat přibližně tak, jak je naznačeno na obrázku 6.3. Bude tedy nabízet přístup k pohledům:

- *Stats (statistiky)* – pohled se statistikami týkajícími se servisních požadavků,
- *List (seznam)* – pohled umožňující zobrazit buď seznam servisních požadavků podle zvolených kritérií, nebo mapu s vyznačenými polohami servisních požadavků,
- *Admin (administrátorská sekce)* – pohled nabízející administrátorské akce,
- *Profile (profil uživatele)* – pohled s informacemi o uživatelském účtu.

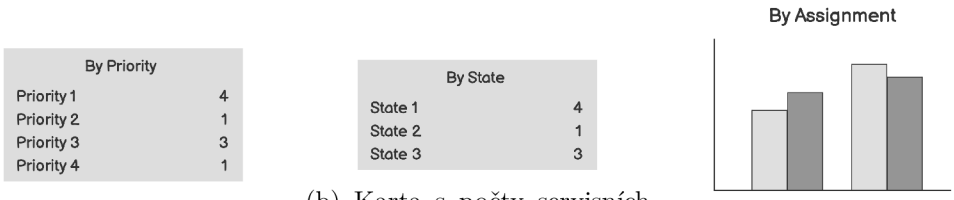


Obrázek 6.3: Dolní lišta mobilní aplikace

Technickému pracovníkovi bude zobrazena podobná lišta, dostupné však budou pouze pohledy *List* a *Profile*.

6.4.1 Stats (statistiky)

Obrazovka *Stats* nabízí pohled na několik statistik. První dvě statistiky jsou vykresleny formou karet a zobrazují servisní požadavky podle priority a stavu. Třetí statistika ukazuje pomocí grafu, kolik otevřených servisních požadavků je komu přiřazeno. Návrhy těchto grafických prvků lze vidět na obrázku 6.4.



(a) Karta s počty servisních požadavků podle priorit

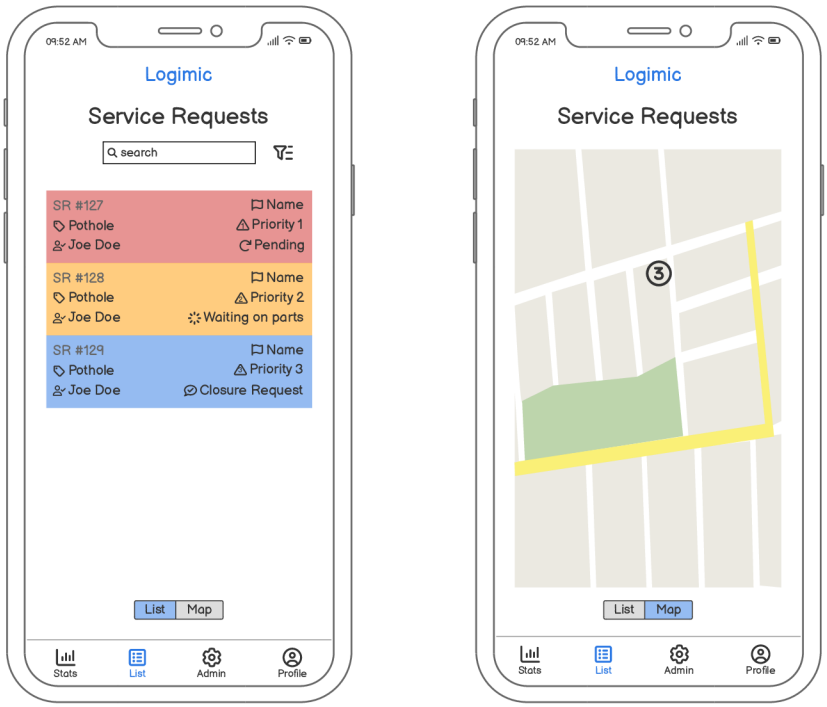
(b) Karta s počty servisních požadavků podle stavů

(c) Graf zobrazující servisní požadavky podle přiřazené osoby

Obrázek 6.4: Grafické prvky pohledu *Stats*

6.4.2 List (seznam)

Pohled *List* je navržen na obrázku 6.5 a nabízí možnost zobrazení dvou odlišných obrazovek. První z nich je na obrázku 6.5a a ukazuje seznam servisních požadavků, ve kterém lze vyhledávat a filtrovat. Jednotlivé servisní požadavky jsou zobrazeny jako řádky s barvou odpovídající prioritě. Kromě barvy je priorita indikována také ikonou a popisem, stejně jako další zobrazené atributy požadavku: identifikační číslo, název, typ, přiřazená osoba a stav zpracování. Na obrázku 6.5b je vykreslena mapa s vyznačenými polohami servisních požadavků. Mezi obrazovkami lze přepnout tlačítkem v dolní části obrazovky.

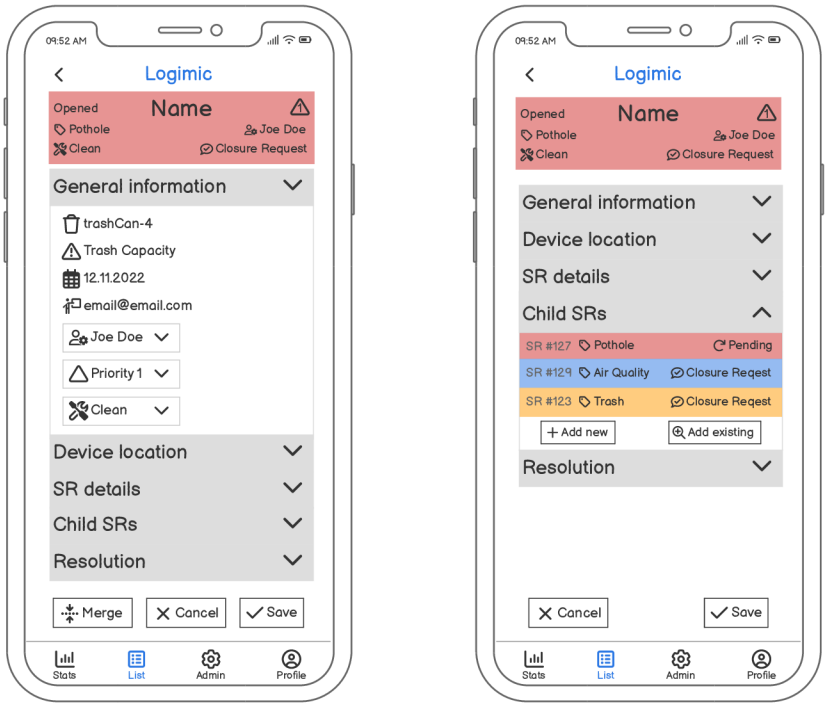


(a) Obrazovka se seznamem servisních požadavků

(b) Obrazovka s mapou s vyznačenými polohami servisních požadavků

Obrázek 6.5: Výchozí obrazovky pohledu *List*

V obou zmíněných obrazovkách pohledu *List* lze po kliknutí na konkrétní servisní požadavek zobrazit jeho detail, tedy obrazovku s detailními informacemi o požadavku. Příklady,



(a) Obrazovka detailu servisního požadavku nejvyšší úrovně

(b) Obrazovka detailu dílčího servisního požadavku

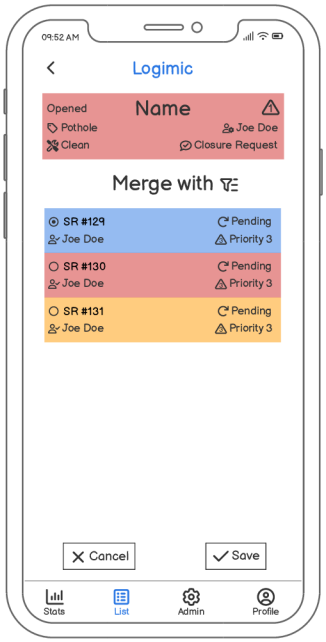
Obrázek 6.6: Obrazovky s detailními informacemi servisního požadavku

jak může taková obrazovka vypadat, jsou ukázány na obrázku 6.6. V horní části obrazovky je barevný řádek se základními informacemi o požadavku a dále jsou na obrazovce vypsány detailní informace. Kvůli nutnosti zobrazit velké množství informací jsou detailní informace rozděleny do rozbalovacích panelů, které tvoří přehlednou strukturu obrazovky detailu. Informace je tak možné postupně procházet, a pokud je k tomu uživatel oprávněn, může některé zobrazené informace také upravovat či doplňovat. Mezi panely se mohou na obrazovce detailu vyskytnout některé z následujících:

- *General information* – panel s obecnými informacemi, jako např. název zařízení, typ upozornění, datum vytvoření nebo kontakt na osobu, která požadavek vytvořila,
- *Merged SRs* – panel se seznamem servisních požadavků, které jsou k aktuálnímu duplicitní,
- *Device location* – panel s informacemi o poloze zařízení, např. město, PSČ nebo GPS souřadnice,
- *Hazard details* – panel s detaily o reportu, např. kontakt na osobu, která report vytvořila, komentář této osoby k nahlášené závadě nebo fotografie dokumentující závadu,
- *SR details* – panel s detaily servisního požadavku, např. datum, kdy začalo zpracování požadavku nebo datum plánovaného konce zpracování,
- *Parent SRs* – panel se seznamem rodičovských servisních požadavků,

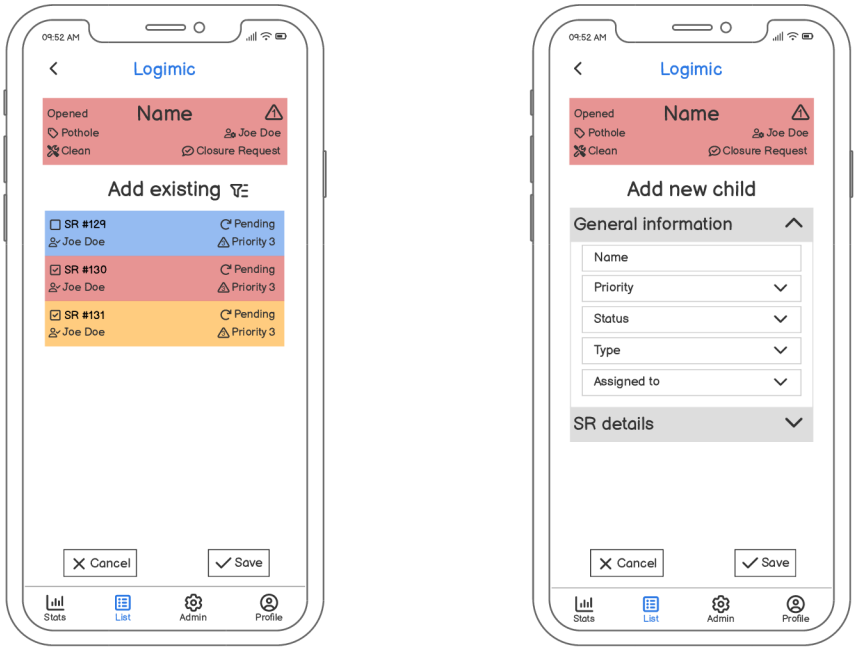
- *Child SRs* – panel se seznamem dílčích servisních požadavků a tlačítky pro přidání dalších dílčích požadavků,
- *Resolution* – panel s možností požádat o uzavření nebo přímo uzavřít požadavek, případně panel s informacemi o uzavření servisního požadavku, jako je např. osoba žádající o uzavření, osoba, která požadavek uzavřela, komentář k uzavření nebo fotografie dokumentující vyřešení požadavku.

Vyjmenované panely mohou být v aplikaci zobrazeny v několika různých kombinacích. Jakýkoli požadavek má zobrazeny panely *General information*, *SR details* a *Resolution*. Dále mají všechny požadavky dostupný buď panel *Device location*, nebo *Hazard details* s detailními informacemi získanými ze zdroje požadavku, tzn. buď od chytrého zařízení, nebo z reportu. Požadavek, ke kterému jsou přiřazeny dílčí požadavky, nebo pokud má přihlášený uživatel právo přidávat dílčí požadavky, má navíc zobrazen panel *Child SRs*. Každý dílčí požadavek má zobrazen také panel *Parent SRs* a každý požadavek, ke kterému existuje duplicitní požadavek, má zobrazen panel *Merged SRs*.



Obrázek 6.7: Obrazovka umožňující sloučení dvou servisních požadavků nejvyšší úrovně

Z obrazovky detailu servisního požadavku se lze dostat do dalších obrazovek. Servisní požadavek nejvyšší úrovně má možnost být sloučen s jiným požadavkem nejvyšší úrovně kvůli řešení duplicit. Pro tento účel je v dolní části obrazovky detailu na obrázku 6.6a tlačítko *Merge* vedoucí na obrazovku na obrázku 6.7. Ta nabízí seznam servisních požadavků pro sloučení, v němž je možné filtrovat a vybrat požadavek pro sloučení. Z obrazovky detailu servisního požadavku lze také přejít pomocí tlačítek *Add new* a *Add existing* na obrazovky pro přidání dílčích požadavků. Obrazovka na obrázku 6.8a slouží k přidání, resp. odebrání již existujících požadavků jako dílčích. Obrazovka na obrázku 6.8b pak nabízí možnost vytvořit nový dílčí požadavek.



(a) Obrazovka umožňující přidání, resp. odebrání existujících požadavků jako dílčích

(b) Obrazovka umožňující přidání nového požadavku jako dílčího

Obrázek 6.8: Obrazovky umožňující přidání servisního požadavku jako dílčího

6.4.3 Admin (administrátorská sekce) a Profile (profil uživatele)

Pohled *Admin* bude umožňovat provádění administrátorských akcí. Pohled *Profile* bude zobrazovat informace o uživatelském účtu aktuálně přihlášeného uživatele. Moduly pro oba pohledy v současně používané webové aplikaci již existují. Jejich vzhled i použití by měly být v nové mobilní aplikaci zachovány, případně může dojít k drobným úpravám, aby pohledy zapadaly do celkového vzhledu aplikace.

Kapitola 7

Implementace mobilní aplikace

Tato kapitola se věnuje implementaci multiplatformní mobilní aplikace dle návrhu popsaného v kapitole 6. Jak bylo vysvětleno v sekci 6.3, pro tvorbu aplikace byl zvolen hybridní přístup, který umožňuje využít existující backendové řešení webové platformy. Před začátkem implementace samotné aplikace tak muselo dojít úpravě platformy podle návrhu, konkrétně ke změnám v datovém modelu a k zavedení nových přístupových bodů (*endpoints*) rozhraní REST. Úpravy byly diskutovány se zaměstnanci firmy Logimic a po několika schůzkách byly úspěšně provedeny. Moje práce zde končila u návrhu změn a následné komunikace se zaměstnanci, kteří konkrétní změny prováděli.

První fází implementace byla volba technologií, kterou shrnuje sekce 7.1. Zásadním výsledkem této fáze bylo zvolení nástrojů Ionic a Capacitor pro získání multiplatformní aplikace. Díky tomu mohla implementace probíhat tak, že nejprve byla vyvíjena webová aplikace pro správu servisních požadavků, a až poté došlo k její transformaci do multiplatformní mobilní aplikace. Fáze implementace webové aplikace je popsána v sekci 7.2. Fázi transformace do multiplatformní aplikace je pak věnována sekce 7.3. Aby byla vytvořená aplikace plně funkční, bylo následně nutné použít některé komponenty nástroje Ionic, což ovlivnilo základní strukturu projektu, jak vysvětluje sekce 7.4. Následně byly implementovány komponenty a funkce používané výhradně v mobilní verzi multiplatformní aplikace. Konkrétně se jedná o komponentu dolního navigačního menu, využívání nativního fotoaparátu a zavedení nativních notifikací, jejichž implementací se zabývá sekce 7.5.

7.1 Použité technologie

Výběr technologií vycházel z informací uvedených v sekci 4.2 nebo z interní dokumentace firmy Logimic. Firma Logimic aktuálně vyvíjí webové aplikace v programovacím jazyce TypeScript za použití frameworku Angular. Volba těchto technologií pro implementaci mobilní aplikace byla proto z pohledu budoucího vývoje a možnosti integrace do platformy pro chytrá města zásadní. V návaznosti na to byl jako hybridní nástroj umožňující vytvoření multiplatformní mobilní aplikace vybrán framework Ionic[1] v kombinaci s nástrojem Capacitor[2]. Stručný přehled obou nástrojů poskytuje sekce 4.2. Ta mimo jiné udává, že Ionic nabízí podporu pro Angular a zároveň že jde o nejpoužívanějším z hybridních frameworků, což jsou dva hlavní důvody pro jeho zvolení. Ionic samostatně nezajišťuje vznik multiplat-

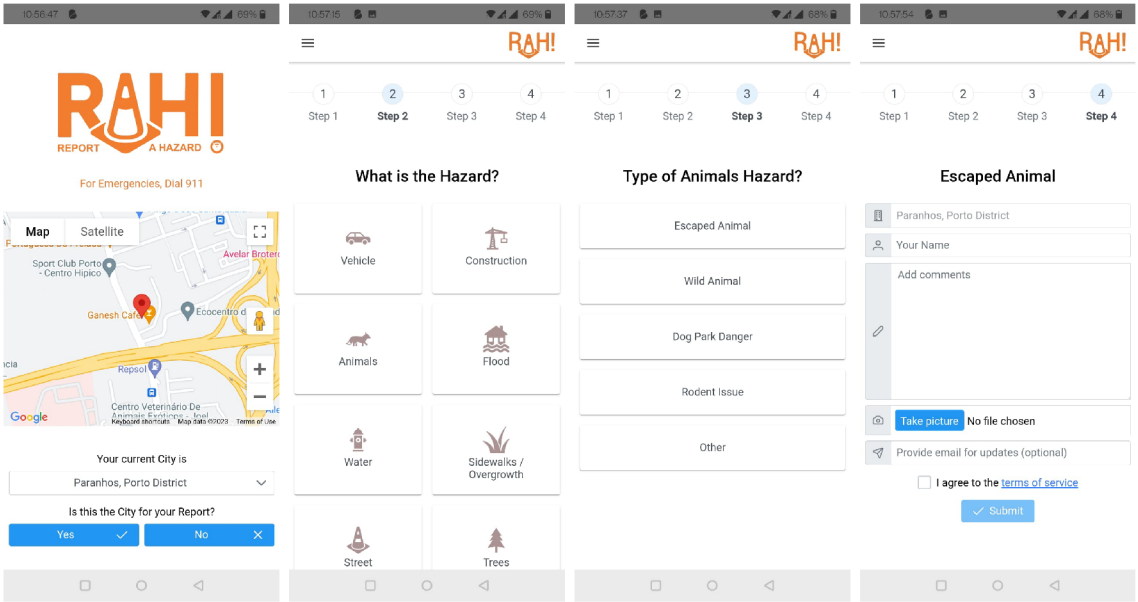
[1] https://ionicframework.com
[2] https://capacitorjs.com

formní aplikace, k tomu je potřeba dalším nástroj. Takovým nástrojem doporučovaným autory frameworku Ionic je právě Capacitor.

Pro tvorbu uživatelského rozhraní (UI, *User Interface*) byla vybrána knihovna PrimeNG[3], která nabízí velké množství předem připravených komponent UI a je rovněž aktuálně používanou technologií ve firmě Logimic. Podobné možnosti pro tvobu UI nabízí také Ionic, jeho použití při implementaci UI je však omezeno na minimum a nahrazeno právě komponentami knihovny PrimeNG, aby výsledná aplikace byla snadno udržovatelná z pohledu budoucího vývoje a provozu firmou Logimic.

7.1.1 Ionic – ověření konceptu

Před začátkem implementace došlo k ověření, že je možné použít Ionic a Capacitor k transformaci existující webové aplikace do multiplatformní mobilní aplikace. K tomuto účelu poskytla firma Logimic jednoduchou webovou aplikaci Město v kapse (distribuovanou pod názvem RAH), jejíž účel stručně vysvětluje sekce 5.1. Aktuálně je tato webová aplikace dostupná také jako mobilní, konkrétně jako progresivní webová aplikace (více o tomto přístupu v sekci 4.1). Jde tedy o aplikaci, která je responzivní, neboli přizpůsobená pro zobrazení na displejích různých rozlišení, např. displejích mobilních zařízení. Podle oficiální dokumentace nástroje Ionic [23] lze do takové aplikace jednoduše integrovat nástroje Ionic a Capacitor, následně vygenerovat nativní mobilní aplikace pro Android a iOS a zároveň aplikaci dále používat také jako webovou.



Obrázek 7.1: Základní obrazovky nativní mobilní aplikace Město v kapse (RAH) pro operační systém Android

Aplikaci Město v kapse se po provedení kroků shrnutých v sekci 7.3 a zavedení základních elementů nástroje Ionic popsaných v sekci 7.4 podařilo transformovat do obou nativních aplikací, přičemž po provedení drobných úprav ve vzhledu jsou obě výsledné aplikace plně funkční, stejně jako webová aplikace. Verze aplikace pro Android je nasazena v obchodě Google Play pro interní testování. Výsledná podoba základních obrazovek je zachycena

[3]https://primeng.org

na obrázku 7.1. Z ověření vyplývá, že Ionic společně s nástrojem Capacitor je vhodný k transformaci webové aplikace implementované za pomoci frameworku Angular do nativní mobilní aplikace pro Android i iOS.

7.2 Aplikace pro správu servisních požadavků

Cílem této fáze implementace bylo vytvořit webovou aplikaci pro správu servisních požadavků tak, aby bylo možné ji jako webový modul integrovat do platformy pro chytrá města. Před začátkem samotné implementace modulu bylo nutné nejprve porozumět architektuře platformy, aby mohly být dodrženy aktuálně používané principy a výsledný zdrojový kód byl pro firmu do budoucna dobře udržovatelný a rozšiřitelný. Dále bylo potřeba prozkoumat existující modul pro správu servisních požadavků a rozhodnout, zda lze jeho zdrojový kód využít pro implementaci nového modulu, nebo zda je vhodné začít s implementací od začátku. Původní modul nebyl napojen na přístupové body rozhraní REST, které by zprostředkovávaly data, byl tedy v současné chvíli nefunkční. Zároveň byly některé komponenty používané v modulu nevhodné pro zobrazení na mobilních zařízeních, jako např. dialogová okna s formuláři. Vzhled modulu by samozřejmě musel projít řadou úprav, aby byly zohledněny nově navržené změny v práci se servisními požadavky. Na základě velkého rozsahu zmíněných úprav v případě využití zdrojového kódu existujícího modulu, jsem se rozhodla začít s implementací nového modulu od začátku. Zdrojový kód původního modulu byl však při tvorbě nového inspirací.

Výsledkem této fáze implementace je webový modul integrovaný do platformy pro chytrá města. Vzhled jednotlivých pohledů modulu vychází z návrhu uživatelského rozhraní v sekci 6.4, je však přizpůsoben nejen pro zobrazení na mobilních zařízeních, ale také pro zobrazení na displejích většího rozlišení, aby mohl být použit právě jako součást webové platformy. Výsledný vzhled aplikace pro správu servisních požadavků zobrazené ve webovém prohlížeči mobilního zařízení lze najít v příloze A.

7.2.1 Uživatelské role

Důležitou součástí implementace bylo zaručit uživatelům aplikace přístup pouze k pohledům a datům, které odpovídají jejich roli. Modul servisních požadavků je určen pro uživatele s rolí manažera nebo technického pracovníka, jak bylo popsáno v sekci 6.2. V celé platformě jsou přístupy k odpovídajícím datům a akcím garantovány pomocí přiřazování práv rolím. Jednotlivá práva jsou reprezentována řetězci, které je možné dát do relace s rolemi, a tím zaručit uživatelům dané role přístup k datům nebo akcím daného práva. Pro potřeby nového modulu byly definovány následující řetězce:

- Práva pro přístup k pohledům modulu:
  - *gui/sr/l2* – přístup k pohledu *Stats*,
  - *gui/sr/l3* – přístup k pohledu *List*,
  - *gui/sr/l4* – přístup k pohledu s detailními informacemi servisního požadavku a k pohledům z něj dostupných.
- Práva pro zobrazení seznamu servisních požadavků
  - *entity/sr/list/all* – zobrazení všech,
  - *entity/sr/list/current* – zobrazení požadavků, ke kterým je uživatel přiřazen.

- Právo pro přidávání nových servisních požadavků: *entity/sr/add*.
- Právo pro zobrazení dat servisního požadavku: *entity/sr/view*.
- Právo pro odebírání servisních požadavků: *entity/sr/remove*.
- Právo pro upravování SR: *entity/sr/edit*.

Role manažera má přiřazena všechna zmíněná práva, role technického pracovníka disponuje pouze právy *gui/sr/l3, gui/sr/l4, entity/sr/list/current* a *view*.

7.3 Transformace webové aplikace do mobilní

Podle oficiální dokumentace nástroje Ionic [23] lze z existující webové aplikace vytvořit mobilní ihned po integraci a inicializaci nástrojů Ionic a Capacitor. Ve skutečnosti je však potřeba vykonat i několik úprav v existující aplikaci a provést sadu kroků ve správném pořadí [11]. Kroky, jejichž provedení vedlo k získání multiplatformní mobilní aplikace pro správu servisních požadavků, shrnuje následující postup. Vstupem transformace byl vytvořený modul pro správu servisních požadavků integrovaný do platformy pro chytrá města, souhrnně označený jako projekt *Smart City*. Přesný popis transformace je důležitým výstupem této diplomové práce pro firmu Logimic.

Transformace webové aplikace do multiplatformní, resp. do nativních mobilních aplikací pro Android a iOS, vyžaduje v první řadě instalaci potřebných nástrojů a balíčků. Prerekvizitami jsou, vzhledem k práci s již existující webovou aplikací, nainstalovaný správce balíčků pro JavaScript *Npm*[4] (testováno na verzi 8.19) a běhové prostředí *Node.js*[5] (verze 16.19). Mimo to je vyžadován správce balíčků pro Swift a Objective-C *CocoaPods*[6]. První fáze celého postupu obsahuje následující kroky:

1. Instalace základních balíčků: *@ionic/cli, @ionic/angular, @angular/cli, @capacitor/-core, @capacitor/cli*
2. Integrace frameworku Ionic do existujícího projektu v terminálu:

   ```
   $ ng add @ionic/angular
   ```

3. Inicializace nástroje Ionic v terminálu vyžadující zadání názvu projektu, který musí odpovídat názvu projektu v již existujícím souboru *angular.json*:

   ```
   $ ionic init
   ```

4. Úprava konfigurace projektu přepsáním obsahu souboru *ionic.config.json* do podoby:

   ```
   {
       "defaultProject": "SmartCity",
       "projects": {
           "SmartCity": {
               "name": "SmartCity",
               "integrations": {
   ```

[4] https://www.npmjs.com
[5] https://nodejs.org/en
[6] https://cocoapods.org

```
                "capacitor": {}
            },
            "type": "angular"
        }
    }
}
```

kde „SmartCity“ označuje název projektu zadaný v předchozím kroku.

Po dokončení této fáze lze aplikaci spustit jako Ionic projekt příkazem `ionic serve`, a tedy používat ve zdrojovém kódu elementy, které Ionic nabízí.

Cílem druhé fáze transformace je vytvoření nativních mobilních aplikací, což zahrnuje kroky:

1. Instalace balíčků pro mobilní platformy: *@capacitor/android, @capacitor/ios*
2. Úprava výstupního adresáře v souboru *angular.json*:

   ```
   "outputPath": "www",
   ```

3. Inicializace nástroje Capacitor v terminálu:

   ```
   $ npx capacitor init
   ```

4. Vytvoření nativních aplikací v terminálu:

   ```
   $ ionic capacitor add android
   $ ionic capacitor add ios
   ```

5. Zkopírování zdrojového kódu webové aplikace do nativních aplikací:

   ```
   $ ionic capacitor sync
   ```

Třetí fází je spuštění vzniklých nativních aplikací. To probíhá ve dvou krocích:

1. Otevření nativní aplikace pro Android ve vývojovém prostředí Android Studio[7]:

   ```
   $ ionic capacitor open android
   ```

   resp. otevření nativní aplikace pro iOS ve vývojovém prostředí Xcode[8]:

   ```
   $ ionic capacitor open ios
   ```

2. Spuštění aplikace v daném vývojovém prostředí na virtuálním nebo fyzickém mobilním zařízení.

[7] https://developer.android.com/studio
[8] https://developer.apple.com/xcode/

7.3.1 Vývoj a údržba vzniklých aplikací

Po úspěšné transformaci nastává fáze vývoje a údržby nativních aplikací. Úpravy ve zdrojovém kódu mohou probíhat stejným způsobem jako před transformací, tzn. úpravy lze provádět ve zdrojovém kódu původní webové aplikace [23]. Promítnutí změn pak zajišťuje příkaz `ionic capacitor sync`, který je možné doplnit o název platformy (`android` nebo `ios`) v případě, že mají být změny promítnuty pouze do aplikace specifikované platformy. Následně může být aplikace spuštěna v odpovídajícím vývojovém prostředí tak, jak bylo popsáno dříve v této sekci.

Existuje také možnost spuštění nativní aplikace tak, že jsou změny prováděné za jejího běhu ihned viditelné. K tomuto účelu slouží příkaz

```
$ ionic capacitor run [android|ios] -l --external
```

Ten spustí aplikaci na zvoleném mobilním zařízení vybrané platformy. Následně mohou být prováděny úpravy ve zdrojovém kódu původní webové aplikace, které jsou po uložení automaticky převedeny do spuštěné nativní aplikace.

K ladění může sloužit jednak výstup z nativní aplikace zobrazovaný ve vývojovém prostředí dané platformy, ale také vývojářské prostředí v prohlížeči (stejně jako pro ladění původní webové aplikace). Vývojářské prostředí prohlížeče pro mobilní aplikaci pro platformu Android lze zobrazit v prohlížeči Google Chrome[9] po přistoupení na adresu `chrome://inspect#devices` a zvolení možnosti „Inspect“ u zařízení, na němž aplikace běží. Pro mobilní aplikaci pro platformu iOS lze prostředí zobrazit po otevření záložky „Develop“ v horním menu prohlížeče Safari[10] a následném zvolení zařízení a konkrétní aplikace, která je na zařízení spuštěna.

7.4 Struktura Ionic projektu

Po provedení transformace webové aplikace do mobilní podle kroků ze sekce 7.3 je aplikace funkční, až na jedno omezení — nelze posouvat zobrazeným obsahem. Řešení tohoto nedostatku spočívá v zavedení dvou základních komponent frameworku Ionic [23]:

- **ion-app**[11] je komponenta, která se v projektu vyskytuje pouze jednou a slouží jako kontejner pro celou aplikaci. Při použití technologie Angular je vhodné tuto komponentu umístit na nejvyšší úroveň kořenové komponenty aplikace (obvykle *app-root*).
- Komponenta **ion-content**[12] slouží jako kontejner pro zobrazovaný obsah aplikace. Nabízí metody pro ovládání oblasti obsahu, jako např. požadované posouvání obsahu na obrazovce, a umožňuje měnit některé vlastnosti této oblasti, jako např. použité barvy. Tato komponenta se v projektu může objevit opakovaně, správný přístup podle oficiální dokumentace technologie Ionic spočívá v použití nejvýše jedné komponenty *ion-content* pro jeden pohled v aplikaci.

Kombinace zmíněných komponent zajistí řešení uvedeného omezení. Mimo to může být velmi užitečné použití také dalších komponent nástroje Ionic, které ovlivňují základní rozvržení aplikace, např.:

[9] https://www.google.com/intl/cs_CZ/chrome/
[10] https://www.apple.com/safari/
[11] https://ionicframework.com/docs/api/app
[12] https://ionicframework.com/docs/api/content

- Komponenta **ion-header**[13] označuje záhlaví stránky a je automaticky zarovnána k hornímu okraji stránky. Obvykle uvnitř obaluje komponentu *ion-toolbar*[14] poskytující akce pro aktuální obrazovku, kdy výsledkem kombinace obou komponent je dodržení požadované velikosti záhlaví včetně zohlednění oblasti bezpečné pro zařízení (*safe area*).
- Komponenta **ion-footer**[15] obsahuje zápatí stránky a je automaticky zarovnána k dolnímu okraji stránky. Stejně jako *ion-header* obvykle obaluje komponentu *ion-toolbar* a zohledňuje zobrazení v oblasti bezpečné pro zařízení.

Obě komponenty vyžadují, aby jejich přímo nadřazeným elementem byla jedna z komponent *ion-app* nebo *ion-content*. Taková komponenta pak slouží jako kontejner, uvnitř něhož je komponenta *ion-header* vždy zarovnána k hornímu okraji a *ion-footer* k dolnímu, a to i při posouvání obsahu na obrazovce. Výsledná základní struktura aplikace pak může vypadat následovně:

```
<ion-app>
    <ion-header>
        <ion-toolbar> Header </ion-toolbar>
    </ion-header>
    <ion-content>
        Content
    </ion-content>
    <ion-footer>
      <ion-toolbar> Footer </ion-toolbar>
    </ion-footer>
</ion-app>
```

Všechny uvedené komponenty přispívají k uživatelské přívětivosti aplikace, přičemž za nezbytné pro získání plně funkční multiplatformní aplikace lze označit užití pouze komponent *ion-app* a *ion-content* na nejvyšší úrovni kořenové komponenty původní aplikace.

Výsledná struktura vytvořené aplikace pro správu servisních požadavků má různou podobu pro webové a mobilní zobrazení, jak je znázorněno v následujícím zjednodušeném zdrojovém kódu komponenty *app-root*:

```
<ion-app>
    <ion-content *ngIf="!isHybrid">
        <lgmc-g-navbar></lgmc-g-navbar>
        <router-outlet></router-outlet>
    </ion-content>

    <ion-header *ngIf="isHybrid">
        <ion-toolbar>
```

[13] https://ionicframework.com/docs/api/header
[14] https://ionicframework.com/docs/api/toolbar
[15] https://ionicframework.com/docs/api/footer

```
9               <lgmc-g-mobile-logo-header></lgmc-g-mobile-logo-header>
10          </ion-toolbar>
11      </ion-header>
12      <ion-content *ngIf="isHybrid">
13          <lgmc-g-mobile-bottom-navbar></lgmc-g-mobile-bottom-navbar>
14      </ion-content>
15  </ion-app>
```

Pokud je aplikace spuštěna ve webovém prohlížeči (proměnná `isHybrid` nabývá hodnoty `false`), využívá její struktura pouze komponenty *ion-app* a *ion-content*. V *ion-content* je zobrazena komponenta navigační lišty (*lgmc-g-navbar*) a dále komponenta vybraná pomocí směrovače (*router-outlet*) podle navigace v aplikaci. Pokud je aplikace spuštěna jako hybridní (proměnná `isHybrid` nabývá hodnoty `true`), zahrnuje struktura kromě *ion-app* a *ion-content* také komponentu *ion-header*. Ta zobrazuje záhlaví aplikace přizpůsobené pro mobilní zobrazení (*lgmc-g-mobile-logo-header*). Komponenta *ion-content* pak zobrazuje dolní navigační menu (*lgmc-g-mobile-bottom-navbar*), které zajišťuje směrování mezi jednotlivými komponentami podle navigace v aplikaci. Více informací o komponentě dolního navigačního menu lze nalézt v podsekci 7.5.1.

7.5 Mobilní komponenty a funkce

Součástí výsledné multiplatformní aplikace je několik komponent a funkcí, které jsou používány pouze v mobilních aplikacích, nikoliv ve webovém modulu. Takové prvky zajišťují lepší uživatelskou přívětivost mobilní aplikace. Konkrétně se jedná o komponentu dolního navigačního menu, využití funkce nativního fotoaparátu a zařazení nativních push notifikací[16], kterým se dále věnuje tato sekce.

7.5.1 Dolní navigační menu

Dolní navigační menu využívá komponenty *ion-tabs*[17] a *ion-tab-bar*[18] z frameworku Ionic [23], jejichž kombinace je využita k navigaci nejvyšší úrovně v aplikaci. Komponenta *ion-tabs* funguje jako směrovač ovládající navigaci (*router outlet*) a je kontejnerovou komponentou. Komponenta *ion-tab-bar* pak musí být vedena jako přímý potomek komponenty *ion-tabs* a její účel spočívá ve zpřístupnění mechanismu pro přepínání mezi položkami menu a poskytování uživatelského rozhraní komponenty. Komponenta *ion-tab-bar* obsahuje několik komponent *ion-tab-button*[19] reprezentujících jednotlivé položky menu. Kromě vzhledu tato komponenta přináší možnost definovat ikonu a název položky v menu, ale také cestu ke komponentě, ke které vede kliknutí na položku v menu.

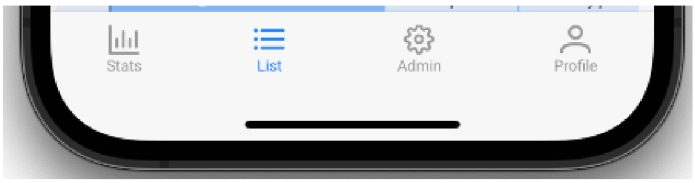
Dolní navigační menu je přímým potomkem komponenty *ion-content* na nejvyšší úrovni aplikace a po zjednodušení vypadá její implementace následovně:

```
1   <ion-tabs>
2       <ion-tab-bar slot="bottom">
```

[16] push notifikace – oznámení zaslané aplikací uživateli, i když aplikace není otevřená
[17] https://ionicframework.com/docs/api/tabs
[18] https://ionicframework.com/docs/api/tab-bar
[19] https://ionicframework.com/docs/api/tab-button

```
        <div *ngFor="let menuItem of menuItems">
            <ion-tab-button [tab]="menuItem.path">
                <i [ngClass]="menuItem.icon"></i>
                {{ menuItem.label }}
            </ion-tab-button>
        </div>
    </ion-tab-bar>
</ion-tabs>
```

Výsledný vzhled komponenty dolního navigačního menu v mobilní aplikaci pro iOS pro uživatele v roli manažera s administrátorskými právy ukazuje obrázek 7.2. Modrou barvou je vždy zvýrazněna aktivní položka menu.



Obrázek 7.2: Mobilní komponenta dolního navigačního menu s aktivní položkou „List“

7.5.2 Nativní fotoaparát

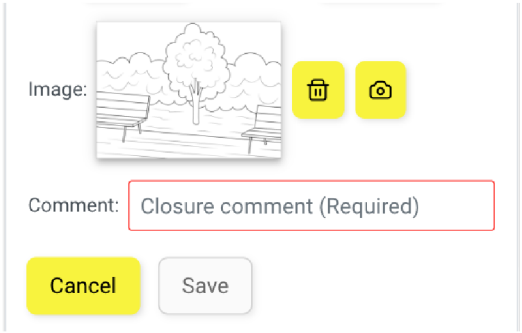
Přístup k nativnímu fotoaparátu mobilního zařízení z multiplatformní aplikace umožňuje balíček *@capacitor/camera*[20] z frameworku Capacitor [8]. Balíček nabízí rozhraní pro pořízení fotografie pomocí fotoaparátu nebo pro vybrání existující fotografie z galerie. V mobilní aplikaci pro správu servisních požadavků jsou tyto funkce využity pro dodání fotografie při vyřešení požadavku. Z balíčku je k tomuto účelu využita asynchronní metoda `getPhoto`, v jejímž vstupním parametru lze zvolit, jakých rozměrů, kvality a typu výsledná fotografie bude. Tato funkce je po stisknutí tlačítka pro pořízení fotografie uživatelem volána následovně:

```
const image: Photo = await Camera.getPhoto({
      resultType: CameraResultType.DataUrl,
      source: CameraSource.Camera,
      quality: 100,
      width: 256,
      height: 256,
    });
```

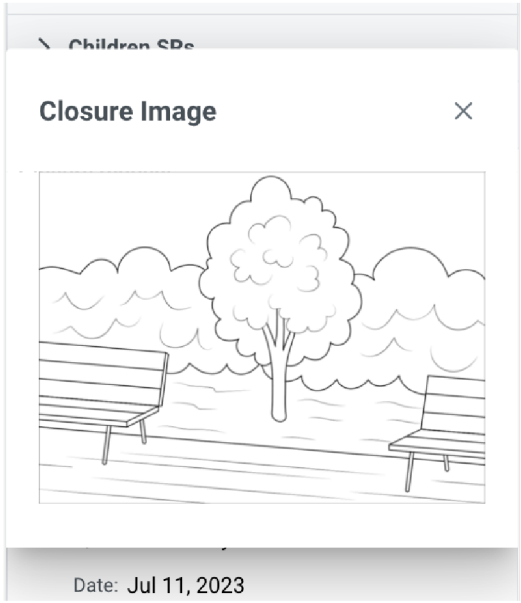
Výsledek je po získání fotografie (z fotoaparátu nebo galerie) uložen v konstantě `image` dodržující rozhraní `Photo` a požadovaný formát je dostupný v položce `image.dataUrl`. Hodnota z `dataUrl` je pak v mobilní aplikaci přímo použita ve zdrojovém atributu `src` HTML elementu `<img>` a je zobrazena uživateli ve zmenšené velikosti, jak lze vidět na

[20] https://capacitorjs.com/docs/apis/camera

obrázku 7.3a. Uživatel si může po kliknutí na zmenšenou fotografii prohlédnout fotografii i ve větší velikosti v dialogovém okně, jak ukazuje obrázek 7.3b.



(a) Zmenšená fotografie ve formuláři pro požádání o uzavření servisního požadavku



(b) Zvětšená fotografie v dialogovém okně

Obrázek 7.3: Náhledy nahraných fotografií v mobilní aplikaci pro správu servisních požadavků

Uložení fotografie do databáze pak probíhá jejím převodem do formátu BLOB (*Binary Large Object*), vytvořením jména fotografie a nahráním na tzv. *Bucket* služby *Amazon Simple Storage Service*[21]. Odtud je fotografie dostupná pod URL (*Uniform Resource Locator*) vytvořeným na základě jména fotografie. Takto získané URL je následně uloženo do databáze, odkud může být kdykoliv snadno načteno a opět použito ve zdrojovém atributu `src` HTML elementu `<img>` a zobrazeno uživateli.

Aby mohla aplikace fotoaparát, resp. galerii mobilního zařízení využívat, je nutné získat povolení od uživatele. O jaké povolení má aplikace uživatele žádat, je však potřeba v aplikaci definovat. Pro platformu iOS jsou tato povolení nastavována v konfiguračním souboru *Info.plist*, kde jsou jednotlivé vlastnosti aplikace strukturovány jako seznam dvojic klíč – hodnota. Pro povolení přístupu k fotoaparátu je třeba do souboru přidat celkem tři dvojice:

```
1 <key>NSCameraUsageDescription</key>
2 <string>To Take Photos</string>
3 <key>NSPhotoLibraryAddUsageDescription</key>
4 <string>To Upload Photos from Library</string>
5 <key>NSPhotoLibraryUsageDescription</key>
6 <string>To Upload Photos from Library</string>
```

[21] https://docs.aws.amazon.com/s3/index.html

Klíčem je zde klíčové slovo definované platformou. Hodnotu tvoří popis vysvětlující, k čemu přístup k fotoaparátu, resp. ke galerii v aplikaci slouží. Pro platformu Android jsou povolení konfigurována v souboru *AndroidManifest.xml*. I zde je třeba doplnit soubor o tři položky:

```
1 <uses-permission android:name="android.permission.READ_MEDIA_IMAGES"/>
2 <uses-permission android:name="android.permission.READ_EXTERNAL_STORAGE"/>
3 <uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE" />
```

Aplikace pak před spuštěním funkce fotoaparátu požádá uživatele o udělení definovaných povolení v modální okně.

Na základě možnosti rozlišení platformy, jak bylo popsáno v sekci 7.4, je přístup k nativnímu fotoaparátu používán pouze pro mobilní aplikace. Ve webové aplikaci je možné fotografii nahrát jako soubor pomocí komponenty *FileUpload*[22] z knihovny *PrimeNG*. Takový soubor je následně uložen do databáze stejsným způsobem jako pro mobilní aplikace.

7.5.3 Nativní push notifikace

Využití nativních push notifikací mobilního zařízení umožňuje balíček *@capacitor/push-notifications*[23] z frameworku Capacitor, jehož metody nabízí možnost registrace a monitorování push notifikací v aplikaci. Tento balíček používá technologii Firebase Cloud Messaging (FCM), která zajišťuje spolehlivé zasílání zpráv bez jakýchkoli nákladů pro různé platformy [17]. Aby mohly být notifikace do aplikace zasílány je potřeba, stejně jako u fotoaparátu, žádat uživatele o udělení povolení. Takové povolení pro aplikaci pro platformu iOS lze nastavit pouze ve vývojovém prostředí Xcode, kde k němu má přístup pouze vývojář s placeným účtem [38]. Placený účet se ale během vývoje této diplomové práce nepodařilo ve spolupráci s firmou Logimic zajistit, a proto jsou push notifikace otestovány pouze pro platformu Android. Zdrojový kód ovládající push notifikace je však pro obě platformy stejný a měl by tak být připraven pro zprovoznění i na platformě iOS. Pro platformu Android je podobně jako u fotoaparátu nutné rozšířit soubor *AndroidManifest.xml* o položku s povolením [31]:

```
1 <uses-permission android:name="android.permission.POST_NOTIFICATIONS"/>
```

Aplikace pak při spuštění požádá uživatele o přidělení těchto povolení v modálním okně.

Před zavedením push notifikací do zdrojového kódu aplikace je nejprve potřeba [38]:

1. vytvořit Firebase projekt ve Firebase Console[24] podle pokynů zobrazených po zvolení „Add project“ a
2. vytvořit Android aplikaci ve Firebase projektu pomocí:
   (a) otevření právě vytvořeného Firebase projektu ve Firebase Console,
   (b) zvolení možnosti „Add app“ zobrazené pod názvem Firebase projektu,

[22] https://primeng.org/fileupload
[23] https://capacitorjs.com/docs/apis/push-notifications
[24] https://console.firebase.google.com/u/0/

(c) zvolení ikony pro Android z nabízených možností,

(d) vyplnění názvu balíčku aplikace, který odpovídá položce *applicationId* v souboru *android/app/build.gradle* (v případě vytvářené aplikace pro správu servisních požadavků je název „com.smartcity.app“),

(e) stažení vzniklého souboru *google-services.json* a jeho přidání do adresáře *android/app*.

V případě zavádění push notifikací pro iOS by bylo navíc potřeba, podobně jako v druhém bodě, vytvořit iOS aplikaci ve Firebase projektu. Způsob zobrazení push notifikací lze konfigurovat v souboru *capacitor.config.ts*. Pro vytvářenou aplikaci pak vypadá konfigurace následovně:

```
PushNotifications: {
    presentationOptions: ['badge', 'sound', 'alert']
}
```

V položce „presentationOptions“ je nastaveno `badge` pro aktualizaci počtu notifikací u ikony aplikace (dostupné pouze pro iOS), `sound` pro zvukové upozornění při přijetí push notifikace a `alert` pro zobrazení notifikace v dialogovém okně při jejím přijetí.

Dalším krokem byla již samotná implementace přijímání notifikací ve zdrojovém kódu, která je díky balíčku push notifikací velmi jednoduchá. Ve vytvářené aplikaci je kód ovládající notifikace umístěn ve službě *LgmcGuiSerqNotificationsService*, kde se nachází inicializační metoda `init` volaná při spuštění aplikace. Její účel spočívá v registraci aplikace k dostávání notifikací (metoda `register`), pokud uživatel přidělil potřebná povolení (metoda `requestPermissions`). Mimo to jsou v inicializační metodě monitorovány události:

- `registrationError` nastávající při chybě v registraci, kdy reakcí je upozornění uživatele na tuto chybu,
- `pushNotificationReceived` nastávající při přijetí nové notifikace, když je aplikace otevřena, kdy reakcí je zobrazení obsahu nové notifikace v dialogovém okně,
- `pushNotificationActionPerformed` nastávající při otevření aplikace po kliknutí na notifikaci zobrazenou v notifikačním panelu mobilního zařízení, kdy reakcí je zobrazení pohledu s detaily servisního požadavku, pokud byl uvnitř notifikace zaslán identifikátor.

Posledním krokem je zasílání notifikací. Běžně by k zasílání notifikací docházelo na základě splnění nějaké podmínky v backendové části platformy, např. po přiřazení technického pracovníka k servisnímu požadavku. Mimo to je však možné odeslat notifikace také z vytvořeného Firebase projektu v jeho webové konzoli (*Firebase Console*) v záložce „Messaging“ [38]. Zde je možné vytvořit novou notifikaci po zvolení „New campaign“ typu „Notifications“ a vyplnění požadovaných údajů. Takto vytvořenou notifikaci lze následně tlačítkem „Publish“ odeslat na všechna zařízení, včetně virtuálních, s nainstalovanou aplikací a povoleným přijímáním push notifikací.

Ve vytvářené multiplatformní aplikaci není vždy vhodné zasílat notifikaci všem uživatelům. Často je třeba pomocí notifikace upozornit pouze určitou skupinu uživatelů, případně pouze konkrétního uživatele, což umožňuje balíček *@capacitor-community/fcm*[25].

[25] https://github.com/capacitor-community/fcm

Jeho metody `subscribeTo` a `unsubscribeFrom` lze využít k přihlášení, resp. odhlášení odběru notifikací z požadovaného kanálu [9]. V multiplatformní aplikaci je pak možné používat notifikační kanály např. pro splnění požadavku **P.3**: *Automaticky informovat technického pracovníka o novém servisním požadavku, ke kterému byl přiřazen.* Pro tento účel je každý uživatel aplikace přihlášen k odběru kanálu, jehož název odpovídá unikátnímu identifikátoru daného uživatele. Na základě toho pak může v backendové části aplikace dojít k zaslání notifikace do kanálu uživatele nově přiřazeného k servisnímu požadavku, neboť je v tu chvíli známý identifikátor přiřazeného uživatele, a tedy i název kanálu. Podobně může být např. zaslána notifikace o změnách v řešení servisního požadavku uživateli, který jej vytvořil.

Kapitola 8

Testování použitelnosti

Testování použitelnosti probíhalo ve dvou fázích. První z nich bylo testování webového modulu, který byl od začátku implementace dostupný zaměstnancům firmy Logimic, včetně vedoucího této práce, jako součást platformy pro chytrá města. Významné posuny v implementaci byly komunikovány na pravidelných schůzkách každý týden, kde byly zároveň diskutovány případné nejasnosti a zpětná vazba od zaměstnanců. Tímto způsobem mohly být změny zaváděny postupně už během vývoje, a výsledný vzhled modulu tak nebyl pro firmu překvapením. Druhou fází bylo testování mobilní aplikace, která, jak bylo popsáno v podsekci 4.1.2, ve skutečnosti pouze zobrazuje webový modul uvnitř nativní komponenty WebView. Šlo tedy především o testování prvků specifických pro mobilní aplikaci a testování použitelnosti aplikace na mobilních zařízeních. Aplikace pro Android byla pro účely testování publikována v obchodě Google Play, a mohla tak být snadno rozšířena mezi účastníky testování. Aplikace pro iOS nemohla být šířena podobným způsobem vzhledem k chybějícímu vývojářskému účtu. K jejímu testování proto docházelo hlavně na virtuálních zařízeních dostupných z vývojového prostředí Xcode. Mimo to jsem mobilní aplikaci testovala na fyzickém zařízení iPhone 12 mini s operačním systémem iOS verze 16.5.

V rámci testování použitelnosti došlo také k ověření, zda byly splněny požadavky definované v sekci 5.3. Splnění základních požadavků pro sledování statistik servisních požadavků, zobrazení seznamů servisních požadavků, zobrazení a úpravy detailních informací servisních požadavků a vytváření servisních požadavků vychází již z obrazovek vytvořené webové aplikace v příloze A. Splnění požadavků na úpravy z podsekce 5.3.1 popisuje následující seznam:

- **P.1**: *Vytvářet více servisních požadavků pro jeden report/upozornění.* Umožněno díky zavedení grafové struktury v řazení servisních požadavků. Ke každému reportu/upozornění existuje jeden servisní požadavek nejvyšší úrovně, k němuž mohou být přidávány nové i existující servisní požadavky jako dílčí.

- **P.2**: *Jednoduše zobrazit servisní požadavky čekající na uzavření z role manažera.* Ke splnění vedly dvě úpravy, z nichž první byla přidání nového stavu servisního požadavku „Čeká na uzavření“. Druhá úprava umožnila filtrování seznamu požadavků podle několika parametrů včetně stavu servisního požadavku.

- **P.3**: *Automaticky informovat technického pracovníka o novém servisním požadavku, ke kterému byl přiřazen.* Ačkoliv tento požadavek vytvořená aplikace nesplňuje, frontendová část aplikace je pro splnění připravena, a to díky využití nativních push

notifikací. Každému uživateli může být podle potřeby z backendové části aplikace zaslána notifikace, jak bylo popsáno v podsekci 7.5.3.

- **P.4**: *Automaticky vytvářet nové servisní požadavky z reportů.* Tento požadavek není z pohledu frontendové části aplikace implementované v rámci této diplomové práce řešitelný. Jeho splnění je třeba řešit v backendové části.

- **P.5**: *Jednoduše spravovat přiřazené servisní požadavky z role technického pracovníka.* Jednodušší správu nabízí aplikace technickému pracovníkovi díky zaměření pouze na servisní požadavky, ale také díky přístupu k fotoaparátu mobilního zařízení, přizpůsobení vzhledu aplikace mobilnímu rozlišení, offline dostupnosti nebo díky přijímání push notifikací.

- **P.6**: *Efektivně řešit duplicity na úrovni servisních požadavků.* Jakýkoliv požadavek může být sloučen s požadavkem, ke kterému je duplicitní. Oba požadavky (originální i duplicitní) potom mají v pohledu s detailními informacemi zobrazen sloučený požadavek. Pokud následně dojde ke změně stavu originálního servisního požadavku, je změněn také stav duplicitního požadavku.

- **P.7**: *Uzavírat servisní požadavky ihned po jejich vzniku.* Manažer může uzavřít jakýkoliv požadavek, a to i takový, který právě vznikl a je ve stavu „Otevřen“. K servisnímu požadavku tak pro jeho uzavření nemusí být přiřazena žádná zodpovědná osoba.

8.1 Uživatelské testování

V rámci uživatelského testování bylo osloveno 6 osob, které dostaly sadu úkolů, o jejichž provedení se mají ve vytvořené mobilní aplikaci pokusit. Zapojené osoby byly lidé ve věku 25 až 55 let s běžnými technickými schopnostmi, kteří žádným způsobem nesouvisí s firmou Logimic. U všech zapojených osob jsem byla pozorovatelem při jejich testování aplikace. Předem jsem každou osobu seznámila se základními pojmy a účelem aplikace, přičemž do samotného provádění úkolů jsem dále aktivně nezasahovala. Pozorované účastníky jsem požádala, aby při testování sdělovaly svoje myšlenky nahlas, a já tak mohla zjistit případné problémy v použitelnosti aplikace.

Účastníci dostaly následující sadu úkolů:

- zobrazit seznam požadavků, které jsou ve stavu „Otevřené“,
- zobrazit detail některého požadavku ze seznamu,
- zobrazenému požadavku přidat nový dílčí požadavek,
- zobrazit detail právě přidaného dílčího požadavku,
- přiřadit zodpovědnou osobu právě přidanému dílčímu požadavku,
- požádat o uzavření právě přidaného dílčího požadavku,
- přijmout žádost o uzavření,
- zobrazit rodičovský požadavek nejvyšší úrovně právě přidaného dílčího požadavku, tzn. zobrazit vždy rodičovský požadavek každého zobrazeného požadavku, dokud takový existuje,

- sloučit rodičovský požadavek nejvyšší úrovně s jakýmkoli jiným servisním požadavkem,
- odstranit toto sloučení v detailu rodičovského požadavku nejvyšší úrovně,
- přidat jakýkoli existující požadavek jako dílčí k rodičovskému požadavku nejvyšší úrovně.

Všem účastníkům testování se zmíněné úkoly podařilo provést. Podstatné však bylo zjistit, jak se účastníci v aplikaci orientují, jaké prvky uživatelského rozhraní jsou pro ně snadno, resp. obtížně použitelné, zda je rozložení prvků uživatelského rozhraní v aplikaci vhodné apod. Poznatky z testování vedly např. ke změně barvy tlačítek ze žluté na bílou s modrými okraji, úpravě názvu tlačítka pro požádání o uzavření servisního požadavku z „Request“ na „Request for closure“ nebo k výraznějšímu oddělení čtyř částí panelu *Resolution* pro řešení požadavku v detailu požadavku. Pozitivním zjištěním z testování bylo, že se uživatelé s aplikací rychle naučili pracovat. Pokud jim např. v seznamu požadavků na první pohled uniklo tlačítko pro filtrování, v dalších seznamech už o tomto tlačítku s jistotou věděli.

Kapitola 9

Závěr

Cílem této práce bylo ve spolupráci s firmou Logimic vytvořit mobilní aplikaci umožňující správu servisních požadavků pro uživatele platforem Android i iOS. K naplnění cíle vedlo několik kroků, z nichž prvním bylo získání potřebného teoretického přehledu v oblastech důležitých pro návrh a implementaci aplikace. Pro pochopení, jak by mohla být aplikace používána, bylo nutné alespoň základní prozkoumání oblasti chytrých měst a oblasti systémů pro řízení požadavků. V oblasti chytrých měst jsem se věnovala možnostem uplatnění tzv. „chytrých konceptů“ a technologiím k tomu používaným. V oblasti systémů řízení požadavků jsem studovala, co představuje požadavek, jaké funkce systémy plní a jaké jsou typické role uživatelů. Následně jsem se zabývala multiplatformními aplikacemi, přístupy k jejich vývoji a používanými nástroji, konkrétně nástroji Flutter, React Native a Ionic. Dále jsem na základě analýzy současného stavu podpory pro řešení servisních požadavků na platformě firmy Logimic definovala cílové uživatele a požadavky na mobilní aplikaci.

Poznatky získané v teoretické a analytické části této práce vedly k volbě hybridního přístupu pro tvorbu multiplatformní aplikace, což umožnilo v aplikaci využít existující backendové řešení platformy a zejména integrovat aplikaci jako modul do současně používané webové platformy. V návaznosti na to došlo k návrhu změn v aktuálně používaném datovém modelu a přístupových bodech rozhraní REST. Současně byly navrženy změny ve způsobu práce se servisními požadavky, uživatelské role a uživatelské rozhraní aplikace. Pro implementaci byl zvolen nástroj Ionic, jehož vhodnost jsem nejprve ověřila na jednoduché aplikaci. Díky volbě hybridního přístupu mohla při implementaci nejprve vzniknout responzivní webová aplikace, která byla následně transformována do multiplatformní aplikace. Po zavedení základních prvků nástroje Ionic do struktury zdrojového kódu aplikace a přidání mobilních komponent a funkcí vznikla mobilní aplikace plně funkční na platformách Android i iOS. Zároveň se zdrojový kód mobilní aplikace podařilo integrovat jako modul do webové platformy. Poslední fází bylo otestování použitelnosti aplikace a ověření splnění definovaných požadavků.

Cíl práce se podařilo naplnit, přičemž největším přínosem pro firmu je fakt, že lze existující webové aplikace snadno převádět do nativních mobilních aplikací, a čerpat tak jejich výhody, jako je zveřejnění v typických obchodech s mobilními aplikacemi nebo přístup k aplikaci bez internetového připojení. Tato práce zároveň poskytuje návod, jak transformaci a další potřebné kroky k získání funkční mobilní aplikace provést, protože takto ucelený postup oficiální dokumentace nástroje Ionic nenabízí. V budoucnu by mohla být aplikace rozšířena o plánování pracovního dne technického pracovníka, což by umožnilo jednoduchý přehled činností, které má daný pracovník během dne provádět. Navíc by takto mohla být naplánována také trasa, po níž se má pracovník mezi požadavky přesouvat.

Literatura

[1] Angulo, E. a Ferre, X. A Case Study on Cross-Platform Development Frameworks for Mobile Applications and UX. In: *Proceedings of the XV International Conference on Human Computer Interaction*. New York, NY, USA: Association for Computing Machinery, 2014. Interacción '14. DOI: 10.1145/2662253.2662280. ISBN 9781450328807. Dostupné z: https://doi.org/10.1145/2662253.2662280.

[2] Berst, J., Enbysk, L. a Williams, C. *Smart Cities Readiness Guide* [online]. 2015 [cit. 2023-01-29]. Dostupné z: https://www.smartcitiescouncil.com/sites/default/files/2023-01/Smart%20Cities%20Council%20Readiness-Guide_V2%202015.pdf.

[3] Bertram, D. *The social nature of issue tracking in software engineering*. Calgary, Alberta, Canada, 2009 [cit. 2022-12-04]. Diplomová práce. University of Calgary.

[4] Biørn Hansen, A., Grønli, T.-M. a Ghinea, G. A Survey and Taxonomy of Core Concepts and Research Challenges in Cross-Platform Mobile Development. *ACM Comput. Surv.* New York, NY, USA: Association for Computing Machinery. nov 2018, sv. 51, č. 5. DOI: 10.1145/3241739. ISSN 0360-0300. Dostupné z: https://doi.org/10.1145/3241739.

[5] Biørn Hansen, A. a Ghinea, G. Bridging the Gap: Investigating Device-Feature Exposure in Cross-Platform Development. In:. Leden 2018. DOI: 10.24251/HICSS.2018.716.

[6] Biørn Hansen, A., Majchrzak, T. A. a Grønli, T.-M. Progressive Web Apps for the Unified Development of Mobile Applications. In:. červen 2018, s. 64–86. DOI: 10.1007/978-3-319-93527-0_4. ISBN 978-3-319-93526-3.

[7] *Bugzilla* [online]. [cit. 2022-12-15]. Dostupné z: https://www.bugzilla.org.

[8] *Capacitor: Cross-platform Native Runtime for Web Apps* [online]. [cit. 2023-02-07]. Dostupné z: https://capacitorjs.com/docs.

[9] *Capacitor FCM* [online]. [cit. 2023-01-07]. Dostupné z: https://github.com/capacitor-community/fcm#readme.

[10] Caragliu, A., Del Bo, C. a Nijkamp, P. Smart Cities in Europe. *VU University Amsterdam, Faculty of Economics, Business Administration and Econometrics, Serie Research Memoranda*. Leden 2009, sv. 18. DOI: 10.1080/10630732.2011.601117.

[11] *Converting Angular Web Application into Ionic App* [online]. [cit. 2023-10-05]. Dostupné z: https://levelup.gitconnected.com/converting-angular-web-application-into-ionic-app-5af678325626.

[12] *Dart documentation* [online]. [cit. 2022-11-19]. Dostupné z: https://dart.dev/guides.

[13] El Kassas, W. S., Abdullah, B. A., Yousef, A. H. a Wahba, A. M. Taxonomy of Cross-Platform Mobile Applications Development Approaches. *Ain Shams Engineering Journal*. 2017, sv. 8, č. 2, s. 163–190. DOI: https://doi.org/10.1016/j.asej.2015.08.004. ISSN 2090-4479. Dostupné z: https://www.sciencedirect.com/science/article/pii/S2090447915001276.

[14] Escoffier, C., Lalanda, P. a Gunalp, O. A component model to manage the heterogeneity and dynamism in mobile applications. In: *2015 18th International ACM SIGSOFT Symposium on Component-Based Software Engineering (CBSE)*. 2015, s. 85–90. DOI: 10.1145/2737166.2737178.

[15] Ettifouri, E. H., Rhouati, A., Berrich, J. a Bouchentouf, T. Toward a merged approach for cross-platform applications (web, mobile and desktop). In: *Proceedings of the 2017 International Conference on Smart Digital Environment*. 2017, s. 207–213. DOI: 10.1145/3128128.3128160.

[16] Falessi, D., Hernandez, F. a Khosmood, F. Issue Tracking Systems: What Developers Want and Use. In: *Proceedings of the 13th International Conference on Software Technologies (ICSOFT 2018)*. 2018, s. 543–548. DOI: 10.5220/0006818405430548. ISBN 978-989-758-320-9.

[17] *Firebase Cloud Messaging* [online]. [cit. 2023-02-07]. Dostupné z: https://firebase.google.com/docs/cloud-messaging.

[18] *Flutter documentation* [online]. [cit. 2022-11-24]. Dostupné z: https://docs.flutter.dev.

[19] Gassmann, O., Böhm, J. a Palmié, M. *Smart Cities: Introducing Digital Innovation to Cities*. červen 2019. ISBN 978-1-78769-614-3.

[20] Greengard, S. *The Internet of Things*. The MIT Press, 2015. ISBN 0262527731.

[21] Grønli, T.-M., Hansen, J., Ghinea, G. a Younas, M. Mobile Application Platform Heterogeneity: Android vs Windows Phone vs iOS vs Firefox OS. In: *2014 IEEE 28th International Conference on Advanced Information Networking and Applications*. 2014, s. 635–641. DOI: 10.1109/AINA.2014.78.

[22] *Introduction* [online]. [cit. 2022-11-24]. Dostupné z: https://reactnative.dev/docs/getting-started.

[23] *Introduction to Ionic* [online]. [cit. 2022-07-14]. Dostupné z: https://ionicframework.com/docs/.

[24] Janák, J. *Issue Tracking Systems [online]*. Brno, CZ, 2009. [cit. 2022-12-04]. Diplomová práce. Masarykova univerzita, Fakulta informatiky, Brno. Dostupné z: https://is.muni.cz/th/pbpa1/.

[25] Kirimtat, A., Krejcar, O., Kertész, A. a Tasgetiren, M. Future Trends and Current State of Smart City Concepts: A Survey. *IEEE Access*. Leden 2020, PP. DOI: 10.1109/ACCESS.2020.2992441.

[26] Knudsen, D. B., Barofsky, A. a Satz, L. R. A Modification Request Control System. In: *Proceedings of the International Conference on Software Engineering*. 1976, s. 187–192. DOI: 10.5555/800253.807673.

[27] Laricchia, F. *Mobile operating systems' market share worldwide from 1st quarter 2009 to 4th quarter 2022* [online]. [cit. 2022-24-10]. Dostupné z: https://www.statista.com/statistics/272698/global-market-share-held-by-mobile-operating-systems-since-2009/.

[28] *Logimic* [online]. [cit. 2023-01-19]. Dostupné z: https://www.logimic.com/cs/.

[29] Lynch, M. *How Capacitor Works* [online]. [cit. 2022-07-10]. Dostupné z: https://tinyletter.com/ionic-max/letters/how-capacitor-works.

[30] Lynch, M. *Smart cities* [online]. [cit. 2022-12-28]. Dostupné z: https://commission.europa.eu/eu-regional-and-urban-development/topics/cities-and-urban-development/city-initiatives/smart-cities_en.

[31] *Notification runtime permission* [online]. [cit. 2023-02-07]. Dostupné z: https://developer.android.com/develop/ui/views/notifications/notification-permission.

[32] *Progressive Web Apps* [online]. [cit. 2022-11-19]. Dostupné z: https://web.dev/progressive-web-apps/.

[33] *Redmine* [online]. [cit. 2022-12-15]. Dostupné z: https://www.redmine.org.

[34] Simonofski, A., Asensio, E. S., De Smedt, J. a Snoeck, M. Citizen Participation in Smart Cities: Evaluation Framework Proposal. In: *2017 IEEE 19th Conference on Business Informatics (CBI)*. 2017, sv. 01, s. 227–236. DOI: 10.1109/CBI.2017.21. ISBN 978-1-5386-3036-5.

[35] Taylor, P. *Market share of mobile operating systems in North America from January 2018 to January 2023* [online]. [cit. 2023-07-02]. Dostupné z: https://www.statista.com/statistics/1045192/share-of-mobile-operating-systems-in-north-america-by-month/.

[36] Umuhoza, E. a Brambilla, M. Model Driven Development Approaches for Mobile Applications: A Survey. In: *Mobile Web and Intelligent Information Systems*. Cham: Springer International Publishing, 2016, s. 93–107. ISBN 978-3-319-44214-3.

[37] *Cities (Urban Audit)* [online]. [cit. 2022-12-31]. Dostupné z: https://ec.europa.eu/eurostat/web/cities/data/database.

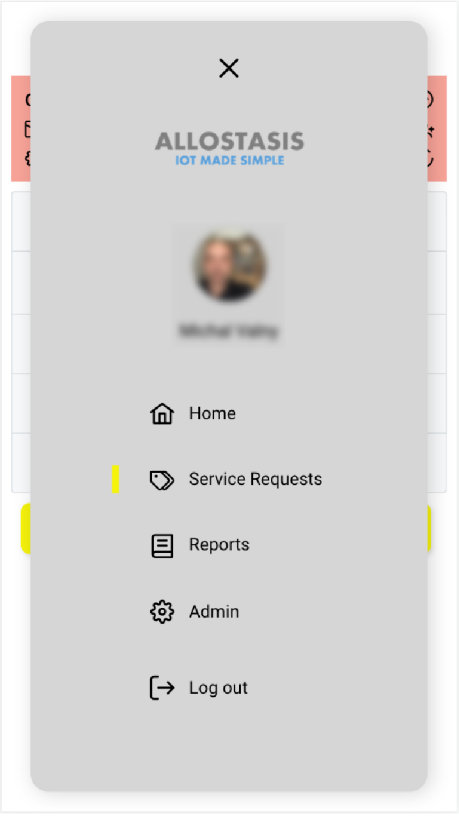
[38] *Using Push Notifications with Firebase in an Ionic + Angular App* [online]. [cit. 2023-02-07]. Dostupné z: https://capacitorjs.com/docs/guides/push-notifications-firebase.

[39] Vailshery, L. S. *Cross-platform mobile frameworks used by software developers worldwide from 2019 to 2021* [online]. [cit. 2022-30-10]. Dostupné z: https://www.statista.com/statistics/869224/worldwide-software-developer-working-hours/.

Příloha A

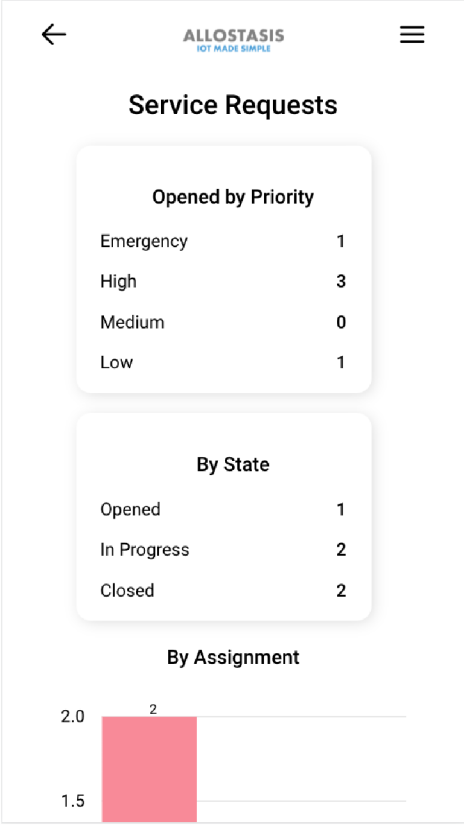
Výsledný vzhled webové aplikace

V této příloze je zobrazen výsledný vzhled webové aplikace pro správu servisních požadavků zobrazené ve webovém prohlížeči na mobilním zařízení. Všechny obrazovky sdílejí komponentu horní lišty, která nebyla implementována v rámci této diplomové práce. Horní lišta umožňuje návrat zpět na předchozí otevřenou obrazovku aplikace, ale také otevření navigačního menu, které lze vidět na obrázku A.1. Z menu je dostupný, jak modul pro správu servisních požadavků, tak i další moduly platformy, lze proto integraci nového modulu do platformy považovat za úspěšnou. Při zvolení možnosti „Service Requests“ v navigačním

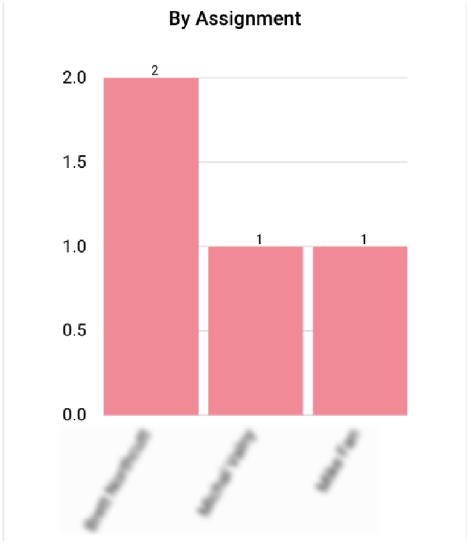


Obrázek A.1: Navigační menu s dostupnými moduly platformy pro chytrá města

menu dojde k přechodu na obrazovku se statistikami, kterou lze vidět na obrázku A.2. Po kliknutí na některou z kategorií zobrazovaných ve statistikách (např. priorita „Emer-



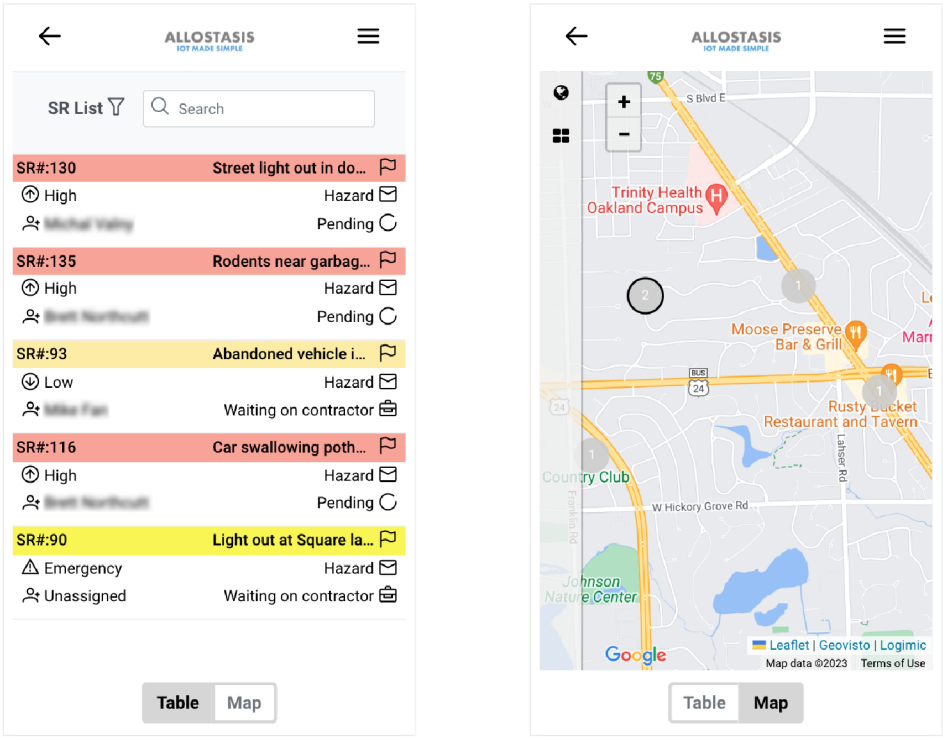
(a) Obrazovka se statistikami servisních požadavků rozdělených podle priority a stavu



(b) Graf zobrazujícím počty servisních požadavků podle přiřazené osoby

Obrázek A.2: Obrazovky se statistikami týkajícími se servisních požadavků

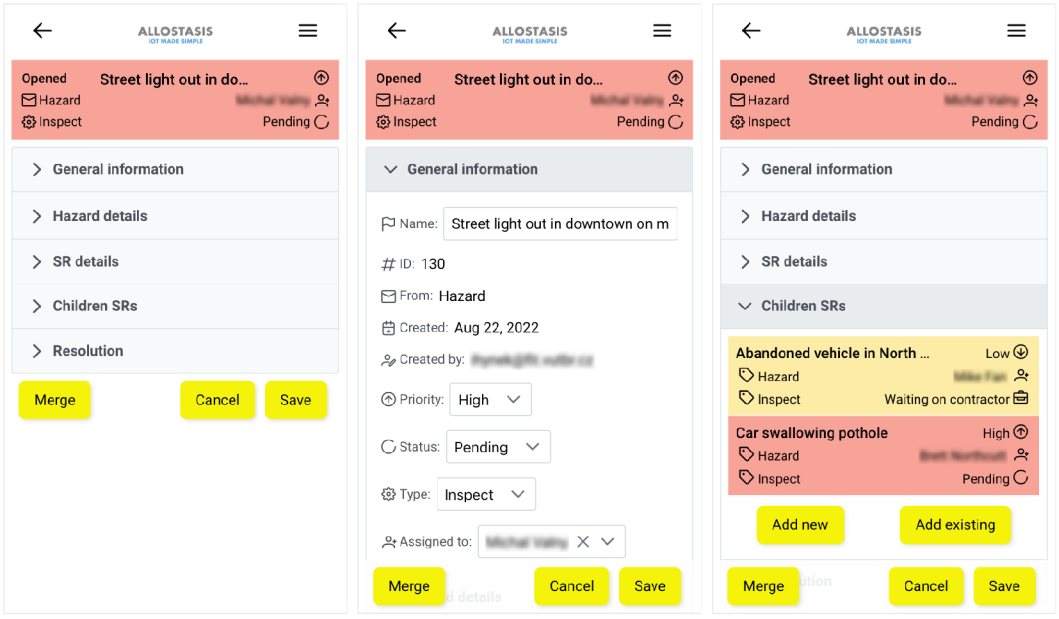
gency“) následuje zobrazení seznamu požadavků, které spadají do této kategorie. Pomocí filtrů v seznamu požadavků lze zobrazené požadavky dále filtrovat, případně zvolené filtry zrušit a zobrazit všechny požadavky pro dané město, jak je ukázáno na obrázku A.3a. Tlačítko v dolní části obrazovky umožňuje zobrazit dané servisní požadavky v mapě, jak lze vidět na obrázku A.3b. Kliknutí na konkrétní požadavek v seznamu nebo mapě vede na obrazovku s detailními informacemi o požadavku. Příklady zobrazených detailních informací prezentuje obrázek A.4. Na obrazovce s detailními informacemi je dále možné kliknout na tlačítko „Merge“, které způsobí přechod na obrazovku na obrázku A.5 umožňující sloučení duplicitních servisních požadavků.



(a) Obrazovka se seznamem servisních požadavků

(b) Obrazovka s mapou s vyznačenými polohami servisních požadavků

Obrázek A.3: Obrazovky s seznamem a mapou servisních požadavků

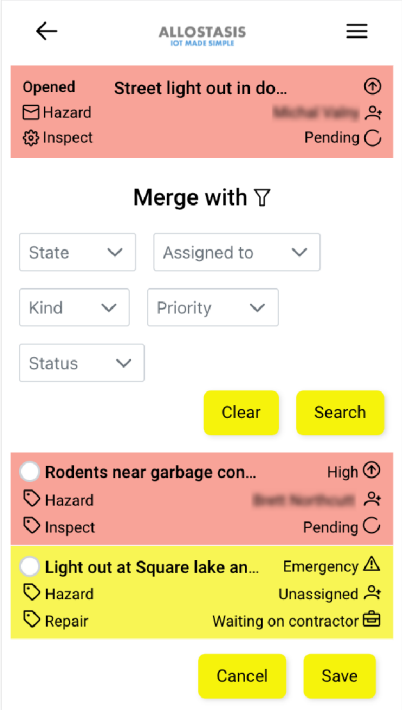


(a) Výchozí obrazovka se zavřenými panely

(b) Otevřený panel s obecnými informacemi

(c) Otevřený panel s dílčími požadavky

Obrázek A.4: Obrazovky s detailními informacemi o servisním požadavku



Obrázek A.5: Obrazovka umožňující sloučení duplicitních servisních požadavků