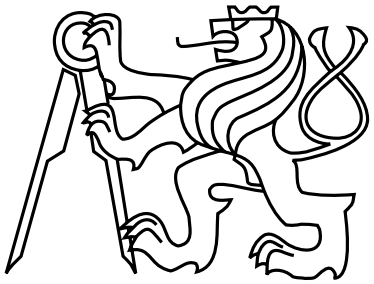
Na tomto místě bude oficiální zadání vaší práce

- Toto zadání je podepsané děkanem a vedoucím katedry,
- musíte si ho vyzvednout na studijním oddělení Katedry počítačů na Karlově náměstí,
- v jedné odevzdané práci bude originál tohoto zadání (originál zůstává po obhajobě na katedře),
- ve druhé bude na stejném místě neověřená kopie tohoto dokumentu (tato se vám vrátí po obhajobě).

České vysoké učení technické v Praze
Fakulta elektrotechnická
Katedra počítačů



Bakalářská práce

Řízení inteligentních budov - rozšíření aplikace pro použití na chytrých telefonech

Josef Lobotka

Vedoucí práce: Ing. Jiří Mlejnek

Studijní program: Softwarové technologie a management, Bakalářský

Obor: Softwarové inženýrství

25. května 2012

Poděkování

Chtěl bych poděkovat především svému vedoucímu, panu Mlejnkovi, za obrovskou pomoc a hlavně trpělivost při vedení této práce. Dále bych rád poděkoval všem klukům ze *SmartBuildings* týmu za bezproblémovou spolupráci a vstřícnost. Děkuji svým rodičům za materiální a psychickou podporu. Nakonec děkuji své přítelkyni, která za mě v minulém půl roce odhodlaně myla nádobí, prala prádlo a statečně mi vařila.

Prohlášení

Prohlašuji, že jsem práci vypracoval samostatně a použil jsem pouze podklady uvedené v přiloženém seznamu.
Nemám závažný důvod proti užití tohoto školního díla ve smyslu §60 Zákona č. 121/2000 Sb., o právu autorském, o právech souvisejících s právem autorským a o změně některých zákonů (autorský zákon).

V Praze dne 25. 5. 2012 .....

Abstract

Intelligent buildings and building automation are one of the fields, which is rapidly growing due to the boom of modern information technologies. Another technic phenomenon are smartphones with independent Internet connection allowing to access network data from almost any location.
Aim of this work is to connect those two trends and provide much more comfortable way how users could manage devices and system in intelligent buildings by simply using their smartphones.
This work describes analysis, design and implementation of useful mobile application, which extends administration interface of existing building management system called *SmartBuildings*.

Abstrakt

Inteligentní budovy a jejich automatizace jsou jednou z oblastí, která díky příchodu moderních technologií zažívá obrovský rozmach. Dalším technickým fenoménem jsou chytré telefony s nezávislým připojením k internetu umožňujícím přistupovat k datům na síti téměř odkudkoliv.
Cílem této práce je spojit oba trendy a poskytnout uživatelům inteligentních budov mnohem pohodlnější způsob, jak pomocí svého chytrého telefonu jednoduše a odkudkoliv spravovat systém a zařízení v budově.
Práce se zabývá analýzou, návrhem a realizací užitečné mobilní aplikace, která rozšiřuje administrační rozhraní existujícího systému pro řízení inteligentních budov *SmartBuildings*.

Obsah

Seznam obrázků

Kapitola 1

Úvod

1.1 Motivace

V posledních dvaceti letech lidstvo prošlo obrovským vývojem v oblasti informačních a telekomunikačních technologií. S příchodem moderních počítačů, mobilních zařízení a zejména internetu tak došlo k razantním změnám v téměř všech oblastech našeho života.

Dnes je již naprosto běžné během několika sekund uspořádat videohovor s člověkem z opačné části planety, z tramvaje pomocí mobilního zařízení přistupovat k nepředstavitelnému množství informací dostupných na internetu, ovládat zařízení hlasem či pohybem ruky nebo například zveřejnit fotky z výletu pár okamžiků po jejich pořízení. Doprava, průmysl, medicína, bezpečnost, sociální interakce a mnoho dalších oborů bychom si už dnes bez moderních IT zařízení ani nedokázali představit. Jednou z významných oblastí, která díky současným technologickým trendům zažívá nebývalý rozmach jsou inteligentní budovy a jejich systémy.

1.1.1 Problém

Systémy inteligentních budov propojují nezávislá zařízení (jako např. kamery, výtahy, požární systém, teplotní čidla) do jednoho celku za účelem automatizace jejich ovládání, zvýšení bezpečnosti, spolehlivosti a komfortu pro obyvatele budovy. Ačkoliv dříve byly takovéto systémy kvůli své finanční i technické náročnosti instalovány převážně do velkých budov a komplexů, dnešní dostupnost levných technologií umožňuje nasadit systém inteligentní budovy například i do většího domu.

V takové chvíli se však uživatelem často stává člověk s průměrnou nebo podprůměrnou úrovní počítačové gramotnosti, kterému může ovládání a administrace systému působit obrovské problémy - obzvlášť, pokud rozhraní systému nabízí rozsáhlé konfigurační možnosti, což bývá kvůli potřebné flexibilitě běžné. V krajním případě může dojít i k situacím, kdy uživatel nechtěně přenastaví nebo dokonce odpojí některý důležitý podsystém a ohrozí tak například bezpečnost svého obydlí.

Dalším problémem bývá závislost na fyzickém umístění administračního rozhraní. Přístupový bod do administrace systému inteligentní budovy totiž musí poskytovat dostatečný

komfort pro pohodlnou konfiguraci nastavení a ovládání. Přístupovým bodem může být například běžný počítač, který se připojí k serveru s běžícím systémem nebo fyzický terminál dodávaný výrobcem. Kdykoliv však uživatel potřebuje provést změny v systému, je nucen dojít k přístupovému bodu osobně, což může být v případě testování nastavení v několikapatrové budově značně náročné.

1.1.2 Řešení

Řešením a cílem této bakalářské práce je vytvořit aplikaci pro chytré telefony, která uživateli umožní vzdáleně se připojit k řídícímu serveru. Velkou výhodou tohoto přístupu je fakt, že uživatel svůj chytrý telefon nosí často u sebe, takže může inteligentní budovu ovládat „odkudkoliv“. Jednoduché uživatelské rozhraní aplikace a podpora dotykového ovládání pak uživateli zajistí snadný a rychlý přístup k nejpoužívanějším funkcím systému řízení inteligentních budov.

1.2 Struktura bakalářské práce

1.2.1 Typ práce a logické členění

Tato práce je kombinací návrhově-implementační bakalářské práce a je rozdělena na několik po sobě jdoucích logických celků:

- **Specifikace požadavků** - Vychází z aktuální funkcionality systému pro řízení inteligentních budov, který bude realizovaná aplikace rozšiřovat, dále provádí analýzu užití aplikace a rešerši konkurenčních aplikací. Na základě všech zjištění nakonec definuje funkční a nefunkční požadavky na systém.
- **Analýza a návrh řešení** - Diskutuje rozdíly mezi vývojem pro desktopové a mobilní aplikace, porovnává vhodné mobilní platformy. Na základě specifikace požadavků navrhuje funkční prototyp, dále se zabývá návrhem SW architektury, strukturou, rozvržením a vzhledem uživatelského rozhraní aplikace.
- **Realizace** - Vybírá nástroje a technologie použité při realizaci aplikace, stanoví pravidla a metodologii postupu implementace. Nakonec detailně popisuje strukturu aplikace a zmiňuje použité techniky.
- **Testování** - Volba a příprava vhodných testů, strategie testování, průběh, výsledky a jejich vyhodnocení.
- **Závěr** - Sebereflexe a shrnutí dosažených výsledků.

1.2.2 Přílohy

Součástí této práce jsou také přílohy, které doplňují základní informace o: *přehled použitých zkratek* (viz A), *výsledky testování prototypu* (viz B), *návod k zprovoznění aplikace* (viz C) v testovacím nebo reálném prostředí a *popis obsahu přiloženého DVD* (viz D).

1.2.3 Použití obchodních značek

Veškeré obchodní značky třetích osob, které budou v této práci použity slouží pouze pro demonstrativní nebo referenční účely. Jako autor této práce si v žádném případě nenárokuji vlastnictví ani autorství žádné z těchto značek, k nim přidružených názvů, grafiky, ani dalších jakkoliv souvisejících materiálů.

1.3 Osobní zkušenosti na začátku práce

Na začátku práce jsem neměl žádné praktické ani teoretické zkušenosti s vývojem aplikací pro chytré telefony a stejně tak jsem se zatím nesetkal s žádným systémem pro řízení inteligentních budov. Značnou část času potřebného k dokončení této práce mi proto zabralo studium teorie, nových technologií a jazyků, hledání vhodných návrhových vzorů, osvědčených řešení (angl. „best practices“) a v neposlední řadě i zkoušení funkčních postupů metodou pokus-omyl.

V závěru této práce všechny nově nabyté poznatky stručně shrnu.

Kapitola 2

Specifikace aplikace

Cílem této kapitoly je seznámení se systémem řízení inteligentních budov *SmartBuildings*, definice kontextu praktického použití mobilního klienta a také sběr podkladů a informací důležitých pro další vývojovou fázi - *Návrh a analýzu aplikace*.

2.1 Inteligentní budovy obecně

Jak už jsem nastínil v úvodu této práce, s expanzí a rozvojem moderních technologií se dostávají do popředí i inteligentní budovy (IB) a jejich systémy. Asi největším cílem a vizí budoucnosti je vytvořit z obydlí integrované „živoucí organismy“, které by se samy přizpůsobily měnícím se potřebám svých obyvatel, změnám okolí, dále by výrazně optimalizovaly spotřebu energií, zvýšily bezpečnost, posílily komfort a zároveň byly šetrné k životnímu prostředí.

Systémy automatizace budov (BAS) a domů (HAS) spolu se systémy pro jejich správu a řízení (BMS) představují jádro dnešních inteligentních budov. Účelem těchto systémů je propojit všechny izolované podsystémy objektu[1] do jednoho řiditelného, centrálního celku. Přes centrálu pak mohou být připojené podsystémy (výtahy, protipožární ochrana, kamery, osvětlení, lokální síť, klimatizace, ...) efektivně ovládány s použitím BMS, které mohou spouštět časem nebo podmínkami řízené akce, poskytovat přehledné informace o celém systému, nebo například umožňovat rozšíření rozhraní či vzdálenou správu [18] [19].

Systémy pro řízení inteligentních budov stále nabízí obrovský potenciál pro rozvoj a inovace, což je jeden z důvodů, proč jsem si pro svou bakalářskou práci vybral právě toto téma.

2.2 Výchozí systém pro řízení inteligentních budov

Systém řízení inteligentních budov, který bude má aplikace rozšiřovat o mobilního klienta je vyvíjen na ČVUT FIT zhruba od počátku roku 2011 pod označením *SmartBuildings* (SB). Celý projekt vede Ing. Jiří Mlejnek a na jeho realizaci se účastní převážně studenti bakalářského, a nově i magisterského programu z ČVUT FIT.

2.2.1 Specifikace systému SB

Systém SB se skládá z několika důležitých částí:

- **Server a jádro** - Nejdůležitější část. Integruje, koordinuje a spravuje jednotlivá zařízení připojená k systému. Umožňuje nastavovat pokročilé chování zařízení a spouštět je na základě předem definovaných podmínek. V neposlední řadě také zprostředkovává rozhraní a veřejné API pro přístup k ovládání a administraci.
- **Ovladače zařízení** - Definují možnosti a schopnosti připojených zařízení. Ovladače například popisují, jakým způsobem lze zařízení k systému připojit, jak ho spravovat a podobně.
- **Webové rozhraní** - Primárně slouží jako ovládací a konfigurační terminál serveru. S jádrem komunikuje pomocí jeho vnitřního rozhraní a je tudíž závislé na jazyce, ve kterém je jádro napsané. K webovému rozhraní se uživatel může připojit s použitím prohlížeče přes lokální síť nebo, pokud to server povoluje, i přes internet. Nevýhodou

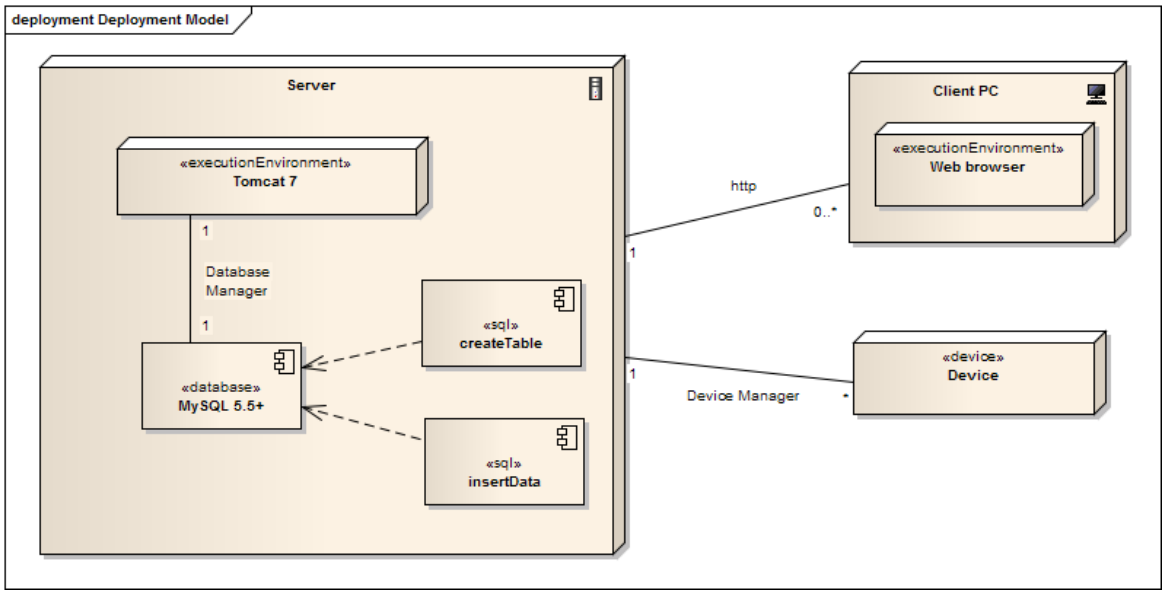
[1] Může se jednat nejen o budovy a domy, ale také o nákupní centra, tunely, křižovatky a další podobné stavby.

však zůstává množství složitých konfiguračních kroků, které uživateli znepříjemňují práci s administrací na zařízeních bez myši a klávesnice.

- **Zařízení a komponenty** - Všechna fyzické zařízení a subsystémy inteligentní budovy, které lze připojit k serveru SB. Připojeno může být naprosto cokoliv, co lze elektronicky ovládat a pro co byl napsán ovladač - tedy např. kamery, světla, zámky, elektronika, čidla a senzory, software, mechanické ovladače a spínače, různé systémy (hasící, bezpečnostní) a mnoho dalších.

Jádro je spuštěno jako server na fyzickém počítači. Zařízení a subsystémy inteligentní budovy jsou pak připojeny přímo k tomuto serveru. Aby jádro vědělo, jak s připojenými zařízeními pracovat, musí být na server nahrány ovladače těchto zařízení. Ty lze nahrát například přes webové rozhraní, které mimo jiné poskytuje detailní a plnohodnotnou administraci serveru. Veřejné API jádra realizované jako REST služby naopak umožňuje připojení platformě nezávislých klientů, kteří tak získají přístup k základním funkcím serveru.

Diagram nasazení systému SB a použité technologie lze vidět na obr. 2.1.



Obrázek 2.1: Diagram nasazení systému *SmartBuildings (autor: Pavel Švec, 2012)*

2.2.2 Dokumentace a vývoj API

Veřejné přístupné REST API, které server SB poskytuje je dostatečně komentováno a zdrojové kódy jsou dostupné v SVN repozitáři projektu. Funkcionalita SB serveru, specifikace i SW architektura byly v případě potřeby konzultovány s vedoucím práce nebo se členy týmu, který server vyvíjel. Všechny nutné změny a úpravy API byly realizovány ve spolupráci s autorem rozhraní jádra Pavlem Švecem.

2.3 Analýza využití

2.3.1 Uživatelé systému, cílová skupina

Cílovou skupinu uživatelů aplikace můžeme rozdělit na dvě hlavní kategorie:

1. *Správce systému inteligentní budovy* - Uživatel s nadprůměrnou počítačovou gramotností často velmi dobře rozumí informačním technologiím. Pro práci se systémem může projít odborným školením, takže zvládne i velmi složitou konfiguraci. Má také vyšší nároky na nastavení, rozšířenou funkcionalitu a bezpečnost aplikace.
2. *Běžný obyvatel inteligentní budovy* - Nejčastěji uživatel s průměrnou nebo podprůměrnou úrovní IT gramotnosti. Systém nejvíce využije při každodenních činnostech k ovládání zařízení v budově a spouštění přednastavených akcí, takže nevyžaduje žádné nadstandardní nebo speciální funkce. Běžnému uživateli jde spíše o jednoduchost ovládání, rychlost a spolehlivost systému.

Správci inteligentních budov přichází do styku se systémem nejčastěji ve firmách, větších budovách a komplexech. Naproti tomu, *běžný obyvatel* je člověk, který tento systém pravděpodobně nechal nainstalovat k sobě domů a očekává od něj zvýšení komfortu ve svém obydlí. Výsledná aplikace proto musí vyhovovat oběma skupinám uživatelů a musí být použitelná ve velmi rozdílných prostředích (firemní vs. domácí).

2.3.2 Současný stav

Bez mobilní aplikace jsou v současnosti uživatelé nuceni používat časově nebo finančně náročnější postupy.

Administrátor systému SB například potřebuje otestovat správnost nastavení vnějšího osvětlení, ale přístup do administrace je až na počítači ve 20. patře té samé budovy. Aby správce nemusel kvůli každé opravě nastavení chodit 20 pater tam a zpět, může hypoteticky:

- spolupracovat s kolegou, který bude stát před budovou a pomocí vysílačky nebo mobilního telefonu komunikovat se správcem sedícím u administrace SB
- použít drahý hardwarový ovladač vyrobený pro nastavení světel inteligentní budovy
- připojit se do administrace přes webový prohlížeč svého chytrého telefonu (tabletu) a pokusit se přenastavit konfiguraci i navzdory složitému ovládání

Ani běžný obyvatel nevyužívá potenciál inteligentní budovy tak, jak by mohl. K ovládání zařízení a přístrojů používá ovladače nebo mechanická tlačítka, aby pokaždé nemusel jít až k terminálu. Garáž proto raději otevře speciálním dálkovým ovladačem, nahrávání na kameře u vchodu spustí tlačítkem na jejím šasí a osvětlení zapíná pouze vypínačem na chodbě. Navíc, kvůli změnám v nastavení musí objednat odborníka, protože je pro něj rozhraní webové administrace příliš složité.

2.3.3 Kontext užití mobilní aplikace

Největší užitek z aplikace pro chytrý telefon spočívá v její mobilitě - pokud bude chtít uživatel spravovat libovolné zařízení, už nebude závislý na poloze ovládacích prvků nebo umístění počítače s administrací. Uživatel může budovu pomocí mobilní aplikace ovládat v dosahu lokální sítě a díky internetu také odkudkoliv, kde je dostupné internetové připojení. Právě díky takovým možnostem bude aplikace používána ve velmi odlišných situacích, které je při tvorbě specifikace potřeba brát v úvahu. Jako konkrétní situace použití aplikace můžeme uvést pár příkladů[2]:

- Uživatel přijíždí ke svému domu autem a chce jednoduše přes svůj telefon vypnout bezpečnostní alarm, otevřít příjezdovou bránu i vrata od garáže a nakonec rozsvítit v garáži a na chodbě.
- Administrátor opravuje kamery na ochozu výškové budovy a jednou rukou se přidržuje žebříku s jištěním. Pomocí svého chytrého telefonu potřebuje kamery v průběhu oprav otestovat, aby ověřil, že už je vše v pořádku.
- Uživatelka musela ve spěchu opustit svůj dům. Není si ale jistá, jestli zhasla všechna světla a zavřela střešní okna v podkroví. Navíc se blíží bouřka. Díky mobilní aplikaci může zkontrolovat stav osvětlení a oken. Pokud se svítí nebo jsou okna otevřená, může vše napravit na dálku.
- Uživatel sedí v obývacím pokoji, když někdo zazvoní na domovní zvonek. Aby nemusel scházet 3 patra ke vchodu a zjišťovat, kdo na něj zvoní, propojí svůj chytrý telefon s hlasovým komunikátorem u vchodových dveří a zeptá se, kdo tam je. Pokud jde například o někoho z rodiny, kdo si zapomněl klíče, uživatel mu je nemusí nosit nebo házet z okna - jednoduše mu dveře odemkne z pohodlí svého gauče.

2.4 Analýza konkurence

Před specifikací funkčních a nefunkčních požadavků na aplikaci je vhodné prozkoumat současný trh a udělat si přibližný „obrázek“ o úrovni a provedení konkurenčních produktů, které jsou v praxi reálně používány. Konkurence navíc své aplikace už nějaký čas vyvíjí, investuje do nich nemalé prostředky a pokud má platící zákazníky, je dost pravděpodobné, že funkcionalita aplikace odpovídá potřebám jejich zákazníků. Dostatečná analýza v takovém případě může upozornit na cesty, které nefungují a naopak vyzdvihnout funkce, které jsou pro aplikaci klíčové (není vždy nutné znovu objevovat kolo).

S prudce rostoucím trendem v oblasti BAS z posledních let přirozeně přibylo i množství více či méně kvalitních aplikací na globálním a českém trhu. Ze všech aplikací pro vzdálený přístup, které jsem prozkoumal, bych rád zmínil ty zajímavější a kvalitnější.

[2]Nesnažím se sepsat výčet všech situací, které mohou nastat. Uvedené příklady mají nastínit, v jak rozdílných kontextech může být aplikace používána.

2.4.1 Facility Prime

Aplikace od Siemensu pro vzdálený přístup pomocí iPadu[3]. Aplikace rozšiřuje robustní BAS *APOGEE* [7], který nabízí komplexní, pokročilé služby a je tudíž vhodný spíše pro větší společnosti. Siemens k APOGEE mimo řídícího softwaru dodává i hardwarová řešení, jako jsou řadiče, transformátory signálů zařízení nebo patentované HVAC systémy.

Facility Prime uživateli poskytuje detailní pohled na celý systém, takže kromě vizualizace stavu a umístění zařízení v budově lze také prohlížet sledované hodnoty, jejich změny v reálném čase, grafy nebo kompletní dokumentaci vybraného zařízení.

2.4.1.1 Klady

- Vizualizace zařízení, jejich reálné polohy a stavu na 3D mapě (viz obr. 2.2(a)).
- Poskytuje bohaté rozhraní, rozsáhlé možnosti nastavení a monitoring spousty hodnot ve formě grafů, tabulek a koláčů.
- Vysoká bezpečnost. Pro důležité změny v systému musí uživatel zadat své bezpečnostní údaje a pokud není k požadované změně oprávněn, systém operaci neprovede.

2.4.1.2 Zápory

- Nemá verzi pro chytré telefony. Tablet v některých situacích může být příliš velký a nedá se například nosit v kapse.
- Kvůli své složitosti se nehodí pro menší budovy a domy.
- Celý systém je závislý na hardwaru výrobce aplikace.
- Poněkud složitý systém. Neškolený uživatel může mít problémy s ovládáním.
- Velmi vysoká cena celého řešení.

2.4.2 iMyHome

Aplikace, za kterou stojí drobná italská společnost UpToWeb je určená primárně k rozšíření systémů *My Home BTicino* [8] pro automatizaci inteligentních domů. Mobilní rozšíření je dostupné ve verzi pro iPhone, iPad a iPod Touch[4] a je schopné komunikovat až se 7 základními typy objektů - se světly, scénáři, kamerami, termoregulací, zvukem, vlastními příkazy a automatizacemi. Tyto objekty pak lze seskupovat do skupin podle reálného umístění v domě (např. do skupiny „kuchyně“), takže uživatel vidí dané objekty pohromadě a snáze se zorientuje.

Použití aplikace je podobně jako u *Facility Prime* a *Loxone* vázané na pořízení specifického hardwaru.

[3] Tablet od společnosti Apple, který běží na operačním systému iOS.
[4] Přenosná zařízení od společnosti Apple, která běží na operačním systému iOS.

2.4.2.1 Klady

- Velice jednoduché uživatelské rozhraní (viz obr. 2.2(b)).
- Podpora pro chytré telefony i tablety.
- Pěkně řešené seskupování objektů do skupin.

2.4.2.2 Zápory

- Přílišná jednoduchost omezuje možnosti širšího využití aplikace.
- Závislost celého systému na komponentách a HW jednoho výrobce. K dispozici je tak relativně malá škála zařízení.
- Velmi slabá podpora, značná část dokumentace a informací pouze italsky.

2.4.3 Loxone

Jediná ze zmíněné trojce, která má zastoupení i v ČR a je dostupná nejen pro iOS, ale také pro Android platformu[5] [6]. Tvůrcem mobilní aplikace *Loxone* je stejnojmenná rakouská společnost, která se zaměřuje na svůj vlastní HAS včetně příslušenství [23]. Přesně tomuto záměru také odpovídá velké množství přehledně zpracovaných informací, rozsáhlá dokumentace k systému a solidní podpora pro vývojáře.

K provozu celého systému stačí tzv. „miniserver“, což je drobná, energeticky úsporná stanice, do které lze rovnou připojit elektrická zařízení, zásuvky, okna, světla a další systémy. *Miniserver* a připojená zařízení může uživatel nastavit přes osobní počítač pomocí speciálního konfiguračního SW, není ale problém připojit se téměř odkudkoliv přes jednodušší webové rozhraní nebo mobilní aplikaci. Uživatel má široké možnosti nastavení, takže může zařízení sdružovat do větších celků, nastavovat a spouštět časem řízené akce, ukládat nastavení do cloudu[7] atd.

Aplikace pro chytré telefony od *Loxone* mi ze všech, na které jsem zatím narazil, připadala nejužitečnější a nejlépe zpracovaná.

2.4.3.1 Klady

- Dostupnost a technická podpora, přítomnost na našem trhu, velké množství informací.
- Přehledné, jednoduché a všeobecně uživatelsky přívětivé UI.
- Aplikace je dobře vyváženým kompromisem mezi příliš „osekaným“ klientem a profesionální administrací s podrobným nastavením[8] (viz obr. 2.2(c)).

[5] Open source mobilní operační systém od společnosti Google založený na Linuxovém jádru.

[6] Aplikace pro Android byla vydána teprve v únoru 2012 a ke dni 6.5.2012 je dostupná pouze v testovací verzi.)

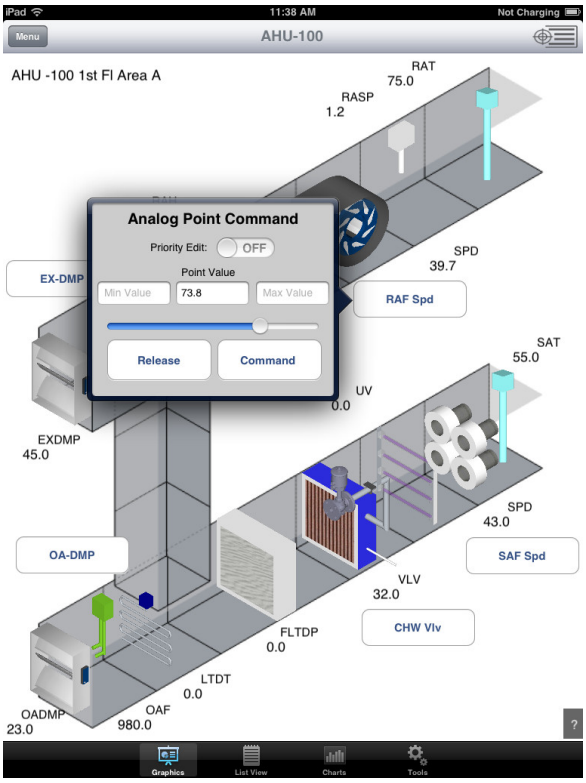
[7] Cloud představuje nepřetržitě běžící servery poskytovatele, na které jsou data ukládána a uživatel k nim má non-stop přístup z internetu.

[8] Jako profesionální administrace je k dispozici konfigurační software pro osobní počítač, který se díky klávesnici a myši ovládá mnohem pohodlněji, než by se ovládal na chytrém telefonu.

- Dostupnost a technická podpora přímo na našem trhu, velké množství informací.
- Dobrý poměr cena:výkon. Platí se za hardware, aplikace jsou zdarma.
- Definice skupin, manuálně a časově řízené spouštění akcí.

2.4.3.2 Zápory

- V současnosti se nehodí pro použití v korporátní sféře a větších budovách, protože neumí komunikovat se složitějšími zařízeními a pravděpodobně ani neustojí vyšší zátěž.
- Závislost na hardwaru výrobce. Aplikace je bez *miniserveru* nepoužitelná, nová zařízení lze připojit jedině přes rozšíření od *Loxone*.



(a) Facility Prime, umístění zařízení na 3D mapě *(zdroj: <http://itunes.apple.com/>)*

(b) iMyHome, jednoduché uživatelské rozhraní *(zdroj: <http://itunes.apple.com/>)*

(c) Loxone, výběr odstínu pro LED osvětlení *(zdroj: <https://play.google.com/store>)*

Obrázek 2.2: Náhledy konkurenčních aplikací

2.5 Funkční a nefunkční požadavky

Požadavky na funkcionalitu a výkonnostní nároky aplikace pro chytré telefony vychází z několika klíčových faktů a zjištění.

V první řadě je při jejich určování důležité, jaký je *současný stav* (viz 2.3.2), *v jakém kontextu bude aplikace používána* (viz 2.3.3) a hlavně, *jací uživatelé budou aplikaci používat* (viz 2.3.1). Zadruhé je potřeba vzít v úvahu konkurenci na českém i zahraničním trhu a *analyzovat podobné implementace* (viz 2.4). Na kvalitní konkurenční aplikacích lze vypozorovat fungující prvky, ale i zřetelné nedostatky. V neposlední řadě musí specifikace požadavků respektovat *možnosti a limity systému* (viz 2.2.1), který bude mobilní aplikace rozšiřovat.

2.5.1 Funkční požadavky

Soubor všech funkcí, které by měla aplikace uživateli poskytovat.

1. **Trvalé přihlášení uživatele** - Pro použití mobilního klienta a přístup k systému je nutné nejdříve se úspěšně přihlásit pomocí validních přihlašovacích údajů. Uživatel na svém chytrém telefonu často potřebuje přepínat mezi aplikacemi nebo se může pohybovat v lokalitě, kde dochází k výpadkům internetu. Abychom ho nenutili při každém obnovení našeho klienta zadávat znovu jméno a heslo (což může být třeba při řízení auta velmi náročné), přihlásíme uživatele k systému trvale.

2. **Odhlášení uživatele** - Umožní uživateli bezpečně se odhlásit od systému, ke kterému je trvale přihlášen. Automaticky bude uživatel odhlášen pouze v případě, pokud aplikaci dlouho nepoužil nebo pokud vypne telefon.

3. **Změna SB serveru** - Aplikace nebude napevno vázaná na jeden konkrétní SB server a uživatel může server změnit zadáním IP[9] nebo DNS adresy.

4. **Zobrazení stavu SB serveru** - Zobrazí současný stav SB serveru. Uživatel tak snadno pozná, jestli je server dostupný nebo ne, což mu může usnadnit řešení problémů s připojením.

5. **Procházení dostupných lokalit** - Dovolí uživateli zobrazit a procházet strom dostupných lokalit a oblastí, na které je inteligentní budova rozdělena.

6. **Zobrazení zařízení na 2D mapě** - Načte 2D mapu lokality (např. „kuchyně“) a zobrazí na ní všechna existující zařízení připojená k systému. Tato funkce uživateli především usnadní orientaci v dané budově a současně poskytne rychlý přístup k zařízením.

7. **Zobrazení detailů zařízení** - Aplikace zobrazí detailní informace o vybraném zařízení, například jeho umístění, název, stav, popis a mnoho dalších včetně ovládacích prvků.

[9] Pokud neuvedu jinak, je vždy myšleno IP ve verzi 4.

8. **Výpis všech zařízení** - Výčet všech zařízení připojených k systémů včetně jejich nejdůležitějších hodnot. Takový výpis uživateli poskytne rychlý přehled o aktuálním stavu všech zařízení v budově.

9. **Automatická aktualizace informací** - Aplikace bude v předem určených intervalech automaticky aktualizovat informace o lokalitách a zařízeních. Uživatel tak získá přístup k aktuálním informacím bez nutnosti neustále aplikaci ručně aktualizovat.

10. **Změna stavů a hodnot zařízení** - Uživatel bude moci u každého zařízení změnit jeho stav nebo přenastavit hodnoty.

11. **Změna profilu** - Aplikace uživateli umožní přepínat mezi různými profily, které si předtím definoval v administračním rozhraní serveru.

12. **Změna nastavení aplikace** - Dovolí uživateli upravovat některé nastavitelné konfigurační hodnoty aplikace. Uživatel si bude moci přizpůsobit chování aplikace podle svých potřeb.

2.5.2 Nefunkční požadavky

Nároky na výkon, technologie, spolehlivost, bezpečnost a další nefunkční rysy aplikace.

1. **Připojení přes Wi-Fi nebo internet** - K serveru se bude možné připojit přes internet nebo s pomocí lokální Wi-Fi sítě.

2. **Podpora ověřování pomocí Digest Access Authentication** - Pro zajištění vyšší bezpečnosti jsou požadavky aplikace proti serveru ověřovány pomocí metody *Digest Access Authentication*[10]. Aplikace proto musí tuto formu komunikaci podporovat.

3. **Podpora pro komunikaci s RESTful API serveru** - Aplikace musí implementovat rozhraní, které bude schopno komunikovat s RESTful API serveru.

4. **Vysoká responzivita aplikace** - Na chytrých telefonech je neresponzivní a pomalá aplikace uživatelem vnímána mnohem negativněji jako je tomu u jiných zařízení [27]. UI aplikace proto musí být plynulé a nesmí „zamrzat“.

5. **Jednoduché a přímočaré ovládání** - Uživatel by měl být schopen použít aplikaci i bez toho, aby si nejdříve musel přečíst návod. Aplikace proto musí obsahovat jednoduché ovládací prvky, jejichž funkcionalita bude uživateli v kombinaci s okolním kontextem UI jasně srozumitelná.

6. **Podpora pro více jazyků** - Aplikace musí být snadno rozšířitelná o další jazykové mutace.

7. **Podpora pro displeje různých velikostí** - Aplikace se musí správně zobrazit a být funkční na co největším počtu chytrých telefonů s různými rozměry displeje. Bezprostřední nutností je podpora všech displejů s rozměry od 3 do 5 palců[11] uhlopříčky.

[10] Digest Access Authentication používá metodu hašování přihlašovacích údajů ještě před jejich odesláním po HTTP.

[11] 1 palec odpovídá přibližně 2,54 cm.

Kapitola 3

Analýza a návrh řešení

Před začátkem implementace je vhodné analyzovat dostupné možnosti a systém nejprve navrhnout. V této kapitole se proto zabývám úskalím vývoje mobilních aplikací a diskutuji o výběru ideální platformy, z čehož později vycházím při tvorbě prototypu, softwarové architektury, datového modelu aplikace a návrhu grafického rozhraní.

3.1 Vývoj pro chytré telefony

Vývoj aplikací pro chytré telefony a tablety se od vývoje pro běžné desktopové systémy značně liší. Nejdůležitější rozdíly, které při návrhu aplikace musíme brát v úvahu bych shrnul takto:

- **Omezené výpočetní zdroje** - Chytrý telefon musí být malý, lehký a energeticky úsporný, aby byl schopen co možná nejdelšího provozu na baterii. Z těchto důvodů ani nemůže být vybaven silnějším HW a ačkoliv už existuje pár modelů přibližně na stejné výpočetní úrovni, jako byly běžné osobní počítače v roce 2004[1], velká spousta dnes používaných chytrých telefonů má výrazně pomalejší procesory, méně paměti a prostoru pro ukládání dat. S takovými podmínkami musí vývojář počítat a přizpůsobit se jim - to znamená nealokovat paměť, když to není potřeba, nevolat zbytečně funkce, optimalizovat výkon a přemýšlet o každém svém implementačním rozhodnutí.

- **Výrazně odlišný kontext použití** - Uživatel chytrý telefon používá v jiných situacích a jiným způsobem než třeba notebook nebo osobní počítač. Často je to ve spěchu, na neobvyklých místech, během jiné souběžné činnosti (např. při řízení), velmi nárazově a v prostředí s omezenými podmínkami (např. v místě se slabým internetem). Aplikace použitelná v podobných situacích musí být zjednodušená, naprosto srozumitelná, snadno ovladatelná a vývojář musí dobře odhadnout, kde a jak ji bude uživatel reálně použije.

- **Rozdílná architektura OS** - Kvůli *omezeným výpočetním zdrojům* a *náročným podmínkám použití* se liší také architektura mobilního OS a hlavně způsob, jakým systém spravuje běžící aplikace. Techniky úplně běžné u desktopových systémů, jako je například multi-tasking, buď na mobilních platformách úplně chybí nebo jsou řešeny alternativní cestou. Aplikace se vypnutím neukončují, zůstávají pozastavené v pozadí pro případ, že by je chtěl uživatel opět rychle použít a pokračovat tam, kde naposledy skončil[2]. Systém pak sám podle stavu volné paměti rozhoduje o tom, kterou z pozastavených aplikací ukončí. Takto nedeterministické chování musí aplikace umět zvládnout.

- **Malá zobrazovací plocha displeje** - Kompaktní, drobné rozměry chytrých telefonů znamenají malý displej, na kterém lze uživateli zobrazit pouze určité množství informací a ovládacích prvků. Aplikace pro mobilní zařízení proto musí být navržena s ohledem na tuto indispozici, musí přizpůsobit svou navigaci, správně seskupovat informace a je mimořádně důležité, aby byla jednoduchá a přehledná.

- **Způsob ovládání** - Tlačítka ovládání a klávesnice bývají zjednodušená nebo mohou úplně chybět. Zařízení s dotykovým displejem se často ovládá pomocí dotykových gest, vstupy uživatel zadává přes softwarovou klávesnici zobrazenou na displeji, otáčením chytrého telefonu lze ovládat nebo měnit rozvržení GUI atd. S jistotou se ale dá říci, že je ovládání mobilních zařízení mnohem náročnější a méně pohodlné jako ovládání osobního počítače nebo notebooku. Z těchto důvodů by aplikace uživatele neměla

[1] Chytrý telefon HTC One X představený začátkem roku 2012 má čtyřjádrový procesor s kmitočtem jádra 1,5GHz a 1GB RAM paměti [17].

[2] Takto situaci řeší minimálně platformy *Android*, *iOS* a *Windows Phone OS*.

nutit zadávat příliš mnoho vstupů nebo používat tlačítka či gesta, která slouží k jiným vžitým akcím.

3.2 Mobilní platforma

Před samotným návrhem SW architektury a modelu aplikace je nejdříve nutné zvolit mobilní platformu, pro kterou budu aplikaci vyvíjet. SW architektura, návrhové vzory, osvědčené postupy a také přístup k tvorbě UI jsou totiž na vybrané platformě silně závislé - co funguje na jedné může být naprosto nepoužitelné pro druhou.

3.2.1 Dostupné platformy

V současnosti má smysl rozhodovat se mezi třemi hlavními platformami, ostatní jsou jen velmi málo rozšířené (nebo do budoucna neperspektivní) a cílem je vytvořit aplikaci dostupnou pro co nejvíce uživatelů. Zvažovat proto budu mobilní operační systémy *Android*, *iOS* nebo *Windows Phone OS*.

3.2.1.1 Android

Open source operační systém založený na Linuxovém jádře, za jehož vývojem stojí společnost Google. Ačkoliv je Android relativně nový OS, každý den je na celém světě aktivováno více jak 850 000 přístrojů s tímto systémem. Android proto můžeme najít v přístrojích přibližně 55 výrobců a více než 300 distributorů [3]. Nejrozšířenější je na mobilech, chytrých telefonech a také tabletech.

Androidí aplikace jsou spouštěny v *Dalvik Virtual Machine* (DVM), což je prostředí vycházející z *Java Virtual Machine*, jehož architektura je upravena pro systémy s malou rychlostí a nízkou pamětí. Vývoj aplikací probíhá v jazyce Java, který je rozšířen o Android SDK. Zdrojové kódy jsou překládány do Java bytecode a potom dále do kódu vhodného pro DVM ještě před instalací aplikace. Aplikace se distribuuje ve formě jednoduchého balíčku *.apk*, který uživatel nainstaluje téměř jedním kliknutím - poté je aplikace okamžitě připravena ke spuštění.

Android poskytuje rozsáhlou, aktualizovanou dokumentaci, mnoho informačních zdrojů, velmi silnou podporu a obrovskou komunitu nadšených vývojářů.

3.2.1.2 iOS

Velmi oblíbený operační systém od společnosti Apple. Ačkoliv systém může běžet pouze na několika málo zařízeních od Apple, bylo již prodáno více než 315 milionů iPadů, iPhonů nebo iPod Touchů [20]. Prodej aplikací pro iOS v porovnání s ostatními platformami navíc vykazuje celkově nejvyšší roční obraty.

Aplikace se vyvíjí v jazyce *Objective C* s využitím iOS SDK a aby mohl vývojář své aplikace vyvíjet a distribuovat, musí si zakoupit licenci. Vývojové prostředí *Xcode* navíc nepod-

poruje ostatní desktopové operační systémy, jako je například Windows, takže je vývoj na čemkoliv jiném jako na počítači od Applu velmi obtížný[3].

Navzdory všem omezením má i iOS velice početnou komunitu vývojářů, kvalitně zpracovanou dokumentaci a hlavně velkou popularitu mezi svými spotřebiteli.

3.2.1.3 Windows Phone OS

Nejmladší a nejméně rozšířený operační systém z uvedené trojce, který byl vyvinul společností Microsoft. *Windows Phone OS* je určen převážně pro chytré telefony a je nasazován na zařízení 8 výrobců, z nichž nejdůležitějším je Nokia. Největší předností platformy je její jednoduché a přehledné uživatelské rozhraní *Metro*, které již získalo několik světových ocenění za design [30]. S roustoucími nároky zákazníků na co nejjednodušší UI je pravděpodobné, že bude podíl *Windows Phone OS* na světovém trhu v budoucnosti růst.

Vývojář může své aplikace vyvíjet ve *frameworku XNA* nebo *Silveright pro Windows Phone*[4] v osekaném IDE *Visual Studio Express Edition*, které je bohužel podporováno pouze systémy Windows Vista a novějšími. Podobně jako *iOS* a *Android*, i *Windows Phone OS* používá k distribuci aplikací centrální online tržiště, takže instalace všech aplikací probíhá co možná nejjednodušeji.

Relativně malá vývojářská komunita je kompenzována kvalitní dokumentací. Na druhou stranu, i když je tento operační systém velmi mladý, je založen na starších technologiích (XNA, Silverlight), takže lze řadu informací dohledat i mimo vývoj pro *Windows Phone*.

3.2.2 Výběr platformy

Při výběru vhodné platformy jsem sledoval hned několik parametrů. *iOS* jsem vyloučil jako první, protože nemám k dispozici počítač nezbytný pro vývoj, žádné z mobilních zařízení důležité pro testování, ani zkušenosti s jazykem Objective C.

U *Windows Phone OS* mě zaujalo jeho jednoduché uživatelské rozhraní Metro, rychlá učící křivka a silná podpora ze strany Microsoftu (finanční, marketingová i technická). Navíc už jsem napsal několik aplikací v C# a pracoval s XNA frameworkem, takže jsem měl přibližné představy o možnostech a potenciálu vývoje. *Windows Phone* je ale velmi málo rozšířený, existuje jen pro dotykové telefony a stále trpí mnoha nedostatky, takže jsem se jednoznačně rozhodl vyvinout aplikaci pro **Android**, což mimo jiné podpořily tyto důvody:

- Aplikace bude dostupná největšímu možnému počtu uživatelů a bude podporovat velkou škálu různých zařízení. Zvyšuje se tak pravděpodobnost, že uživatel, který bude chtít použít mou mobilní aplikaci, už doma zařízení s Androidem má a není proto nucen kupovat nové.
- Android je open source, který kolem sebe vybudoval obrovskou uživatelskou a vývojářskou základnu. To sebou přináší velkou spoustu výhod, mimo jiné například množství kvalitních informací dostupných na internetu, možnost upravovat a rozšiřovat zdrojové kódy Android frameworku a mnoho dalších.

[3] Xcode sice lze zprovoznit pomocí tzv. „jailbreaku“, ale nehodí se pro vývoj seriozního projektu. Obsahuje plno zvláštních chyb a neočekávaných problémů.

[4] Oba frameworky jsou založeny na .NET jazycích, konkrétně na C# a nově i Visual Basicu.

- Výhody, které přináší vývoj v Javě - jednoduchost, nezávislost IDE i kompilátoru na operačním systému, podpora široké řady vývojových nástrojů atd.
- Celkové provedení systému, řada pěkných funkcí a propojení včetně synchronizace s online službami Googlu.
- Mám přístup k několika různým typům zařízení s Androidem, což značně usnadní testování a zvýší kvalitu aplikace.

3.2.3 Podpora zařízení

Aplikace bude přímo podporovat zařízení se systémem Android od verze 2.3.3[5] (API úrovně 10) a vyšší, dále multijazyčnost, displeje se středním a vysokým DPI a také chytré telefony s dotykovým displejem (viz 2.5.2 Nefunkční požadavky).

3.3 Návrh funkcionality, prototyp

Před *analýzou a návrhem modelu aplikace* (viz 3.4) je potřeba definovat hlavní případy užití (UC) - tedy navrhnout, jakým konkrétním způsobem systém uživateli zprostředkuje funkcionalitu podle specifikace požadavků (viz 2.5.1 Funkční požadavky).

3.3.1 Metodika

Mobilní aplikace by neměla obsahovat zbytečné funkce nebo složité ovládací prvky. Abych podpořil jednoduchost aplikace, rozhodl jsem se nesepisovat klasické UC scénáře a promítnout případy užití do prototypu ve formě drátěného modelu aplikace, tzv. „wireframes“ (WF). Výroba drátěného modelu je velmi rychlá a ve své nejjednodušší podobě se k zachycení funkcionality používají pouze jednoduché čáry a textové popisky.

Pro účely této práce připravím komplexnější drátěný model, který mi pomůže lépe si představit funkcionalitu a ovládací prvky pospolu tak, jak by mohly fungovat ve skutečné aplikaci. Další výhodou wireframů je, že zachovávají nezávislost na vzhledu GUI, umožňují snadno odhalit chyby v návrhu a provádět rychlé změny.

První finální verzi prototypu aplikace použiji pro testování metodou heuristické evaluace, kterou podrobněji rozebírám v kapitole o testování (viz 5.1). Výsledky těchto testů poslouží jako podněty k vylepšení prototypu, z něhož následně výjdu při návrhu SW architektury, modelu aplikace a grafickém návrhu UI.

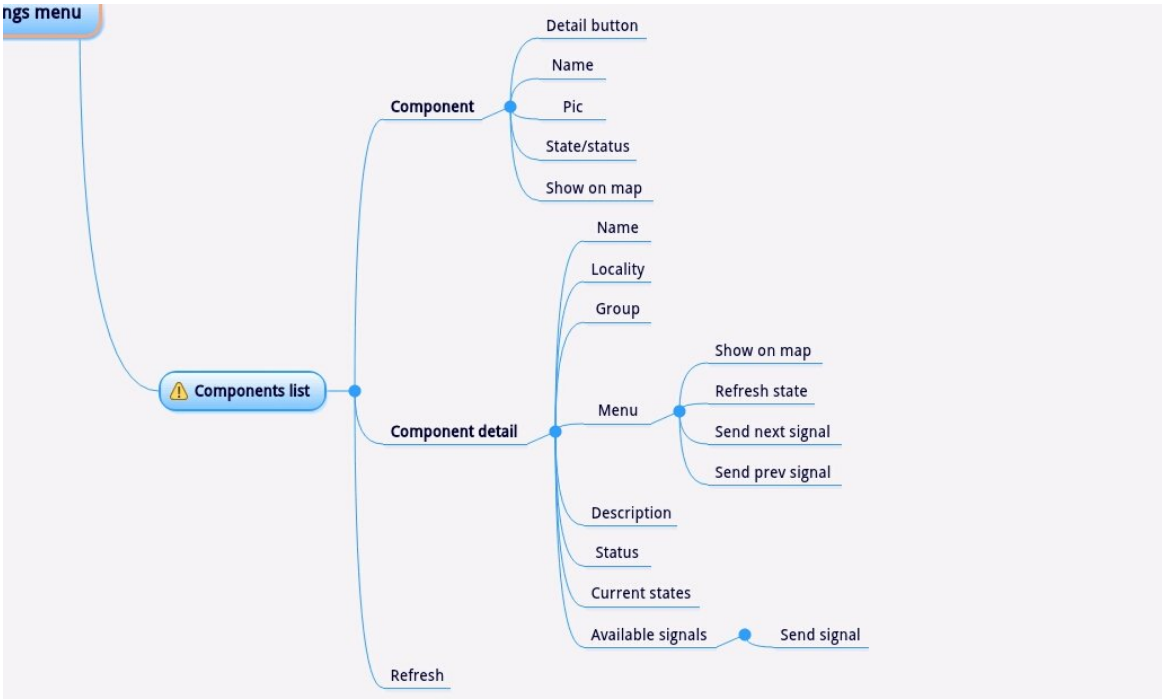
[5] Tedy více jak 70% všech Android zařízení [5] s tím, že aplikace ve velké většině případů poběží i na ostatních zařízeních s nižší verzí OS.

3.3.2 Prototypování

Vývoj prototypu probíhal iteracemi následujících kroků:

1. Definice jednotlivých scénářů a závislostí pomocí myšlenkové mapy.
2. Zachycení scénářů a funkcionality podrobněji formou jednoduchého drátěného modelu.
3. Vylepšení prototypu zachycením navigace mezi jednotlivými obrazovkami WF do jednoho interaktivního celku.

Začal jsem tvorbou myšlenkové mapy v aplikaci *MindJet pro Android* [24]. Nejprve navrhnu hrubou strukturu modelu a tu postupně rozšiřuji do čím dál podrobnějších detailů, k čemuž mi stačí i použití jednoduchých značek a hesel. Tato metoda mi pomáhá rozdělit aplikaci na menší segmenty a každému segmentu přiřadit funkce a ovládací prvky, které by měl obsahovat. Pro lepší představu, funkční požadavek č. 9: *Výpis všech zařízení* jsem myšlenkovou mapou rozpracoval tak, jak to ukazuje obr. 3.1[6]



Obrázek 3.1: Myšlenková mapa - rozvržení výpisu zařízení

Myšlenková mapa mi pomáhá s přípravou drátěného modelu. Pro výrobu wireframů jsem použil jednoduchou open source aplikaci *Pencil 1.3* [25], která se výborně hodí na tvorbu prototypů pro mobilní aplikaci a je k dispozici i jako rozšíření pro prohlížeč Firefox.

[6] Tento i všechny mé další vývojové materiály (dokumentace, kódy, diagramy, modely apod.) jsou v angličtině. Důvodem je snaha zpřístupnit materiály co největšímu počtu potencionálních vývojářů, kteří by v budoucnosti mohli mít zájem o další vývoj a vylepšení aplikace.

V poslední fázi jsou mezi sebou všechny navržené obrazovky s wireframy propojeny pomocí hypertextových odkazů, takže v prototypu vznikne jednoduchá navigace. Výsledek je nakonec exportován do HTML formátu a vzniklý návrh lze procházet na běžném počítači klikáním na ovládací prvky.

Celý proces je možné několikrát opakovat a prototyp tak vylepšit na požadovanou úroveň. Obr. 3.2(a) a obr. 3.2(b) ukazují náhledy mého rozšířeného prototypu, který byl určen pro první testy použitelnosti.

(a) Scénář „výpis zařízení“ (b) Scénář „detail zařízení“

Obrázek 3.2: Rozšířený prototyp aplikace

3.4 SW architektura

Jak podrobněji rozebírám v podkapitole 3.1 *Vývoj pro chytré telefony*, aplikace navržené pro desktopové systémy se od těch mobilních razantně liší, takže při návrhu architektury nemohu vycházet ze svých zkušeností s vývojem pro desktop a web.

Proto nejprve specifikuji požadavky a nároky na SW architekturu a na jejich základě poté analyzuji a vyberu vhodnou strukturu, kterou upravím a popíšu.

3.4.1 Požadavky na architekturu aplikace

Požadavky na SW architekturu vyplývají ze *Specifikace* (viz 2) a *Návrhu funkcionality* (viz 3.3). Zároveň musí odpovídat struktuře frameworku Android. Architektura aplikace musí:

1. Být **dostatečně robustní** - Pokud dojde k rozšíření serveru SB o novou funkcionalitu a technologie, aplikace musí být dostatečně robustní na to, aby šla vylepšit bez zbytečných komplikací. Jestliže server například začne podporovat streamování zvuku z mikrofonů, nesmí se stát, že bude rozšíření aplikace znamenat změny v architektuře.

2. Vhodně řešit **vícevláknové operace** - Časově náročnější operace nemohou být spouštěny v jednom vlákně, protože by docházelo k „zamrzání“ GUI. Aplikace rovněž bude kešovat data na pozadí nebo stahovat mapy, což ani nelze bez použití vláken rozumně realizovat. SW Architektura musí být zvolena s ohledem na potřeby vícevláknového zpracování.

3. Mít **striktně oddělené vrstvy** - Aby docházelo k menším závislostem mezi objekty v odlišných vrstvách, je nutné použít architekturu, která tyto vrstvy navzájem dobře oddělí.

4. Zohledňovat nároky na **synchronizaci mezi zařízením a serverem** - Při stahování dat z SB serveru může být aplikace pozastavena operačním systémem nebo může na krátkou chvíli přijít o internetové připojení. Vhodná architektura a mechanika synchronizace dat mezi serverem a klientem je klíčová pro spolehlivost a stabilitu aplikace.

3.4.2 Hledání vhodné architektury

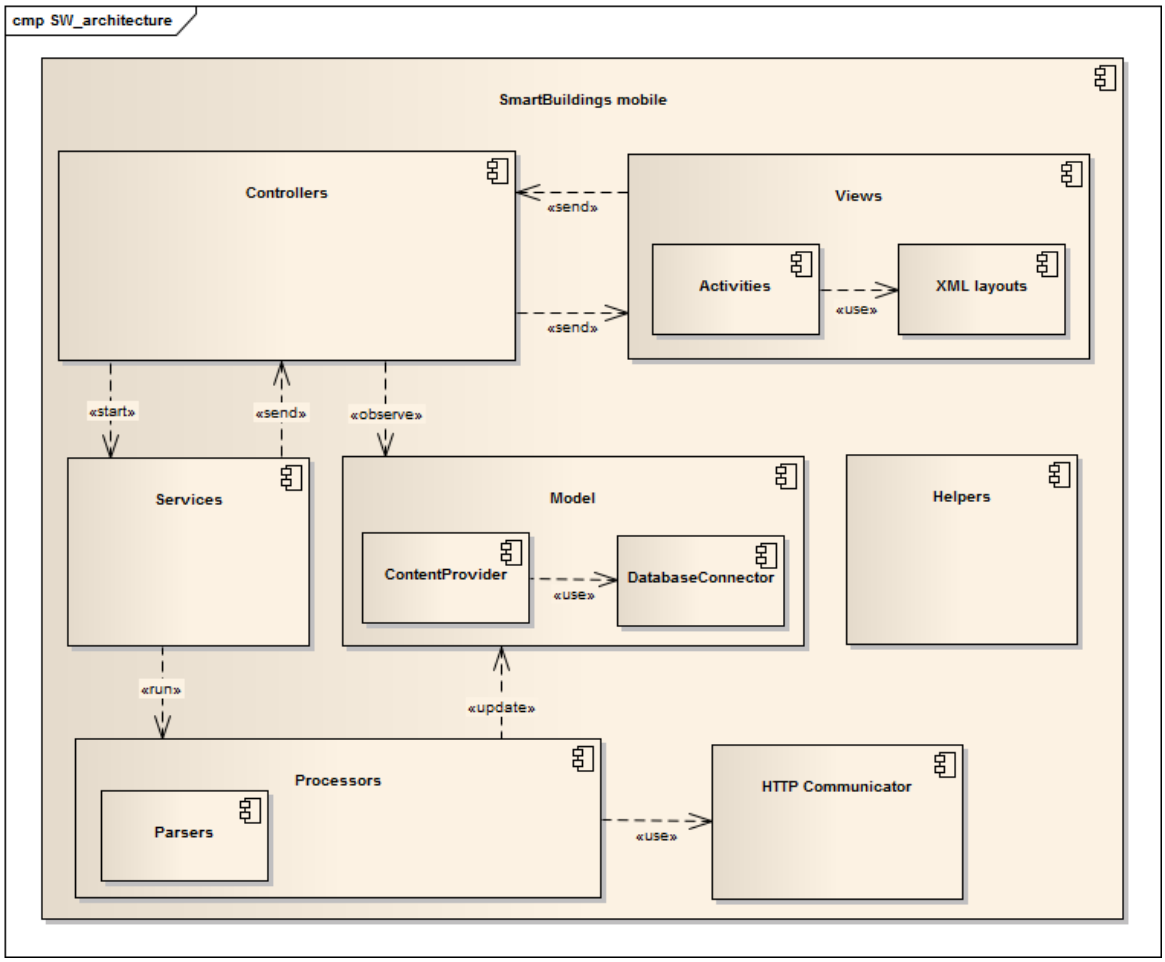
Abych netrávil desítky hodin času objevováním objeveného, rozhodl jsem prozkoumat knihy o vývoji pro Android, zdrojové kódy podobných aplikací a internetové diskuze řešící nejlepší přístup k architektuře REST klientů.

Jediné rozumné řešení, které nejvíce odpovídalo výše uvedeným *požadavkům na architekturu aplikace* (viz 3.4.1) byl koncept, jenž v roce 2010 stručně představil vývojář Androidu Virgil Dobjanschi [9] a který lze jednoduše popsat jako MVC architektura s robustní síťovou vrstvou. Na podobnou architekturu jsem také narazil ve velmi kvalitní knížce *Programming Android* [2] a i zkušenosti vývojářů s tímto přístupem byly vesměs velmi pozitivní. Také existuje několik aplikací, které architekturu implementují. Z těch, jejichž zdrojový kód jsem blíže analyzoval mohu jmenovat například aplikace *Eli-G* [11] nebo *Google I/O App* [16].

3.5 Návrh struktury a modelu komunikace

Navzdory skutečnosti, že jsem vycházel z nalezených architektonických doporučení, počet a typ vrstev mé aplikace a způsob komunikace mezi nimi se v mnohém liší.

Android framework svou strukturou vývojáře nutí k použití vícevrstvé (angl. „n-tier“) architektury. Uživatelské rozhraní je v Androidu definováno především pomocí XML souborů a k zobrazení GUI je nutné inicializovat tzv. *Aktivitu*[7], kterou si pro jednoduchost lze představit jako jednu samostatnou obrazovku. Problém je v tom, že podle původní SW architektury *Aktivita* modifikuje uživatelské rozhraní a současně také provádí řídící operace. Z toho vyplývá, že se chová jako částečné *View*[8] a také jako *Controller*, což je nežádoucí, protože by tak mohlo docházet k duplikacím v situacích, kdy budou chtít dvě různé *Aktivity* provést stejnou řídící operaci.



Obrázek 3.3: Návrh architektury a komunikace mezi vrstvami

[7] Aktivita je v Android frameworku realizována jako třída `Activity`. Aktivitou tedy může být i jakýkoliv její potomek.

[8] Druhou polovinu *View* v tomto případě tvoří výše zmíněné XML definice rozhraní.

Jedním z možných řešení je oddělit řídící logiku a vše, co by měl provádět *Controller* do vlastní vrstvy a považovat XML definice rozhraní spolu s *Aktivitami* za čisté *View*. Návrh s ohledem na tento přístup lze spolu s ostatními vrstvami, které podrobněji popisuji dále, vidět v diagramu na obr. 3.3. K návrhu všech diagramů jsem použil software *Enterprise Architect* ve verzi 8.0.

3.5.1 View

View vrstva má na starosti zobrazení uživatelského rozhraní a manipulaci s ním. Zároveň uživatelské rozhraní striktně odděluje od zbytku aplikace.

Skládá se ze dvou důležitých částí - z *XML definic*, což jsou statické XML soubory popisující, jak má grafické rozhraní vypadat, a z *Aktivit*, což jsou naopak Java třídy, které slouží k inicializaci a dynamickým změnám v uživatelském rozhraní. *Aktivity* rovněž odchytávají události, které uživatel vyvolá manipulací s ovládacími prvky uživatelského rozhraní. Podle zachycené události daná *Aktivita* pošle požadavek na zpracování všem *Controllerům*, ke kterým je v dané chvíli zaregistrována. Během zpracování požadavku jednotlivými *Controllery* od nich *Aktivita* postupně přijímá žádosti o aktualizace a změny uživatelského rozhraní, které následně provádí.

Celý proces si můžeme představit například takto: Uživatel na displeji svého chytrého telefonu stiskne tlačítko „Aktualizovat stav zařízení“. Tím vznikne v uživatelském rozhraní událost, kterou zachytí *Aktivita* reprezentující aktuální zobrazení GUI. *Aktivita* pošle *Controllerům* zprávu s požadavkem, aby provedly aktualizaci stavů zařízení a nezajímá se o to, jak to *Controllery* zařídí. *Aktivita* od *Controllerů* akorát na začátku obdrží žádost o zobrazení dialogu s textem „Aktualizuji, počkejte prosím“ a jakmile *Controllery* zpracování požadavku dokončí, pošlou *Aktivitě* další dvě žádosti. První o skrytí právě zobrazeného dialogu a druhou o aktualizaci textů s informacemi zařízení.

3.5.2 Controllery

Controller slouží jako řídící centrála, výpočetní logika a „srdce aplikace“.

Jeho úkolem je zachycovat a vyřizovat požadavky příchozí z *View* vrstvy. Náročnější operace *Controller* musí spouštět v novém vlákně, aby neblokovaly uživatelské rozhraní. Pokud *Controller* k vyřízení požadavku potřebuje komunikovat se SB serverem, spustí *Service*, který se o celou transakci postará a zároveň synchronizuje data s lokálním úložištěm. *Controller* od *Service* vrstvy pouze dostává notifikace s informacemi o současném stavu celé operace, aby mohl reagovat a podle situace aktualizovat *View*.

Controller se současně stará o přípravu dat pro *View*, takže sleduje vrstvu *Model* a pokud v ní dojde ke změně v datech, aktualizuje datové objekty, které s *View* sdílí.

V praxi může práce *Controlleru* vypadat například takto: *Controller* přijme od *View* (konkrétně od Aktivity) požadavek na aktualizaci stavu zařízení a okamžitě *View* odpoví odesláním žádosti na zobrazení grafického prvku signalizujícího načítání[9]. Tato operace vyžaduje, aby aplikace ověřila, zda-li na serveru došlo ke změně stavu zařízení. *Controller*

[9] *Controller* je od *View* izolovaný, takže při zasílání žádosti o zobrazení prvku signalizujícího načítání neví, jak naslouchající *Aktivity* žádost zpracují a co vlastně uživatel ve výsledku na svém displeji uvidí. Každá *Aktivita* může žádost vyřídit po svém - jedna například zobrazí dialog, druhá malou animovanou ikonku atd.

proto musí založit nové vlákno, ve kterém spustí *Service*. *Service* už se postará o synchronizaci stavu zařízení mezi serverem a lokálním datovým úložištěm. V této fázi *Controller* opět nezajímá, jakým způsobem *Service* celou synchronizaci vyřeší. *Service* akorát *Controlleru* pravidelně posílá notifikace o průběhu požadované operace, takže *Controller* přesně ví, kdy zpracování začalo a kdy skončí. *Controller* taky předpokládá, že pokud dojde ke změně stavu zařízení, *Service* provede aktualizaci dat na lokálním úložišti. Z tohoto důvodu *Controller* sleduje změny u datového modelu zařízení. Pokud k nějaké změně dojde a *Service* již poslal notifikaci o dokončení operace, *Controller* jednoduše aktualizuje instanci s atributy zařízení, kterou sdílí s *View*. Hned poté všem *Aktivitám* pošle žádost o skrytí nebo zastavení grafického prvku signalizujícího načítání a také žádost o aktualizaci textů s informacemi o daném zařízení.

3.5.3 Service

Service vyřizuje a spravuje veškeré požadavky týkající se komunikace se SB serverem.

Jde o vrstvu, která stejně jako vrstvy *View* nebo *Model* využívá přirozené struktury Android frameworku, konkrétně schopností třídy `IntentService`. Třída `IntentService` je objekt, který Android automaticky inicializuje na pozadí a ten pak běží nezávisle na tom, jestli daná aplikace právě vlastní kontext nebo ne[10]. Tyto vlastnosti *Service* vrstvě umožňují provádět časově náročné operace, jako je například stahování velkého množství dat ze SB serveru s tím, že uživatel během celého procesu nemusí čekat na jeho dokončení, může aplikaci minimalizovat a vrátit se až bude operace dokončena.

Service vrstva samotná se ale ve skutečnosti stará hlavně o správu požadavků a pravidelné zasílání notifikací *Controlleru*. Mimo to také připravuje data pro *Processory* a zpracovává chyby, ke kterým případně může dojít v průběhu celé operace. Vrstva s *Processory* je pak dále odpovědná za manipulaci s daty, které jsou přijímány ze serveru nebo odesílány na server.

V aplikaci pak může být *Service* vrstva reálně využita přibližně takto: *Controller* spustí *Service* s požadavkem synchronizovat se SB serverem informace o konkrétním zařízení. *Service* připraví potřebná data a spustí Android službu (instanci třídy `IntentService`). V této službě pak inicializuje příslušný *Processor*, který celou operaci zpracuje a *Service* víceméně nezajímá, jakým způsobem to *Processor* udělá. Současně se startem *Processoru Service* pošle notifikaci *Controlleru* o tom, že zpracování započalo. Jakmile *Processor* vše dokončí, Androidí služba se ukončí a *Service* pošle *Controlleru* notifikaci o úspěšném dokončení operace. Pokud by se stalo, že při zpracování *Processorem* dojde k jakékoliv chybě, *Service* ji zachytí a opět pošle *Controlleru* notifikaci včetně popisu chyby.

3.5.4 Processory

Processory jsou vrstva, která zpracovává data určená pro odeslání na SB server a také data, která aplikace obdržela od serveru. Jejich hlavní činností je mimo jiné synchronizace přijatých a odesílaných dat s lokálním úložištěm.

[10] Jinými slovy, jestli je aplikace právě na popředí nebo jestli je minimalizovaná na pozadí a systém ji proto pozastavil. Blíže tohle chování popisuji v podkapitole 3.1 *Vývoj pro chytré telefony*, v části *Rozdílná architektura OS*.

Celý účel *Processorů* se dá stručněji popsat tak, že mapují jednotlivé REST operace[11] na odpovídající CRUD operace[12]. Ve skutečnosti je celý průběh mnohem složitější.

Pokud chce *Processor* získat data ze serveru, nejdříve pomocí *HTTP Communicatoru* zašle požadavek na získání dat definovaný RESTful URL adresou. Vrstva *HTTP Communicator* daný požadavek odešle, zpracuje odpověď a pokud vše proběhne v pořádku, vrátí *Processoru* zpět JSON objekt nebo pole JSON objektů. *Processor* posléze využije vhodný *Parser* a převede JSONy na odpovídající Data Transfer Objekty (DTO), které aplikace běžně používá pro výměnu informací. *Processor* nakonec takto získané DTO vloží do lokálního datového úložiště pomocí *Content Provideru* z vrstvy *Model*.

Jestliže chce *Processor* odeslat data na SB server, postup je přibližně obrácený. *Processor* nejprve získá data z lokálního úložiště reprezentovaného vrstvou *Model*, tato data podle požadavků pozmění a vloží do DTO, které následně přes vhodný *Parser* převede do JSON objektů. JSONy spolu s cílovou RESTful URL adresou předá vrstvě *HTTP Communicator*, která se postará o jejich odeslání na SB server a vrátí *Processoru* odpověď. V případě, že byla původní data pozměněna a odpověď ze strany serveru potvrzuje úspěšnou aktualizaci, *Processor* opět využije *Content Provider* z vrstvy *Model*, aby data aktualizoval i na lokálním úložišti.

3.5.5 Model

Model představuje hlavní datovou vrstvu aplikace a zprostředkovává rozhraní pro ukládání a získávání dat z lokálního úložiště.

Základem datové vrstvy *Model* je implementace abstraktní třídy `ContentProvider`. Tato třída poskytovaná Android frameworkem tvoří další ze základních stavebních kamenů každé pokročilejší aplikace a při správné implementaci metod a URI cest včetně deskriptoru popisujícího tyto cesty `ContentProvider` poskytuje univerzální rozhraní pro přístup k datové vrstvě[13]. Pokud se pro to vývojář rozhodne, může navíc aplikace přes rozhraní `ContentProvideru` poskytovat data z lokálního úložiště cizím aplikacím v OS Android.

Druhou částí vrstvy *Model* je `DatabaseConnector`, tedy třída, pomocí které vrstva `ContentProvider` ukládá data přímo do lokálního úložiště nebo je odtamtud načítá. Obrovskou výhodou těchto dvou tříd je relativní nezávislost `ContentProvideru` na implementaci `DatabaseConnectoru`. `ContentProvideru` nezáleží způsobu, jakým `DatabaseConnector` řeší ukládání a načítání dat, ani na tom, do jakého úložiště jsou data ve skutečnosti ukládána, dokud `DatabaseConector` korektně implementuje požadované chování. Díky těmto vlastnostem se dá říci, že je architektura nezávislá na lokálním úložišti, které v budoucnu můžeme měnit pouhou změnou implementace `DatabaseConnectoru`.

[11] Myšleny jsou operace *Get*, *Put*, *Post* a *Delete*.

[12] V Android Frameworku a jeho třídě `ContentProvider`, která se stará o správu datového úložiště, jsou tyto operace známy jako *Insert*, *Query*, *Update* a *Delete*, v tomto uvedeném pořadí.

[13] Používání tohoto rozhraní se velmi podobá používání RESTful URL adres s tím rozdílem, že `ContentProvideru` je URI předávána jako parametr jedné z metod *Insert*, *Query*, *Update* nebo *Delete*.

3.5.6 HTTP Communicator

Jednoduchá vrstva *HTTP Communicator*, která se stará o HTTP komunikaci s REST rozhraním SB serveru.

HTTP Communicator konfiguruje HTTP klienta, stará se o odesílání a přijímání HTTP požadavků a spravuje RESTful URL adresy. Jedinou vrstvou, která *HTTP Communicator* smí používat jsou *Processory*. Pouze *Processory* totiž ví, jakým způsobem zpracovávat JSON objekty, které *HTTP Communicator* od serveru může přijmout a jedině *Processory* umí připravit JSONy pro aktualizaci dat na serveru.

3.5.7 Helpery

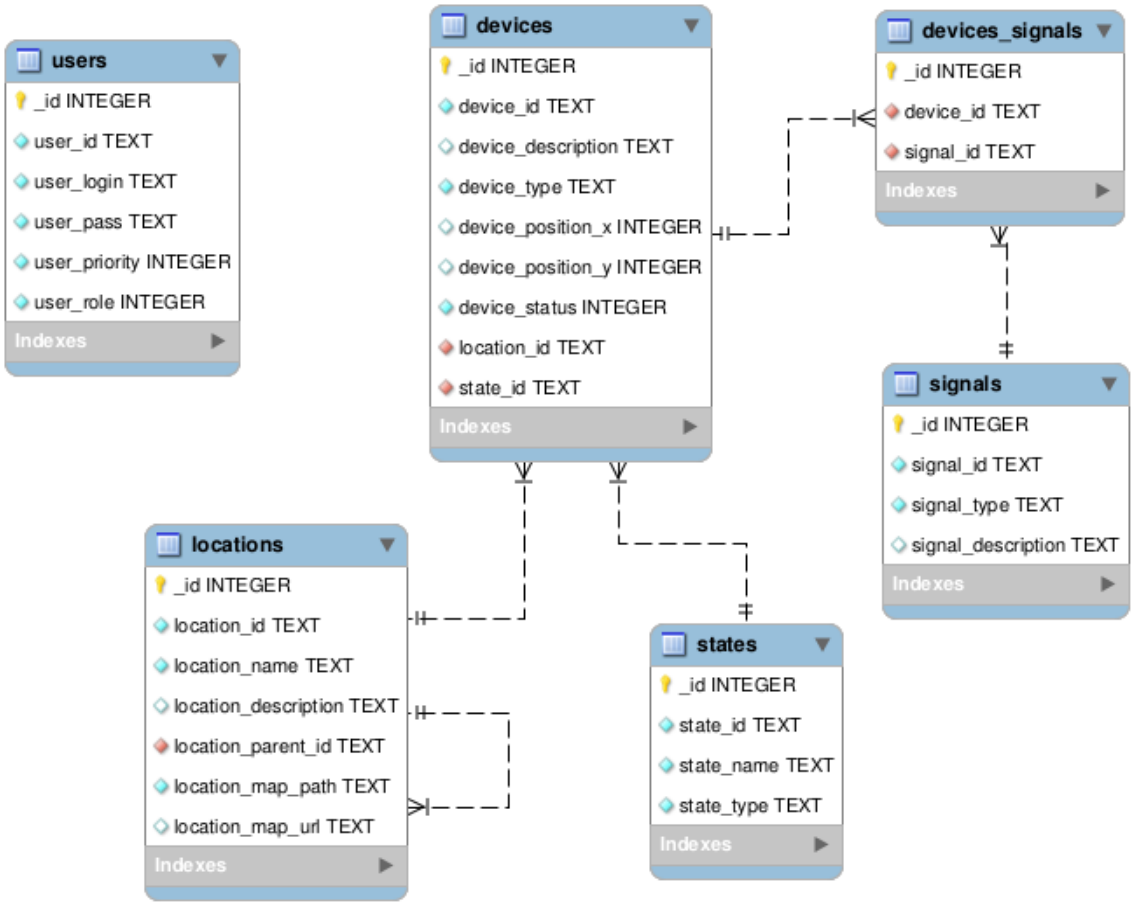
Helpery sdružují všechny univerzální třídy a rozhraní, které mohou být využity libovolnou vrstvou pro usnadnění práce, zamezení duplicitám a zvýšení přehlednosti kódu. Nejčastěji se jedná o statické třídy, z nichž nejvíce používané bývají například `StringHelper` pro zpracování řetězeců, `Logger` jako nástroj pro výpis a ukládání informačních i chybových zpráv nebo `ResourceHelper` pro mapování DTO atributů na obrázky, texty a další užitečné zdroje.

3.6 Datový model a úložiště

Datový model aplikace vychází z dat a struktur, které lze získat z RESTful API serveru *SmartBuildings* a současně zahrnuje pouze ta data, která mobilní aplikace reálně využije.

Datové úložiště bude realizováno pomocí relační databáze *SQLite 3*, kterou Android nativně integruje přímo do svého OS [28]. Ostatní alternativy, tedy zapisování do souborů nebo ukládání do *Shared Preferences* realizovaných jako XML, nepřipadaly v úvahu, protože by se aplikace musela starat o správný průběh transakcí[14], implementovat závislosti mezi jednotlivými datovými strukturami a v neposlední řadě také zajišťovat kaskádové aktualizace a mazání dat. I kdyby však aplikace všechny tyto požadavky sama vyřešila, velkým problémem by zůstávala pomalejší rychlost zpracování.

Datový model byl navržen a upravován v aplikaci *MySQL WorkBench 5.2*, která poskytuje kvalitní nástroje pro přípravu pokročilejších E-R modelů. Výsledný model, který lze vidět na obr. 3.4 je navíc rozšířen o základní datové typy databáze *SQLite 3*.



Obrázek 3.4: Datový model aplikace

[14]Tím je myšleno zajištění ACID vlastností, tedy: *atomicity*, *konzistence*, *izolovanosti* a *trvalosti*.

3.7 Grafický návrh uživatelského rozhraní

Návrh grafického rozhraní vychází z prototypu popsaného v podkapitole 3.3.2 *Návrh funkcionality, prototyp*. V průběhu návrhu grafiky byly zohledněny také:

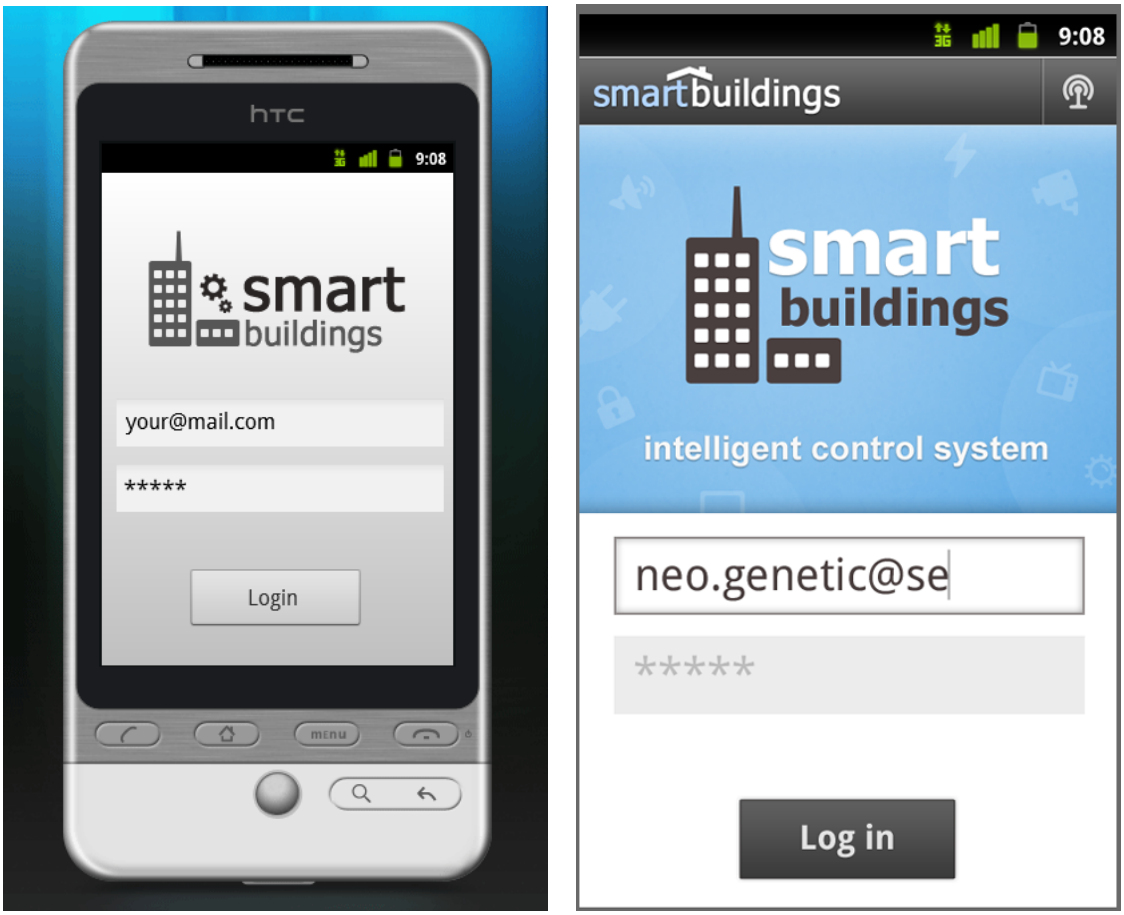
- **Výsledky testování prototypu**, převážně týkající se použitelnosti, UX a vizuálních doporučení.
- **Obecné zásady návrhu UI pro Android**, zejména návrhové vzory pro uživatelské rozhraní a pravidla vzhledu grafických prvků jako jsou ikonky, tlačítka, pozadí a další [4]. Respektování obecných zásad návrhu pro Android je důležité, protože se těmito pravidly řídí velké množství aplikací, takže jsou majitelé Androidu na hodně standardizovaných grafických ovládacích prvků navyklí. Pokud návrh mobilní aplikace zohlední řešení, které uživatelé už dávno zná a ví, jak daný prvek používat, stane se pro něj ovládání mnohem jednodušším.
- **Malé rozměry displeje mobilního zařízení**. Všechny grafické prvky proto musí být dostatečně velké a snadno čitelné.

Po dokončení návrhu vzhledu jsou jednotlivé grafické prvky „nařezány“ po obdélnících a vyexportovány ve formátu PNG s 8 bitovým alfa, aby byla zajištěna průhlednost. Export probíhá ve dvou kolech. V prvním dochází k exportování prvků v originální velikosti, což odpovídá displeji s vysokým rozlišením, v Androidu označovaným jako HDPI. V druhém kole je celá grafika nejdříve zmenšena na 66,67% původní velikosti[15] a poté exportována pro zařízení se středním rozlišením displeje MDPI. Mezi grafikou určenou pro HDPI a grafikou určenou pro MDPI Android automaticky přepne v průběhu instalace podle toho, na jaké zařízení se .apk balíček právě instaluje.

Na obr. 3.7 lze vidět porovnání přihlašovací obrazovky prototypu s finálním grafickým návrhem rozhraní, který z prototypu vychází. Pro návrh veškeré grafiky včetně uživatelského rozhraní byl použit grafický editor *Adobe Photoshop CS5*.

Jedinou výjimkou jsou 2D mapy podlaží nebo lokalit - ty jsou připravovány pomocí jednoduché webové služby *FloorPlanner* <http://www.floorplanner.com/>. Mapy navržené ve *FloorPlanneru* umožňují zachytit interiér a prostředí uvnitř budovy tak, jak přibližně doopravdy vypadá. Barevné, s pěknou grafikou, bez zbytečných kót a architektonických čar a navíc tvorbu mapy zvládne každý *běžný uživatel*. Návrh mapy pro 1. podlaží malého rodinného domku demonstruje obr. 3.6.

[15] Zmenšení odpovídá poměru mezi velikostmi displeje HDPI (240 DPI) a MDPI (160 DPI).



(a) Vzhled prototypu, scénář „přihlášení“ (b) Grafický návrh, scénář „přihlášení“

Obrázek 3.5: Porovnání prototypu s grafickým návrhem

Obrázek 3.6: Mapa 1. podlaží včetně zahrady navržená webovou službou *FloorPlanner* (zdroj: <http://www.floorplanner.com/>)

Kapitola 4

Realizace

Kapitola *Realizace* popisuje procesy a postupy, ze kterých jsem vycházel při implementaci mobilní aplikace. Jako první pojednává o výběru vhodných technologií a nástrojů, které mi pomohly vylepšit efektivitu vývoje a kvalitu kódu. V druhé části popisuji postup při implementaci, podrobně rozebírám diagram tříd a uvádím příklad komunikace modelu aplikace.

4.1 Výběr nástrojů a technologií

Výběr vhodných nástrojů a užitečného softwaru může vývojáři značně pomoci - v ideálním případě zkrátí čas potřebný k dokončení úkolu a vylepší kvalitu výsledné aplikace. Nástroje a techniky, které jsem při implementaci použil, vychází buď z mých vlastních zkušeností, nebo byly vybrány na základě doporučení komunity Android vývojářů.

4.1.1 Vývojové prostředí (IDE)

Asi nejdůležitějším nástrojem, který je k implementaci aplikace pro Android potřeba, je právě vývojové prostředí. V tomto případě byla volba přímočará, protože se Google rozhodl plně podporovat pouze jediné vývojové prostředí - *Eclipse*. Právě pro *Eclipse* je ve formě pluginu dostupná celá řada nástrojů, která se souhrnně nazývá *Android Development Tools* (ADT) [10].

Eclipse IDE tak vývojáři poskytuje například kompletní debugování aplikace na reálném nebo virtuálním zařízení, přehlednou konzoli zachycující logování v reálném čase, dále nabízí nástroje pro správu a profilování paměti, vizuální editor uživatelského rozhraní, statickou analýzu kódu nebo mnoho utilit užitečných při refaktorování kódu. Další funkce pro práci s Java kódem poskytuje sám *Eclipse*, takže třeba integrovaná dokumentace nebo nešeptávání metod pomáhají urychlit učící křivku, pokud pro Android vývojář píše poprvé.

4.1.2 Build skripty

Eclipse IDE s ADT pluginem ví, jakým způsobem provést build Android projektu, na kterém vývojář právě pracuje. Problém však nastává tehdy, pokud například chcete přeložený balíček digitálně podepsat, spouštět během buildu další závislé nástroje a programy nebo pokud se má překlad odstartovat automaticky. Některé věci *Eclipse* vůbec neumí, některé mu dělají problémy. Navíc, jestliže zdrojové kódy přenesete na jiný počítač, aplikaci tak jak je bez *Eclipsu* nesestavíte a nespustíte.

Z tohoto důvodu jsem se rozhodl používat build nástroj *Apache Ant*, který k sestavování aplikace využívá jednoduché, ale mocné příkazy zapsané ve formě XML skriptů [6]. Díky *Antu* jsem mohl realizovat například:

- Automatickou instalaci nebo odinstalaci aplikace na zařízení s Androidem, která jsou právě připojená k operačnímu systému.
- Generování kompletní dokumentace, výsledků statické analýzy a informačních dokumentů.
- Spouštění jednotkových testů před kompilací ostré aplikace.
- Kaskádovou kompilaci závislých projektů a pomocných knihoven.
- Generování hodnot do zdrojových kódů ještě před jejich překladem.
- Spouštění nástroje *ProGuard* pro optimalizaci kódu.

Obrovskou výhodou je, že *Eclipse* umí build skripty od *Antu* spouštět, takže v průběhu vývoje není potřeba přepínat mezi příkazovou řádkou a IDE.

4.1.3 Verzování

Verzování aplikace vývojáři pomáhá zálohovat odvedenou práci, sdílet ji s ostatními členy týmu a současně zaznamenat průběžné změny prováděné v čase. V případě, že později vývojář ztratí důležité soubory nebo v kódu udělá změny vedoucí k znefunkčnění části aplikace, která ještě před pár dny bez problémů fungovala, může se pomocí verzovacího systému „vrátit“ několik dní zpět a porovnat změny mezi aktuální a starší verzí, případně obnovit soubory, které právě ztratil. Nástroje pro verzování toho ale umí ještě mnohem více.

Pro své účely jsem si jako verzovací systém vybral *Git*, který lze používat jako decentralizovaný repozitář, takže je při lokálním verzování velmi rychlý [15]. Změny jsou „commitovány“ pouze při splnění těchto pravidel:

- Aplikaci lze zkompilovat pomocí build skriptů, lze ji spustit a je funkční.
- Verzovací systém ignoruje citlivé soubory s hesly a nastavením vlastního prostředí.
- Mezi soubory, které budou verzovány jsou zahrnuty i veškeré závislé knihovny, komponenty a související dokumentace.

Do centrálního SVN repozitáře projektu *SmartBuildings* jsou odesílány pouze plně funkční verze, určené pro předvedení nebo jako oficiální vydání.

4.1.4 Dokumentace

Dokumentace k implementaci je pomocí nástroje *Android JavaDoc* [21] generována z komentářů obsažených v kódu aplikace. Výsledný formát dokumentace je několik vzájemně provázaných HTML dokumentů, které může čtenář pohodlně procházet pomocí hypertextových odkazů a navigace. Tímto způsobem je zdokumentována většina jádra a všechny důležité nebo složitější části aplikace.

Pro lepší pochopitelnost kódu jsem se snažil používat jmenné konvence Javy a pojmenovávat metody i členy podle jejich účelu. Velká část kódu je proto samodokumentující.

4.1.5 Kvalita kódu

Pro zajištění vyšší kvality, čistoty a všeobecné přehlednosti kódu využívám kromě klasických utilit integrovaných do *Eclipse IDE* také následující externí nástroje:

- *FindBugs* [14] - Statická analýza Java kódu. Umí odhalit pravděpodobné chyby a neefektivní volání, které IDE nehlásí a kompilátor ignoruje.
- *Android Lint* [22] - Statická analýza těch částí kódu, které využívají Android SDK. *Android Lint* pokrývá situace, jenž *FindBugs* nedokáže detekovat, protože nezná principy vývoje v Androidu.
- *EMMA* [12] - Měření a statistiky pokrytí kódu jednotkovými testy (angl. „unit testy“). Má podporu pro *Ant*, takže jsou reporty s informacemi o kódu generovány automaticky.

- *ProGuard* [26] - Optimalizuje a zmenšuje výslednou aplikaci prekompilačními úpravami v kódu. Odstraňuje nepoužívané části kódu, nahrazuje jména metod a dalších členů krátkými řetězci a provádí řadu dalších úprav. Takto modifikovanou aplikaci je navíc velmi obtížné zneužít pomocí metod reverzního inženýrství.

4.2 Implementace

4.2.1 Postup při implementaci

Před začátkem programování a realizací cílové aplikace jsem si nejdříve vyhledal, zprovoznil a nastavil nástroje zmíněné v podkapitole 4.1 *Výběr nástrojů a technologií*. V průběhu implementace vycházím z návrhů a modelů připravených během *Analýzy* (viz kapitola 3). Postup vývoje je přibližně následující:

1. Na základě návrhu architektury a komunikace mezi jednotlivými vrstvami (viz podkapitola 3.5) jsem strukturu aplikace detailněji rozepsal a navrhl *zjednodušený diagram tříd*, který lze vidět na obr. 4.1 a blíže jej popisuje podkapitola 4.2.2 *Popis komunikace tříd*.

2. Podle hrubé struktury *diagramu tříd* jsem začal implementovat tělo aplikace směrem od datové vrstvy[1], přes řízení a logiku aplikace až po prezentační vrstvu. Cílem je zprovoznit nejmenší možnou verzi aplikace, která bude umět propagovat informaci z prezentační vrstvy do datové včetně odeslání na server a opačně.

3. Ve chvíli, kdy vrstvy aplikace zvládají alespoň základní vzájemnou komunikaci, implementuji v XML grafické rozhraní prvního scénáře (např. „přihlašování“) podle grafického návrhu (viz 3.7). Pak postupuji s implementací od prezentační vrstvy směrem po datovou a snažím se zprovoznit daný scénář. Při tomto procesu dochází k přirozenému vylepšování jednotlivých vrstev, protože jsou upraveny pro potřeby scénářů. Tento krok je iterován tolikrát, kolik je potřeba implementovat scénářů. V průběhu celé realizace navíc dochází k refaktorování kódu.

4. Jako poslední je funkčnost aplikace testována, což do detailu rozebírá kapitola 5 *Testování*, část 5.2 *Testování uživatelského rozhraní a funkcionality*.

Během implementace se může stát, že vymyslím lepší řešení daného problému, než jaké je navrženo nebo dokonce implementováno. V takovém případě dojde k aktualizaci modelů a rekurzivním úpravám v kódu.

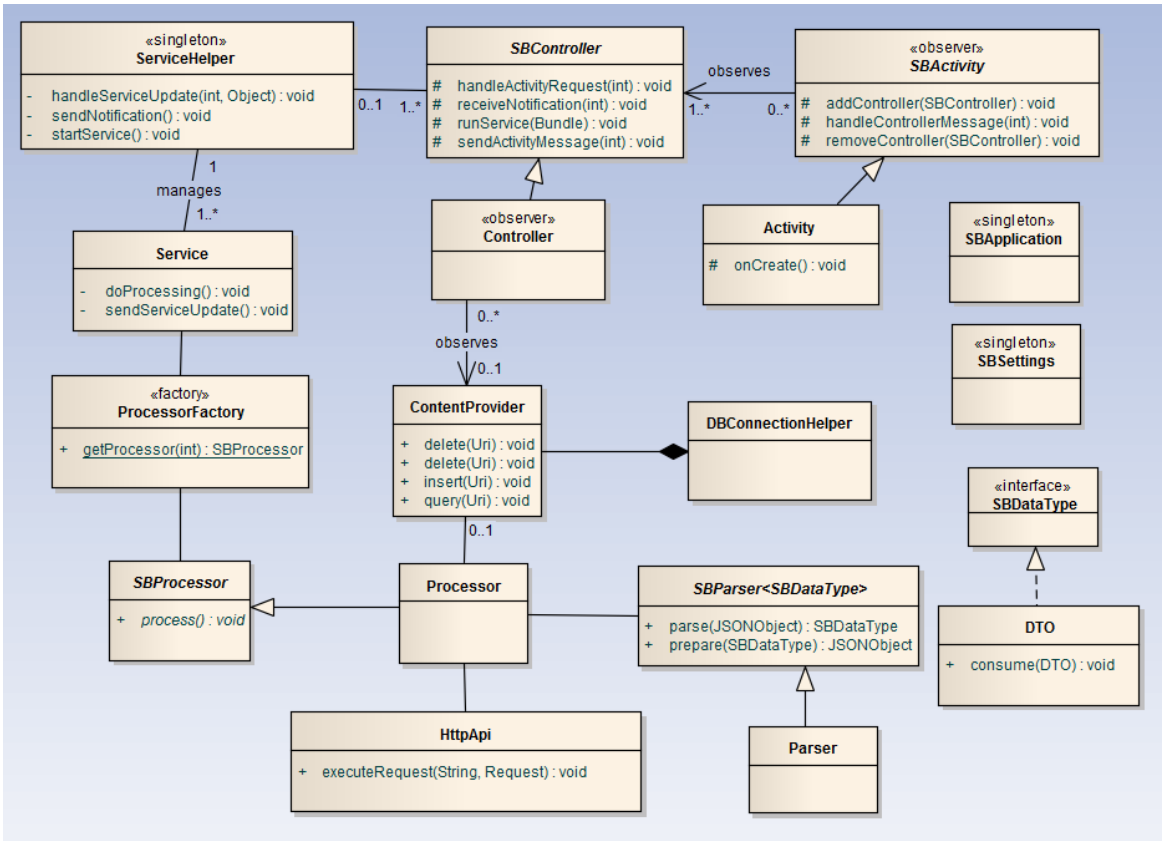
Po dokončení realizace definovaného počtu scénářů jsou nejdůležitější části kódu pečlivě okomentovány a po odpovídajícím otestování je aplikace označena příslušnou verzí[2] a je oficiálně vydána.

[1] Databáze a obecně model datové vrstvy byl realizován podle datového modelu z obr. 3.4.

[2] Číslo verze je určováno podle počtu a důležitosti změn mezi předchozí a současnou aplikací.

4.2.2 Popis komunikace tříd

Diagram tříd z obr. 4.1 detailněji přibližuje komunikaci mezi jednotlivými vrstvami, konkrétně mezi jejich klíčovými třídami. Abych zachoval jednoduchost a čitelnost diagramu, zredukoval jsem ho na důležité minimum. Celá struktura tříd aplikace je však ve skutečnosti mnohem složitější.



Obrázek 4.1: Zjednodušený diagram tříd

Vzájemný vztah tříd nejlépe vysvětlím popisem propagace informací z prezentační vrstvy do datové vrstvy.

Pro zobrazení uživatelského rozhraní na displeji zařízení systém nejprve vytvoří instanci příslušné *Aktivity* - například třídy `DeviceDetailActivity` představující GUI obrazovky s detailem zařízení[3]. Abstraktní třída `SBActivity`, od které `DeviceDetailActivity` dědí, slouží k definování veškerého chování, které by se dokola opakovalo ve většině *Aktivit*. To se týká třeba registrace *Aktivity* ke *Controlleru*, zobrazení akční lišty nebo zpracování obecné chybové hlášky. Klíčovou funkcionalitu může dědic vždy přetížit a obecně tenhle přístup přispívá k principu DRY! („Don't repeat yourself!“).

[3]V diagramu tříd na obr. 4.1 jsou třídy *Activity*, *Controller*, *Service*, *Processor*, *Parser* a *SBDTO* zobecněny. Reálná aplikace ale obsahuje jejich konkrétní implementace, tedy například třídy `DeviceController`, `DeviceProcessor`, `DeviceParser`, `DeviceDTO` a tak dále.

Okamžitě při vzniku `DeviceDetailActivity` systém sám zavolá metodu `onCreate`. V té dojde k vytvoření *Controllerů*, ke kterým se `DeviceDetailActivity` zaregistruje jako *Listener*, dále je inicializováno GUI a jsou nastaveny původní hodnoty. Nakonec může být spuštěn jakýkoliv implicitní příkaz, například požadavek na aktualizaci informací o daném zařízení. `DeviceDetailActivity` proto zavolá metodu `handleActivityRequest( int requestId )` na instanci třídy `DeviceController`.

`DeviceController` dědí od obecnější abstraktní třídy `SBController`, která opět řeší mnoho opakujících se situací. Podle ID požadavku, který `DeviceController` od `DeviceDetailActivity` obdržel je zavolána některá ze soukromých metod a ta se o celý požadavek postará. Jestliže se jedná o časově a výpočetně nenáročnou operaci, metoda připraví data pro `DeviceDetailActivity` a odešle všem registrovaným *Aktivitám* jednu nebo více zpráv přes notifikační metodu `sendActivityMessage ( int messageId )`. `DeviceDetailActivity` by nakonec tyto zprávy přijala, zpracovala a zobrazila na displeji telefonu odpovídající změny. Ve chvíli, kdy však musí `DeviceController` provést časově náročnější operaci, jako například synchronizaci informací o zařízení, komunikace je mnohem složitější.

V metodě, která má na starosti zpracování požadavku vyžadujícího síťovou komunikaci je nejdříve spuštěno nové, nezávislé vlákno. V tomto vlákně jsou připravena data, která bude *Service* potřebovat pro synchronizaci informací. Data serializovaná v instanci třídy `Bundle` jsou metodou `runService ( Bundle bundle )` předána *Singletonu* `ServiceHelper`.

Třída `ServiceHelper` musí být implementována jako *ThreadSafe*, protože jde o *Singleton* a tak mohou k této jediné instanci přistupovat ostatní vlákna. Bez zajištění vhodné synchronizace by mohlo docházet k poškození dat nebo k deadlocku[4]. `ServiceHelper` rozbalí a deserializuje data, která mu předal `DeviceController` ve formě instance `Bundle`, vygeneruje unikátní ID požadavku, zvolí vhodný *Service*, připraví pro něj další důležitá data včetně přiděleného ID a *Service* metodou `startService ()` odstartuje. Nakonec `ServiceHelper` pošle `DeviceControlleru` notifikaci, že se požadavek s daným ID začal zpracovávat, načež `DeviceController` pošle žádost o zobrazení indikátoru načítání v `DeviceDetailActivity`.

V tomto konkrétním případě je k dokončení požadavku potřeba synchronizační služba, takže `ServiceHelper` odstartuje instanci třídy `SyncService`. `SyncService` dědí od nativní třídy Android frameworku `IntentService`. Android OS takovou instanci spouští mimo hlavní kontext aplikace a ta běží na pozadí ve stejném vlákně, které ji spustilo, což se hodí například při stahování dat prostřednictvím sítě. `SyncService` nejprve deserializuje data získaná od `ServiceHelper` a část jich předá *Factory* třídě `ProcessorFactory`. `ProcessorFactory` se rozhodne, který *Processor* dokáže zpracovat daný požadavek a ten vrátí při volání metody `getProcessor ( int actionId )`. Tímto získá `SyncService` správný *Processor*, na kterém zavolá metodu `process ()` a ta se postará o komunikaci se serverem a synchronizaci s lokální databází. Pokud někdy dál dojde při zpracování k chybě, bude odchycena v `SyncService`, který pošle *Singletonu* `ServiceHelper` informaci, že požadavek s jeho ID skončil chybou. `ServiceHelper` dále analogicky notifikuje `DeviceController`, který pošle požadavek na aktualizaci GUI instanci `DeviceDetailActivity`.

Zpracování požadavku dále pokračuje ve správném *Processoru*, který dědí obecnou třídu `SBProcessor`. V této situaci požadavek zpracuje instance třídy `DeviceProcessor`, která inicializuje `HttpApi` a zavolá jeho metodu `executeRequest ( String url, Request request )`.

[4] K zvládnutí problematiky vývoje aplikace s více vlákny mi pomohla perfektní kniha *Java Concurrency in Practice* [1] od autorů, kteří se podíleli na vývoji Java concurrency modelu.

`HttpApi` připraví klienta, zkonstruuje HTTP požadavek a ten odešle na URL serveru, který by měl odpovědět aktuálním stavem dotazovaného zařízení, a to ve formě vráceného JSON objektu. `DeviceProcessor` předá JSON *Parseru*, který dokáže JSON podle specifikace SB serveru převést na odpovídající Data Transfer Objekt (DTO). Správnou konverzi provede instance `DeviceParseru` a `DeviceProcessor` tak získá `DeviceDTO`, se kterým mohou ostatní vrstvy a objekty snadno pracovat.

V poslední fázi zpracování je instance `DeviceDTO` předána `ContentProvideru` a jeho metodě `insert ( Uri uri )`, která DTO zpracuje a vloží do relační databáze. Jestliže dojde ke změně dat v databázi, `ContentProvider` notifikuje *Observer* `DeviceController` a ten aktualizuje `DeviceDTO`, kterou spolusdílí s `DeviceDetailActivity`. Během tohoto zpracování v `DeviceProcessor` končí, takže `SyncService` posílá na `ServiceHelper` informaci, že byl požadavek s daným ID dokončen. To vede k tomu, že `ServiceHelper` notifikuje `DeviceController`, který konečně pošle jednu nebo více žádostí `o` aktualizaci uživatelského rozhraní instanci `DeviceDetailActivity`.

Tento model komunikace tvoří jádro aplikace.

4.2.3 Použití návrhových vzorů

V průběhu implementace jsem použil hned několik návrhových vzorů, které mi pomohly zjednodušit návrh a vylepšit čistotu kódu:

- **Singleton** - Zajištění inicializace maximálně jedné instance. *Singleton* jsem využil například pro realizaci třídy `ServiceHelper`, která se stará o správu služeb a tak se nesmí stát, aby některé z vláken přistupovalo k jiné instanci nebo pro třídu `SBApp`, jejíž instance představuje spuštěnou aplikaci.
- **Observer** - Asi nejpoužívanější návrhový vzor. Funguje tak, že se libovolné množství posluchačů zaregistruje k přijímání notifikací od určitého objektu a ten pak posílá hromadně zprávy všem, kteří se k němu registrovali. Tímto způsobem funguje posílání zpráv mezi *Controllery* a *Activitami*, odchytávání zpráv UI vyvolaných akcí uživatele, komunikací nebo pozorování změn v databázi *Controllerem*.
- **Factory** - Třída, která podle konkrétních podmínek vrací určitý typ objektu typovaného jako rozhraní nebo supertřída. *Factory* v mé aplikaci řeší vracení správného *Processoru*, který po zavolání metody `process ()` zpracuje odpovídající data.
- **Data Transfer Object** - Z Javy známé také jako *POJO* (Plain Old Java Object). Jde o objekt, který slouží jako kontejner pro atributy a používá se pro transport informací napříč vrstvami. Kromě *accessorů* a *mutátorů*[5] většinou *DTO* nemívá žádné další metody. Skladba *DTO* a jejich atributů často do velké míry odráží strukturu datové vrstvy. V aplikaci jsou jako *DTO* implementovány například třídy `DeviceDTO`, `UserDTO`, `SignalDTO` a mnoho dalších.

[5] Accessor a mutator je obecnější označení pro *getter* a *setter*.

Kapitola 5

Testování

Testování je důležitou součástí každého vývoje. Pomáhá upozornit na špatná rozhodnutí při návrhu a později odhaluje nedostatky a chyby v aplikaci. Tato kapitola zmiňuje a popisuje hned několik testovacích metod, které jsem v průběhu vývoje absolvoval. Výsledky testů pak zcela nepochybně přispěly ke zvýšení spolehlivosti, použitelnosti a kvality aplikace.

5.1 Testování prototypu aplikace

Jako první byla aplikace testována ve fázi *prototypu* (viz 3.3). Cílem bylo zjistit, jestli je navržená funkcionalita dostatečně smysluplná a rozmístění prvků intuitivní. Jako nejvhodnější k dosažení vytyčeného cíle připadalo v úvahu testování formou *heuristické analýzy* (angl. „heuristic evaluation“). Jde o rychlou a málo nákladnou metodu, při které prototyp testuje několik nezávislých odborníků na použitelnost z daného oboru. Jejich zkušenosti výrazně přispějí k jednoduchosti a čistotě výsledné aplikace, protože jsem se s vývojem něčeho podobného doposud nesetkal a tak bude prototyp jistě obsahovat spoustu slabin a špatných řešení.

5.1.1 Příprava testů

Jako první jsem připravil sérii otázek, které budou expertům kladeny pravidelně u každé obrazovky prototypu:

1. *Ví uživatel, jaký je aktuální stav systému a kde se zrovna nachází?*
2. *Pozná uživatel, co se systémem může dělat dál?*
3. *Odpovídají ovládací prvky kontextu, ve kterém bude aplikace používána - tzn. lze aplikaci pohodlně používat jednou rukou, při řízení auta, na žebříku ve sklepě a v dalších náročných situacích?*
4. *Je dostatečně srozumitelné, jakou funkci mají veškerá tlačítka, která systém poskytuje?*
5. *Jsou klíčové změny systému uživateli oznamovány jasně a zřetelně?*
6. *Zabraňuje systém ve všech kritických předčasnému a nechtěnému dokončení či spuštění kritické operace?*
7. *Je v systému v současnosti nějaký nadbytečný grafický nebo ovládací prvek, případně funkce, která by tam být nemusela?*

K provedení heuristické analýzy potřebuji minimálně dva odborníky. Hledal jsem jednoho člověka se zkušenostmi z řízení inteligentních budov, druhého z oblasti vývoje aplikací pro Android. Nakonec se mi oba podařilo najít mezi studenty fakulty informačních technologií na ČVUT.

5.1.2 Průběh

Testování probíhalo po internetu formou online komunikace s oběma experty. Před samotným začátkem byl každý z nich poučen o účelu a cíli testování a byly jim vysvětleny základní principy heuristické analýzy. Dále bylo oběma vysvětleno, jakým způsobem mají pracovat s HTML prototypem a jak přesně zaznamenávat své názory a odborné hodnocení. Současně jsem upozornil na to, že při testování nejde o zkoušku jejich znalostí, ale pouze o pomoc s odhalením nedostatků v návrhu mé aplikace.

Celý test trval přibližně 60 - 75 minut a až na pár technických dotazů proběhl naprosto bez problémů.

5.1.3 Vyhodnocení

Podle výsledků a odpovědí obou odborníků (viz příloha B) je patrné, že má návrh 2 hlavní slabiny - *nedostačující navigaci* a *málo intuitivní ovládací prvky*. Obrovskou výhodou je, že bylo testování provedeno hned u prototypu, takže mohu výsledky testů zohlednit při *tvorbě grafického rozhraní* (viz 3.7) a změny nebudou tak časově náročné, jako by byly u kompletně zpracovaného rozhraní dokončené aplikace.

Při návrhu UI vezmu v potaz následující změny:

- Zjednodušení zobrazení a pohybu po mapě, zejména přepracování šipek. Nabídka seznamu pater a lokalit.
- Používat nativní Android menu místo spodního panelu.
- Přidat do horní části displeje „Action bar“, tedy panel s nejpoužívanějšími ovládacími prvky a klíčovou navigací (tlačítko „hlavní menu“, název aktuální obrazovky).
- Navigační tlačítka „Zpět“ a „Menu“ jsou v Androidu u každého zařízení řešena hardwarovými tlačítky, které umí každý uživatel intuitivně používat. Nesuplovat tuto přirozenou funkcionalitu grafickými ovládacími prvky.
- Zjednodušení seznamů s informacemi.
- Vylepšit texty a popisky, které jsou málo deskriptivní. Přidat popisky tam, kde úplně chybí.
- Odstranění tlačítka „Exit“ z menu.
- Předělat tlačítka tak, aby vypadala opravdu jako tlačítka. Zvýšit použitelnost, aby byla tlačítka dostatečně velká, srozumitelná.

5.2 Testování uživatelského rozhraní a funkcionality

Testování uživatelského rozhraní a funkčnosti celé aplikace (tzv. „blackbox testing“) je absolvováno vždy těsně před vydáním oficiální verze. Jde o manuální test všech scénářů a možných situací prováděný na dvou lehce rozdílných fyzických zařízeních.

5.2.1 Testovací zařízení

První zařízení je *HTC Rhyme* s 3,7 palců dotykovým displejem (rozlišení HDPI), 768MB RAM, 1GHz procesorem a Androidem 2.3.3 včetně nadstavby systému *HTC Sense*.

Druhý chytrý telefon je mnohem starší a slabší *HTC HD2* s 4 palcovým dotykovým displejem (rozlišení XHDPI), pouze 448 MB RAM, 1GHz procesorem a Androidem 2.3, který toto zařízení oficiálně ani nepodporuje, takže mu byl pozměněn firmware[1].

Testování na dvou různých chytrých telefonech pomáhá odhalit velké množství chyb souvisejících se správným zobrazením grafiky a uživatelského rozhraní. V méně častých případech pak dochází i k neočekávanému chování aplikace a tím i nalezení nových chyb.

[1] HTC HD2 bylo dostupné pouze s OS Windows Mobile 6.5.

5.2.2 Metodika

Při testování rozhraní a funkcionality jsou sledovány převážně následující parametry:

- Daná funkce nebo tlačítko se chová tak, jak se má chovat.
- Jsou zobrazovány korektní a očekávané informace.
- Aplikace správně přizpůsobuje své grafické rozhraní rozměrům a rozvržení displeje.
- Jsou vhodně ošetřeny všechny situace a možné chyby, které mohou při používání nastat (např. při zadání krajních hodnot).
- Uživatelské rozhraní je responzivní a práce s aplikací uživatele nezdržuje tak dlouho, aby to uživatele obtěžovalo.

Nalezené chyby jsou opravovány ihned a testování daného scénáře je poté opakováno.

5.3 Whitebox testing

Testování, které si klade za cíl odhalit chyby uvnitř logiky aplikace.

5.3.1 Jednotkové testy

Jednotkovými testy (angl. „unit tests“) je z většiny pokryto jádro aplikace a centrální logika, která se stará o propagaci dat z datové do prezentační vrstvy a zpět. Pokud při rozšiřování aplikace dojde ke vzniku nových chyb uvnitř jádra, testy je okamžitě pomohou odhalit a chyby jsou opraveny ještě předtím, než způsobí pád nebo neočekávané chování aplikace.

Části aplikace, u kterých dochází k častým změnám struktury a logiky testovány nejsou, protože by bylo pokrytí testy kontraproduktivní. Toto se týká například komunikace uvnitř prezentační vrstvy, kdy některé třídy jsou během vývoje kompletně od základu přepsány nebo z větší části pozměněny, protože dochází k průběžnému vylepšování celé aplikace. Dále nejsou testovány části systému, u kterých je vznik chyby velmi nepravděpodobný nebo chyba neohrožuje běh aplikace - to zahrnuje některé *Helpery*, *Singletony* nebo *DTO*.

100% úspěšnost všech testů je automatickou podmínkou při buildu oficiální verze aplikace. Pokud testy neprojdou, aplikace není přeložena a nelze ji veřejně distribuovat.

5.3.2 Logování

Vlastní implementace logování realizovaná třídou `Logger` pomáhá určit, kde chyba přibližně vzniká a s jakými hodnotami v dané chvíli třídy a metody pracovaly. Pokud k chybě dojde na straně uživatele, log pomáhá k simulaci, odchycení a opravě nahlášené chyby. Logovány jsou všechny důležité a složitější procesy v systému.

Kapitola 6

Závěr

6.1 Splnění cílů

Původním cílem bylo vytvořit rozsáhlou aplikaci s celou řadou funkcí, která by byla plnohodnotným rozšířením systému řízení inteligentních budov *SmartBuildings*. Moje původní představy o náročnosti takové aplikace byly značně zkreslené a naivní, k čemuž ještě přispěla má nezkušenost s vývojem pro chytré telefony a také fakt, že jsem se s automatizací budov setkal poprvé v životě.

První specifikace požadavků nezohledňovala faktory, které jsou při používání mobilní aplikace klíčové. To jsem si uvědomil až ve fázi implementace první verze aplikace, kdy jsem se poprvé dostal do vážných problémů pramenících ze špatného návrhu a nevhodně zvolené architektury systému. Při realizaci se projevily i vážné nedostatky plynoucí z návrhu uživatelského rozhraní, ke kterému jsem kvůli chybějícím zkušenostem přistupoval jako k návrhu rozhraní pro web. V průběhu vývoje jsem také několikrát zjistil, že je stanovený požadavek nesmysluplný, protože by příliš komplikoval ovládání a tím pádem výrazně snižoval použitelnost aplikace nebo protože SB server ve skutečnosti fungoval úplně jinak, než jsem si původně představoval.

Celou aplikaci jsem proto několikrát od základu předělal - došlo k přepracování nároků na funkcionalitu, díky přípravě prototypu a důkladnému testování byla změněna grafika rozhraní a identity a v neposlední řadě byl značně zjednodušen návrh a rovněž přepsáno celé jádro včetně logiky.

Výsledný produkt sice není klientem s hromadou sofistikovaných konfiguračních funkcí, zato jsem se ale snažil vytvořit elegantní a co možná nejjednodušší doplněk k systému *SmartBuildings*, který může začít okamžitě aktivně používat kdokoliv z více než čtvrt miliardy Android uživatelů. Jeden z hlavních cílů aplikace - přinést svým uživatelům větší pohodlí - zůstal zachován a aplikaci je možné využívat i v praxi.

6.2 Doporučení, pokračování vývoje

Tato práce i výsledná aplikace určují směr, kterým by se měl další vývoj rozšíření pro chytré telefony nadále ubírat. Do budoucna by měla být zachována hlavně jednoduchost, vzhled a struktura aplikace, což jsou věci, jejichž zdokonalování jsem věnoval celkově nejvíce času a péče. Spolu s mobilní aplikací bude pochopitelně nutné rozšířit i RESTful API rozhraní serveru.

Na úplný závěr bych rád uvedl několik tipů a námětů na vylepšení aplikace:

- Streamování zvuku a videa z kamer a dalších audiovizuálních zařízení do chytrého telefonu.
- Větší provázanost aplikace se systémem a chytrým telefonem obecně - například spuštění mobilního alarmu, pokud je spuštěn domovní alarm.
- Přidání jednoduchých akcí do dávkových operací, které může uživatel spouštět manuálně, na základě času nebo podmínky.
- Verze pro tablety, která bude umožňovat složitější konfiguraci systému jako je například určování podmínek, definice akcí, tvorba profilů.

Literatura

[1] BRIAN GOETZ, T. P. et al. *Java Concurrency in Practice*. Addison Wesley Professional, 1st edition, 2006.

[2] MEDNIEKS, Z. et al. Programming Android. 2011, 1, s. 331–356.

[3] PAGE, L. 2012 Update from the CEO — Investor Relations. <http://investor.google.com/corporate/2012/ceo-letter.html>, stav z 10. 5. 2012.

[4] web:android-ui. Android Design — Design Principles. <http://developer.android.com/design/get-started/principles.html>, stav z 15. 5. 2012.

[5] web:androidverze. Android Developers — Platform Versions. <http://developer.android.com/resources/dashboard/platform-versions.html>, stav z 10. 5. 2012.

[6] web:ant. Apache Ant. <http://ant.apache.org/>, stav z 18. 5. 2012.

[7] web:apogee. APOGEE — Building automation system. <http://www.buildingtechnologies.siemens.com/bt/us/Products__and_Systems1/Building_automation_and_control/APOGEE_Building_Automation_System/Pages/apogee_building_automation_system.aspx>, stav ze 7. 5. 2012.

[8] web:bticino. My Home BTicino — Home automation system. <https://www.myhome-bticino.it/dispo/login/login.page>, stav z 8. 5. 2012.

[9] web:dobjanschi. Google I/0 2010 — Developing Android REST client applications. <http://www.google.com/events/io/2010/sessions/developing-RESTful-android-apps.html>, stav z 12. 5. 2012.

[10] web:eclipse-adt. Android Developers — ADT Plugin for Eclipse. <http://developer.android.com/sdk/eclipse-adt.html>, stav z 18. 5. 2012.

[11] web:elig. GitHub — Eli-G — Zdrojové kódy. <https://github.com/necronet/Eli-G>, stav z 12. 5. 2012.

[12] web:emma. EMMA: a free Java code coverage tool. <http://emma.sourceforge.net/>, stav z 18. 5. 2012.

[13] web:emulator. Android Developers — Using the Android Emulator — Limitations. <http://developer.android.com/guide/developing/devices/emulator.html>, stav z 24. 5. 2012.

[14] web:findbugs. FindBugs — Find Bugs in Java Programs. <http://findbugs.sourceforge.net/>, stav z 18. 5. 2012.

[15] web:git. Git — Homepage. <http://git-scm.com/>, stav z 18. 5. 2012.

[16] web:googleio. Google Project Hosting — Google I/0 App for Android — Zdrojové kódy. <http://code.google.com/p/iosched/source/browse/>, stav z 12. 5. 2012.

[17] web:htcone. HTC Smartphones — HTC One X Product Overview. <http://www.htc.com/www/smartphones/htc-one-x/>, stav z 11. 5. 2012.

[18] web:ibclanek. Inteligentní budovy. <http://www.inteligentni-budovy.cz/>, stav z 9. 5. 2012.

[19] web:ibteorie. iHNed.cz — Inteligentní budovy, technologie a systémy. <http://tech.ihned.cz/c1-43941000-inteligentni-budovy-technologie-a-systemy>, stav z 9. 5. 2012.

[20] web:ios. TechCrunch — Tim Cook Talks iOS Device Stats. <http://techcrunch.com/2012/03/07/tim-cook-talks-ios-device-stats-315-million-sold-62-million-in-q4-alone/>, stav z 10. 5. 2012.

[21] web:javadoc. Android JavaDoc. <http://www.androidjavadoc.com/>, stav z 18. 5. 2012.

[22] web:lint. Android Tools Project Site — Android Lint. <http://tools.android.com/tips/lint>, stav z 18. 5. 2012.

[23] web:loxone. Loxone — Domácí automatizace — Kontakty. <http://www.loxone.com/Pages/cz/service/Support-Kontakt/Kontakt-CSY.aspx>, stav z 8. 5. 2012.

[24] web:mindjet. MindManager by Mindjet — Android and iOS Mobile Apps. <http://www.mindjet.com/android>, stav z 17. 5. 2012.

[25] web:pencil. Pencil project. <http://pencil.evolus.vn/en-US/Home.aspx>, stav z 10. 5. 2012.

[26] web:proguard. ProGuard. <http://proguard.sourceforge.net/>, stav z 18. 5. 2012.

[27] web:responzivita. Android developers blog — Improving App Quality — Improve UI responsiveness. <http://android-developers.blogspot.com/2010/10/improving-app-quality.html>, stav z 9. 5. 2012.

[28] web:sqlite. Android Developers — Data Storage — Using Databases. <http://developer.android.com/guide/topics/data/data-storage.html>, stav z 17. 5. 2012.

[29] web:virtualbox. VirtualBox Manual — First steps — Import and Export. <http://www.virtualbox.org/manual/ch01.html>, stav z 24. 5. 2012.

[30] web:windows. TechNet Blogs — Windows Phone 7.5 wins interaction award. <http://blogs.technet.com/b/next/archive/2012/02/05/windows-phone-7-5-wins-interaction-award.aspx>, stav z 10. 5. 2012.

[31] ŠVEC, P. *SmartBuildings — rozšíření logiky jádra systému: Bakalářská práce*. ČVUT v Praze, Fakulta informačních technologií, 2012.

Příloha A

Seznam použitých zkratek

ADT Android Development Tools - Nástroje pro vývoj aplikací pro OS Android dostupné přes Eclipse IDE.

API Application programming interface.

AVD Android Virtual Device - Emulátor virtuálního zařízení s Androidem.

BAS Building automation system - Systém automatizace budov.

BMS Building management system - Systém pro správu a řízení budov.

CRUD Create, Read, Update, Delete - Soubor operací označující manipulaci s daty na datové vrstvě.

ČVUT České vysoké učení technické

DB Databáze

DPI Dots per inch - Jednotka určující počet pixelů na jeden palec reálné velikosti

DTO Data Transfer Object - Návrhový vzor, který používá objekt pouze pro přenos informací mezi vrstvami

DVM Dalvik Virtual Machine - Virtuální stroj podobný JVM, který běží na OS Android a slouží k provozu aplikací

E-R Entity-relationship - Vztah mezi relacemi, často se používá k zachycení vztahu mezi prvky relačního modelu

FIT Fakulta informačních technologií

GUI Graphic user interface - Grafické uživatelské rozhraní

HAS Home automation system - Systém automatizace domů a domácností

HDPI High density - Označení pro zobrazení s vysokým DPI (240)

HTTP Hyper Text Transfer Protocol - Nejpoužívanější protokol pro výměnu informací na internetu

HVAC Heating, ventilation and air conditioning - Systém automaticky zajišťující vhodné enviromentální podmínky uvnitř budovy

HW Hardware

IB Inteligentní budovy

ID Identifikátor

IDE Integrated development environment - Vývojové prostředí poskytující pokročilé nástroje užitečné pro efektivnější vývoj

IT Informační technologie

IP Internet Protocol - Unikátní identifikátor představující adresu rozhraní v síti

JSON JavaScript Object Notation - Standard pro zápis a výměnu dat na webu

JVM Java Virtual Machine - Virtuální stroj pro spouštění aplikací v Javě

MDPI Medium density - Označení pro zobrazení se středním DPI (160), v Androidu je tato Density výchozí („baseline“)

MVC Model-View-Controller - Třívrstvá softwarová architektura, která od sebe striktně odděluje data, řídící logiku a uživatelské rozhraní

OS Operační systém

PNG Portable Network Graphics - Standardizovaný grafický formát s bezztrátovou kompresí

SB SmartBuildings - Název systému pro správu a řízení inteligentních budov

SDK Software development kit - Sada softwarových nástrojů vhodná pro vývoj specifického typu aplikací

SVN Subversion - Typ verzovacího systému

SW Software

UML Unified modeling language - Modelovací jazyk

RAM Random access memory - Typ rychlé paměti, která přichází o svá data po odpojení napájení

REST Representational State Transfer - Softwarová architektura rozhraní pro komunikaci mezi distribuovanými systémy

UI User interface - Uživatelské rozhraní

UC Use case - Případy užití aplikace a zachycení interakce mezi uživatelem a systémem

URL Uniform Resource Locator - Textový řetězec (adresa) jednoznačně identifikující umístění zdroje na internetu

URI Uniform Resource Identifier - Textový řetězec (adresa) specifikující umístění zdroje na internetu, častokrát obecnější jako URL

UX User experience - Celkový dojem uživatele z interakce s aplikací

WF Wireframes - Drátěný model aplikace zachycující přibližný obsah, vzhled a funkcionalitu

XHDPI Extremely high density - Označení pro zobrazení s velmi vysokým DPI (320)

XML Extensible Markup Language - Obecný, univerzální značkovací jazyk často používaný pro výměnu dat

XNA Framework určený pro vývoj her a graficky náročných aplikací v .NET jazycích

Příloha B

Heuristická analýza - výsledky

Výsledky testování prototypu (viz podkapitola 3.3) metodou heuristické analýzy (viz podkapitola 5.1). Odpovědi obou odborníků, Pavla V. a Davida K., jsou zde uvedeny přesně tak, jak je oba testeři zaznemenali u každé ze zkoumaných otázek a kromě opravy gramatiky nebyly změněny.

B.1 Pavel V., vývojář aplikací pro Android

„Vycházím ze zvyklostí a doporučení (design patterns), které jsou zažité pro aktuálně používanou verzi OS Android, tedy 2.3 Gingerbread a 3.0 Honeycomb“.

1. Ví uživatel, jaký je aktuální stav systému a kde se zrovna nachází?

- *Po přihlášení* - Je zvykem v horní liště zobrazovat název aktivity. Současné všudypřítomné logo bych nahradil názvem obrazovky. Toto se týká všech obrazovek, nebudu to tedy vícekrát vypisovat.

2. Pozná uživatel, co se systémem může dělat dál?

- *Přihlášení* - Chybí výzva k akci, například „Please login“.
- *Přihlášení* - Použití samotných placeholderů je sporné, já bych přidal i samostatné popisky políček: „E-mail“ a „Password“.

3. Odpovídají ovládací prvky kontextu, ve kterém bude aplikace používána - tzn. lze aplikaci pohodlně používat jednou rukou, při řízení auta, na žebříku ve sklepě a v dalších náročných situacích?

- *Seznam zařízení* - Šipka vedoucí na mapku s patrem je příliš malá. Nepůjde trefit prstem bez velkého soustředění. Řešení: buď úplně vypustit, nebo zvětšit.

4. Je dostatečně srozumitelné, jakou funkci mají veškerá tlačítka, která systém poskytuje?

- *Dávky* - Tlačítko „Run“ by nemělo vést na obrazovku s detailem, místo toho pouze zapínat a vypínat.
- *Zobrazení mapy* - Není mi jasné, kam mě posunou tlačítka „Prev“ a „Next“. Na další a předchozí patro (jestliže ano, které kam)? Doleva na patře? Do jiné budovy? Jsem pro přesnější označení.
- *Zobrazení mapy* - Tlačítko „Menu“ vede na domovskou obrazovku. Mělo by být označené jako „Home“. Označení „Menu“ se v OS Android používá pro vyvolání kontextového menu.
- *Chyba „síť nenalezena“* - Tlačítko „Try again“ je nedostatečně označeno jako tlačítko. Mělo by mít okraj, ideálně se stínováním, aby bylo jasné, že se jedná o aktivní oblast.

5. Jsou klíčové změny systému uživateli oznamovány jasně a zřetelně?

- *Chyba „síť nenalezena“* - Chybová hláška mi připadá z nevyhovující.

6. Zabraňuje systém ve všech kritických předčasnému a nechtěnému dokončení či spuštění kritické operace?

- *Dávky* - Tlačítka „Run“ a „>“ (odkaz na detail) jsou hned vedle sebe, zvyšuje se tedy riziko překlepů.

7. Je v systému v současnosti nějaký nadbytečný grafický nebo ovládací prvek, případně funkce, která by tam být nemusela?

- *Dávky* - Tato obrazovka obecně je nepříliš optimalizovaná. Doporučuji vzít si příklad ze seznamu budíků.
- *Dávky* - Detailní informace na této obrazovce jenom zabírají místo. Stačí, když budou přístupné na stránce s detailem. Měla by zůstat jedině informace, zda je služba aktivní, která se dá znázornit on/off tlačítkem.
- *Hlavní menu* - Android aplikace by neměly využívat tlačítko „Exit“ kromě kritických situací, například internetové bankovnictví. Tuto funkci plní samostatné tlačítko „Home“ umístěné přímo na přístroji.
- *Hlavní menu* - Volba „Settings“ by měla být přístupná až v nabídce tlačítka „Menu“.
- Spodní lišta s tlačítky, přítomná na některých obrazovkách, vypadá jako kontextové menu, ale ve skutečnosti není.

B.2 David K., vývojář, správce a administrátor

1. Ví uživatel, jaký je aktuální stav systému a kde se zrovna nachází?

- *Zobrazit mapu* - Modrá bublinka znamená, že je zařízení aktivní? Nebo jen zobrazuje zařízení? Je nějak rozlišeno aktivní a neaktivní (případně nefunkční) zařízení? V případě zařízení, kde jsou možnosti pouze aktivní/neaktivní by bylo pěkné nevyvolat obrazovku zařízení, ale rovnou onu akci zapnutí nebo vypnutí.
- *Přehled zařízení* - Bylo by vhodné napsat uživateli, že se nachází v „Device list".
- *Dávky* - Doporučuji napsat uživateli velkým textem „Batch list" nebo tak něco, je tam jen malým textem co je to „batch" a není tomu rozumět.

2. Pozná uživatel, co se systémem může dělat dál?

- *Přihlášení* - Bylo by vhodné napsat tam nějaký text typu „Please, log in".
- *Změna informace zařízení* - Uživatel sice pozná, jaké má možnosti, ale bylo by vhodné umisťovat zařízení do různých kategorií, které by bylo možné ovládat najednou (např. zhasnout všechna světla v celém bytě, zhasnout 1. patro apod.).
- *Detail dávky* - Přidat tlačítko „zpět" nebo „do menu".
- *Hlavní menu* - U tlačítka „Automatic control" není zcela jasné, že zobrazí seznam rutin. Lepší by bylo „routines list" nebo „batches list".
- *Přehled zařízení* - Uživatel nemá možnost vrátit se na předchozí obrazovku. Musí si vybrat nějaké zařízení nebo patro.
- *Seznam dávek* - Opět chybí možnost snadného návratu o obrazovku zpět (nebo do menu). Není jasné zda se „routine" spouští tlačítkem „Run!" nebo i tlačítkem „>" nad ním.
- *Nová dávka* - Chybí možnost návratu zpět nebo do menu.
- *Hlavní menu* - V případě, že uživatel skočil do menu z nějaké jiné obrazovky, nemá návratu zpět.
- *Zobrazení mapy* - Není zcela jasné, jakou funkci mají tlačítka „next" a „prev" (Další patro? Nebo místnost? Nebo úsek mapy?).

3. Odpovídají ovládací prvky kontextu, ve kterém bude aplikace používána - tzn. lze aplikaci pohodlně používat jednou rukou, při řízení auta, na žebříku ve sklepě a v dalších náročných situacích?

- *Detail zařízení* - Pokud se jedná o zařízení, které má jen 2 stavy zapnuto/vypnuto tak by se mi líbil ovládací prvek posuvníku mezi oněmi 2 stavy (něco jako je na iPhone).
- *Přihlášení* - Bylo by vhodné přidat odrážku „zapamatovat heslo", aby ho nemusel uživatel pokaždé zadávat (psát uživatelské jméno a heslo může být jednou rukou problematické).

4. Je dostatečně srozumitelné, jakou funkci mají veškerá tlačítka, která systém poskytuje?

- *Změna informací zařízení* - Systém nepovoluje neuložit změny nastavení, přidat tlačítko „Cancel“ nebo „Back“.

5. Jsou klíčové změny systému uživateli oznamovány jasně a zřetelně?

- *Zobrazení mapy* - Uživatel může být zmaten, že se mu při rozkliknutí „Building“ ihned objeví např. 2. patro. Správné by bylo nechat uživatele vybrat si patro nebo případně přímo konkrétní místnost

6. Zabraňuje systém ve všech kritických předčasnému a nechtěnému dokončení či spuštění kritické operace?

-

7. Je v systému v současnosti nějaký nadbytečný grafický nebo ovládací prvek, případně funkce, která by tam být nemusela?

-

Příloha C

Instalační příručka

C.1 Minimální požadavky

Minimální požadavky na konfiguraci chytrého telefonu:

Verze platformy: Android 2.3.3 (API 10)
Procesor: 600 MHz
Paměť RAM: 400 MB
Volné místo v úložišti: 30 MB
Rozlišení: 480x800
Density: MDPI

C.2 Nasazení systému *SmartBuildings*

Pro jednoduchost je na přiloženém DVD (viz příloha D) k dispozici obraz virtuálního stroje `image.ora` pro virtualizační systém *Oracle VirtualBox*. Na tomto obraze je nainstalován Linux *Fedora 16 „Verne“ i686* s předinstalovaným a nakonfigurovaným systémem pro správu inteligentních budov *SmartBuildings* [31] a s emulátorem virtuálního Android zařízení.

Virtuální stroj je možné zprovoznit pomocí importu souboru `image.ora` do vlastní instalace VirtualBoxu. Toho lze docílit buď dvojitým kliknutím na obraz nebo volbou `„File" -> „Import appliance"` v manažeru virtuálních strojů a následným výběrem souboru s příponou .ova na DVD [29].

C.3 Spuštění serveru, konfigurace zařízení

Po spuštění virtuálního stroje a naběhnutí systému je potřeba nejdříve zapnout aplikační server a zprovoznit databázi. Veškeré příkazy je pro jistotu lepší zadávat jako administrátor (přihlašovací údaje: *root/smartbuildings*), ke kterému se lze přihlásit pomocí příkazu `su`.

Následně spustíme aplikační server Tomcat:

```
[root@smartbuildings ]# /opt/tomcat7/bin/startup.sh
```

Dále databázi SB serveru:

```
[root@smartbuildings ]# systemctl start mysqld.service
```

A nakonec funkčnost serveru ověříme ve webové administraci Tomcatu, která běží na adrese `localhost:8080/manager` (přihlašovací údaje: *tomcat/tomcat*). Modul */SmartBuildings* by měl být ve stavu „running“ - pokud komponenta neběží, pokusíme se ji zprovoznit tlačítkem `„start"`.

Jako simulátor chování reálných zařízení je k dispozici jednoduchá aplikace *DeviceSimulator*, kterou zapneme příkazem:

```
[root@smartbuildings ]# java -jar DeviceSimulator/DeviceSimulator.jar
```

V *DeviceSimulatoru* lze ručně přidat libovolný počet různých zařízení. K dispozici je však i přednastavená konfigurace, kterou lze načíst volbou `„Program" -> „Load config ..."` a dále výběrem souboru `component_settings.xml` umístěného ve složce `/home/smartbuildings/`. Po načtení konfigurace nebo nastavení zařízení k simulátoru necháme připojit server stisknutím tlačítka `„Listening"`[1]

Konfigurovat připojená zařízení a nastavovat systém *SmartBuildings* můžeme přes webové administrační rozhraní dostupné na adrese `localhost:8080/SmartBuildings` (přístupové údaje: *su/su*). Veškeré změny provedené serveru jsou ihned viditelné v mobilní aplikaci.

C.4 Spuštění mobilního klienta

Mobilního klienta lze spustit dvěma způsoby - na reálném mobilním Android zařízení nebo pokud žádné nemáte, v emulátoru virtuálního stroje.

C.4.1 Reálné zařízení

Spuštění na reálném zařízení je doporučeno pro lepší a věrohodnější zážitek z použití a pro bezproblémový chod aplikace[2]. Aby se chytrý telefon dokázal připojit k SB serveru na virtuálním stroji, je nejdříve potřeba vytvořit síťový most mezi virtuálem a fyzickým počítačem. Toho je možné docílit volbou `„Devices" -> „Network adapters..."` v okně virtuálního stroje a dále výběrem jednoho ze dvou dostupných adaptérů a změnou jeho nastavení z „NAT“ na „Bridged Adapter“.

[1] Předpokladem je, že je na SB serveru již nahrán ovladač k *DeviceSimulatoru* a tak server ví, jakým způsobem se ke komponentě může připojit.

[2] Emulátor může obsahovat chyby, je pomalý a navíc nepodporuje některé klíčové funkce každého chytrého telefonu [13].

Adresu síťového rozhraní lze zjistit v konzoli Fedory příkazem:

```
[root@smartbuildings ]# ifconfig -a | grep "inet adr"
```

Aplikaci do chytrého telefonu nainstalujeme překopírováním a spuštěním instalačního balíčku `SmartBuildingsApp.apk`, dostupném na přiloženém DVD (viz D). Telefon musí být připojen ke stejné síti jako počítač, na kterém je spuštěn virtuální stroj se serverem. K serveru se pak lze připojit zadáním správné IP adresy na přihlašovací obrazovce aplikace.

Přihlašovací údaje k přístupu na server jsou *su/su*.

C.4.2 Virtuální zařízení

Emulátor virtuálního Android zařízení lze spustit ve virtuálním stroji přes předkonfigurovaný *Android Virtual Machine Manager* (AVD manažer) příkazem:

```
[root@smartbuildings ]# ./android-sdk-linux/tools/android avd
```

V AVD manažeru je připraveno typické zařízení, které lze emulovat kliknutím na tlačítko „`Start...`" -> „`Launch`". Načítání virtuálního Android zařízení může trvat i několik minut a jakmile je vše připraveno, aplikace může být spuštěna přes ikonku dostupnou na ploše emulátoru.

K SB serveru se připojíme na přihlašovací obrazovce zadáním IP adresy rozhraní[3] a následným přihlášením pomocí přihlašovacích údajů *su/su*.

[3] **Pozor!** Nezadávejte adresu `127.0.0.1` nebo `localhost`, protože by šlo o rozhraní Android zařízení. Do adresy serveru musí být zadána IP adresa vnějšího rozhraní sítě virtuálního stroje.

Příloha D

Obsah přiloženého DVD

Na DVD přiloženém k této práci jsou k dispozici všechna data a zdrojové soubory, které byly v průběhu vývoje projektu vytvořeny. Obsah a struktura disku je znázorněna níže, na obr. D.1.

- `DVD`
  - `image.ova` ..... obraz virtuálního stroje s operačním systémem Fedora
  - `README.txt` ..... instalační a konfigurační instrukce, minimální požadavky
  - `SmartBuildingsApp.apk` ..... instalační balíček nejnovější oficiální verze aplikace
  - `analysis` ..... soubory týkající se návrhu a analýzy aplikace
  - `docs`
    - `coverage` ..... report pokrytí poslední verze jednotkovými testy
    - `javadoc` ..... dokumentace kódu aplikace ve formátu *JavaDoc*
  - `graphic` ..... grafické návrhy, ikony, loga, zdrojová grafika atd.
  - `src`
    - `BP` ..... zdrojové soubory k textu bakalářské práce
    - `SmartBuildingsApp` ..... zdrojové soubory a kódy k aplikaci
      - `build.xml` ..... ant skript pro kompilaci a sestavení aplikace
      - `secure.properties` ... klíče k certifikátu potřebného pro build oficiální verze
      - `build` ..... dynamické soubory a konfigurace důležité pro `build.xml`
      - `ide` ..... konfigurace a nastavení *IDE Eclipse*
      - `libs` ..... podpůrné knihovny a moduly
      - `src` ..... zdrojové kódy aplikace
      - `tests` ..... soubory k jednotkovým testům
  - `text`
    - `bp.pdf` ..... text bakalářské práce v PDF
    - `bp.ps` ..... text bakalářské práce v PS formátu

Obrázek D.1: Struktura a obsah přiloženého DVD