ZÁPADOČESKÁ UNIVERZITA V PLZNI
FAKULTA ELEKTROTECHNICKÁ

KATEDRA TECHNOLOGIÍ A MĚŘENÍ

DIPLOMOVÁ PRÁCE

Mapový systém pro orientaci v budově (areálu) pro mobilní telefony

vedoucí práce: **Ing. Petr Kropík, Ph.D.**
autor: **Bc. Pavel Předota**

Plzeň 2012

ZÁPADOČESKÁ UNIVERZITA V PLZNI
Fakulta elektrotechnická
Akademický rok: 2011/2012

ZADÁNÍ DIPLOMOVÉ PRÁCE

(PROJEKTU, UMĚLECKÉHO DÍLA, UMĚLECKÉHO VÝKONU)

Jméno a příjmení:	**Pavel PŘEDOTA**
Osobní číslo:	**E10N0047P**
Studijní program:	**N2612 Elektrotechnika a informatika**
Studijní obor:	**Komerční elektrotechnika**
Název tématu:	**Mapový systém pro orientaci v budově (areálu) pro mobilní telefony**
Zadávající katedra:	**Katedra technologií a měření**

Zásady pro vypracování:

1. Prostudujte problematiku tvorby aplikací pro mobilní telefony se zaměřením na ukládání perzistentních dat s využitím RMS a práce s vektorovou SVG grafikou.
2. Vytvořte aplikaci pro mobilní telefon s vektorově vykreslovanými mapami podlaží areálu budov, s mini databází s vhodně volenou standardní strukturou pro snadnou aktualizaci.
3. Umožněte aktualizaci dat a aplikace technologií OTA (Over The Air).
4. Vytvořte aplikaci tak, aby umožnila navigaci dle zadání čísla místnosti a vlastní polohy, body zájmu, více orientačních bodů, vygenerování trasy a navigace na cíl, zadání požadovaných parametrů trasy (např. bezbariérová nebo naopak bez výtahu).
5. Umožněte aplikaci pracovat pokud možno v režimu offline a zajistěte, aby aplikace nepřesáhla svými nároky omezení daná parametry mobilních telefonů. Vytvořte též potřebnou aplikaci pro osobní počítač pro údržbu mobilní verze. Aplikaci vytvořte pod některou z volných licencí (OpenSource).

Rozsah grafických prací: **dle doporučení vedoucího**

Rozsah pracovní zprávy: **30 - 40 stran**

Forma zpracování diplomové práce: **tištěná/elektronická**

Seznam odborné literatury:

1. **Topley, K. : J2ME V kostce, Grada Publishing, Praha, 2004.**
2. **Mahmoud Q. H.: Naučte se Java 2 Micro Edition, Grada Publishing, Praha, 2002.**
3. **Roedl M.: Využití jazyka MicroJAVA na platformě mobilních zařízení, bakalářská práce ZČU, 2006.**
4. **Kraus V.: Multimediální rozhraní mobilních zařízení na platformě J2ME, bakalářská práce ZČU, 2007.**
5. **Popelka J.: Nástroje pro vývoj aplikací na platformě Java Micro Edition pro mobilní zařízení, bakalářská práce ZČU, 2007. Internetové zdroje.**

Vedoucí diplomové práce: **Ing. Petr Kropík, Ph.D.**
Katedra teoretické elektrotechniky

Datum zadání diplomové práce: **17. října 2011**

Termín odevzdání diplomové práce: **11. května 2012**

Doc. Ing. Jiří Hammerbauer, Ph.D.
děkan

L.S.

Doc. Ing. Vlastimil Skočil, CSc.
vedoucí katedry

V Plzni dne 17. října 2011

Anotace

Práce se zabývá problematikou vývoje aplikací pro mobilní telefony, zvláště se zabývá vývojem pro mobilní operační systém Android. Cílem práce je shrnout principy operačního systému Android a vytvořit aplikaci pro mobilní telefon, která bude sloužit jako mapový systém pro orientaci v budovách v areálu Bory Západočeské univerzity v Plzni.

Klíčová slova

Android, mapový systém, mobilní telefon, navigace

Abstract

Map system for in-building orientation for cellular phones

This work deals with issues of mobile application development especially the development of the mobile operating system Android. The objective of this work is to summarize principles of Android OS and to create an application for mobile phones which will work like a map system for in-building orientation in the area of Bory Campus of the University of West Bohemia in Pilsen.

Key words

Android, map system, smartphone, navigation

Prohlášení

Předkládám tímto k posouzení a obhajobě diplomovou práci, zpracovanou na závěr studia na Fakultě elektrotechnické Západočeské univerzity v Plzni.

Prohlašuji, že jsem tuto diplomovou práci vypracoval samostatně, s použitím odborné literatury a pramenů uvedených v seznamu, který je součástí této diplomové práce.

Dále prohlašuji, že veškerý software, použitý při řešení této diplomové práce, je legální.

V Plzni dne 8. 5. 2012 Jméno příjmení

……..

Poděkování

Tímto bych poděkovat vedoucímu mé bakalářské práce Ing. Petru Kropíkovi PhD. za poskytnutí cenných rad, připomínek a za veškerý čas, který mi při psaní této práce věnoval.

Obsah

Seznam použitých pojmů a zkratek

- API (Application Programming Interface) – rozhraní určující interakci mezi součástmi softwarové struktury
- OpenGL (Open Graphic Library) – multiplatformní standard specifikující grafické rozhraní pro aplikace využívající 2D či 3D grafiku
- Rámec (Framework) – softwarová struktura obsahující knihovny, API apod., sloužící jako podpora při programování
- SDK (Software Development Kit) – sada vývojových nástrojů sloužících k vytváření aplikací
- SQL (Structured Query Language) – dotazovací jazyk používaný pro práci s daty v relačních databázích
- SQLite – odlehčené SQL používající zjednodušené rozhraní pro práci s daty relační databáze
- SVG (Scalable Vector Graphic) – vektorový grafický formát na bázi XML
- VGA (Video Graphic Array) – standard pro zobrazovací techniku definovaný společností IBM
- XML (Extensible Markup Language) – značkovací jazyk, který umožňuje strukturovaně uspořádávat data pomocí značek, tzv. tagů, a atributů.

Úvod

Od doby, kdy byl zkonstruován první mobilní telefon na světě, uplynuly již více než tři desetiletí. Když v roce 1973 sestrojila společnost Motorola první funkční prototyp mobilního telefonu, začala tím novou éru v možnostech komunikace. Časem se však definice mobilního telefonu jako prostředku pouze pro přenos hlasu výrazně změnila. Zatímco prototyp společnosti Motorola, který byl po deseti letech od svého zkonstruování uveden na trh pod označením DynaTAC 8000X, sloužil pouze k uskutečňování hovorů a svými rozměry připomínal cihlu, dnešní moderní mobilní telefony disponují obrovským množstvím funkcí a mnohem kompaktnějšími rozměry. [1]

V současné době mobilním telefonům vévodí chytré telefony, tzv. smartphony, které jsou ve své podstatě výkonné kapesní počítače, které kombinují mobilní telefon s kamerou a s malým kapesním počítačem PDA. Obrovskou výhodou těchto chytrých telefonů je fakt, že v jediném zařízení přináší veliké množství funkcí. Aby však bylo možné tyto funkce využívat, bylo nutné vytvořit uživatelské prostředí, které by bylo schopné s uživatelem komunikovat co možná uživatelsky nejpřátelštější formou. Z tohoto důvodu byly vytvořeny mobilní operační systémy, jejichž řady se v poslední době s dostupností chytrých telefonů rozšířily.

V této práci jsem se rozhodl věnovat právě jednomu z nejčerstvějších zástupců těchto operačních systému, mobilnímu operačnímu systému Android. A proč právě tomuto systému? Prvním důvodem je vzrůstající obliba systému Android, který se za svoji zhruba čtyřletou existenci stal velice oblíbeným v celé řadě chytrých telefonů. Dalším, pro tuto práci velmi důležitým důvodem, je poměrné dobrá podpora pro vývoj aplikací pro tento operační systém, jednak přímo od vývojářů systému, tak i od neoficiálních komunit vývojářů.

Cílem této práce je vytvoření mobilní aplikace pro orientaci v budovách univerzitního areálu Bory, Západočeské univerzity v Plzni. K tomu bylo nutné se seznámit, jak s obecnými principy vývoje aplikace pro operační systém Android, jakými jsou například návrh uživatelského rozhraní a podpora různých typů telefonů, tak i se složitějšími principy, jakými jsou použití externí SQLite databáze, zobrazování SVG formátu, použití vláken či ošetřování změn aplikace způsobených při rotaci zařízení.

1. Vývoj aplikací pro mobilní operační systémy

S nárůstem mobilních operačních systémů přibylo i různých vývojových prostředků pro tvorbu aplikací tzv. třetích stran, tedy aplikací, jejichž vývoj nebyl přímo spojen se společností vyvíjející daný operační systém. Následující řádky se budou věnovat problematice vývoje aplikací pro několik, v poslední době nejvíce rozšířených, mobilních operačních systémů.

1.1. Windows Phone

Tento operační systém je nástupce mobilní platformy Windows Mobile od společnosti Microsoft, ačkoliv s ní není zpětně kompatibilní.[2] Veškeré aplikace vyvíjené pro tuto platformu jsou postaveny na sadě nástrojů Microsoft XNA Application Framework nebo na Microsoft Silverlight.[3] K návrhu uživatelského rozhraní se využívá designerský jazyk Metro[4], který k vizuální komunikaci mezi uživatelem a aplikacemi používá systém aktivních dlaždic. Z hlediska vývojového prostředí je k dispozici sada Windows Phone Developer Tools jako rozšíření vývojového prostředí Microsoft Visual Studio nebo pro návrh uživatelského rozhraní.[5]

Pro vývoj aplikací v sadě Windows Phone Developer Tools je zatím možné využít programovacích jazyků C# a Visual Basic. Pro návrh grafického rozhraní je navíc k dispozici značkovací jazyk XAML.[6] Veškeré aplikace třetích stran vyvinuté na této platformě určené k dalšímu šíření přes oficiální Windows Phone Marketplace, musí být nejdříve schváleny samotnou společností Microsoft.[7]

1.2. Symbian

Mobilní operační systém Symbian využívá k vývoji aplikací hned několik různých programovacích jazyků. Od roku 2010 se však oficiálním vývojovým jazykem stal programovací jazyk C# s využitím Qt Framework pro vývoj grafických i negrafických uživatelských rozhraní.[8] K tomuto jazyku je k dispozici například vývojové prostředí Qt Creator nebo Carbide. [9]

Mezi další programovací jazyky, které mohou být využity k vývoji pro Symbian patří jazyk Python, Java Micro Edition nebo Adobe Flash verze Lite.[10] Nově vytvořené aplikace lze ověřit zdarma digitálním podpisem od výrobce.[11]

1.3. BlackBerry

BlackBerry je mobilní operační systém, který vyvinula společnost Research In Motion pro jejich vlastní řadu stejnojmenných chytrých telefonů. K vývoji aplikací třetích stran slouží vývojářům programovací jazyk Java s využívaním knihoven vlastního BlackBerry API. [12]

Jedno ze zajímavostí na operačním systému BlackBerry je fakt, že v budoucnu bude každý nový BlackBerry smartphone plně, bez modifikací, podporovat aplikace vytvořené pro operační systém Android.[13]

1.4. iPhone OS (iOS)

Jedná se o mobilní operační systém od společnosti Apple primárně vytvořený pro chytré telefony iPhone. Přesto však našel uplatnění i v jiných zařízeních od této společnosti jako například v multimediálním přehrávači iPod Touch nebo tabletu iPad. V době psaní této práce byl pro zařízení společnosti Apple k dispozici iOS verze 5.x.[14]

Co se týče vývoje pro tuto platformu, společnost Apple uvolnila sadu nástrojů s názvem Xcode, která obsahuje balíček iOS SDK a simulátor skutečného zařízení iOS Simulator.[15] Velkou nevýhodou je ale fakt, že sada Xcode je podporovaná pouze operačním systémem Apple MacOS, který je k dispozici, pokud pomineme nelegální úpravu zdrojového kódu, pouze na počítačích elektronicky označených společností Apple. Programování aplikací je objektově orientované využívající jazyk Objective-C.[16]

Primárně by se měli veškeré aplikace třetích stran instalovat do telefonu prostřednictvím oficiálního obchodu App Store, kde veškeré aplikace nejprve projdou schválením. Toto omezení však ze strany uživatelů vedlo k procesu tzv. Jailbreaku, kdy je systém modifikován tak, aby bylo možné do něho nahrávat i neautorizované aplikace. [17]

1.5. Android

Vzhledem k tomu, že tomuto mobilnímu operačnímu systému bude věnován zbytek práce, je v této kapitole pouze zmíněn pro úplnost seznamu a jeho bližšímu popisu se budu věnovat v dalších kapitolách.

1.6. Bada

Tento mobilní operační systém vyvinula společnost Samsung pro své zařízení typu smartphone a tablet. Pro vývojáře je k dispozici Bada SDK, který obsahuje potřebné knihovny, včetně emulátoru zařízení, a pro programování využívá jazyka C++ či Flash.[18]

Tento systém se ale do budoucna v telefonech společnosti Samsung příliš neprosadil a proto je v nových modelech nahrazován platformou Android či Windows Phone.

1.7. MeeGo

Tato mobilní platforma byla zatím použita pouze v jednom chytrém telefonu a dále není zcela jistý její osud v budoucnu. Jedná platformu vyvíjenou společností Intel a Nokia, využívající pro psaní aplikací jazyk C# s využitím Qt Framework. [19]

Budoucnost této platformy je však nejistá z důvodu odstoupení společnosti Nokia od vývoje a přechodu na platformu Windows Phone. Přesto je zde zmíněna, protože tato platforma budí velké očekávání nejen v případě použití v chytrých telefonech ale i v jiných zařízeních.

2. Operační systém Android

Jak jsem již zmiňoval v úvodu, v této práci se budu primárně věnovat právě mobilnímu operačnímu systému Android od společnosti Google.[20] Důvody tohoto rozhodnutí jsou především velice dobrá základna pro vývojáře aplikací a dále rostoucí obliba tohoto systému v nových telefonech či tabletech.

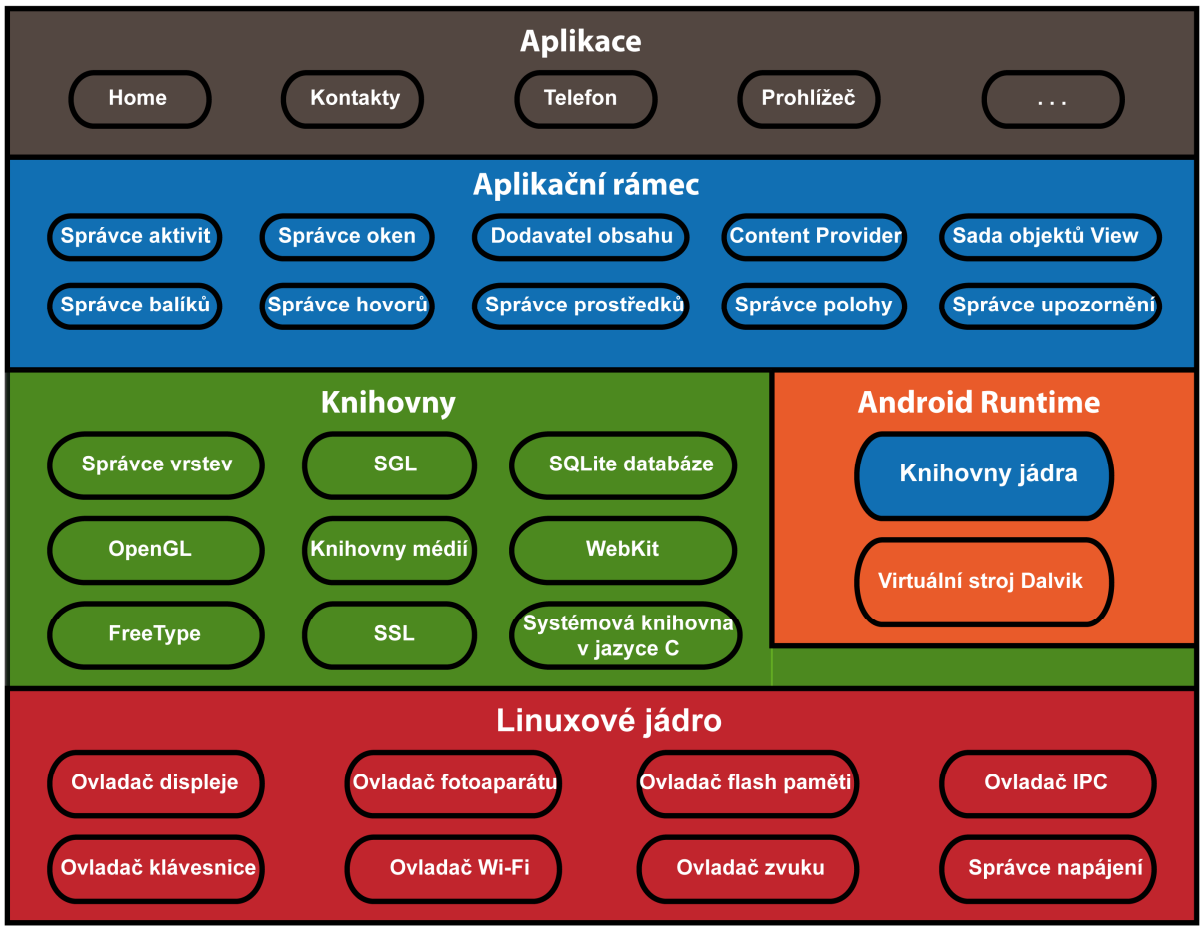
2.1. Historie

Operační systém byl oficiálně představen 5. listopadu 2007. Spolu s ním bylo představeno a tím i oficiálně vzniklo sdružení společností zabývající se softwarem, hardwarem a telekomunikacemi pod názvem Open Handset Alliance (OHA). Zároveň byl systém Android předán tomuto sdružení společností Google, která zakoupila začínající společnost Android již v roce 2005. [21]

Téměř rok po oficiálním představení systému, 23. října 2008, byla představena první verze Android 1.0 a s ní začal i vývoj nových aplikací pro tuto platformu. Později v říjnu roku 2008 byl systém uvolněn jako open-source a tím mohl jeho zdrojový kód využívat každý. Toto rozhodnutí nepochybně pomohlo k rozšíření této platformy. [21]

2.2. Architektura

Architektura systému Android je rozdělena celkem do 5 vrstev. Na obrázku 1 jsou zobrazené jednotlivé vrstvy systému. V následujících odstavcích budou jednotlivé vrstvy podrobněji popsány.



Obrázek 1: Vrstvy architektury systému Android [22]

2.2.1. Aplikace

Aplikace představují nejvyšší vrstvu a v podstatě se jedná o vrstvu, kterou využívá běžný uživatel. Mezi tyto základní aplikace patří například emailový klient, SMS, webový prohlížeč, prohlížeč kontaktů a jakákoliv předinstalovaná či dodatečně nainstalovaná aplikace. [23]

2.2.2. Aplikační rámec

Tato vrstva je nejdůležitější pro vývojáře aplikací. Díky této vrstvě mají vývojáři plný přístup k API, úrovně odpovídající konkrétní verzi systému. Umožňuje také to, že data určité aplikace či služby mohou být k dispozici aplikaci či službě jiné, tzn., dovoluje využívat hardware konkrétního zařízení, komunikovat se službami na pozadí atd. [24]

Nyní bych se zaměřil na popis samotné vrstvy, která obsahuje sadu služeb a prostředků zásadních pro chod veškerých aplikací:[24]

- **Rozšiřitelná sada objektů View**, která se využívá pro vytvoření uživatelského rozhraní aplikace
- **Správce prostředků**, který poskytuje přístup k „nekódovým" zdrojům, jako jsou XML soubory s uloženými textovými řetězci, XML soubory s rozvržením objektů v okně atd. Vesměs je využíván k načtení XML návrhů, obecných XML souborů nebo obrázků, využívaných v aplikaci.
- **Správce upozornění**, který dovoluje všem aplikacím zobrazit vlastní upozornění ve stavovém řádku
- **Správce aktivit**, který spravuje jednotlivé aktivity aplikace. Řídí jejich životní cyklus, přepíná mezi jednotlivými aktivitami atd.
- **Správce hovorů**, který zajišťuje přístup k telefonním službám zařízení. Pomocí něho je například možné zjistit stav sítě, ke které je telefon připojen nebo zjistit stav SIM karty telefonu.
- **Správce balíků**, pomocí kterého je možné získat data z balíků aplikace uložené v zařízení.
- **Správce polohy**, pomocí kterého je možné získat aktuální polohu zařízení, prostřednictvím dat z GPS nebo vyvolat specifický záměr při dosažení určité polohy

2.2.3. Knihovny

Tato vrstva obsahuje sadu knihoven psaných v jazyku C/C++. Tyto knihovny jsou využívané různými komponentami systému a vývojáři aplikací jsou přístupné prostřednictvím vrstvy Aplikační rámec. [25]

Některé základní knihovny:

- **Systémová knihovna C** – standartní knihovna jazyka C (libc) odvozená od distribuce BSD.
- **Knihovna médií** – knihovna pro přehrávání audio a video formátů.
- **Správce vrstev** – umožňuje zkombinovat vrstvy 2D a 3D grafiky z více aplikací
- **LibWebCore** – knihovna s podporou webového prohlížeče
- **SGL** – základní knihovna 2D grafického enginu
- **3D knihovny** – knihovny jsou implementovány na základě OpenGL ES, knihovny využívají buďto hardwarovou akceleraci grafického rozhraní nebo optimalizované softwarové 3D rastrování.
- **FreeType** – knihovna určená k renderování vektorových a bitmapových fontů
- **SQLite** – knihovna odlehčené relační databáze, která je k dispozici všem aplikacím

2.2.4. Android Runtime

Každá Android aplikace je spuštěna ve vlastním procesu, s vlastní instancí ve virtuálním stroji Dalvik. Tento virtuální stroj by speciálně vytvořen pro systém Android. Soubory jsou spouštěny ve formátu Dalvik Executable, s příponou .dex, který je optimalizován na co nejmenší paměťové požadavky. Protože veškeré spouštěné třídy jsou kompilovány v jazyku Java, musí být poté ještě převedeny do souboru .dex pomocí přiložených nástrojů.[26]

Dalvik je registrově založený virtuální stroj, který využívá vlastností Linuxového jádra, například podpory vláken, nízkoúrovňové správy paměti atd. [27]

2.2.5. Linuxové jádro

Nejnižší vrstva obsahuje linuxové jádro, které tvoří abstraktní vrstvu mezi hardwarem a softwarovým vybavením. Jádro je postaveno na operačním systému Linux ve verzi 2.6 a využívá jeho vlastností, například správu paměti a procesů, zabudované ovladače, síťovou správu atp. [28]

2.3. Základní součásti aplikace

Aplikace pro systém Android se píší v běžně používaném a rozšířeném programovacím jazyce Java. Zároveň využívá i některé standardní knihovny jazyka Java, které jsou rozšířené o knihovny specificky vytvořené pro systém Android. Aplikace je kompilována do souboru s příponou *.apk, který zároveň Android zařízení využívá k instalaci aplikace. [29]

V tomto odstavci bych se zaměřil na popis hlavních a nejdůležitějších komponent každé aplikace pro systém Android.[30] Patří mezi ně:

- **Aktivity** – Jedná se o nejdůležitější součásti každé aplikace. V podstatě se jedná o komponenty, které zajišťují zobrazení okna s aktuálním uživatelským rozhraním. Další podrobnosti ohledně aktivit a jejich životního cyklu budou rozvedeny v kapitole 3. [30]
- **Dodavatel obsahu** (Content Provider) – Dodavatel obsahu dovoluje uživatele přistupovat k jakýmkoliv sdíleným datům, která jsou uložena v zařízení. Pomocí dodavatele obsahu je možné také zabezpečit sdílená data a určit jakým způsobem budou data k dispozici jiným aplikacím. Příkladem využití může být například přístup do telefonního seznamu zařízení. [30]
- **Služby** (Services) – Na rozdíl od aktivit, které slouží k zobrazení aktuálního okna aplikace a mají životnost pouze do ukončení aplikace, mohou služby pokračovat dále v práci, nezávisle na kterékoliv aktivitě. Tímto způsobem lze například přehrávat hudbu spuštěnou v hudebním přehrávači i po tom co, byla spouštěcí aktivita přehrávače ukončena. [30]
- **Záměry** (Intents) – Jedná se o systémové zprávy a upozornění, pomocí kterých jsou spuštěné aplikace informovány o výskytu některých událostí.

Typickým příkladem může být příchozí hovor či textová zpráva nebo manipulace s paměťovou kartou. Nejčastější využití záměrů je v aplikacích s více aktivitami, kde záměr souží ke spouštění nových aktivit. Lze však vytvářet i záměry pro jiné aktivity, než jen pro aktivity vlastní aplikace. [30]

2.4. Soubor Manifest

Žádná z komponent popsaných v předchozí podkapitole a tudíž žádná aplikace by nemohla fungovat bez souboru AndroidManifest.xml[31], který se nachází v kořenovém adresáři každé aplikace. V tomto souboru se registrují jednotlivé komponenty aplikace, jako jsou aktivity, filtry záměrů, dodavatelé obsahu či služby. Mimo samotného registrování aktivit v tomto souboru určujeme, jaká aktivita bude hlavní a bude tedy spuštěná po startu aplikace.

Velice důležité je, chceme-li v naší aplikaci přistupovat například k vibracím telefonu, GPS přijímači či jiné systémové či hardwarové součásti zařízení, musíme využívanou součást povolit právě v manifestu. Pokud by se totiž aplikace domáhala systémové funkce, která před kompilací nebyla povolena, způsobí to nevyhnutelně chybu aplikace. Dále je možné v tomto souboru omezit, pro jaké verze systému Android je aplikace napsána a tím se vyhnout případným nekompatibilitám s nižší verzí systému.

Pro lepší popis souboru následuje část `AndroidManifest.xml`, používaného v aplikaci vytvořené v rámci této práce. Ukázka však obsahuje pouze zlomek ze všech nastavení a povolení, které lze v souboru provést, jelikož výčet všech možností by byl nad rozsah této práce.

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="ppredota.android.navigation.view.activities"
    android:versionCode="1"
    android:versionName="1.0" >

    <uses-sdk android:minSdkVersion="4" />
    <uses-permission android:name="android.permission.VIBRATE"/>

    <application
        android:icon="@drawable/ic_launcher"
        android:label="@string/app_name" >

        <activity
            android:label="@string/app_name"
            android:name=".MainMenuActivity" >

         <intent-filter >
           <action android:name="android.intent.action.MAIN" />
            <category android:name="android.intent.category.LAUNCHER" />

         </intent-filter>
       </activity>
```

Na této části souboru manifest je vidět, jak se provádí dříve zmíněné registrování aktivit, kde každá aktivita má prostřednictvím elementu `<intent-filter>` specifikováno jaký typ záměru může přijmout a zároveň může filtrovat záměry jiného typu. Element `<category>` určuje postavení aktivity v rámci aplikaci, v tomto případě se jedná o hlavní aktivitu, která bude spuštěna při startu aplikace. Ještě bych se zmínil o elementu `<uses-sdk>`, která v tomto případě určuje minimální verzi API, pro kterou je aplikace určena a nakonec o elementu `<uses-permission>`, který zde povoluje používání vibrací telefonu.[32]

2.5. Verze

Od první verze systému Android bylo do současné doby vydáno několik aktualizací. Každá aktualizace je k dispozici s novou úrovní rozhraní pro programování aplikací, tzv. API Level. Obecně API systému Android obsahuje základní sadu balíků a tříd, sadu XML elementů pro deklarování souboru Manifest a pro přístup ke zdrojům (Resources), sadu záměrů a sadu oprávnění pro aplikace.

V následujícím odstavci budou vyjmenovány jednotlivé verze systému spolu s příslušnou úrovní API. Pro každou novou verzi systému se vžilo pojmenování podle zákusků. V případě poslední verze se jedná o verzi, na které se v době dopsání této práce pouze pracovalo a nebyla ještě oficiálně vydaná. [33]

- **Android 1.5 Cupcake** (API Level 3)
- **Android 1.6 Donut** (API Level 4)
- **Android 2.0 – 2.1 Eclair** (API Level 5-7)
- **Android 2.2 Froyo** (API Level 8)
- **Android 2.3 Gingerbread** (API Level 9-10)
- **Android 3.0 Honeycomb** (API Level 11-13)
- **Android 4.0 Ice Cream Sandwitch** (API Level 14)
- **Android 5.0 Jelly Beans** (API Level ?)

3. Aktivity

Jak již bylo dříve uvedeno, nejdůležitější součástí každé aplikace jsou jedna či více aktivit.[34] Jedná se o komponentu, která zprostředkovává obsah okna a dovoluje interakci mezi uživatelem a aplikací. Aktivita je veřejná třída, která je odvozena děděním ze třídy `Activity.java` a která se nachází v balíku android.app.Activity. Každá aktivita, která je součástí aplikace, musí být registrována v souboru `AndroidManifest.xml`, aby mohla být v aplikaci zobrazena.

3.1. Spouštění nové aktivity

Ve většině aplikací se však nelze obejít pouze s jednou aktivitou, z toho důvodu je nutné nějakým způsobem aplikaci sdělit, že má být spuštěna jiná aktivita (například stisknutím tlačítka v aktivitě). To se provádí pomocí záměrů.[35]

3.1.1. Spuštění aktivity pomocí záměru

Pokud máme v úmyslu pouze spustit novou aktivitu a přitom není potřeba nově spuštěné aktivitě předat žádná data ze spouštěcí aktivity, stačí použít následující řádky kódu:

```
Intent intent = new Intent(this, NavigateMenuActivity.class);
startActivity(intent);
```

V tomto případě je nutné k vytvoření záměru předat objektu typu `Intent` dva parametry. Prvním je parametr `contextPackage`, kterým předáváme objektu informace o aplikačním prostředí, v tomto případě instanci spouštěcí aktivity. Druhým parametrem je třída aktivity, která má být spuštěná. Samotné spuštění se provede metodou `startActivity(intent)` s parametrem vytvořeného záměru. [36]

3.1.2. Spuštění aktivity pomocí záměru s předáním dat aktivitě

Někdy však samotné spuštění nové aktivity nestačí a je potřeba předat některá data spouštěcí aktivity spouštěné aktivitě. Může se například jednat o data, která uživatel vybral z rozbalovací nabídky a v následující aktivitě bude potřeba vybraná data dále zpracovávat.[37]

Následující kód ukazuje jak přibalit data do vytvořeného záměru:

```
Intent i = new Intent(this, SVGViewActivity.class);
i.putExtra("startPointX", startPoint.x);
i.putExtra("startPointY", startPoint.y);
```

V této ukázce jsou do vytvořeného záměru přibaleny souřadnice bodu místnosti, kterou uživatel vybral ve spouštěcí aktivitě, jako místo odkud se bude vypočítávat trasa. Každá uložená data musí být určena klíčem, v tomto případě je klíčem první parametr, který je typu `String`. Takto předaná data získáme v nově spuštěné aktivitě následujícím způsobem:

```
Intent i = getIntent();
int x = i.getIntExtra("startPointX", 0);
int y = i.getIntExtra("startPointY“, 0);
```

Pomocí metody `getIntent()` získáme záměr, kterým byla aktivita spuštěna. V tomto případě lze očekávat, že programátor aplikace má pod kontrolou, jakého typu byla data uložena do vytvořeného záměru (zde typu integer). Z toho důvodu je k získání dat využita metoda `getIntExtra()` a hodnota kterou chceme vrátit je určena pomocí klíče. Nula jako druhý parametr znamená výchozí návratovou hodnotu pro případ, že by klíčová hodnota nebyla nalezena.

Ne vždy je ale nutné využívat záměrů jen pro spouštění aktivit vlastní aplikace. Existuje možnost pomocí správně určeného záměru spustit i aktivity jiných aplikací. [36] Může se například stát, že z vlastní aplikace budeme potřebovat odeslat email některým příjemcům. V tomto případě je zbytečné vytvářet vlastního emailového klienta, pokud už je v zařízení nainstalován, místo toho lze využít záměr, jehož ukázka je v následujícím kódu.

```
Intent i = new Intent(Intent.ACTION_SEND);
i.putExtra(Intent.EXTRA_EMAIL, recipientsArray);
startActivity(i);
```

Jako parametr záměru je předána akce, pro kterou je záměr určen. Aktivita, která bude zpracovávat tento záměr, musí mít v elementu `<intent-filter>` souboru `AndroidManifest.xml` tuto akci registrovanou následujícím způsobem:

```
<action android:name="android.intent.action.SEND" />
```

Nakonec pomocí konstanty EXTRA_EMAIL sdělujeme aktivitě, která bude záměrem vyvolána, že odesíláme seznam příjemců, kterým chceme email zaslat. Pokud by se stalo, že je záměr schopno zpracovat více aktivit (aplikací), dostane uživatel možnost si cílovou aplikaci zvolit.

3.2. Životní cyklus aktivity

Každá spuštěná aktivita se řídí přesně stanoveným životním cyklem, který určuje, v jakém stavu se daná aktivita právě nachází.[38] Každá aktivita, jejíž životní cyklus nebyl ještě ukončen, se nachází v jednom z těchto stavů:

- **Aktivní** – Aktivita se nachází v popředí a je schopná interakce s uživatelem
- **Pozastavená** – V tomto případě je sice aktivita viditelná, ovšem v popředí se nachází buďto jiná aktivita, která částečně překrývá pozastavenou aktivitu, či nějaký druh upozornění. Pozastavená aktivita není schopna interakce s uživatelem
- **Zastavená** – Aktivita je sice pořád v paměti a veškerá data, která obsahuje, jsou uložena, ovšem není viditelná z důvodu spuštění nové aktivity. Nově spuštěná aktivita odsunula zastavenou aktivitu do pozadí.

Pokud se aktivita nachází v jednom z posledních dvou uvedených stavů, může být systémem kdykoliv z důvodu paměťových požadavků kdykoliv ukončena (někdy může být z důvodů extrémního nedostatku paměti ukončen i proces právě aktivní aktivity).

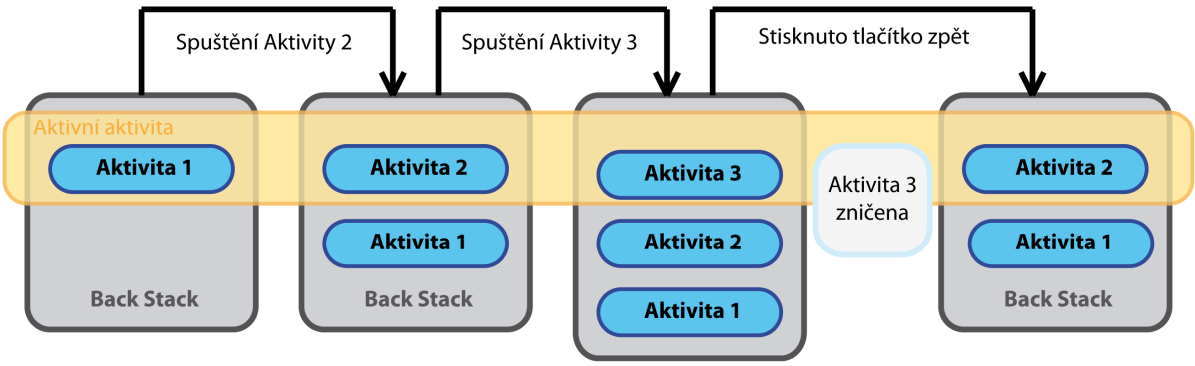
3.2.1. Callback metody životního cyklu aktivity

K potvrzení toho, že se mění stav aktuální aktivity, slouží několik různých callback metod.[39] Všechny callback metody mají dané přesné pořadí, podle kterého jsou volány při změnách stavu určité aktivity. Tabulka 1 obsahuje všechny dostupné callback metody s jejich stručným popisem. Pokud chceme tyto metody překrýt a implementovat do nich vlastní kód, je nutné nejprve zavolat rodičovskou třídu pomocí klíčového slova super, například `super.onStart()`.

Callback metoda	Popis metody	Další volaná metoda
`onCreate()`	Volána při prvním spuštění aktivity s parametrem null. Pokud již byla aktivita spuštěna a následně bude znovu vytvořena, volá se s parametrem typu `Bundle` získaným pomocí metody `onSaveInstanceState()`.	`onStart()`
`onRestart()`	Volána v případě, že byla aktivita zastavena a bude znovu přesunuta do popředí.	`onStart()`
`onStart()`	Volána těsně před tím než je obsah aktivity zobrazen uživateli, aktivita se dostává do popředí.	`onResume()` nebo `onStop()`
`onResume()`	Volána v případě, že bude obsah aktivity schopný interakce s uživatelem.	`onPause()`
`OnPause()`	Volána v případě, že se do popředí dostává jiná aktivita nebo nějaký druh upozornění	`onResume()` nebo `onStop()`
`onStop()`	Volána v případě, že byla vyvolána jiná aktivita, která zcela překrývá aktuální aktivitu a tím ji i odsouvá do pozadí. Volána i v případě, že dochází k ukončení aktivity.	`onRestart()` nebo `onDestroy()`
`OnDestroy()`	Volána při ukončení aktivity buď uživatelem, který ukončuje aplikaci nebo z důvodu paměťových požadavků systému.	-

Tabulka 1: Callback metody životního cyklu aktivity [39]

V tabulce 1 je uvedeno, že nově spuštěné aktivity jsou přesouvány do popředí a původní aktivity naopak přesouvány do pozadí, odkud mohou být znovu vyvolány. Z toho vyplývá, že si systém musí určitým způsobem pamatovat hned několik aktivit, v přesném pořadí, v jakém byly spuštěny. K tomu systém Android využívá tzv. backstack [40], tedy zásobník, ve kterém ukládá zastavené aktivity. Jakým způsobem to provádí je naznačeno na obrázku 2.



Obrázek 2: Back Stack - ukládání zastavených aktivit [40]

Z obrázku 2 lze dobře vyčíst, jakým způsobem zásobník funguje. Z aktivity 1 je spouštěna aktivita 2. Aktivita 1 je v tuto chvíli zastavená a je přesunuta na dno zásobníku. Stejným způsobem, v okamžik spuštění aktivity 3, je aktivita 2 zastavena a přesunuta do zásobníku. V tuto chvíli je tedy aktivita 3 v aktivním režimu v popředí a ostatní 2 aktivity jsou zastavené (metoda `onStop()`) nebo pozastavené (metoda `onPause()`) v pozadí. Pokud nyní uživatel zmáčkne tlačítko zpět, v případě aktivity 3 dojde k vyvolání metody `onDestroy()` a aktivita bude ukončena. Na její místo se přesune aktivita 2, u které dojde k volání metody `onRestart()` či `onResume()`. V případě opětovného spuštění aktivity 3, nebude aktivita obnovena, ale dojde k jejímu znovuvytvoření.

3.2.2. Ukládání stavu aktivity

V předchozím odstavci bylo uvedeno, že zastavené či pozastavené aktivity zůstávají, i přes jejich přesunutí do pozadí, uloženy v paměti a čekají na jejich případné obnovení. V tomto případě je zřejmé, že je zachována kompletní instance aktivity se všemi jejími daty. Ve skutečnosti je ale potřeba počítat i s možností, že aktivita bude ukončena v jakýkoliv moment. Naštěstí existuje způsob, jak i v takovém případě zachránit důležitá data aktivity.

Tento způsob využívá dvou metod. První metodou je metoda `onSaveInstanceSave()`, která ukládá informace o stavu aktivity do objektu typu `Bundle`. Tato metoda provede uložení před tím, než je zavolána metoda `onDestroy()` a ukončena aktivita. Druhou metodou je metoda `onRestoreSaveInstance()`. Tato metoda je zavolána znovu při sestavování aktivity a vrací objekt typu `Bundle` s uloženým stavem aktivity získaným metodou `onSaveInstanceSave()`. Tento objekt je předán metodě `onCreate(Bundle savedInstance)` a data z něho jsou využita při sestavování aktivity. Pokud předtím ještě nebyla aktivita spuštěna a jedná se o první spuštění aktivity, je metoda `onCreate()` spuštěna s parametrem `null`.[41]

Využívání těchto metod však není dokonalou zárukou, že při náhlém ukončení aktivity resp. aplikace dojde k jejich volání. Může se totiž stát, že aktivita, u které očekáváme, že pomocí těchto metod dojde k uložení jejího stavu, byla přesunuta do zásobníku jako zastavená. Z toho důvodu nedochází k volání metody `onDestroy()` a tedy ani k uložení stavu, jelikož instance aktivity nebyla ukončena, ale jen zastavena. Proto je doporučeno některé zvlášť důležité operace, jako například uzavření databáze, ukončení vláken a podobně, provádět v metodách `onStop()` či ještě lépe v `onPause()`, která se provede vždy. Zároveň by však implementace vlastního kódu neměla být příliš složitá a měli by v ní být provedeny jen nejnutnější operace.

3.2.3. Ošetření změn aktivity při změnách konfigurace

Náhlé ukončení aktivity z důvodu nedostatku paměti nebývá příliš časté, vzhledem k tomu, že nové telefony mívají dostatek paměťové kapacity. Častějším případem náhlého ukončení aktivity je změna orientace telefonu nebo vysunutí klávesnice, pokud je jí telefon vybaven. V těchto případech dochází ke zničení a znovuvytvoření aktivity a každá aplikace musí být schopna na podobnou situaci reagovat.[42]

Jednou možností jak ošetřit tuto situaci je prosté zakázání změny orientace tak, že zvolené aktivitě určíme, přidáním následujícího řádku do souboru `AndroidManifest.xml`, jakou orientaci bude mít nezávisle na orientaci telefonu.

```
android:screenOrientation="portrait"
```

Tento postup ale v mnoha případech není vhodný a také úplně nezaručuje, že nebude aktivita destruována. Například v případě vysunutí klávesnice je aktivita znovuvytvořena i přesto, že byla přímo určena orientace.

Další možností je využití metod pro uložení stavu. Vzhledem k tomu, že metody `onSaveInstanceState()` a `onRestoreInstanceState()` jsou omezeny pouze na data, která lze uložit do objektu typu `Bundle`, je občas výhodnější využít metodu `onRetainNonConfigurationInstance()`, ve které lze uložit jakýkoliv objekt typu `Object`, a metodu `getLastNonConfigurationInstance()`, pomocí které lze uložený objekt opět načíst.[43]

Ve skutečnosti existují i jiné způsoby, podrobně vyjmenovávat všechny je však nad rámec této práce a veškeré informace lze najít v příslušné dokumentaci.[44]

Uložení stavu nemusí být vždy úplně jednoduchou záležitostí, zvláště při využívání vláken na pozadí apod. Podrobnějším postupem, jak ošetřit změny při orientaci zařízení, se budu věnovat později v kapitole 15, v rámci popisu samotné aplikace.

4. Uživatelské rozhraní

V této kapitole bych se zmínil pouze obecně o principech tvoření uživatelského rozhraní v systému Android. Praktické ukázky a podrobnější popis vytváření rozhraní bude uveden dále v této práci, při popisu vytvořené aplikace.

V systému Android se pro tvorbu uživatelského rozhraní využívá objektů, které jsou podtřídami tříd `View` a `ViewGroups`. V případě `ViewGroups` se jedná o podtřídy, které charakterizují návrh okna, kontejnery (objekty typu `Layout`). Ve třídě `View` se naopak nacházejí jednotlivé prvky uživatelského rozhraní nazývané widgety, mezi které patří tlačítka, textová pole a podobně. Všechny widgety ve spojení s vhodnými kontejnery tvoří hierarchii náhledů uživatelského rozhraní.[45]

4.1. Vytváření uživatelského rozhraní pomocí XML návrhu

Přestože je možné tvořit uživatelské rozhraní pouze v rámci zdrojového kódu, nabízí systém Android sofistikovanější způsob, jakým vytvářet uživatelské rozhraní, a to za pomocí návrhu v jazyce XML. Tento návrh má jednu obrovskou výhodu v tom, že odděluje XML návrh od zdrojového kódu v jazyce Java a tudíž je možné kdykoliv pozměnit vytvořené uživatelské rozhraní bez nutnosti zásahu do zdrojového kódu.

Vytváření uživatelského návrhu pomocí XML by se dalo rozdělit do dvou kroků, které budou naznačeny v následujících odstavcích.

4.1.1. Deklarace elementů uživatelského rozhraní

Každý XML návrh se skládá z elementů charakterizujících, o jaký objekt uživatelského rozhraní se jedná. Každý element navíc obsahuje sadu atributů, pomocí kterých se určuje vzhled a chování konkrétního objektu. Následující část XML ukazuje, jak vypadá část návrhu hlavního menu aplikace.

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:orientation="vertical"
    <Button
        android:id="@+id/button_navigate"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:text="@string/navigate_label" />
</LinearLayout>
```

Ukázka kódu je záměrně zjednodušena, vzhledem k tomu, že v tomto oddílu se nechci věnovat podrobnému popisu. Obsahem tohoto návrhu je jen jedno tlačítko, pro jeho použití ho je však nutné ještě inicializovat ve zdrojovém kódu.

4.1.2. Vložení návrhu jako obsahu aktivity

Každý XML návrh, aby mohl být zobrazen, musí být přidán do určité aktivity jako její obsah. Vytvořené návrhy musí být uloženy ve složce res/layout v adresáři projektu. V případě, že je vytvořen návrh, pojmenovaný například `main_layout.xml`, přistupujeme k němu pomocí třídy `R`, která charakterizuje dostupné prostředky (`Resources`). Návrh vložíme jako obsah aktivity pomocí metody `setContentView()` následujícím způsobem:

```
setContentView(R.layout.main_layout);
```

Tato metoda musí být volána v metodě `onCreate()`, aby při každém sestavování aktivity bylo zajištěno, že bude načtený příslušný návrh.

4.1.3. Inicializování prvků XML návrhu ve zdrojovém kódu

Samotný XML návrh a jeho přidání do obsahu aktivity zajistí pouze statické uživatelské rozhraní bez možnosti interakce s jednotlivými widgety. Pokud je potřeba s některým widgetem pracovat ve zdrojovém kódu, což je například u tlačítka žádoucí, je nutné inicializovat konkrétní widget ve zdrojovém kódu.

Každému widgetu, který chceme použít ve zdrojovém kódu, můžeme přiřadit ID, což je unikátní řetězec, kterému systém při kompilaci přiřadí unikátní celočíselný identifikátor. Nové ID lze přiřadit widgetu následujícím způsobem:

```
android:id="@+id/button_navigate"
```

Znak @ znamená, že tento atribut bude identifikován jako prostředek a řetězec +id říká, že do dostupných prostředků bude přidán řetězec button_navigate, pomocí kterého se bude přistupovat ke konkrétnímu tlačítku.

Tlačítko bylo v tuto chvíli přidáno mezi dostupné prostředky a ve zdrojovém kódu je k dispozici prostřednictvím třídy `R`. Inicializace tlačítka ve zdrojovém kódu bude vypadat následovně:

```
Button navigateBtn;
navigateBtn = (Button) findViewById(R.id.button_navigate);
```

K inicializaci tlačítka je použita metoda `findViewById()`, které předáme z dostupných prostředků ID objektu, který chceme inicializovat. V tomto případě se jedná o tlačítko, z toho důvodu je voleno přetypování na třídu `Button`.

4.2. Layout

Objekty typu Layout by se daly popsat jako kontejnery, které sdružují jednotlivé widgety a jiné kontejnery do jednoho návrhu. Všechny dostupné kontejnery jsou podtřídami třídy `ViewGroup` a liší se způsobem koncepce návrhu.[46]

4.2.1. LinearLayout

Tento návrh vychází z myšlenky modelů založených na využití boxů, kde jsou všechny objekty (jiné kontejnery nebo widgety) řazeny jeden vedle druhého v určeném směru do svého rodičovského kontejneru.[47]

4.2.2. RelativeLayout

Zatímco `LinearLayout` pracuje s myšlenkou, že všechny objekty návrhu jsou řazeny a umísťovány vůči svému rodičovskému kontejneru, filosofie návrhu v případě `RelativeLayout` je mírně odlišná. Jednotlivé objekty jsou umisťovány vedle sebe tak, že řeší vztahy nejen vůči svému rodičovskému kontejneru, ale i vztahy vůči jiným objektům v návrhu. K určení vztahů mezi objekty v návrhu jsou využívány identifikátory specifikované v atributu `android:id`. [48]

4.2.3. TableLayout

V případě `TableLayout` se jednotlivé objekty okna vkládají do mřížky se specifickým počtem řádků a sloupců. Každému řádku je možné určit vlastnosti a způsob, jakým se bude přizpůsobovat obsahu. Počet sloupců se určuje nepřímo tak, že systém určí jejich počet podle počtu widgetů, které se nacházejí v jednom řádku.[49]

4.2.4. ScrollView

V některých případech se stane, že návrh obsahuje tolik objektů, že je displej telefonu nedokáže zobrazit všechny. V tomto případě lze využít kontejner `ScrollView`, ve kterém sice nelze přímo umisťovat widgety, ale je možné tímto kontejnerem obalit již hotový návrh. Díky tomu bude viditelná pouze část návrhu, jejíž zobrazení se vejde do displeje telefonu a ke zbytku návrhu se může uživatel dostat posunutím celého návrhu.[50]

4.3. Widgets

Objekty okna, widgety[51], jsou objekty předefinované systémem Android a slouží k vytváření uživatelského rozhraní. Mezi widgety patří jednoduché objekty typu tlačítko, textové pole, rozbalovací nabídka atd., ale také složitější objekty jako jsou hodiny a kalendář. Seznam všech widgetů je poměrně dlouhý, všechny druhy lze najít v balíku `android.widget`.

Návrh uživatelského rozhraní však není limitován pouze widgety definovanými systémem android. Je možné definovat nové widgety nebo obecně objekty třídy `View`, upravovat stávající děděním z třídy `View` nebo kombinovat stávající objekty.

4.4. Prostředky (Resources)

Prostředky by se daly charakterizovat jako data uložená odděleně od zdrojového kódu.[52] Kromě návrhů, které byly zmíněny v předchozích odstavcích, se řadí mezi prostředky například obrázky, řetězce, xml soubory a mnoho dalších souborů a informací používaných aplikací.

Výhoda využívání prostředků spočívá hlavně v tom, že je lze upravovat odděleně od zdrojového kódu. Další obrovská výhoda spočívá v univerzálnosti, kdy je možné pomocí správně definovaných prostředků přizpůsobit aplikace různým zařízením, například zvolit vhodné rozlišení obrázku pro displej zařízení, ve kterém je aplikace spuštěna.

4.4.1. Přístup k prostředkům

Přístup k prostředkům se liší podle toho, zda se k nim přistupuje ze zdrojového kódu, nebo zda chceme získat prostředek v XML souboru.

- **Přístup k prostředkům ve zdrojovém kódu** [53] – Ve zdrojovém kódu se k prostředkům přistupuje pomocí statické třídy R, která obsahuje dostupné prostředky pro konkrétní aplikaci. Následující příklad ukazuje načtení řetězce obsahujícího titulek hlavního menu:

```
R.string.mainmenu_title
```

- **Přístup k prostředkům v XML souboru** [53] – K získání prostředků v XML souboru se používá znak @, který nahrazuje třídu R z předchozího příkladu. Získání prostředku v XML souboru vypadá následovně:

```
@string/mainmenu_title
```

4.4.2. Typy prostředků

Všechny vlastní prostředky, které bude aplikace využívat, by měly být uloženy v některém z podporovaných adresářů, které se nacházejí v adresáři res/. Tabulka 2 zobrazuje názvy podporovaných adresářů, které se mohou nacházet v adresáři res/.

Název složky	**Popis uloženého prostředku**
animator/	XML soubory definující animace vlastností objektů (podporované až od verze Android 3.0 – API level 11)
anim/	XML soubory definující doplňkové animace, používané pro objekty typu `View`.
color/	XML soubor, ve kterém lze definovat vlastní barvy a vytvořit si tak barevnou paletu vhodnou pro aplikaci
drawable/	Soubory obrázků (s příponami .png, .9.png, .jpg, .gif) nebo XML soubory, které definují vlastnosti a strukturu některého drawable objektu, například nadefinování vzhledu widgetu (nastavení ostrosti hran, definování barevného přechodu, definování barev při určitých stavech apod.).
layout/	XML soubory definující návrhy uživatelského rozhraní.
menu/	XML soubory, ve kterých lze definovat různé menu programu (menu vlastností, kontextové menu atd.).
raw/	Adresář, do kterého lze ukládat jakékoliv libovolné soubory. K těmto souborům lze přistupovat pomocí metody `Resources.openRawResources()`, které je předáno ID prvku, který chceme načíst např. `R.raw.vlastni_soubor`. Někdy je ale potřeba přistupovat a načíst soubor ne pomocí ID ale podle vlastního názvu souboru. Pro tento případ lze využít pro uložení souboru složku assets/, ke kterému pak přistupujeme pomocí metody `getAssets()`.

values/	XML soubory, které mohou obsahovat jednoduché hodnoty, řetězce, pole řetězců a různé konstanty. Na rozdíl od ostatních složek, ve kterých definované prostředky odpovídají názvům souborů, soubory uložené v této složce mohou definovat více zdrojů zároveň. V každém souboru je jako prostředek definován každý element, který se nachází v kořenovém elementu `<resources>`. Nebylo by ovšem příliš praktické v jednom XML souboru definovat v elementu `<resources>` více typů prostředků (například barvy a řetězce míchat v jednom souboru), proto je vhodné jednotlivé typy zdrojů oddělit do souborů. Pro tyto soubory by měla platit určitá konvence: • arrays.xml – pro prostředky typu pole • colors.xml – pro konstanty uchovávající barvy • dimens.xml – pro hodnoty rozměrů • strings.xml – pro řetězcové konstanty • styles.xml – pro uživatelem definované styly
xml/	Složka pro libovolné XML soubory, ze kterých mohou být načítána data při běhu programu. K XML souboru se přistupuje s využitím metody `Resources.getXML()` a ID získaného pomocí `R.xml.vlastni_xml_soubor`.

Tabulka 2: Názvy podporovaných složek pro ukádání vlastních zdrojů[52]

4.4.3. Alternativní prostředky

Vzhledem k tomu, že aplikace pro systém Android mohou být nainstalovány na mnoha rozdílných zařízeních, je možné k výše uvedeným názvům adresářů přidat ještě konfigurační kvalifikátor.[52] Ten dovoluje aplikaci zvolit vhodný prostředek, který nejlépe odpovídá aktuální konfiguraci zařízení. Příkladem mohou být dva rozdílné návrhy uživatelského rozhraní, z nichž se bude volit podle toho, zde je orientace displeje na šířku či na výšku. Názvy adresářů, spolu s konfiguračními kvalifikátory, budou vypadat následovně:

- `layout-port` – pro návrh, který aplikace použije při vertikální orientaci displeje
- `layout-land` – pro návrh, který aplikace použije při horizontální orientaci displeje

V obou adresářích bude název XML návrhu uživatelského rozhraní stejný, jen jeho obsah bude přizpůsoben tak, aby vypadal co nejlépe pro požadovanou konfiguraci.

Podobných kvalifikátorů je poměrně velké množství a umožňují, kromě výše zmíněné orientace, rozlišovat i různé prostředky pro různé velikosti displejů, pro displeje s různou hustotou bodů, pro rozdílnou úroveň API nebo pro volbu jazykového nastavení aplikace podle aktuálního jazykového nastavení zařízení. Jednotlivé kvalifikátory a možnosti jejich kombinací jsou vyjmenovány a podrobně popsány v dokumentaci.[52]

5. SVG grafika a Android

Protože cílem této práce bylo vytvořit mobilní aplikaci tak, aby dokázala pracovat s vektorově vykreslovanými mapami, v následujících odstavcích bych krátce popsal, co je vektorový grafický formát SVG[54] a jak je s ním možné pracovat v aplikacích systému Android.

5.1. Grafický formát SVG

SVG (Scalable Vector Graphics) je grafický vektorový formát, který byl původně vyvinut pro oblast webové grafiky, avšak rozšířil se i do mnoha jiných oblastí a postupně se stal standardem pro přenos grafiky mezi různými platformami. Protože implementace SVG formátu nebyla vždy bezproblémová, formát se postupně rozdělil na dvě specifikace SVG.[55] První je SVG Basic, která je cílená na větší zařízení nebo zařízení s většími paměťovými prostředky (PDA a podobně) a druhou specifikací je SVG Tiny, která je určena pro zařízení s menšími paměťovými prostředky (klasické mobilní telefony).

Ve své podstatě má SVG soubor strukturu XML souboru, který k vykreslování využívá sadu geometrických prvků, křivky (cesty) a další funkce jako například pokročilou práci s textem a podobně.

5.2. Mobilní OS a podpora SVG

V současné době, nové mobilní operační systémy postrádají nativní podporu SVG formátu, tudíž pro využívání tohoto formátu je nutné většinou využívat knihovny vývojářů třetích stran, které přidávají funkce pro práci s SVG grafikou.

V případě systému Android tomu není jinak a přímo ze strany systému neexistuje nativní podpora formátu SVG. Naštěstí existuje řešení právě v použití neoficiální knihovny, která implementuje funkce, kterými lze tento formát zpracovat.

5.3. Knihovna svg-android

Knihovna svg-android[56] umožňuje připojit SVG soubor k aplikaci a využít třídu `Canvas` k jeho vykreslení. Knihovna dokáže zpracovat specifikaci SVG Basic 1.1 a všechny její podmnožiny, ovšem ne úplně se všemi jejími vlastnostmi. Nedokáže například zpracovat úplně všechny vlastnosti fontů, připojené rastrové obrázky, skripty,

animace a další pokročilejší vlastnosti. Další mírnou nevýhodou je i fakt, že v případě zobrazování SVG obrázku pomocí této knihovny na displejích s různými parametry, dochází k automatickému přizpůsobování obrázku rozměrům displeje, aniž by měl vývojář přehled, v jakém poměru byl obrázek přizpůsoben

.

6. Ukládání dat

Systém Android poskytuje několik možností jak ukládat perzistentní data určená konkrétní aplikaci či aplikacím. Výběr nejvhodnějšího způsobu se odvíjí od toho, zda uložená data budou sdílená mezi několika aplikacemi nebo zda se jedná o interní data jedné konkrétní aplikace.

6.1. Sdílené preference (Shared Preferences)

V případě sdílených preferencí[57] se jedná o ukládání dat v podobě páru klíč – hodnota, která jsou k dispozici konkrétní aplikaci (aktivitě) a nejsou sdílena jiným aplikacím. K získání nebo modifikaci sdílených preferencí se využívá rozhraní `SharedPreferences` a k získání konkrétního objektu aplikace (aktivity) těchto dvou metod:

- `getSharedPreferences()` – pro získání objektu sdílených preferencí pro více aktivit
- `getPreferences()` – pro získání objektu preferencí pro aktuální aktivitu

Do získaného objektu preferencí lze uložit jakýkoliv primitivní datový typ, kterému je nutné přiradit klíč datového typu String.

```
SharedPreferences settings = getSharedPreferences(JMENO_PREF,0);
SharedPreferences.Editor editor = settings.edit();
editor.putString("klic","hodnota");
editor.commit();
```

Část zdrojového kódu ukazuje, jakým způsobem se s preferencemi pracuje. Po vložení či modifikování jakékoliv hodnoty do objektu `SharedPreferences` je vždy nutné potvrdit provedení změn pomocí metody `commit()`. Pokud se neprovedlo volání této metody, změny provedené výše se vůbec neprovedou. Nová data jsou ihned po potvrzení k dispozici, tudíž pokud jedna část aplikace změní některou hodnotu preference, jiná část aplikace bude mít tuto hodnotu ihned k dispozici.

6.2. Interní úložiště

Interním úložištěm[58] může být jakýkoliv soubor, který uchovává privátní data konkrétní aplikace. Takto uložená data jsou k dispozici právě jen konkrétní aplikaci a vymazáním aplikace ze zařízení dojde k vymazání i všech jejích interních dat. V tomto případě se tedy nejčastěji jedná o data uložené v některém adresáři prostředků.

Zvláštním druhem interních dat jsou dočasná data aplikace, cache data. Cache data se ukládají do dočasných souborů, které jsou vymazány po ukončení aplikace nebo v případě, že zařízení potřebuje více paměti.

6.3. Externí úložiště

V případě využívání externího úložiště[59] jsou data uložena mimo aplikaci, nejčastěji do jiného umístění v interní paměti zařízení nebo na paměťovou kartu. Nebezpečím pro takto uložená data je jejich dostupnost i jiným aplikacím, které je mohou libovolně modifikovat či úplně smazat.

Přestože jsou data aplikace uložena externě mimo aplikaci, při odinstalování dojde k jejich odstranění. Pokud je potřeba uložit data, která musí být zachována i po odstranění aplikace, které jsou součástí, musí se data uložit do jednoho z veřejných adresářů. Mezi tyto adresáře patří například Music/, Pictures/, Download/ a nacházejí se zpravidla v kořenovém adresáři úložiště.

6.4. SQLite databáze

Systém Android poskytuje plnou podporu SQLite databáze[60], která může poskytovat data jakékoliv aktivitě či procesu v rámci aplikace, ovšem nelze se k ní připojit mimo aplikaci.

V systému Android slouží SQLite databáze jako interní úložiště dat, které se vytváří vždy za běhu aplikace. Z toho důvodu je zřejmý jeden celkem podstatný problém a to je problém využití externě vytvořené SQLite databáze. Způsobu, jak lze tento problém obejít a využívat externě vytvořenou databázi se budu věnovat v kapitole 8 při popisu aplikace.

6.5. Síťové úložiště

Pokud má aplikace přístup k internetovému připojení a zároveň je pro aplikaci k dispozici webová služba, která umožňuje ukládat či vracet nějaká perzistentní data, je možné využít, jako alternativu k ostatním úložištím i síťové úložiště.[61] K síťovým operacím poskytuje systém Android třídy, které se nacházejí v balíkách `java.net.*` a `android.net.*`.

7. Vlákna

V ideálním případě by měla být každá aplikace systému Android dostatečně rychlá, aby z pohledu uživatele pracovala zcela plynule. Ideální doba odezvy na jakýkoliv uživatelský vstup nebo obecně na jakoukoliv interakci s uživatelem se pohybuje přibližně do 200 a 300 ms. V případě systému Android, pokud aktivita neodpovídá po dobu pěti sekund, třída `ActivityManager` ji automaticky ukončí.

Problémům pomalé nebo žádné odezvy aplikace se dá předejít použitím vláken, která mohou provádět časově náročné operace a tím někdy podstatně zrychlit činnost aplikací. Systém Android dává vývojářům k dispozici několik druhů vláken.[62]

7.1. Vlákno uživatelského rozhraní

Vlákno uživatelského rozhraní[62] je vždy automaticky spouštěno s určitou aktivitou a stará se o vykreslování všech objektů uživatelského rozhraní. Velice důležité je, že veškeré změny uživatelského rozhraní, vyvolání upozornění, ukazatele průběhu a podobně, se musí provádět přímo z tohoto vlákna. Jakýkoliv přímý přístup k některému objektu typu `View` z jiného než tohoto vlákna, končí zpravidla chybou programu.

Pokud má být aplikace dostatečně rychlá, neměly by se v tomto vlákně zpracovávat časově náročné úkoly, ale měly by se přenechat některému vláknu na pozadí. Časově náročné operace v tomto vlákně zdržují vlákno od vykreslení uživatelského rozhraní, čímž brání plynulosti aplikace.

7.2. Pracovní vlákno

V případě pracovního vlákna[63] se využívá třídy `Thread` a rozhraní `Runnable`, implementujícího metodu `run()`, ve které se provádí definované operace. Z takto spuštěného vlákna však nelze komunikovat přímo s žádným objektem uživatelského rozhraní. Ke komunikaci s objektem uživatelského rozhraní je nutné využít několik metod:

- **Activity.runOnUiThred(Runnable)** – Tento způsob je asi nejméně vhodný, protože jednoduše spustí uživatelské vlákno přímo ve vlákně uživatelského rozhraní.

- **View.post(Runnable)** – Tato metoda přidá objekt typu `Runnable` do fronty a o jeho zpracování se postará vlákno uživatelského rozhraní.
- **Využití objektu typu Handler** – Z těchto tří možností se jedná o nejflexibilnější řešení, jelikož je možné komunikovat s uživatelským vláknem pomocí zpráv. V tomto případě se v aktivitě aplikace zaregistruje objekt typu `Handler`, kterému jsou z pozadí zasílány patřičné zprávy o aktualizaci některého objektu uživatelského rozhraní.

7.3. Asynchronní vlákno

Lepším řešením pro použití vlákna na pozadí komunikujícího s uživatelským rozhraním je využít třídu `AsyncTask`.[63] Tato třída implementuje celkem čtyři metody, z nichž tři slouží ke komunikaci s uživatelským rozhraním a jedna pracuje jako klasické pracovní vlákno. Třída využívá generika, tudíž je nutné pro tři metody specifikovat datové typy. Vlastní třída asynchronního vlákna se vytváří jako podtřída `AsyncTask` takto:

```
private class Task extends AsyncTask<String, Void, Integer >
```

Zvolené datové typy lze samozřejmě volit i jiné nebo nastavit všechny parametry jako objekty `Void`, tudíž se vláknu nebudou předávat žádná data. První parametr předává data potřebná k provedení některé operace, než se vlákno spustí. Druhý parametr se využívá k indikaci průběhu vlákna. Nakonec třetí parametr obsahuje data, která se předávají po ukončení vlákna. První dva datové typy jsou navíc parametry o více prvcích, z toho důvodu je nutné v těchto parametrech předávat pole objektů.

Samotné metody, které jsou součástí třídy a mohou být implementovány, jsou tyto:

- **onPreExecute**() – Metoda se volá před metodou `doInBackground()` a slouží nejčastěji k inicializaci proměnných nebo spuštění ukazatele průběhu před samotným prováděním operací ve vlákně. Metoda je volána z vlákna uživatelského rozhraní a může s ním komunikovat.
- **doInBackground**() – Provádí operace jako vlákno na pozadí a nemůže komunikovat přímo s uživatelským rozhraním. K průběžné komunikaci uživatelského rozhraní může využít metodu `onProgressUpdate()`.

- **onPostExecute**() – Tato metoda se provede po skončení operací na pozadí. Je jí předána hodnota vrácena metodou `doInBackground()` nebo je v ni možné zrušit ukazatele průběhu. Tato metoda je volána z vlákna uživatelského rozhraní a může s ním komunikovat.
- **onProgressUpdate**() – Slouží k průběžné komunikaci s uživatelským vláknem a může hlídat průběh operací prováděných na pozadí.

Pro použití asynchronního vlákna platí několik pravidel:

- Instance vlákna musí být vytvořena ve vlákně uživatelského rozhraní.
- Instance se spouští metodou `execute(Parametry…)`, také ve vlákně uživatelského rozhraní.
- Jednotlivé metody vlákna se nesmějí volat manuálně.
- Každá instance vlákna smí být spuštěna pouze jednou, při vícenásobném spuštění dojde k chybě.

8. Sdílení a update Android aplikací

Většina aplikací určených pro operační systém Android jsou uživatelům k dispozici prostřednictvím webu Google Play.[64] Ještě před tím než je aplikace nahrána na tento web, je vhodné provést některé úpravy v souboru `AndroidManifest.xml`.

První úpravou by měla být specifikace verze aplikace.[65] Ta se provádí přidáním následujících řádek do `AndroidManifest.xml`:

- `android:versionCode` – Jedná se o celočíselnou hodnotu, pomocí které se zjišťuje, zda je na webu Google Play vyšší verze aplikace, než je nainstalována v zařízení. Aktualizace jsou zjišťovány automaticky, jakmile je zařízení připojené k internetu. Parametr versionCode není uživateli aplikace zobrazován.
- `android:versionName` – V tomto případě verzi aplikace určuje libovolný řetězec, který si může vývojář aplikace určit. Tato hodnota bývá uživateli zobrazována, ovšem nemá žádný vliv na vyhledávání případných aktualizací aplikace na webu Google Play.

Kromě specifikace verze, by měla být u každé aplikace určena i verze systému Android, pro kterou je aplikace vytvořena a na které byla otestována. Určení verze systému lze provést několika způsoby.

- `android:minSDKVersion` – minimální úroveň API systému, na které může být aplikace instalována
- `android:targetSDKVersion` – úroveň API systému, pro kterou je aplikace přesně navržena
- `android:maxSDKVersion` – maximální úroveň API systému, na které může být aplikace nainstalována

K uvolnění aplikace ostatním uživatelům je nutné všechny součásti aplikace zabalit do souboru s příponou .apk, pomocí kterého je možné aplikaci nainstalovat do zařízení se systémem Android. Součástí tohoto souboru jsou zkompilované zdrojové kódy, veškeré prostředky využívané aplikací a soubor manifest. Při samotném vytváření .apk souboru je nutné aplikaci ještě digitálně podepsat, což lze jednoduše provést například

pomocí vývojového prostředí Eclipse. Pravidla pro digitální podpis lze najít v příslušné dokumentaci.[66]

Není však podmínkou, že veškeré aplikace musí být dány uživatelům k dispozici pouze prostřednictvím webu Google Play. Vytvořenou aplikaci je možné uložit a sdílet i na osobním webu. Jelikož se ale jedná o instalaci aplikace neoficiální cestou, je nutné mít v nastavení zařízení povolenu instalaci aplikací z neznámého zdroje. V opačném případě by totiž instalace aplikace z privátního webu skončila chybovým hlášením.

9. Aplikace – Mapový systém pro orientaci v budově

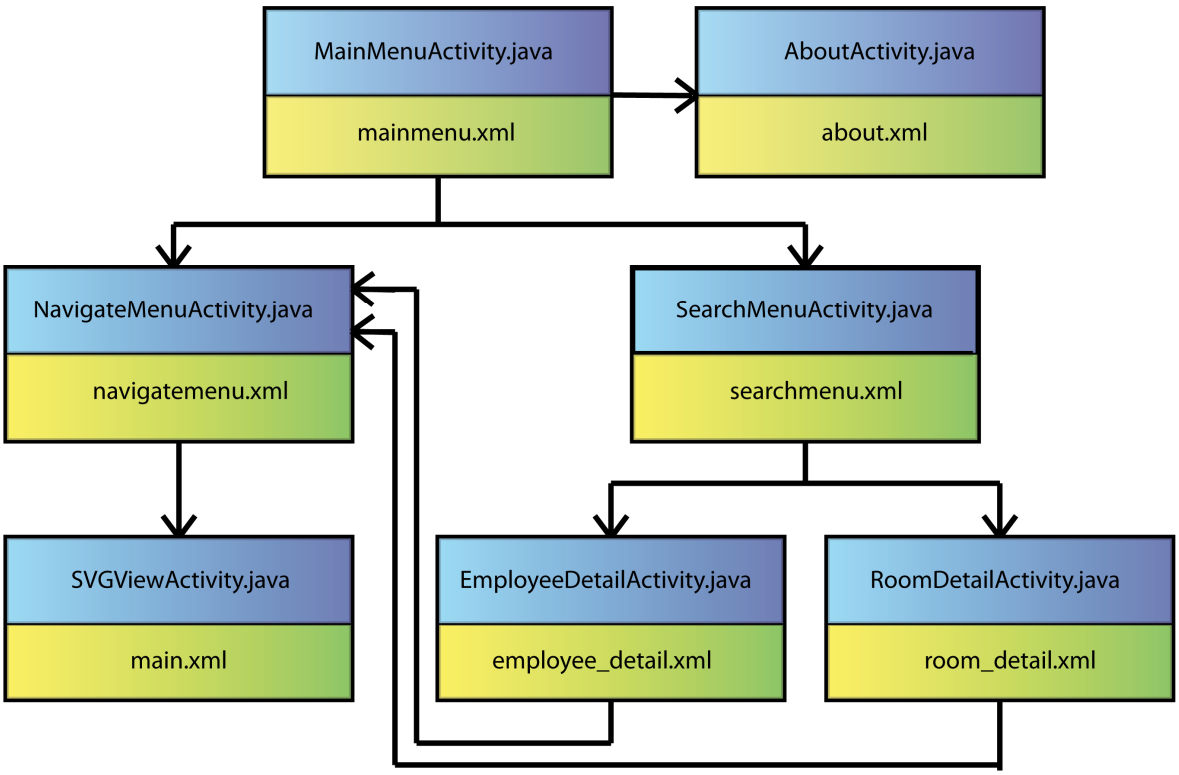
Cílem této práce bylo vytvořit mobilní aplikaci, která by posloužila k orientaci v budovách Západočeské univerzity v Plzni. Nejprve bych krátce vyjmenoval jednotlivé funkce a vlastnosti vytvořené mobilní aplikace.

- Vygenerování trasy k cíli dle uživatelem zadaného počátečního a koncového bodu
- Vyhledávání trasy v režimu offline
- Podpora mapových podkladů ve formátu SVG
- Možnost zoomu a pohybu po mapě pouze jedním prstem
- Možnost volby bezbariérového přístupu
- Možnost vyhledávání zaměstnanců a místností
- Zobrazování detailů o zaměstnancích a místnostech
- Možnost volání na telefonní číslo konkrétního zaměstnance
- Možnost spuštění emailového klienta k odeslání zprávy konkrétnímu zaměstnanci

Aplikace je cílená na všechny mobilní telefony využívající systém Android verze 1.6 a vyšší. Zdrojový kód je součástí balíku `ppredota.android.navigation`, který obsahuje ještě několik vnořených balíků pro přehlednější strukturu aplikace.

9.1. Struktura aplikace

Součástí aplikace je celkem sedm aktivit, jejichž třídy se nacházejí v balíku `ppredota.android.navigation.activities`. Pro každou aktivitu jsou vytvořeny vlastní XML návrhy. Navíc pro aktivitu hlavního menu je vytvořen návrh, který je použit při orientaci zařízení na šířku. Aktivity jsou spolu propojeny pomocí konkrétních záměrů a hierarchie aplikace je na obrázku 3.



Obrázek 3: Hierarchie aktivit aplikace

Z hierarchie na obrázku 3 je vidět, že aplikace začíná v hlavním menu, ze kterého je možné pokračovat k menu navigace a vyhledat zadanou trasu, k aktivitě umožňující vyhledávat zaměstnance nebo místnosti a zjišťovat o nalezených položkách detaily, nebo je možné vyvolat aktivitu zobrazující detaily o samotné aplikaci. Navíc je možné z aktivit zobrazujících detaily zaměstnance či místnosti plynule přejít k samotnému menu navigace, kde jako cíl bude automaticky zadána místnost, o které byly zobrazeny detailní informace.

10. Uživatelské rozhraní aplikace

Uživatelské rozhraní aplikace je, až na pár výjimek, vytvořené čistě za pomocí XML návrhů. Každá aktivita v aplikaci má vytvořena svůj XML návrh, přesně tak jak bylo vidět na obrázku 3 v předchozí kapitole. K vytvoření návrhů uživatelského rozhraní bylo záměrně použito více druhů kontejnerů (`LinearLayout`, `RelativeLayout`, `TableLayout` a `ScrollView`) proto, aby bylo možné demonstrovat práci s různými typy kontejnerů.

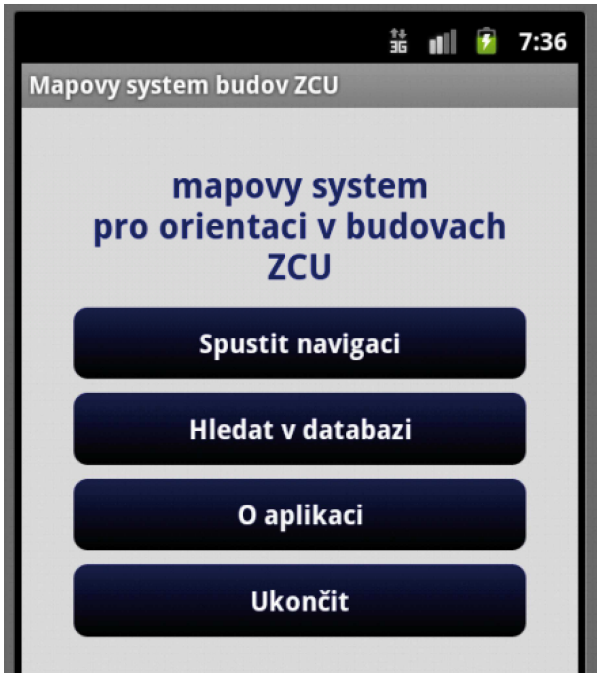
10.1. Hlavní menu

Součástí hlavního menu jsou tři tlačítka, která odkazují na ostatní aktivity, a čtvrté tlačítko umožňující aplikaci ukončit. Součástí návrhu je i textové pole zobrazující název aplikace.

Pro hlavní menu aplikace byly vytvořeny dva různé návrhy, jeden návrh pro orientaci telefonu na výšku, druhý pro orientaci na šířku.

10.1.1. Layout pro orientaci na výšku

V případě návrhu při orientaci telefonu na výšku jsou tlačítka seřazena pod sebou a nad všemi tlačítky je vložen název aplikace. Vzhled menu při této orientaci je zobrazen na obrázku 4.



Obrázek 4: Hlavní menu aplikace

K vytvoření tohoto návrhu byl použit kontejner typu `LinearLayout`. Obsah návrh `mainmenu.xml`, resp. jeho část, vypadá následovně:

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:orientation="vertical"
    android:padding="30dp"
    android:background="@color/white2">

    <TextView
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:gravity="center"
        android:layout_gravity="center"
        android:layout_marginBottom="20dp"
        android:text="@string/mainmenu_title"/>
     <Button
         android:id="@+id/button_navigate"
         android:layout_width="fill_parent"
         android:layout_height="wrap_content"
         android:layout_gravity="center"
         android:text="@string/navigate_label"
         android:layout_marginBottom="5dp"
         android:textStyle="bold"
         android:textSize="15dp"
         android:textColor="@android:color/white"
         android:background="@drawable/custombutton"/>
```

Uvedený XML kód návrhu mainmenu.xml je zkrácený pouze na widget `TextView` a jedno tlačítko, tedy widget typu `Button`. Zbylá tři tlačítka jsou vytvářena naprosto shodně s prvním. Co se týče atributů v XML návrhu, jejich použití je velice intuitivní, jelikož podle jejich jména lze snadno odhadnout, k jakému nastavení slouží. Přesto bych důležité a často se opakující atributy a jejich možné hodnoty přiblížil.

- `layout_width`, `layout_height` – atributy určující velikost objektů, místo přesného rozměru se používají častěji dvě hodnoty
    - `fill_parent`: Nastaví velikost objektu tak, že zabere plochu objektu (rodiče), ve kterém se nachází.

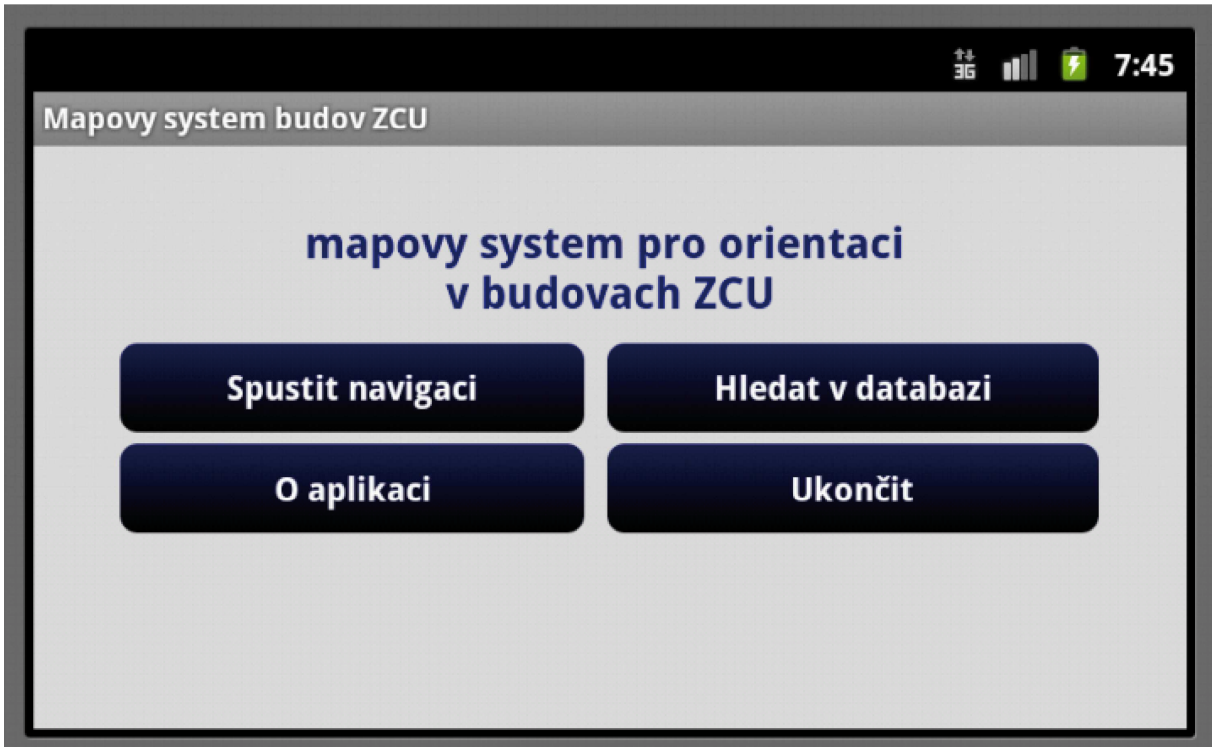
  - `wrap_content`: Velikost objektu je určena a přizpůsobena jeho obsahem

- `orientation` – nastaví orientaci, ve které se budou objekty řadit za sebou
  - `vertical`: Objekty návrhu budou řazeny od shora směrem dolů
  - `horizontal`: Objekty návrhu se budou řadit směrem zleva doprava.

Z uvedené ukázky návrhu pomocí kontejneru `LinearLayout` jsou dobře vidět vlastnosti tohoto kontejneru. Nejprve je určen samotný kontejner, jeho rozměry a směr jakým se budou jednotlivé objekty do kontejneru vkládat. Poté je možné vkládat jednotlivé objekty a nastavovat jejich vlastnosti. Jednotlivé kontejnery je možné do sebe zanořovat a měnit tak orientaci dalších objektů, oddělovat jednotlivé části návrhu a podobně.

10.1.2. Layout pro orientaci na šířku

Přestože by šel tento návrh vytvořit pohodlně jen pomocí kontejneru `LinearLayout`, zvolil jsem nakonec jeho kombinaci s kontejnerem `TableLayout`, abych mohl demonstrovat práci i s jiným typem kontejneru. Na obrázku 5 je vidět, jak vypadá návrh hlavního menu při orientaci na šířku.



Obrázek 5: Vzhled hlavního menu při orientaci telefonu na šířku

Jednotlivá tlačítka návrhu byla seřazena tak, aby co nejlépe pokryla plochu kontejneru, a zároveň se nemůže stát, že by při orientaci telefonu na šířku, některá tlačítka nebyla vidět. Způsob vytvoření podobného návrhu s využitím kontejneru `TableLayout` ukazuje následující část XML kódu.

```
<LinearLayout <!-- kód shodný s předchozím návrhem  --> >
      <TextView <!-- kód shodný s předchozím návrhem  --> />

      <TableLayout
      xmlns:android="http://schemas.android.com/apk/res/android"
          android:layout_width="fill_parent"
          android:layout_height="fill_parent"
          android:stretchColumns="0,1">
          <TableRow
              android:gravity="center_horizontal">
              <Button
                 android:id="@+id/button_navigate"
                 android:layout_width="fill_parent"
                 android:layout_height="fill_parent"
                 android:text="@string/navigate_label"
                 android:textStyle="bold"
                 android:textSize="15dp"
                 android:textColor="@android:color/white"
                 android:background="@drawable/custombutton"
                 android:layout_margin="10dp"/>
               <Button
                 <!-- kód shodný s předchozím widgetem Button --> />

          </TableRow>
          <TableRow
              android:gravity="center_horizontal">
              <Button
                     android:id="@+id/button_about"
```

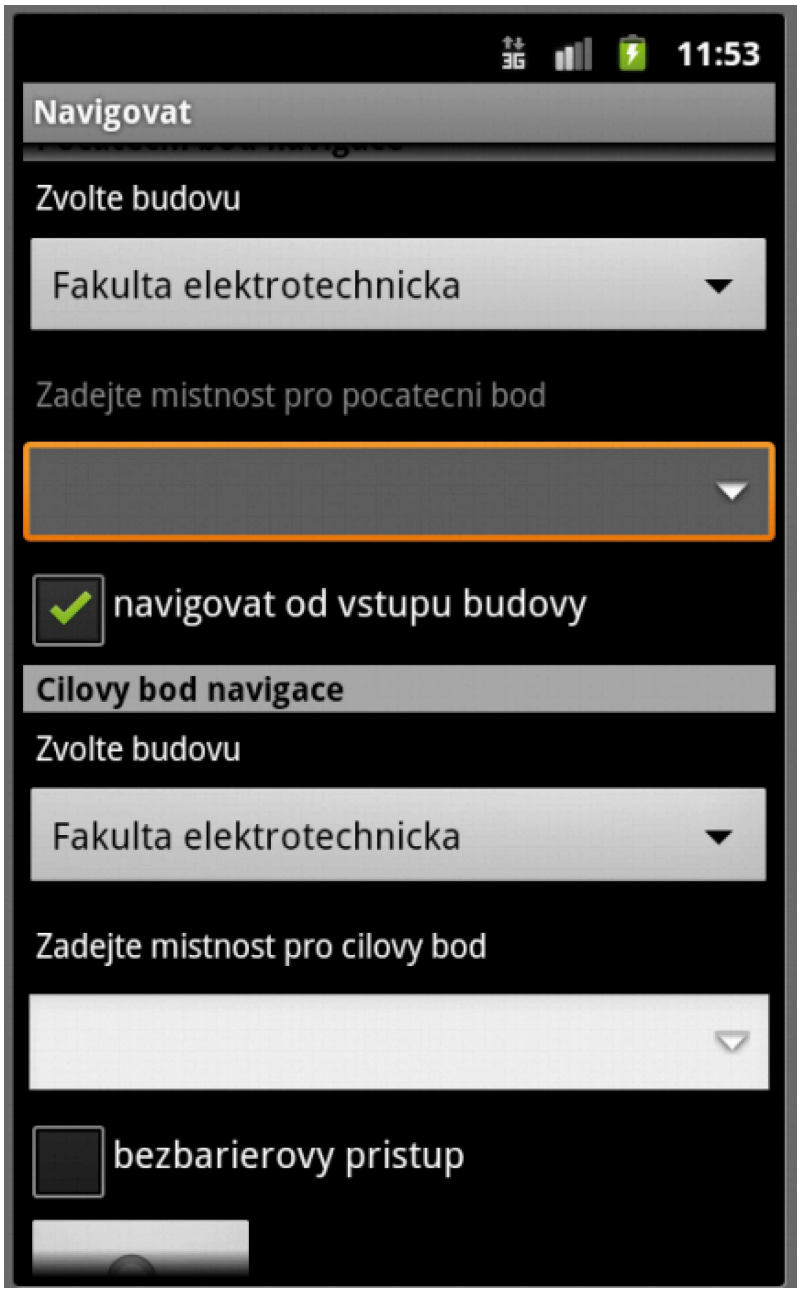
V případě tohoto kontejneru se návrh vytváří pomocí objektů, které se nacházejí mezi elementy `<TableRow></TableRow>`. Každý element `TableRow` specifikuje objekty nacházející se v jedné řádce. Počet sloupců v řádce nelze přímo určit, ale je určen počtem objektů, v tomto případě se v řádce nacházejí dvě tlačítka, tudíž řádek obsahuje dva sloupce, které jsou indexovány od nuly.

Implicitně jsou jednotlivé sloupce široké tak, jako je standardní šířka objektu, který se v něm nachází. V případě, že je potřeba prvek ve sloupci nějakým způsobem roztáhnout, je možné určit sloupce, které se mají takto přizpůsobit. To se provádí atributem `stretchColumns` a jeho hodnota odpovídá sloupci, který má být roztažen, případně je možné zadat více sloupců oddělené čárkami.

V kontejneru `TableLayout` je samozřejmě k dispozici celá řada atributů na přizpůsobení sloupců obsahu a operace s objekty v řádkách a sloupcích, ovšem vyjmenovávat a popisovat všechny atributy je nad rámec této práce, veškeré atributy lze najít v příslušné dokumentaci.[45]

10.2. Okno pro zadání vstupů navigaci

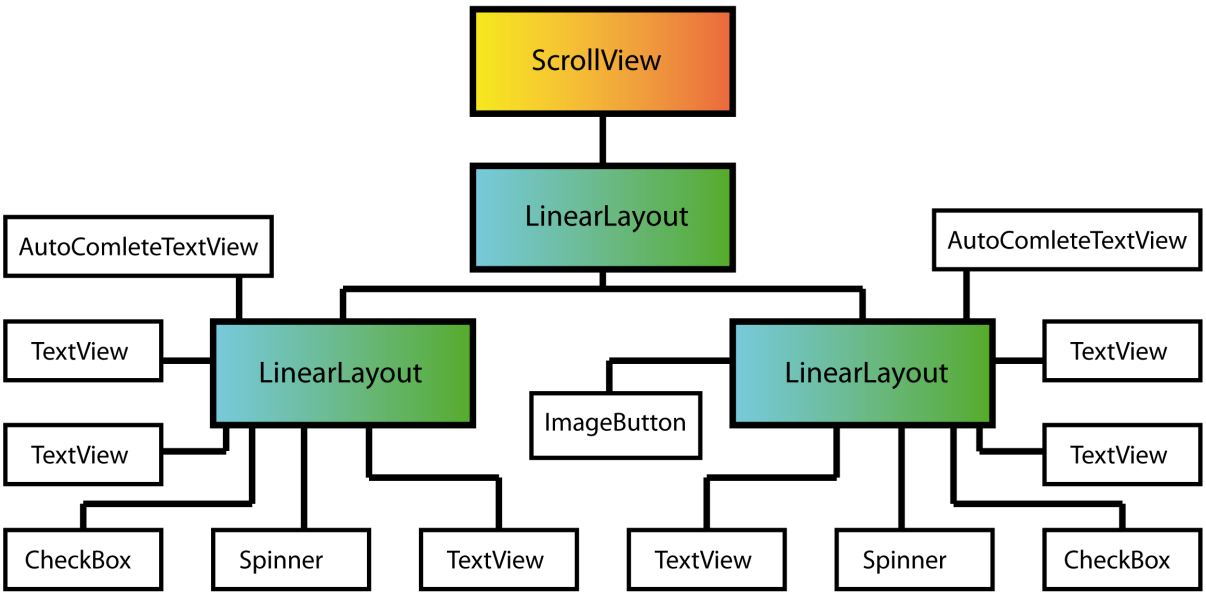
Tento návrh je o něco komplikovanější než předchozí návrhy, vzhledem k tomu, že obsahuje mnohem více objektů. Návrh pro zadání vstupů pro navigaci obsahuje dvě rozbalovací nabídky, widgety `Spinner`, které slouží k volbě počáteční a cílové budovy, dále dvě textová okna s automatickým dokončováním textu (`AutoCompleteTextView`), dvě zaškrtávací tlačítka pro možnost volby navigace od vstupu budovy a pro nastavení bezbariérového přístupu. Zadané vstupy se potvrzují tlačítkem pro vyhledávání. Vzhled návrhu je na obrázku 6.



Obrázek 6: Návrh pro zadání vstupů pro navigaci

Na návrhu je z prvního pohledu vidět, že obsahuje tolik objektů, že displej telefonu ani nedokáže celý návrh zobrazit. Z toho důvodu byl pro tento XML návrh využit kontejner `ScrollView`, který umožňuje posouvat obsah okna tak, aby byly zobrazeny všechny jeho části. Díky tomu není ani nutné řešit dva různé návrhy pro různé orientace, v případě orientace displeje na šířku bude sice zobrazeno méně objektů, ale ke zbylým objektům se může uživatel posunout právě díky `ScrollView`.

V tomto případě bych vynechal přesnou strukturu XML návrhu, jelikož obsahuje atributy použité a popsané v dřívějších odstavcích. Zaměřil bych se spíše na hierarchii kontejnerů a způsob jejich zanořování. Obrázek 7 ukazuje, jak vypadá hierarchie kontejnerů pro tento konkrétní návrh.



Obrázek 7: Hierarchie kontejnerů v XML návrhu navigatemenu.xml

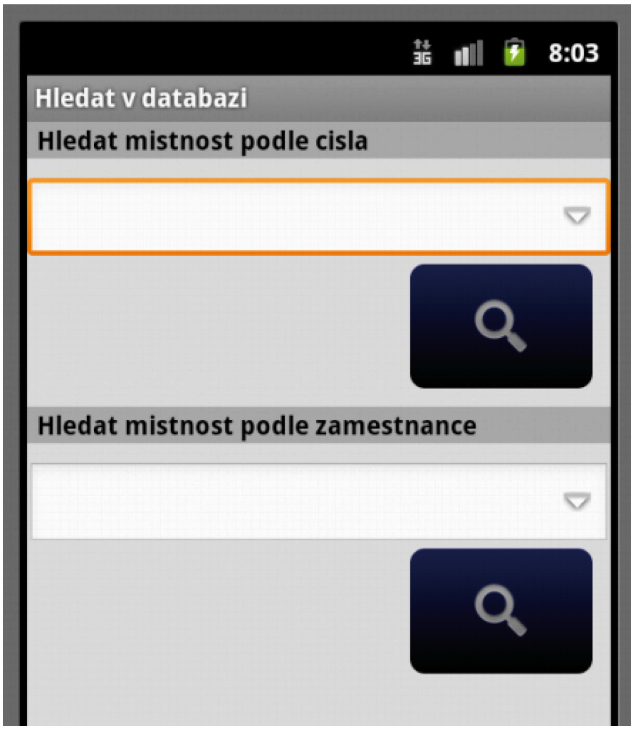
Z obrázku 7 je dobře vidět, že `ScrollView` se používá tím způsobem, že obaluje ostatní kontejnery v návrhu. To je z důvodu, že samotné `ScrollView` nemůže sloužit jako kontejner jednotlivým objektům okna a z toho důvodu je v hierarchii návrhů nejvýše.

Jak jsem uváděl, ostatní kontejnery mohou obsahovat jiné kontejnery a podobným způsobem se větvit. V tomto případě je návrh rozdělen tak, že každá část okna je sestavena ve vlastním kontejneru, tudíž je oddělena část pro zadání vstupních hodnot navigace a část pro zadání cílových hodnot navigace.

Větvení kontejnerů je nutné provádět tak, že každý nový kontejner vychází ze svého rodiče. Pokud by byl sestaven návrh tím způsobem, že dva elementy např. `LinearLayout`, jsou v návrhu registrovány bez společného předka nebo nevychází jeden z druhého, pokus o načtení takového návrhu do aktivity končí chybou programu.

10.3. Okno prohlížení záznamů

V této aktivitě je možné vyhledat jednotlivé položky, které obsahuje připojená SQLite databáze. První formulář slouží k vyhledávání detailů o místnostech, druhý formulář slouží k vyhledávání informací o zaměstnancích. Vzhledem k tomu, že tento návrh není nijak složitý, nebudu zde uvádět popis kódu v XML návrhu, ale pouze vzhled návrhu. Vzhled návrhu ukazuje obrázek 8.



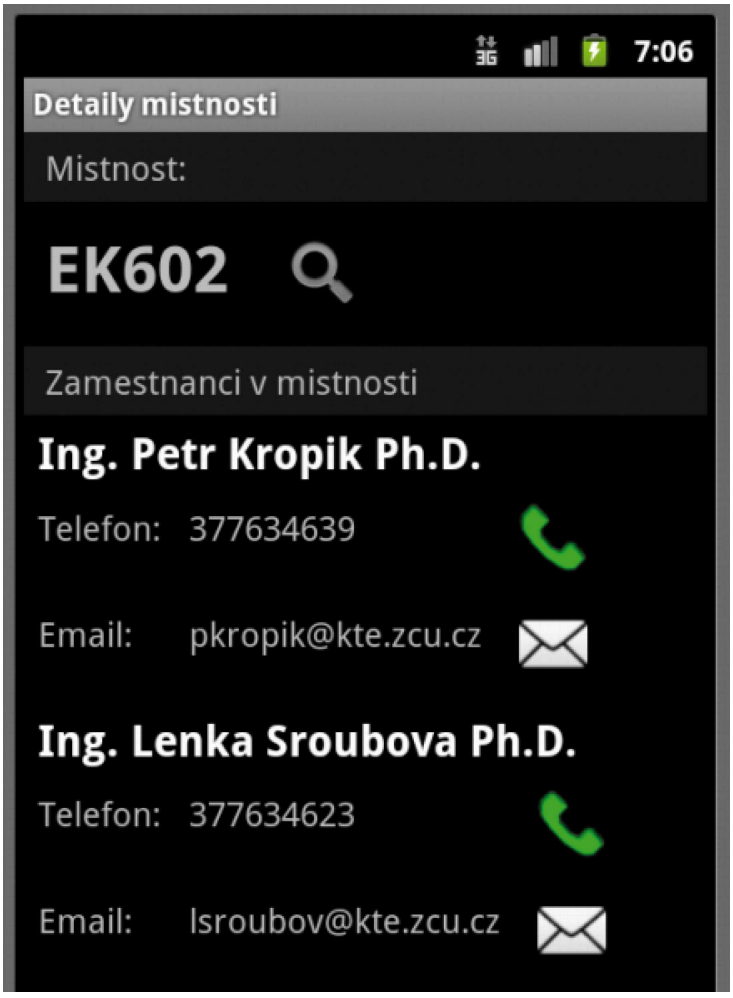
Obrázek 8: Návrh pro prohlížení záznamů

Pro vyhledávání místností je použit widget `AutoCompleteTextView`, který funguje jako formulář s nápovědou a podle počátečních písmen v rozbalovací nabídce pomáhá uživateli zvolit místnost. Vzhledem k tomu, že `AutoCompleteTextView` dovoluje hledání položek s nápovědou obsahující jen jedno slovo, pro vyhledávání zaměstnanců byl zvolen widget `MultiAutoCompleteTextView`, který umožňuje vyhledávání v položkách o více slovech.

10.4. Okno detailů hledaného záznamu

Součástí aplikace jsou dvě různé aktivity sloužící k zobrazení detailů, první aktivita s názvem `EmployeeDetailActivity.java` slouží k zobrazení detailních informací pro jednoho konkrétního zaměstnance. Druhá aktivita s názvem `RoomDetailActivity.java` naopak slouží k zobrazování detailů o místnosti, včetně výpisu zaměstnanců, kteří mají tuto místnost přidělenou jako kancelář. Vzhledem k tomu, že jsou si aktivity podobné, v této kapitole bych se věnoval popisu pouze aktivity `RoomDetailActivity.java`.

Vzhled návrhu okna pro zobrazování detailů pro zvolenou místnost ukazuje obrázek 9.



Obrázek 9: Okno pro zobrazování detailů místnosti

Jak je vidět na obrázku 9, prostřednictvím tohoto okna je možné ihned plynule přejít k navigaci do konkrétní místnosti pomocí tlačítka v podobě lupy hned vedle

vyhledané místnosti. Navíc, pokud jsou v databázi u konkrétního zaměstnance vyplněny i záznamy o telefonním čísle a emailové adrese, je možné zahájit hovor či spustit emailového klienta přímo prostřednictvím této aktivity.

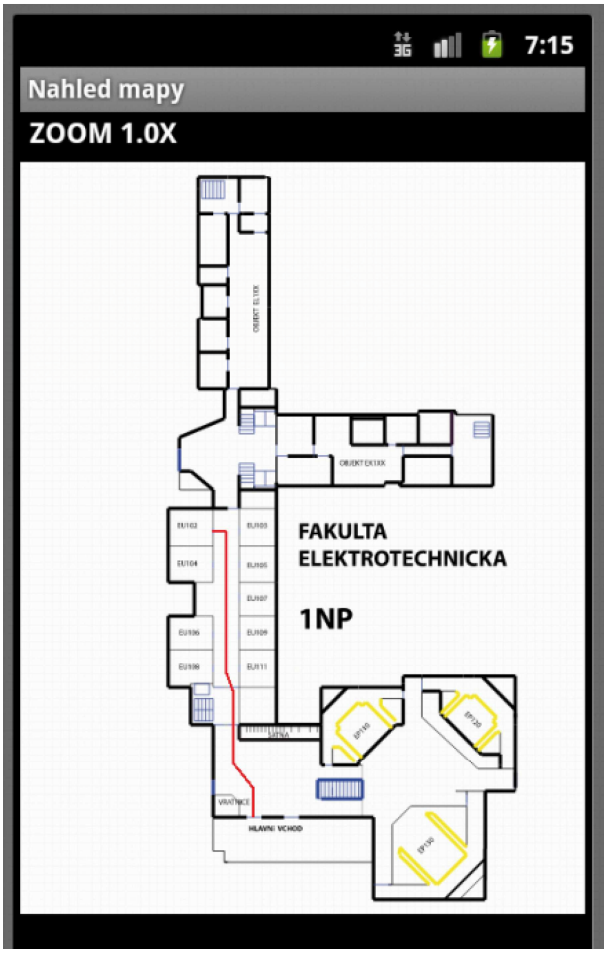
Samotná část návrhu uživatelského rozhraní, která slouží k výpisu seznamu zaměstnanců, kteří mají místnost přidělenou jako kancelář, nebyla vytvářena statickým způsobem pomocí XML návrhu, tak jako u předchozích návrhů. Vzhledem k tomu, že nikdy není přesně znám počet zaměstnanců v místnosti, je návrh vytvářen dynamicky přímo ve zdrojovém kódu za běhu aplikace.

11. Uživatelské rozhraní pro zobrazení mapy

Vzhledem k tomu, že hlavní funkce aplikace má být mapový systém s vyhledáváním trasy, bylo nutné vytvořit okno s prostředím k vhodnému zobrazení mapy. V následujících odstavcích podrobněji popíšu návrh uživatelského prostředí pro mapu, způsob načtení mapy do návrhu a možnosti interakce uživatele s oknem mapy.

11.1. Návrh uživatelského rozhraní

Uživatelské prostředí pro mapu je v podstatě jednoduché okno téměř bez objektů uživatelského rozhraní, z důvodu většího prostoru pro zobrazenou mapu. Jedinými objekty okna jsou ukazatel, který uživatele informuje, že je v režimu přibližování či oddalování mapy (tento objekt je v normálním režimu neviditelný), textové pole zobrazující hodnotu zoomu a objekty typu `ImageButton`, které zobrazují šipky po stranách mapy, v případě, že vygenerovaná trasa vede přes více map. Jak vypadá mapa v návrhu, ukazuje obrázek 10.



Obrázek 10: Návrh pro zobrazování SVG mapy

K návrhu mapy byl jako kontejner využit `RelativeLayout` a pro zobrazení mapy objekt typu `ImageView`, který umožňuje mapu vykreslovat. Důležitá část XML kódu návrhu uživatelského rozhraní vypadá následovně:

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout
xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:orientation="vertical"
    android:background="@android:color/black" >

    <ppredota.android.navigation.view.map.SVGView
        android:layout_centerInParent="true"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:id="@+id/svgview"/>
```

Z návrhu je vidět, že do kontejneru `RelativeLayout` je vložen jakýsi objekt `SVGView`. `SVGView` je třída, která dědí vlastnosti třídy `ImageView`, ale je přizpůsobená pro zobrazování konkrétního obsahu, v tomto případě SVG mapy, a spolu s jinými třídami umožňuje interakci mezi uživatelem a oknem mapy. Protože se jedná o uživatelem definovanou třídu, která se nachází ve složce zdrojového kódu, musí být v elementu XML návrhu uvedena kompletní cesta se všemi balíky až ke zvolené třídě.

Kontejner `RelativeLayout` není v tomto případě zvolen náhodně. Protože lze jakýkoliv jeho obsah vycentrovat přesně na střed pomocí atributu `android:layout_centerInParent`, je ideálním kontejnerem pro toto uživatelské rozhraní.

11.2. Načtení SVG mapy

Jak již bylo uvedeno, systém Android neobsahuje nativní podporu SVG formátu, z toho důvodu je k načtení SVG mapy využita knihovna svg-android ver1.1. Použití knihovny je poměrně jednoduché, stačí použít tento zdrojový kód:

```
int idMap = R.drawable.fel_1;
SVG svg = SVGParser.getSVGFromResource(getResources(), idMap);
setImageDrawable(svg.createPictureDrawable());
```

Metodou `getSVGFromResources()` ve třídě `SVGParser` je podle zadaného ID prostředku, v tomto případě parametr `idMap`, uložen SVG soubor v instanci třídy `SVG`. Pomocí metody `createPictureDrawable()` je objekt převeden na typ `Drawable`, což je objekt, který může být vykreslený metodou `onDraw(Canvas canvas)`. Před vykreslením je však nutné `Drawable` objekt nastavit jako obsah třídy `ImageView` metodou `setImageDrawable()`.

11.3. Přiblížení a pohyb po mapě

Interakce s uživatelským rozhraním pro zobrazení mapy probíhá pomocí dotykového displeje. Přibližování a oddalování mapy lze provést jedním prstem. Pro přiblížení stačí na moment přidržet prst na displeji, telefon odpoví krátkým zavibrováním a v pravém dolním rohu se objeví ikona režimu přiblížení. V tuto chvíli lze posunem v horizontálním směru doprava přibližovat obsah okna nebo posunem doleva oddalovat obsah okna. Uživatelské rozhraní umožňuje až pětinásobné zvětšení obsahu.

Jakmile je přiblížena určitá část okna, je jasné, že nelze na displeji zobrazit kompletně celý obsah. Z toho důvodu aplikace umožňuje posun okna jednoduchým posunem prstu po displeji. Tento posun je samozřejmě omezen hranicí obsahu okna.

11.3.1. Struktura uživatelského prostředí pro zobrazování map

Samotná struktura uživatelského prostředí pro zobrazování map se skládá ze tří tříd. Hlavní třídou, která zprostředkovává zobrazení mapy je třída `SVGView.java`. Důležité části kódu jsou zobrazeny v následujícím zdrojovém kódu.

```
public class SVGView extends ImageView implements Observer {

	private ViewState viewState;

	public void setViewState(ViewState viewState){
		if(this.viewState==null){
			this.viewState = viewState;
			this.viewState.addObserver(this);
		}
	}
	public void update(Observable observable, Object data) {
		invalidate();
	}
}
```

Třída implementuje rozhraní `Observer`. Pomocí tohoto rozhraní je možné tuto třídu přidat do seznamu tzv. sledovaných tříd, přesněji do seznamu objektů typu `Observable` (sledovatelných tříd). To se provádí pomocí metody `addObserver(Observer observer)`.

Ve zdrojovém kódu popsaném výše byla třída `SVGView` implementující rozhraní `Observer` registrována pomocí metody `this.viewState.addObserver(this)`. Aby to bylo možné, musí třída `ViewState`, jíž je instance `viewState`, dědit objekt typu `Observable`. Následující zdrojový kód ukazuje důležité metody třídy `ViewState`.

```
public class ViewState extends Observable {

	private float _x;
	private float _y;
	private float _zoom;

	public void setX(float x){
		_x = x;
		setChanged();
	}

	public void setY(float y){
		_y = y;
		setChanged();
	}

	public void setZoom(float zoom){
		_zoom = zoom;
		setChanged();
	}
}
```

Třída `ViewState` dědí třídu `Observable` a díky tomu v sobě může registrovat objekty implementující rozhraní `Observer`. Zároveň třída obsahuje sadu settrů, pomocí kterých lze registrovat změny konkrétních proměnných. Každá změna proměnné se ve třídě `ViewState` registruje pomocí metody `setChanged()`.

Poslední třídou uživatelského prostředí pro zobrazování map je třída `ViewListener`, která implementuje listener typu `OnTouchListener`, tedy listener reagující při dotyku displeje. Kód zobrazený níže je velice zestručnělý a ukazuje pouze jednoduchá volání metod. Kompletní zdrojový kód, včetně komentářů je součástí zdrojového kódu aplikace.

```
public class ViewListener implements OnTouchListener {

	public void setViewState(ViewState viewState){
		this.viewState = viewState;
	}
	public boolean onTouch(View view, MotionEvent event) {

		viewState.setX(x);
		viewState.setY(y);
		viewState.setZoom(zoom);
		viewState.notifyObservers();
	}
}
```

Metoda `onTouch(View view, MotionEvent event)` obsahuje settry třídy `ViewState`, díky kterým lze nastavit aktuální souřadnice a přiblížení okna. Nejdůležitější je ale v tomto případě metoda `notifyObservers()`, která potvrzuje změny provedené v instanci třídy `viewState`. Pokud byl volán alespoň jeden setter a nastaven tak příznak změny metodou `setChanged()` ve třídě `ViewState`, potvrzením metodou `notifyObservers` dojde k zavolání metody `update(Observable observable, Object data)` a tím k překreslení okna.

11.3.2. Rozlišování posuvu a přiblížení

K tomu, aby mohl uživatel spolehlivě ovládat okno zobrazující mapy, je nutné rozlišovat situaci, kdy se jedná o posuv mapy nebo o přiblížení mapy. Vzhledem k tomu že dotyk prstu na displeji nelze považovat za příliš přesné ovládání, je důležité sdělit zařízení toleranci, při které má brát dotyk na displej jen jako dotyk na jednom místě nebo už jako posun.

Toleranci s jakou zařízení pracuje v případě rozlišování posuvu nebo jen dotyku je možné získat způsobem zobrazeným níže. Hodnota tolerance v pixelech je uložena do proměnné `scaledTouchSlop`.

```
int scaledTouchSlop;
scaledTouchSlop = ViewConfiguration.get(context).getScaledTouchSlop();
```

Dotyková zařízení dále rozlišují i způsoby dotyku displeje. V případě, že se uživatel dotkne displeje, zařízení dotyk zaregistruje. Pokud trvá dotyk displeje v jednom místě určitou dobu, zařízení zaregistruje dotyk jako na tzv. dlouhý dotyk (long press či

long touch). Pomocí této časové konstanty aplikace pozná, zda má přejít do režimu zoom. Hodnotu v milisekundách pro konkrétní zařízení lze získat následujícím způsobem.

```
int touchTimeout;
touchTimeout = ViewConfiguration.getLongPressTimeout();
```

12. Data aplikace

Samotná aplikace musí uchovávat a poskytnout uživateli poměrně velké množství údajů. Jedná se o údaje zahrnující body, po kterých se vykresluje trasa na mapě, seznam místností, seznam zaměstnanců a další informace sloužící ke správnému fungování aplikace. Některé informace musí být dokonce propojeny, například když uživatel chce zjistit podrobnosti o zaměstnanci, musí mu aplikace také zobrazit, v jaké místnosti se zaměstnanec nachází. Pro tyto a jiné informace byly zvoleny níže uvedené způsoby uložení dat.

12.1. Hlavní body mapy

Hledání trasy mezi body a její vykreslování se musí řídit určitými pravidly. Pravidlo, jakým směrem vygenerovaná trasa povede, se řídí body uloženými v XML souboru, který názvem odpovídá mapě, pro kterou jsou body určeny. Jedním z důvodů, proč je v aplikaci, jako prostředek pro uložení bodů mapy použit právě XML soubor, je jeho přehlednost a dále jeho snazší údržba při případném updatu aplikace.

Popis složitějších struktur XML souborů není v této kapitole podstatný (je popsána v kapitole Hledání trasy), pro přiblížení struktury načítaných dat je proto volen XML soubor uchovávající vstupy do budov. Tento XML soubor má jednoduchou strukturu, jak ukazuje jeho obsah

```
<?xml version="1.0" encoding="UTF-8"?>
<entrances>
      <entrance name="FEL" x="198" y="558"/>
      <entrance name="FAV_FST" x="0" y="0"/>
</entrances>
```

Cílem zdrojového kódu zobrazeného níže, resp. metody, je načíst specifický bod, podle atributu name a uložit ho jako instanci `RoutePoint`, což je instance třídy zděděná od třídy Point. Uvedená metoda `loadSpecificPoint(String pointName, int xmlID, String elementName)` je součástí třídy `WayPoints`, uložené v balíku `ppredota.android.navigate.route.WayPoints.java`. Tato třída obsahuje metody pro načítání bodů, pro vykreslování mapy a pro přepočítávání bodů, ze kterých se vykresluje trasa.

```
public RoutePoint loadSpecificPoint(String pointName, int xmlID,
                                                String elementName ){
RoutePoint point = null;

try {
   XmlPullParser xmlParser = context.getResources().getXml(xmlID);

   while(xmlParser.getEventType()!=XmlPullParser.END_DOCUMENT){
      if(xmlParser.getEventType() == XmlPullParser.START_TAG){
         if(xmlParser.getName().equals(elementName)){
            String name = xmlParser.getAttributeValue(0);
            if(name.equals(pointName)){
                  point = new RoutePoint(Integer.valueOf
                  (xmlParser.getAttributeValue(1)),
                  Integer.valueOf(xmlParser.getAttributeValue(2));
                  break;
            }
          }
        }
        xmlParser.next();
    }
 }
 catch (Throwable t){
      Log.i("loadSpecificPoint", "Chyba nacteni XML souboru");
 }
 return point;
}
```

Uvedená metoda načítá body tak, že do `xmlParser`, což je proměnná instance třídy `XMLPullParser`, načte obsah XML souboru předaného jako prostředek pomocí ID. Samotné čtení XML souboru probíhá do doby než cyklus while nenarazí na konec dokumentu, přičemž v každém cyklu se metodou `next()` posouvá na další řádek. Další dvě podmínky kontrolují, zda se čte právě ten element, který byl předán jako parametr `elementName`, v tomto případě by byl předán řetězec `"entrance"`. Poslední podmínka kontroluje, zda první atribut, tedy atribut `name`, má hodnotu `"FEL"`. Pokud ano, je načten druhý atribut, souřadnice x, a třetí atribut, souřadnice y, do proměnné instance `RoutePoint`.

12.2. SQLite databáze

Další možností jak, uložit data pro aplikaci, je využít SQLite databázi. Z důvodu rozsáhlejší problematiky použití SQLite databáze v systému Android, bude tomuto tématu zvlášť věnována následující nová kapitola.

13. Použití SQLite databáze

Důvodem použití databáze v aplikaci pro uložení persistentních dat bývá požadavek na strukturovanou formu a na možnost propojení jednotlivých dat mezi sebou. V tomto případě se přímo nabízí využití databáze pro uložení seznamu místností a seznamu zaměstnanců spolu s příslušnými detailními informacemi.

13.1. Struktura SQLite databáze

Databáze použitá v aplikaci obsahuje dvě tabulky. První tabulka s názvem Rooms uchovává seznam všech místností. Součástí této tabulky jsou následující sloupce:

- _id – Jedná se o primární klíč tabulky, obsahuje ID každého záznamu a z principu nedovoluje duplicitní záznamy
- RoomName – Záznam uchovávající název místnosti
- IDmap – ID mapy, ve které se daná místnost nachází
- IDbuilding – ID budovy, ve které se daná místnost nachází
- X – Souřadnice místnosti na konkrétní mapě
- Y – Souřadnice místnosti na konkrétní mapě

Jak ukazuje seznam sloupců tabulky, u každé místnosti je obsažen i záznam o souřadnicích bodu, kde se konkrétní místnost nachází. Důvody, proč byly jednotlivé body místností uloženy do databáze a ne do XML souboru, jsou hlavně zamezení duplicity (seznam místností by byl jak v databázi tak v XML souboru) a také možnost ohlídat, zda jsou záznamy kompletní. V případě, že databáze není zcela vyplněna a některým místnostem schází záznam o souřadnicích, jsou tyto místnosti vynechány z položek načítaných do uživatelského prostředí.

Druhá tabulka databáze, pojmenovaná Employees, obsahuje seznam zaměstnanců a jejich detailní údaje. Sloupce, které tabulka obsahuje, jsou následující:

- _id – Sloupec primárního klíče, nepovoluje duplicitní položky
- Name – Sloupec obsahující křestní jméno zaměstnance
- Surname – Sloupec obsahující příjmení zaměstnance
- Preffix – Sloupec obsahující titul před jménem

- Suffix – Sloupec obsahující titul za jménem
- IDroom – ID místnosti (kanceláře) konkrétního pracovníka (_id z tabulky Rooms)
- Dept – Sloupec obsahujíc název katedry, pod kterou pracovník spadá
- IDbuilding – ID budovy, ve které se pracovník nachází

Jednotlivé záznamy byly převzaty z informačního systému STAG, ze záznamů byly odstraněny duplicitní položky a získaná data byla přizpůsobena databázi aplikace.

13.2. Spojení externí databáze s aplikací

Externě připravenou databázi je nutné do aplikace nejprve připojit tak, aby z ní bylo možné číst. V tuto chvíli ale nastává poměrně vážný problém, systém Android nedovoluje číst z SQLite databáze vytvořené externě a připojené k aplikaci. Naštěstí lze tento problém obejít.

Před samotným připojením databáze do aplikace je nutné databázi upravit tak, aby měla stejnou strukturu jako databáze vytvořená za běhu aplikace.

13.2.1. Vytvoření tabulky android_metadata

V předem vytvořené databázi je nutné provést několik úprav. V prvé řadě, ve všech tabulkách musí být obsažen klíčový sloupec, který má název _id. Pro správnou funkci je nezbytné, aby se takto pojmenovaný sloupec nacházel úplně ve všech uživatelem definovaných tabulkách.[67]

Další úpravou je přidání nové tabulky android_metadata a jeho jediného sloupce s názvem locale. K tomu je dobré využít některý z SQLite databázových prohlížečů (Například volně šiřitelný SQLite Database Browser) a v něm zadat následující SQL příkaz:

```
CREATE TABLE "android_metadata" ("locale" TEXT DEFAULT 'en_US')
```

Tento příkaz vytvoří výše zmíněnou tabulku včetně sloupce locale. Nakonec je nutné ještě do sloupce přidat následujícím příkazem záznam obsahující řetězec en_US:

```
INSERT INTO "android_metadata" VALUES ('en_US')
```

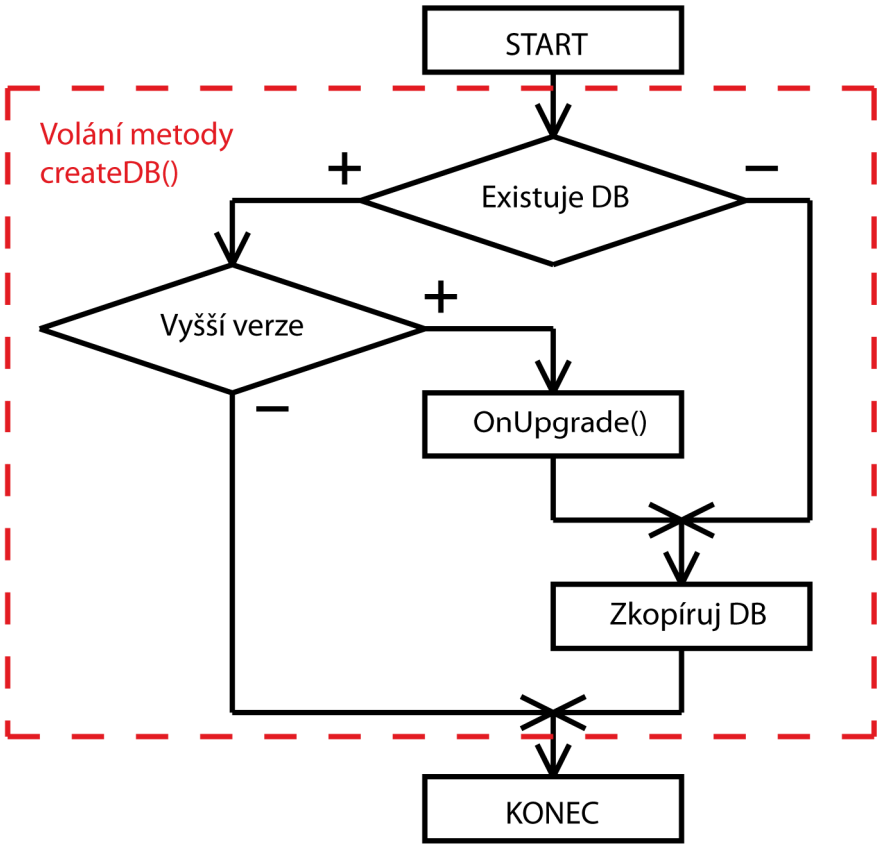
V tuto chvíli je databáze připravena a může být připojena k aplikaci

13.2.2. Spojení databáze s aplikací

Databáze vytvořená za běhu aplikace je uložena v adresáři databases, který se nachází uvnitř nainstalované aplikace v interní paměti zařízení. Z toho důvodu je nutné do tohoto umístění dostat i externě vytvořenou databázi.[67]

Jednou z možností, jak připojit vlastní databázi do projektu, je uložit databázi do adresáře assets/ či raw/ a poté za běhu programu databázi zkopírovat do konkrétního umístění v zařízení. V aplikaci popisované v rámci této práce je databáze uložena ve složce assets.

Veškeré potřebné metody k vytvoření a ověření, zda databáze existuje, jsou součástí třídy `SQLiteDB.java`, která se nachází v balíku `ppredota.android.navigate.database`. Pro připojení databáze k aplikaci je nutné volat metodu `createDB()`. Veškeré operace, které se tímto voláním provedou, znázorňuje vývojový diagram na obrázku 11.



Obrázek 11: Způsob připojení databáze do aplikace

Třída `SQLiteDB.java` poskytuje metody pro ověřování, kopírování a otevírání databáze a je vytvořená dědením od třídy `SQLiteOpenHelper`. Pro práci s databází jsou nejdůležitější tři veřejné metody:

- `public void createDB()` – zjistí existenci databáze a případně ji vytvoří nebo provede upgrade
- `public void openDB()` – ověří, zda je databáze otevřená, v případě, že není, databázi otevře
- `public void close()` – před pokusem o zavření nejprve ověří, zda je databáze otevřena

Uvedený postup zpracování databáze, podle jednoduchého vývojového diagramu zobrazeného výše na obrázku 11, probíhá kompletně v asynchronním vlákně na pozadí, tedy úplně mimo vlákno uživatelského rozhraní. Tento postup je výhodný hlavně v tom, že při delší době kopírování a zpracovávání databáze, nedochází ke zdržování vlákna uživatelského rozhraní a zároveň je možné informovat uživatele o činnosti na pozadí prostřednictvím ukazatele průběhu.

13.3. Upgrade databáze

Upgrade databáze probíhá v metodě `onUpgrade()` ve třídě SQLiteDB.java. Verze aktuální databáze se předává jako parametr při dědění konstruktoru třídy SQLiteOpenHelper.java v konstruktoru aktuální třídy SQLiteDB.java. Verze databáze je uložena jako prostředek s označením `database_version` a je typu `String`. Hodnota tohoto prostředku je uložena v XML souboru `database_version.xml`. Verze je do rodičovského konstruktoru předána následujícím způsobem:

```
public SQLiteDB(Context context) {

    super(context, DBConstant.DB_NAME, null,
    context.getResources().getInteger(ppredota.android.navigation.view.
                                activities.R.string.database_version));

    this.context = context;
}
```

Pokud je tedy hodnota `database_version` větší než byla původní hodnota, je po zavolání metody `getReadableDatabase()` nebo `getWritableDatabase()` (metody

budou popsány v nové podkapitole) vyvolána metoda `onUpgrade()`. Upgrade databáze je proveden jednoduše překopírováním nové databáze, čímž v podstatě dojde k přepsání staré databáze, což je mnohem jednoduší řešení než porovnávat a aktualizovat jednotlivé položky ve staré a nové databázi. Navíc není nutné před kopírováním nové verze databáze starou verzi smazat.

13.4. Přístup do databáze

Pro přístup k datům uložených v databázi slouží metody ve třídě `SQLiteData.java`, která je součástí balíku `ppredota.android.navigate.database`. Při práci s metodami je však nutné dodržet jedno velmi důležité pravidlo. Žádné metodě nesmí být jako parametr `sqliteDatabase` předána přímá instance na databázi vytvořenou prostřednictvím třídy `SQLiteDB.java`. Obecně není doporučeno pracovat s instancí vytvořené databáze mimo třídu, kde byla vytvořena, tedy mimo třídu `SQLiteDB.java`. Předávání přímého odkazu na tuto instanci může v reálném zařízení způsobovat chyby aplikace, které se navíc objevují nahodile.

Pro získání instance dostupné databáze je vhodnějším a doporučovaným způsobem používání jedné z těchto dvou metod:

- `getReadableDatabase()` – vytvoří nebo otevře dostupnou databázi pro čtení a vrátí její instanci
- `getWritableDatabase()` – vytvoří nebo otevře dostupnou databázi pro zápis a vrátí její instanci (nesmí být volána rekurzivně)

Pro samotné čtení dat z databáze je možné použít dotazy metodami `query()` nebo `rawQuery`. Tyto metody vrací instanci typu `Cursor`, ve které je možné procházet data získaná z databáze. Jak vypadá čtení dat z databáze pomocí metody `rawQuery` ukazuje následující zdrojový kód.

```
public Point getRoomPoint(String room, SQLiteDatabase sqliteDatabase) {
      Point roomPoint = new Point();
      Cursor xyCursor = null;
      try{
         String sql = "select "+DBConstant.DB_ROOM_X_COL+","+
                  DBConstant.DB_ROOM_Y_COL+" from "
                  +DBConstant.DB_TABLE_ROOMS+
                  " where "+DBConstant.DB_ROOM_NAME_COL+"='"+room+"'";

         xyCursor = sqliteDatabase.rawQuery(sql, null);

         if(xyCursor.moveToFirst()){
            roomPoint.set(xyCursor.getInt(xyCursor.getColumnIndex
            (DBConstant.DB_ROOM_X_COL)),xyCursor.getInt
            (xyCursor.getColumnIndex (DBConstant.DB_ROOM_Y_COL)));
         }
       }
       finally {
          if(xyCursor!=null){
             if(!xyCursor.isClosed()){
                  xyCursor.close();
             }
           }
        }
     return roomPoint;
}
```

Metoda `rawQuery()` dovoluje přímé použití SQL příkazu, libovolně dlouhého. Nalezený záznam je předán proměnné `xyCursor`, která jakožto instance třídy `Cursor` dovoluje prohlížet načtená data. Metoda `moveToFirst()` přesune ukazatel v proměnné `xyCursor` k prvnímu záznamu a je načtena jeho hodnota. Pokud by se dalo očekávat více záznamů, následovala by metoda `moveToNext()` pro přesun ukazatele na další záznam, v tomto případě se však jednalo o načtení jednoho specifického záznamu, proto je přečten jen první záznam v proměnné `xyCursor`. Po ukončení čtení je nutné použitou instanci třídy `Cursor` zavřít. V případě, že by instance nebyla zavřena, aplikace by po určitém čase zahlásila chybu.

Po každém otevření databáze je nutné se postarat o to, aby byla vždy zavřena. Pokud by totiž uživatel opustil aktivitu nebo byla aktivita přerušena a databáze nebyla uzavřena, aplikace by vyvolala chybu. Proto bych doporučoval uzavírat databázi ihned po přečtení potřebných dat, případně implementovat metodu `close()` do metody `onPause()`, která se vykonává při každém přerušení aktivity. Tím bude zajištěno, že bude databáze vždy spolehlivě uzavřena.

14. Hledání trasy

Důležitá součást této aplikace je samozřejmě algoritmus pro vyhledávání trasy v budově. Cílem bylo vytvořit takový algoritmus, který by byl spolehlivý a zároveň by pracoval s takovými daty, které by bylo možné co nejjednodušeji aktualizovat v případě updatu mapových podkladů.

Vyhledávání trasy se provádí na základě bodů, které charakterizují možnou trasu na mapě, v podstatě by se dalo říct, že se jedná o tzv. waypointy, které využívají klasické navigace, ovšem ve velice zjednodušené podobně.

14.1. Struktura uložených bodů v XML souboru

Body mapy a ostatní informace důležité pro vyhledávání trasy jsou uložené v XML souboru, který je pojmenovaný stejně jako mapa, ke které informace v XML souboru přísluší. Struktura všech XML souborů je stejná a obsahuje

- Hlavní body trasy, včetně uzlů a jejich ID
- Body výtahů včetně jejich unikátních ID
- Body schodů včetně jejich unikátních ID
- Seznam uzlů a jejich sousedů

Struktura XML souboru uchovávajícího data konkrétní mapy tedy vypadá následovně (počet elementů byl zmenšen vzhledem k rozsahu XML souborů):

```
<points>
     <!-- Body hlavni trasy v posloupnosti -->
     <mainpoints>
            <mainpoint x="437" y="521" id="1"/>
            <mainpoint x="175" y="372"/>
            <mainpoint x="176" y="280"/>
            <mainpoint x="176" y="247" id="2"/>
            <mainpoint x="191" y="245"/>
            <mainpoint x="388" y="245"/>
            <mainpoint x="412" y="245" id="3"/>
            <mainpoint x="176" y="247" id="2"/>
            <mainpoint x="182" y="231"/>
            <mainpoint x="182" y="3" id="4"/>
            <mainpoint x="307" y="521"/>
            <mainpoint x="288" y="521"/>
            <mainpoint x="270" y="521" id="5"/>
            <mainpoint x="182" y="3" id="4"/>
            <mainpoint x="261" y="521"/>
            <mainpoint x="248" y="521"/>
            <mainpoint x="248" y="528" id="6"/>
     </mainpoints>
     <!-- Body pro vytahy, ID pro hledani trasy mezi patry -->
     <elevator>
            <epoint x="203" y="226" id="1"/>
            <epoint x="213" y="226" id="2"/>
     </elevator>
     <!-- Body pro schody, ID pro hledani trasy mezi patry -->
     <stairs>
            <spoint x="192" y="231" id="1"/>
            <spoint x="192" y="257" id="2"/>
            <spoint x="252" y="534" id="4"/>
     </stairs>
     <!-- sousedni uzly -->
     <neighbors>
            <neighbor id="1" n1="2"/>
            <neighbor id="2" n1="1" n2="3" n3="4"/>
            <neighbor id="3" n1="2"/>
            <neighbor id="4" n1="2" n2="5" n3="6"/>/>
            <neighbor id="5" n1="4"/>
            <neighbor id="6" n1="4"/>
```

Popis pravidel pro vytváření XML souborů bude součástí kapitoly Údržba mobilní aplikace, v tuto chvíli postačí pro lepší pochopení vyhledávacího algoritmu pouze ukázka struktury uložených informací.

14.2. Vyhledávání trasy

Vyhledávací algoritmus aplikace vychází z principů používaných v algoritmech pro prohledávání grafu.[72] Veškeré metody spojené s vyhledáváním trasy jsou součástí třídy `WayPoint.java`, která se nachází v balíku `ppredota.android.navigate.route`.

14.2.1. Načtení bodů trasy

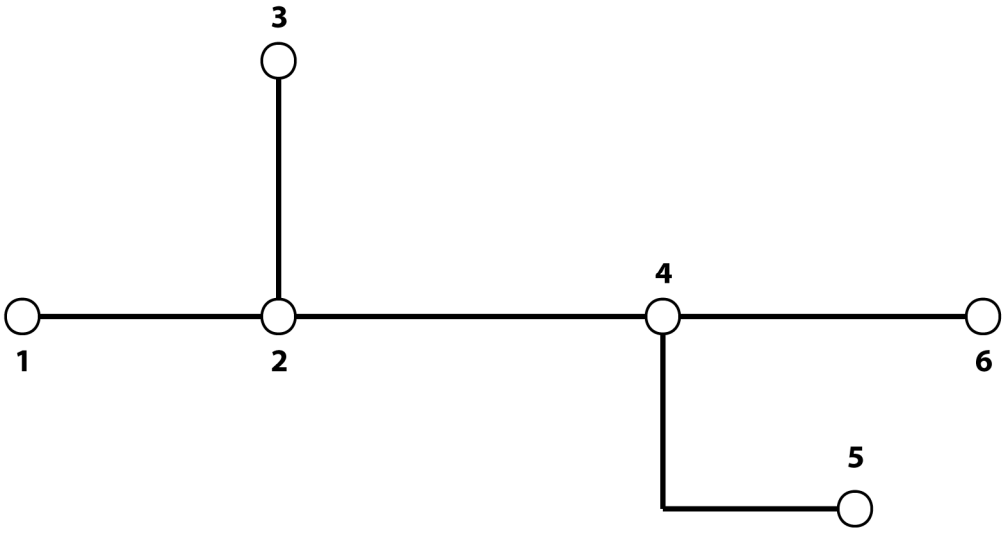
Body hlavní trasy, elementy `<mainpoint>`, jsou načteny do pole objektů typu `RoutePoint`. Třída `RoutePoint` dědí vlastnosti třídy `Point`, která je standardní součástí systému Android, ovšem jsou do ní přidány některé vlastnosti potřebné pro vyhledávání trasy. V objektu `RoutePoint` je možné uchovávat tyto informace:

- Souřadnice bodu
- Barvu bodu – V případě obyčejného bodu trasy není barva určena, v případě uzlu může být bod bílý, šedý, černý, zelený nebo červený
- ID bodu – pouze v případě uzlu, každý uzel má své jedinečné ID
- Sousední body – pouze pokud se jedná o uzel, uchovává informaci o ID sousedních uzlů

S takto načtenými body je možné libovolně manipulovat, měnit barvy, měnit sousední uzly anebo vytvářet nové uzly.

14.2.2. Grafické znázornění bodů trasy

Strukturu takto načtených dat lze graficky znázornit tak, jak je vidět na obrázku 12. Struktura odpovídá struktuře grafu.



Obrázek 12: Grafické znázornění bodů hlavní trasy

Všechny uzly jsou v případě načtení hlavních bodů trasy nejprve bílé, což je jejich výchozí hodnota. V tuto chvíli je nutné pro začátek prohledávání grafu určit počáteční bod a cílový bod.

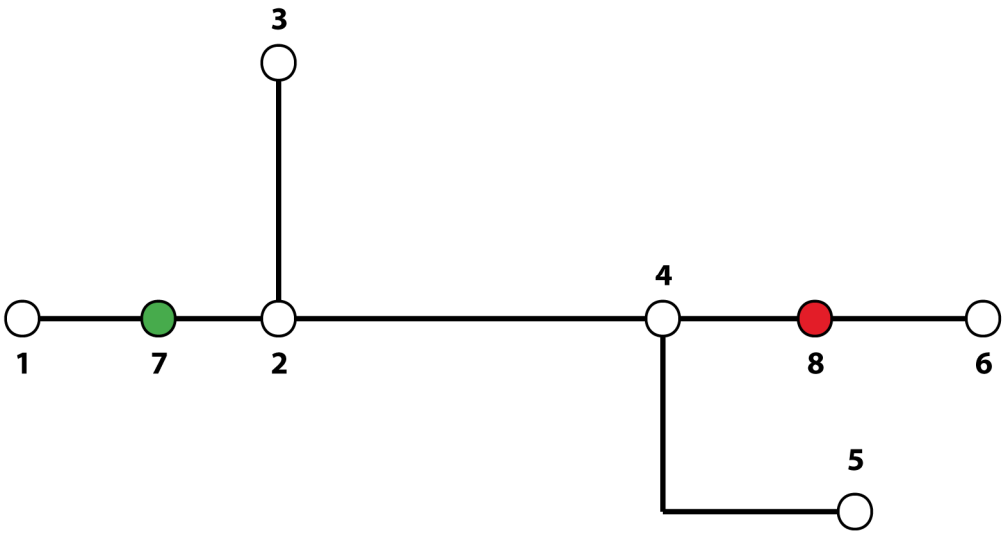
14.2.3. Určení počátečního a cílového bodu

Pro určení počátečního a cílového bodu jsou použité body místností z databáze, případně při přechodu mezi patry jsou použity body výtahů `<epoints>` nebo body schodů `<spoints>` z XML souboru. Pro každý z bodů je nutné vybrat z hlavních bodů trasy nejbližší bod. To lze provést pomocí metody `findClosePoint(Point point, Point[] mainPoints)`, která najde a vrátí z hlavních bodů trasy nejbližší bod k bodu, předaného parametrem `point`.

14.2.4. Vložení počátečního a cílového bodu do grafu

V případě, že jsou určeny blízké body k počátečnímu a koncovému bodu, dalším krokem je jejich vložení do grafu. Obrázek 13 znázorňuje vložení počátečního bodu a cílového bodu do grafu. Počátečnímu bodu je nastavena zelená barva a ID = 7 a cílovému bodu červená barva a ID = 8. Tím se z nich v podstatě staly nové uzly. Vložení se provádí

metodou `addNodeToPoint (RoutePoint[] mainPoints, int index, int id, int pointColor)`, kde je navíc specifikováno kromě umístění nového uzlu i jeho ID a barva.



Obrázek 13: Vložení počátečního a cílového bodu do grafu

Samotným vložením však nastavení počátečního a koncového bodu nekončí. Nově vloženým uzlům sice byly zároveň i nastaveny sousední uzly, ovšem okolním uzlům zůstali původní sousedi. Z toho důvodu je z metody `addNodeToPoint()` volána navíc metoda `changeNeightbors(RoutePoint[] mainPoints, int newId, int index)`, která změní okolním bodům sousední body tak, aby odpovídaly skutečnosti.

14.2.5. Sestavení matice sousednosti

Pro potřeby algoritmu je v dalším kroku nutné sestavit matici sousednosti, která je složená pouze z jedniček a nul. Matice se sestavuje tak, že pokud indexy řádku a sloupce odpovídají ID dvou uzlů, které jsou sousední, bude na této souřadnici hodnota 1. Matice pro aktuální graf je na obrázku 14.

	1	2	3	4	5	6	7	8
1	0	0	0	0	0	0	1	0
2	0	0	1	1	0	0	1	0
3	0	1	0	0	0	0	0	0
4	0	1	0	0	1	0	0	1
5	0	0	0	1	0	0	0	0
6	0	0	0	0	0	0	0	1
7	1	1	0	0	0	0	0	0
8	0	0	0	1	0	1	0	0

Obrázek 14: Matice sousednosti pro aktuální graf

14.2.6. Prohledávání grafu

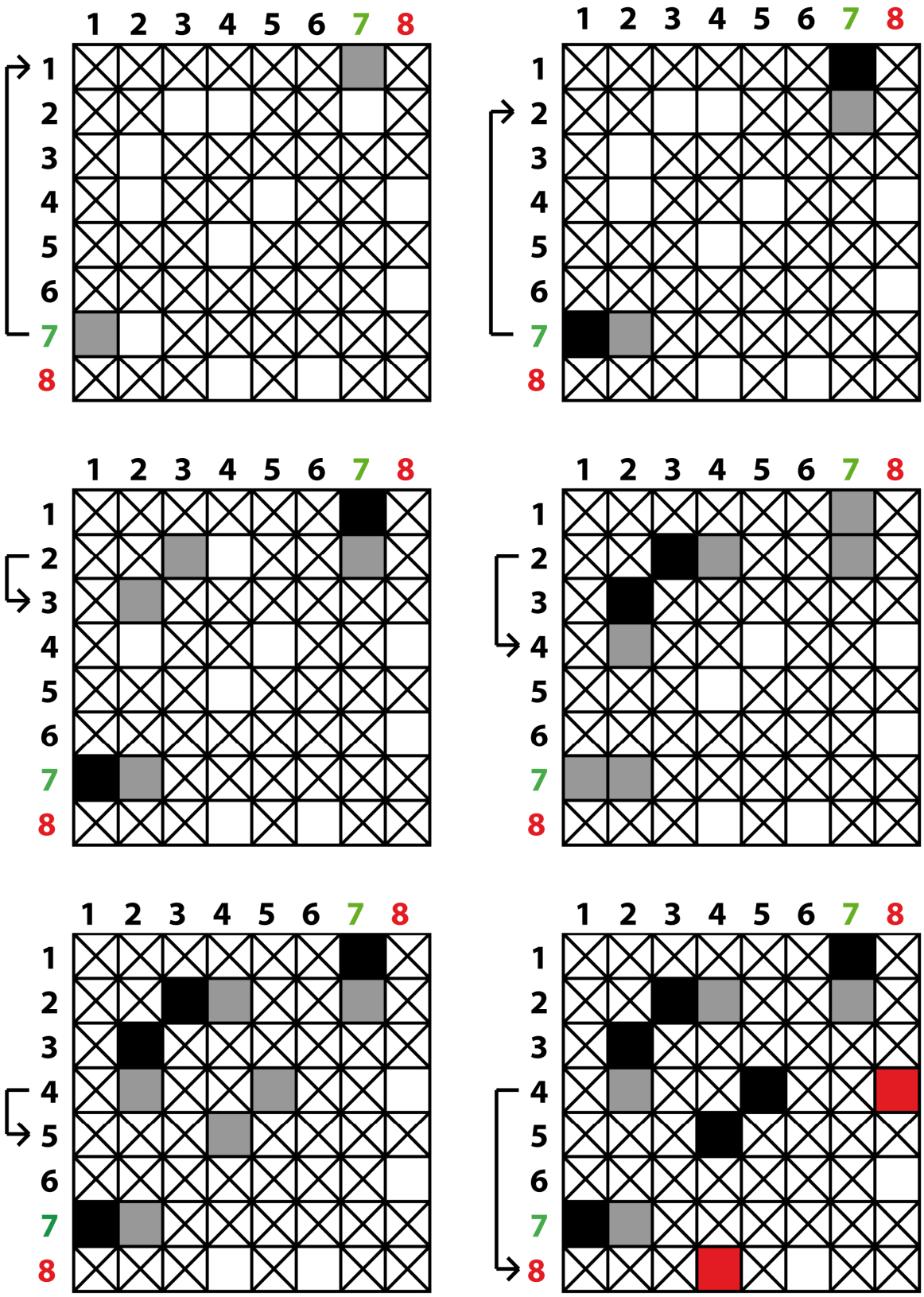
Po sestavení matice sousednosti přechází algoritmus k vyhledávání nejkratší trasy. Hledání probíhá tak, že se prochází jednotlivé řádky a navštívené uzly se obarvují následujícím způsobem:

Bílý uzel – Uzel nebyl ještě navštíven.

Šedý uzel – Uzel byl navštíven poprvé.

Černý bod – Uzel byl navštíven podruhé

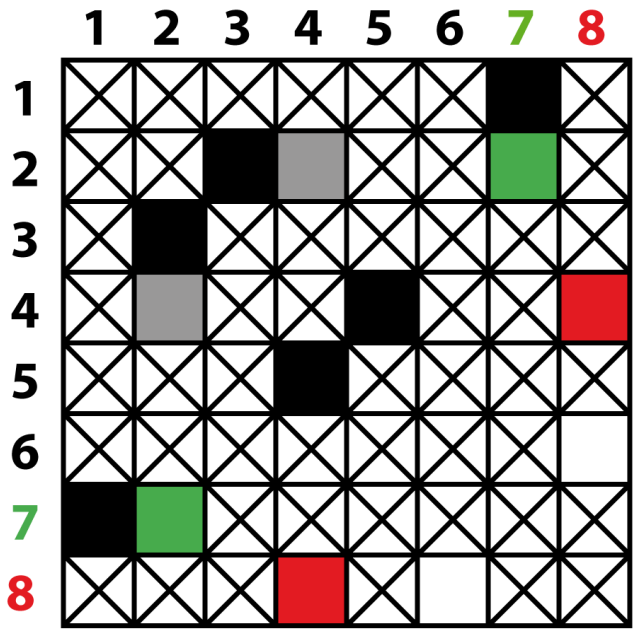
Samotné prohledávání krok za krokem přiblíží následující obrázek 15. Matici sousednosti, která byla na obrázku 14, je možné graficky znázornit způsobem, jaký ukazuje obrázek 15. Na obrázku 15 jsou zároveň znázorněny všechny kroky, které algoritmus provede, než nalezne cílový červený uzel.



Obrázek 15: Hledání trasy v matici sousednosti

Algoritmus znázorněný na obrázku 15 pracuje následujícím způsobem. Začíná na řádkovém indexu 7 a hledá první bílý uzel. Po nalezení bílého uzlu zbarví uzel šedě a přesune se na řádkový index, který odpovídá sloupcovému indexu nalezeného uzlu. Znovu hledá bílý uzel. Ten však v řádku není, proto zkouší hledat šedý uzel. Nalezený šedý uzel nabarví na černo a vrací se na index řádku, který odpovídal sloupcovému indexu, nyní již černého, uzlu. Černá barva uzlu označuje uzly, které již nemají žádné sousedy, kromě navštívených uzlů a jedná se o slepé větve grafu.

Výše popsaným způsobem algoritmus postupně prohledává matici sousednosti, dokud nenarazí na souřadnici, která odpovídá uzlu s červenou barvou, tedy cílovému uzlu. Po nalezení červeného uzlu je obarven šedý uzel na počátečním indexu 7 zpět na zeleno a tím je vyhledávání trasy ukončeno. Na obrázku 16 je výsledná matice sousednosti.



Obrázek 16: Výsledná matice sousednosti s vyhledanou trasou

V takto vytvořené matici řádkový index se zeleným políčkem znázorňuje ID počátečního uzlu, šedá políčka znázorňují ID uzlů, přes které vede trasa a červené políčko znázorňuje ID cílového uzel.

14.2.7. Ukládání bodů pro vykreslení

Veškeré operace výše popsané, včetně metod, které jsou použité k vyhledání trasy, jsou součástí veřejné metody `public Point[] findPath(Point startPoint, Point endPoint, RoutePoint mainPoints[])`, která zároveň zajišťuje i uložení nalezených uzlů a bodů trasy mezi uzly do pole typu `Point`. Zároveň zajišťuje i dodatečné připojení skutečného počátečního a cílového bodu na začátek a konec pole, jelikož tyto body nebyly přímo součástí vyhledávání.

14.3. Vykreslení bodů na mapě

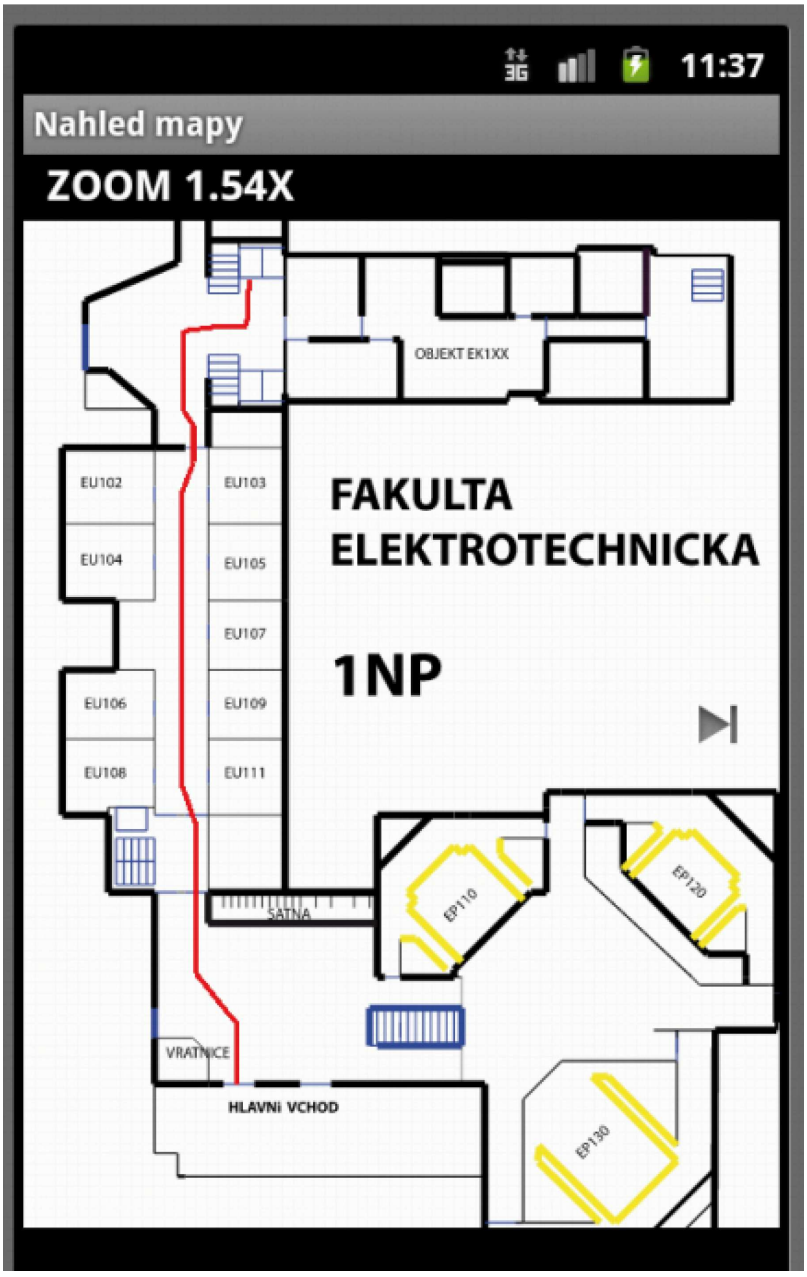
Pro vykreslení bodů na mapě je k dispozici třída `PathDrawer.java`, která je uložena v balíku `ppredota.android.navigation.routes`. Tato třída obsahuje metodu `public void drawPath(Canvas canvas, Point pathPoint[],float heightQuotient, float widthQuotient, Paint paint)`, která kromě ostatních a na první pohled jasných parametrů, očekává i následující dva parametry

`float heightQuotient` – poměr výšky mapy a výšky okna zobrazovaného zařízením (výška získaná metodou `getHeight()`)

`float widthQuotient` - poměr šířky mapy a šířky okna zobrazovaného zařízením (výška získaná metodou `getWidth()`)

Tyto parametry jsou předávány metodě pro případ, že mapa nemá stejné rozměry jako okno, ve kterém má být zobrazena a tudíž je přizpůsobena. Z toho důvodu je nutné přizpůsobit a přepočítat i body, které vykreslují trasu.

Na obrázku 17 je vzhled okna mapy s vygenerovanou trasou. V tomto případě se jedná o vykreslení trasy mezi prvním a druhým patrem s bezbariérovým přístupem. Pro přesun k mapě dalšího (vyššího) patra je vpravo uprostřed okna tlačítko ve tvaru šipky.



Obrázek 17: Vzhled okna s vygenerovanou trasou

15. Ošetření změn konfigurace

V případě psaní aplikací pro mobilní zařízení, je nutné, pro bezchybné fungování aplikace, řešit specifické události, které by například v případě desktopových aplikací nikdy nenastaly.

Jednou takovou událostí je například příchozí hovor v momentě, kdy uživatel pracuje s aplikací. Tento problém není nijak závažný, ve většině případů tuto událost uživatel ani nemusí nijak řešit a systém Android po ukončení hovoru pouze pozastaví a znovu obnoví aktivitu aplikace, jak bylo popsáno v kapitole 3. Aktivity.

Existují ovšem události, se kterými je nutné počítat a pro správnou práci aplikace je v některých situacích ošetření takovýchto událostí nezbytné.

15.1. Možné změny konfigurace

První takovou událostí může být změna orientace zařízení. Při změně orientace zařízení dochází k ukončení a znovuvytvoření aktuální aktivity. Pro vertikální a horizontální orientaci mohou existovat dva různé návrhy uživatelského rozhraní a právě z toho důvodu systém Android při každé změně orientace volá pro aktuální aktivitu metodu `onCreate()`, která zajistí, že bude načten správný návrh uživatelského rozhraní.[68]

Druhou událostí může být vysunutí hardwarové klávesnice zařízení, pokud je tedy zařízení klávesnicí vybaveno. V tomto případě se systém zachová stejně jako v případě změny orientace zařízení. Dokonce je aktivita restartována i přesto, že je v souboru `AndroidManifest.xml` nastavena orientace uživatelského návrhu na pevnou hodnotu.

Výše popsané situace není potřeba příliš řešit v jednoduchých aplikacích, ovšem existují situace, kdy nesprávné ošetření těchto stavů vede až chybě aplikace.

15.2. Použití asynchronního vlákna a ukazatele průběhu

V aplikaci této práce je pro načítání databáze využívané asynchronní vlákno děděné ze třídy `AsyncTask`. Použití tohoto vlákna je velice jednoduché a navíc nabízí velice jednoduchý způsob, jak informovat uživatele o tom, že na pozadí běží některá činnost v tomto vlákně.

15.2.1. Přidání ukazatele průběhu do vlákna na pozadí

Pro přidání jednoduchého ukazatele průběhu je možné použít třídu `ProgressDialog`, která ve své výchozí hodnotě zobrazuje jednoduchý ukazatel průběhu ve formě bodu kroužícího dokola.

Pro pohodlnější použití `ProgressDialog` je nejlepší způsob překrýt metodu `onCreateDialog(int id)` a implementovat do ní vlastní vzhled ukazatele. Následující kód ukazuje jakým způsobem je překryta metoda `onCreateDialog(int id)` pro ukazatel, zobrazovaný v aplikaci při načítání databáze.

```
@Override
protected Dialog onCreateDialog(int id){
      switch(id){
            case LOADING_DB_DIALOG:
                ProgressDialog progressDialog = new ProgressDialog(this);
                progressDialog.setMessage("Nacitani databaze...");
                progressDialog.setCancelable(false);
                return progressDialog;
            default:
               return null;
      }
}
```

Součástí aktivity, ve které je metoda překryta je i konstanta `public static final int LOADING_DB_DIALOG = 0;`, s jejíž pomocí je možné `ProgressDialog` vytvořit. Co se týče samotného zdrojového kódu, je v něm specifikována zpráva, která se objeví u ukazatele průběhu a navíc je metodou `setCancelable(false)` zakázáno vypnutí ukazatele před jeho ukončením.

Protože je ukazatel průběhu objekt uživatelského rozhraní, je ho možné spouštět pouze z vlákna uživatelského rozhraní. Z toho důvodu se pro přidání ukazatele do asynchronního vlákna perfektně hodí jeho dvě metody volané z vlákna uživatelského rozhraní, tedy metody `onPreExecute()` a `onPostExecute()`. Zdrojový kód pro zobrazení a vypnutí ukazatele průběhu je následující.

```
public class OpenDatabaseTask extends AsyncTask<Void, Void, Void> {

@Override
protected void onPreExecute() {
      super.onPreExecute();
      searchActivity.showDialog(SearchMenuActivity.LOADING_DB_DIALOG);
}

@Override
protected Void doInBackground(Void... params) {

}

@Override
protected void onPostExecute(Void unused) {
      searchActivity.dismissDialog(SearchMenuActivity.LOADING_DB_DIALOG);
}
}
```

Jak je vidět na zdrojovém kódu, použití ukazatele průběhu je poměrně jednoduché, pokud ovšem nenastane nějaká neočekávaná událost.

15.2.2. Problém při změně orientace displeje

Řešení popsané výše bude fungovat spolehlivě do té doby, než uživatel otočí telefon nebo vysune hardwarovou klávesnici. Pokud uživatel provede jednu z těchto činností až po ukončení průběhu vlákna, bude sice provedena změna uživatelského rozhraní, ale ukazatel průběhu zůstane zobrazen na displeji a neumožní uživateli další práci s aplikací. V případě, že uživatel změní orientaci displeje ještě před ukončením vlákna, dojde nejspíše k chybě aplikace, jelikož se aplikace pokusí vypnout ukazatel průběhu, jehož instance byla ukončena a s novým vytvořením aktivity byla zároveň vytvořena jiná.

Naštěstí lze poměrně elegantně vyřešit obě situace najednou.[69] Jednou z podmínek je vytvořit třídu pro asynchronní vlákno jako vlastní třídu, nikoliv jako vnitřní třídu aktivity, ve které je používaná. Pro praktickou ukázku jsem z aplikace zvolil třídu `SearchMenuActivity.java`, která spolupracuje s třídou asynchronního vlákna `OpenDatabaseTask.java`. Následující kód ukazuje důležité části a metody třídy `SearchMenuActivity.java`.

```
public class SearchMenuActivity extends Activity implements
                                                    OnClickListener{
private OpenDatabaseTask openDatabaseTask;

      public void onCreate(Bundle savedInstanceState) {
            super.onCreate(savedInstanceState);
            Log.i("SearchMenuActivity", "spusten onCreate()");
            setContentView(R.layout.searchmenu)
            Object retained = getLastNonConfigurationInstance();

        if(retained instanceof OpenDatabaseTask){
            Log.i("onCreate", "Vraci predchozi asyncTask");
            openDatabaseTask = (OpenDatabaseTask)retained;
            openDatabaseTask.setActivity(this);
        }
        else{
            Log.i("onCreate","Vytvareni noveho asyncTask");
            openDatabaseTask = new OpenDatabaseTask(this, sqliteDB);
            openDatabaseTask.execute();
        }
      }

      @Override
      public Object onRetainNonConfigurationInstance() {
            openDatabaseTask.setActivity(null);
            return openDatabaseTask;
      }
      public void onTaskComleted(){
            Log.i("onTaskCompleted()","asyncTask complete");
            roomTextView.setAdapter(openDatabaseTask.getRoomAdapter());
            employeeTextView.setAdapter
                              (openDatabaseTask.getEmployeeAdapter());
            if(showDialog){
                  dismissDialog(LOADING_DB_DIALOG);
            }
      }
}
```

V tomto zdrojovém kódu jsou velmi důležité dvě metody. První z nich je metoda `public Object onRetainNonConfigurationInstance()`, která je volána vždy s metodou `onDestroy()`, může vracet jakýkoliv objekt (nejen tedy primitivní datové typy) a tudíž její překrytí poskytuje příležitost k uložení původní instance aktivity. Opakem je metoda `getLastNonConfigurationInstance()`, která umožňuje získat objekt, který byl vrácen metodou `onRetainNonConfigurationInstance()`. V případě, že získaný objekt je instancí třídy `SearchMenuActivity`, je instance obnovena a je zavolána metoda `setActivity(this)` ze třídy `OpenDatabaseTask`.

Tato metoda zajistí při restartu aktivity zavoláním metody `public void onTaskComleted()`, zrušení ukazatele průběhu, případně aktualizuje okno uživatelského rozhraní původními daty.

Důležité části třídy `OpenDatabaseTask`, která obsluhuje asynchronní vlákno a obsahuje metody, které informují o dokončení vlákna, vypadají následovně.

```
public class OpenDatabaseTask extends AsyncTask<Void, Void, Void> {

  private SearchMenuActivity searchActivity;
  private boolean completed;
  private SQLiteDB sqliteDB;

  public OpenDatabaseTask(SearchMenuActivity activity, SQLiteDB
                                                        sqliteDB){
      searchActivity = activity;
      this.sqliteDB = sqliteDB;
  }

  @Override
  protected void onPreExecute() {
      super.onPreExecute();
      Log.i("onPreExecute()", "Spusten progressDialog");
      searchActivity.showDialog(SearchMenuActivity.LOADING_DB_DIALOG);
  }
  @Override
  protected void onPostExecute(Void unused) {

      completed = true;
      notifyActivityTaskCompleted();
  }

  public void setActivity(SearchMenuActivity searchMenuActivity){
      this.searchActivity = searchMenuActivity;
      if(completed){
            notifyActivityTaskCompleted();
      }
  }

  public void notifyActivityTaskCompleted(){
      if(searchActivity != null){
            searchActivity.onTaskComleted();
      }
  }
}
```

Třída informuje o dokončení vlákna metodou `notifyActivityTaskCompleted()`. Tato metoda informuje aktivitu o dokončení vlákna, na což může aktivita reagovat patřičným způsobem, viz minulý odstavec.

Úplně na konec bych ještě dodal, že pro zobrazení ukazatele průběhu je kromě překrytí metody `onCreateDialog(int id)` vhodné ještě překrýt metodu `onPrepareDialog()`. Tato metoda je volána s každým volání ukazatele průběhu, tudíž je v ní možné nastavit příznak informující o tom, že je ukazatel průběhu aktivní. Následující zdrojový kód ukazuje implementaci této metody tak, jak je implementována ve třídě `SearchMenuActivity.java`.

```
@Override
protected void onPrepareDialog(int id, Dialog dialog){
      super.onPrepareDialog(id, dialog);
      if(id == LOADING_DB_DIALOG){
            showDialog = true;
      }
}
```

15.3. Ošetření změn konfigurace v okně mapy

V případě uživatelského rozhraní pro zobrazení mapy byla situace s ošetřováním změn konfigurace o něco jednoduší. Vzhledem k tomu, všechny mapové podklady jsou určeny jen pro zobrazení na výšku, nebylo by příliš vhodné otáčet mapu spolu s telefonem, protože by se nedosáhlo lepších výsledků zobrazení.

Pro tuto situaci je vhodné použít radikálnější řešení a jednoduše zakázat konfigurační změnu okna v souboru `AndroidManifest.xml`. Pro zakázání je nutné do elementu aktivity `SVGViewActivity` přidat následující řádku.

```
android:configChanges="orientation|keyboardHidden"
```

V tuto chvíli je možné konfigurační změny obsluhovat ze zdrojového kódu, což v tomto případě není nutné ani žádoucí, ale co je důležitější, samotný systém neprovede žádnou změnu aktivity při změně orientace displeje ani při vysunutí hardwarové klávesnice.[68]

16. Údržba mobilní aplikace

Aby mohla být mobilní aplikace reálně využitelná, je ji nutné udržovat co možná nejaktuálnější. V této kapitole budou popsány postupy, jakými lze mobilní aplikaci udržovat aktuální či postupy, kterými lze stávající aplikaci rozšiřovat.

16.1. Způsob vytváření bodů trasy

Hlavní součástí mobilní aplikace je část určená pro vyhledávání trasy ve zvolené budově. Aby vyhledávání probíhalo vždy bezproblémově, je nutné se při vytváření nových mapových podkladů řídit pravidly popsanými dále.

16.1.1. Velikost mapových podkladů

Pro co nejlepší zobrazení mapových podkladů a pro co nejpřesnější vykreslení trasy, bylo nutné určit velikost mapových podkladů. Po mnoha pokusech se jako nejlepší rozlišení pro podklady ukázala hodnota standardu VGA, tedy 640x480 obrazových bodů.[70]

Hodnota 640x480 obrazových bodů vychází z rozlišení displejů, používaných v telefonech se systémem Android. Jedná se o velikosti, které vycházejí ze standardu VGA a jedná se nejčastěji o tyto rozlišení:[52]

- QVGA – 320x240
- HVGA – 320x480
- WVGA – 480x800

16.1.2. Vytváření XML souborů s body trasy

Struktura XML souborů obsahujících body trasy byla ukázána v kapitole 13. Struktura hlavních bodů trasy (elementy `<mainpoint/>`) je načítána za běhu aplikace do jednorozměrného pole, ze kterého je následně vypočítávána konkrétní trasa. Z toho důvodu se struktura těchto bodů musí řídit následujícími zásadami.

- Všechny body trasy musí být uloženy v posloupnosti.
- Body trasy musí mít vždy minimálně dva uzly (počáteční a koncový)
- Každá větev musí začínat uzlem, ze kterého vychází, a musí mít koncový uzel (z toho důvodu může být v hlavních bodech více uzlů se stejnými ID)

Další součástí tohoto XML souboru jsou elementy se souřadnicemi výtahů (elementy `<epoint/>`) a schodů (elementy `<spoint/>`). Kromě souřadnice obsahuje každý element jako třetí atribut ještě ID, které je jedinečné pro konkrétní výtah nebo schodiště v rámci celého patra. Pomocí tohoto ID je možné vyhledávat cestu mezi patry.

Poslední součástí XML souboru jsou elementy `<neightbor/>`, které jako atributy uchovávají seznam sousedů konkrétního uzlu. První atribut určuje ID uzlu a ostatní atributy určují jeho sousední uzly. V každém souboru musí být minimálně dva tyto elementy, pro počáteční a pro koncový uzel.

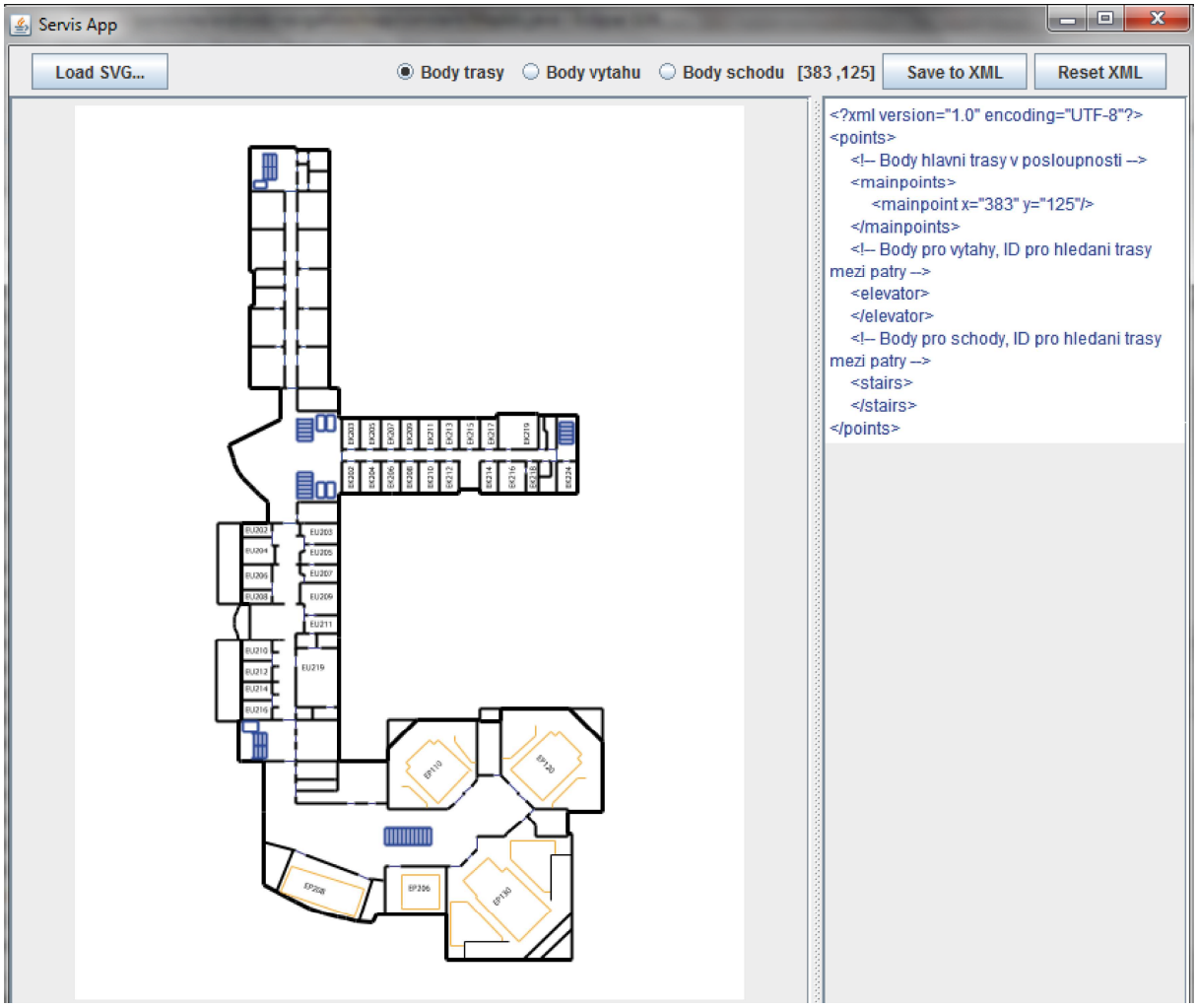
Ke každé budově je nutné přidat souřadnici hlavního chodu. Tuto souřadnici spolu s atributem se jménem budovy je nutné zadat do souboru `entrances.xml`, který se nachází v adresáři xml/.

16.1.3. Odkazy na jednotlivé součásti ve zdrojovém kódu

Veškeré soubory s uloženými body a veškeré soubory mapových podkladů je nutné zpřístupnit zdrojovému kódu tak, aby s nimi mohla aplikace pracovat. To se provádí pomocí třídy `MapIds.java`, která se nachází v balíku `ppredota.android.navigation.map.constant`. V této třídě jsou (prozatím) dvě jednorozměrná pole. První obsahuje ID prostředků pro jednotlivé mapové podklady. Druhé pole pak obsahuje ID prostředků pro jednotlivé XML soubory s uloženými body a to ve stejném pořadí jako první pole. Toto řazení je nutné zachovat z důvodu, aby bylo možné jednotlivým podkladům přiřadit odpovídající body trasy a naopak.

16.2. Aplikace pro údržbu mobilní aplikace

Pro samotné vytváření XML souborů s body trasy je možné využít aplikaci napsanou v jazyce Java a vytvořenou pro tuto mobilní aplikaci. Aplikace obsahuje dvě okna, levé pro načtení SVG mapového podkladu a pravé, které generuje a zobrazuje strukturu vytvářeného XML souboru. Vzhled aplikace je zobrazen na obrázku 18.



Obrázek 18: Vzhled aplikace pro údržbu mobilní verze

Kliknutím na mapu je do XML struktury vlevo přidán element, který je určen podle toho, zda je z nabídky na liště zvolena volba Body trasy, Body výtahu nebo Body schodů. Po tom, co jsou naklikány body trasy, je možné hotovou strukturu uložit do souboru pod libovolným jménem.

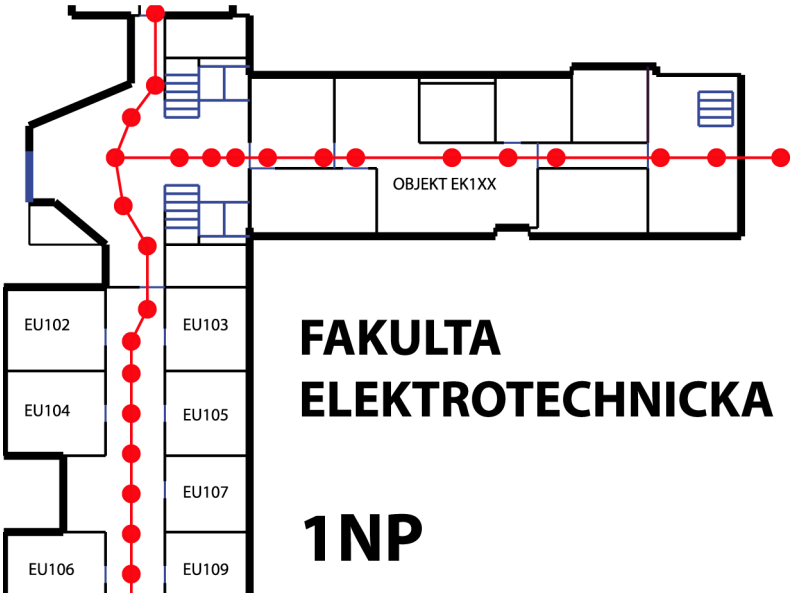
Samotná aplikace však nevytvoří úplně kompletně strukturu tak, aby ji bylo možné ihned použít v aplikaci. Pro dokončení připravené struktury je nutné ještě v hotovém souboru nebo v samotné aplikaci těsně před uložením, ručně přiřadit ID konkrétním

bodům (uzlům, výtahům a schodištím) a ručně doplnit elementy pro určení sousedních uzlů.

Pro samotné nastavování bodů na mapě platí několik následujících pravidel či doporučení.

- Žádný uzel nesmí být zároveň nejbližším bodem jakéhokoliv objektu na mapě (dveří místnosti, výtahu, schodiště).
- Přímo před každým objektem musí být bod trasy, který ale zároveň není uzlem
- Je vhodné používat větve jen v nezbytných případech. Přílišný počet větví nejen komplikuje XML soubor, ale také prodlužuje dobu vyhledávání.
- Čím více bodů bude trasa mít, tím detailněji bude vykreslena, ale výpočet zabere o něco více času. Naopak příliš málo bodů se může negativně projevit na vzhledu trasy.

Optimální způsob rozložení bodů na mapě je znázorněn na obrázku 19. Body jsou seřazené tak, že trasa co možná nejvíc kopíruje přímku (to však samozřejmě není vždy nejlepší řešení a v některých situacích to není ani proveditelné).



Obrázek 19: Grafické znázornění optimálního rozložení bodů trasy

16.3. Upgrade databáze

Pro aplikace je také velice důležité udržovat přibalenou databázi aktuální a to nejen z pohledu jednotlivých souřadnic pro místnosti. Protože mobilní aplikace umožňuje prohledávat a zobrazovat detailní informace o jednotlivých položkách databáze, bylo nutné vytvořit způsob, jakým by bylo možné přiloženou databázi upgradovat.

Upgrade databáze lze provést v následujících jednoduchých krocích. Nejprve je nutné novou databázi nakopírovat do složky assets/ v projektu aplikace a tím přepsat stávající databázi. Poté je nutné v souboru `database_version.xml`, ve složce values/ v projektu aplikace, zvýšit číslo, které označuje verzi databáze. Po spuštění již aplikace provede upgrade databáze zcela automaticky.

Pro samotnou práci s databází bych doporučil například aplikaci SQLite Browser[71], která je licencovaná jako freeware a umožňuje manipulovat s databází jak pomocí přímé editace tabulek, tak i pomocí SQL příkazů.

Závěr

Rychlost vývoje nových technologií v oblasti hardwaru ale i softwaru jen dokazuje, že obliba mobilních operačních systémů a obliba mobilních technologií obecně roste opravdu obrovským tempem. Zatím nic nenasvědčuje tomu, že by se tento rostoucí trend měl zpomalit či nějakým způsobem zastavit. Naopak, jen za dobu vzniku této práce se toho v oblasti mobilních operačních systémů odehrálo spoustu, ať už se jednalo o představení nových zařízení, technologií či nového softwaru.

Co se týče vývoje pro operační systém Android, dovoluji si tvrdit, že se tento operační systém i přes svoji silnou konkurenci dokáže do budoucna dále prosazovat nejen v mnoha chytrých telefonech ale i v jiných mobilních zařízeních. Tomuto tvrzení nahrává fakt, že se Android stal hlavním operačním systémem nasazeným v nových telefonech společností Samsung a HTC. Navíc oblibě tohoto systému přispívá i množství lehce dostupných doplňkových aplikací, z důvodu poměrně vstřícného přístupu společnosti Google k uvedení nových aplikací na webu Google Play.

Co se týče aplikace vyvíjené v rámci této práce, samotná navigace je funkční v rámci budovy fakulty elektrotechnické. Bohužel se v době vývoje aplikace ukázala možnost sehnání kvalitních mapových podkladů jako značně problematická, z toho důvodu je samotná navigace omezena pouze na tuto jednu budovu. Aplikace je ovšem připravena tak, aby bylo možné přidáním dalších mapových podkladů a dalších záznamů do přiložené databáze, s minimálními zásahy do zdrojového kódu, umožnit bezproblémovou navigaci i v ostatních budovách. Ke zjednodušení budoucího upgradu databáze byla pro mobilní aplikaci vytvořena aplikace pro PC, která umožňuje připravovat podklady pro navigaci ve formátu přizpůsobeném mobilní aplikaci.

Navíc, kromě výše zmíněné navigace, byla do aplikace přidána možnost prohledávat jednotlivé položky přiložené databáze. Tudíž je například možné vyhledat místnost podle zaměstnance a poté si k ní nechat vygenerovat trasu pomocí navigace. Nebo je možné konkrétnímu zaměstnanci zatelefonovat nebo odeslat email přímo z aplikace.

Aplikace byla vyvíjena tím způsobem, aby pracovala i na starších verzích operačního systému Android a to již od verze 1.6, tedy od API úrovně 4, což by mělo zabezpečit kompatibilitu aplikace se všemi dostupnými Android telefony. Z důvodu

spolehlivosti byla aplikace testována jak na emulovaných zařízeních se všemi verzemi systému Android, tak i na reálných zařízeních ve verzích 2.1 a 2.2.

K mírným odlišnostem však může docházet při zobrazování mapových podkladů na různých displejích z toho důvodu, že systém Android přímo nepodporuje formát SVG. Tyto odlišnosti by však neměly být nijak dramatické. Lze jen těžko odhadovat, zda bude SVG formát podporován v některé z nově vydaných verzí systému Android. Prozatím se bohužel o chystané podpoře neobjevují žádné zprávy, tudíž pro využití tohoto formátu je zatím nutné využívat neoficiální knihovny.

Mobilní aplikace pracuje kompletně v režimu offline, což je dáno tím, že v budovách není téměř možné používat GPS přijímač v režimu online. V případném budoucím rozšíření aplikace by však mohla být zajímavá možnost navigace za pomocí připojeného GPS při pohybu mezi vzdálenějšími budovami s využitím mapových podkladů Google Maps, jejichž použití je systémem Android přímo podporováno na úrovni API.

Je jen velmi těžké odhadovat jaké typy zařízení a s jakými možnostmi se pro operační systém Android do budoucna objeví. Z toho důvodu je mobilní aplikace podrobně okomentovaná a obsahuje i dokumentaci popisující jednotlivé třídy a metody tak, aby bylo možné do budoucna tuto aplikaci modifikovat nebo ji dále rozšiřovat.

Použitá literatura

[1] *Vývoj mobilních telefonů* [online]. c2003 [cit. 2011-12-16]. Galaxie. Dostupné z WWW: <http://www.galaxie.name/index.php?clanek=vyvoj-mobilnich-telefonu-1-dil>.

[2] Windows Phone Development [online]. c2011 [cit. 2011-12-16]. MSDN. Dostupné z WWW: <http://msdn.microsoft.com/en-us/library/ff402535(v=vs.92).aspx>.

[3] The Silverlight and XNA Frameworks for Windows Phone [online]. c2011 [cit. 2011-12-16]. MSDN. Dostupné z WWW: <http://msdn.microsoft.com/en-us/library/ff402528(v=VS.92).aspx>.

[4] Metro design language. Microsoft [online]. © 2012 [cit. 2012-05-06]. Dostupné z: http://www.microsoft.com/design/toolbox/tutorials/windows-phone-7/metro/

[5] Windows Phone SDK Tools. MSDN Microsoft [online]. © 2012 [cit. 2012-05-06]. Dostupné z: http://msdn.microsoft.com/en-us/library/ff402523(v=vs.92).aspx

[6] Creating Windows Phone Apps. MSDN [online]. © 2012 [cit. 2012-05-06]. Dostupné z: http://create.msdn.com/en-US/education/quickstarts/Get_Started_Creating_a_Windows_Phone_Application

[7] Publishing your app in the Marketplace. MSDN [online]. © 2012 [cit. 2012-05-06]. Dostupné z: http://create.msdn.com/en-US/education/quickstarts/marketplace

[8] *Qt* [online]. c2011 [cit. 2011-12-16]. Nokia developer. Dostupné z WWW: <http://www.developer.nokia.com/Develop/Qt/>.

[9] *Symbian C++* [online]. c2011 [cit. 2011-12-16]. Nokia developer. Dostupné z WWW: <http://www.developer.nokia.com/Develop/Featured_Technologies/Symbian_C++/>.

[10] *Featured Technologies* [online]. c2011 [cit. 2011-12-16]. Nokia developer. Dostupné z WWW: <http://www.developer.nokia.com/Develop/Featured_Technologies/>.

[11] *Symbian Signed* [online]. c2011 [cit. 2011-12-16]. Nokia developer. Dostupné z WWW: <https://www.symbiansigned.com/signedui/welcome>.

[12] *Java Application Development* [online]. c2011 [cit. 2011-12-16]. BlackBerry. Dostupné z WWW: <http://us.blackberry.com/developers/javaappdev/>.

[13] *Press Release* [online]. 18.10.2011 [cit. 2011-12-16]. Research In Motion. Dostupné z WWW: <http://press.rim.com/release.jsp?id=5230>.

[14] *IOS* [online]. c2011 [cit. 2011-12-17]. Apple. Dostupné z WWW: <http://www.apple.com/cz/ios/>.

[15] *IOS developer library* [online]. c2011 [cit. 2011-12-17]. Apple developer. Dostupné z WWW: <http://developer.apple.com/library/ios/#documentation/Xcode/Conceptual/ios_development_workflow/000-Introduction/introduction.html>.

[16] App Design Basics. Apple iOS [online]. © 2012 [cit. 2012-05-06]. Dostupné z: https://developer.apple.com/library/ios/#documentation/iPhone/Conceptual/iPhoneOSProgrammingGuide/AppDesignBasics/AppDesignBasics.html#//apple_ref/doc/uid/TP40007072-CH2-SW1

[17] *Co je Jailbreak* [online]. c2011 [cit. 2011-12-17]. Apple poradna. Dostupné z WWW: <http://www.appleporadna.cz/iphone/co-je-to-jailbreak-u-iphone/>.

[18] *Bada developers home* [online]. c2011 [cit. 2011-12-17]. Bada developers. Dostupné z WWW: <http://developer.bada.com/apis/index.do>.

[19] *MeeGo developer* [online]. c2011 [cit. 2011-12-17]. MeeGo. Dostupné z WWW: <http://developer.meego.com/>.

[20] Android [online]. © 2012 [cit. 2012-05-06]. Dostupné z: www.android.com

[21] *Google Android historie* [online]. 2011 [cit. 2011-12-17]. Cnews. Dostupné z WWW: <http://www.cnews.cz/google-android-velky-vylet-do-historie>.

[22] What is Android. Android Architecture [online]. © 2012 [cit. 2012-05-06]. Dostupné z: http://developer.android.com/guide/basics/what-is-android.html

[23] Applications. Android Architecture [online]. © 2012 [cit. 2012-05-06]. Dostupné z: http://developer.android.com/guide/basics/what-is-android.html

[24] Application Framework. Android Architecture [online]. © 2012 [cit. 2012-05-06]. Dostupné z: http://developer.android.com/guide/basics/what-is-android.html

[25] Libraries. Android Architecture [online]. © 2012 [cit. 2012-05-06]. Dostupné z: http://developer.android.com/guide/basics/what-is-android.html

[26] Android Runtime. Android Architecture [online]. © 2012 [cit. 2012-05-06]. Dostupné z: http://developer.android.com/guide/basics/what-is-android.html

[27] Dalvik Technical Information. Android Source [online]. © 2012 [cit. 2012-05-06]. Dostupné z: http://source.android.com/tech/dalvik/index.html

[28] Linux Kernel. Android Architecture [online]. © 2012 [cit. 2012-05-06]. Dostupné z: http://developer.android.com/guide/basics/what-is-android.html

[29] Android 2: Průvodce programováním mobilních aplikací. Brno: Computer Press, 2011. ISBN 978-80-251-3194-7.

[30] Android 2: Průvodce programováním mobilních aplikací. Brno: Computer Press, 2011, s. 23. ISBN 978-80-251-3194-7.

[31] The AndroidManifest.xml File. Developer Android [online]. © 2012 [cit. 2012-05-06]. Dostupné z: http://developer.android.com/guide/topics/manifest/manifest-intro.html

[32] Structure of the Manifest File. Developer Android [online]. © 2012 [cit. 2012-05-06]. Dostupné z: http://developer.android.com/guide/topics/manifest/manifest-intro.html

[33] Android API levels [online]. 2011 [cit. 2011-12-17]. Android developers. Dostupné z WWW: <http://developer.android.com/guide/appendix/api-levels.html>.

[34] Activities. Developer Android [online]. © 2012 [cit. 2012-05-06]. Dostupné z: http://developer.android.com/guide/topics/fundamentals/activities.html

[35] Intent and Intent Filters. Developer Android [online]. © 2012 [cit. 2012-05-06]. Dostupné z: http://developer.android.com/guide/topics/intents/intents-filters.html

[36] Starting Activities. Developer Android [online]. © 2012 [cit. 2012-05-06]. Dostupné z: http://developer.android.com/guide/topics/fundamentals/activities.html

[37] Intent Class Overview. Developer Android [online]. © 2012 [cit. 2012-05-06]. Dostupné z: http://developer.android.com/reference/android/content/Intent.html

[38] Android 2: Průvodce programováním mobilních aplikací. Brno: Computer Press, 2011, s. 174. ISBN 978-80-251-3194-7.

[39] Activities: Implementing the lifecycle callbacks. Developer Android [online]. © 2012 [cit. 2012-05-06]. Dostupné z: http://developer.android.com/guide/topics/fundamentals/activities.html

[40] Activities: Tasks and Back Stack. Developer Android [online]. © 2012 [cit. 2012-05-06]. Dostupné z: http://developer.android.com/guide/topics/fundamentals/tasks-and-back-stack.html

[41] Activities: Saving activity state. Developer Android [online]. © 2012 [cit. 2012-05-06]. Dostupné z: http://developer.android.com/guide/topics/fundamentals/tasks-and-back-stack.html

[42] Handling Runtime Changes. Developer Android [online]. © 2012 [cit. 2012-05-06]. Dostupné z: http://developer.android.com/guide/topics/resources/runtime-changes.html

[43] Handling Runtime Changes: Retaining an Object During a Configuration Change. Developer Android [online]. © 2012 [cit. 2012-05-06]. Dostupné z: http://developer.android.com/guide/topics/resources/runtime-changes.html

[44] Android Class Overview. Developer Android [online]. © 2012 [cit. 2012-05-06]. Dostupné z: http://developer.android.com/reference/classes.html

[45] Android Class Overview. Developer Android [online]. © 2012 [cit. 2012-05-06]. Dostupné z: http://developer.android.com/guide/topics/ui/index.html

[46] Android 2: Průvodce programováním mobilních aplikací. Brno: Computer Press, 2011, s. 58. ISBN 978-80-251-3194-7.

[47] Android 2: Průvodce programováním mobilních aplikací. Brno: Computer Press, 2011, s. 58-63. ISBN 978-80-251-3194-7.

[48] Android 2: Průvodce programováním mobilních aplikací. Brno: Computer Press, 2011, s. 64-67. ISBN 978-80-251-3194-7.

[49] Android 2: Průvodce programováním mobilních aplikací. Brno: Computer Press, 2011, s. 68-70. ISBN 978-80-251-3194-7.

[50] Android 2: Průvodce programováním mobilních aplikací. Brno: Computer Press, 2011, s. 70-72. ISBN 978-80-251-3194-7.

[51] User Interface: Widgets. Developer Android [online]. © 2012 [cit. 2012-05-07]. Dostupné z: http://developer.android.com/guide/topics/ui/index.html

[52] Resources: Providing Resources. Developer Android [online]. © 2012 [cit. 2012-05-07]. Dostupné z: http://developer.android.com/guide/topics/resources/providing-resources.html

[53] Resources: Accessing Resources. Developer Android [online]. © 2012 [cit. 2012-05-07]. Dostupné z: http://developer.android.com/guide/topics/resources/accessing-resources.html

[54] About SVG. W3C [online]. 2004 [cit. 2012-05-07]. Dostupné z: http://www.w3.org/Graphics/SVG/About.html

[55] Vektorový grafický formát SVG. Root [online]. 2007 [cit. 2012-05-07]. Dostupné z: http://www.root.cz/clanky/vektorovy-graficky-format-svg/

[56] Vector Graphic Support for Android. Google Codes [online]. © 2012 [cit. 2012-05-07]. Dostupné z: http://code.google.com/p/svg-android/

[57] Data Storage: Using Shared Preferences. Developer Android [online]. © 2012 [cit. 2012-05-07]. Dostupné z: http://developer.android.com/guide/topics/data/data-storage.html#pref

[58] Data Storage: Using Internal Storage. Developer Android [online]. © 2012 [cit. 2012-05-07]. Dostupné z: http://developer.android.com/guide/topics/data/data-storage.html#filesInternal

[59] Data Storage: Using External Storage. Developer Android [online]. © 2012 [cit. 2012-05-07]. Dostupné z: http://developer.android.com/guide/topics/data/data-storage.html#filesExternal

[60] Data Storage: Using databases. Developer Android [online]. © 2012 [cit. 2012-05-07]. Dostupné z: http://developer.android.com/guide/topics/data/data-storage.html#db

[61] Data Storage: Using Network Connection. Developer Android [online]. © 2012 [cit. 2012-05-07]. Dostupné z: http://developer.android.com/guide/topics/data/data-storage.html#netw

[62] Threads. Developer Android [online]. © 2012 [cit. 2012-05-07]. Dostupné z: http://developer.android.com/guide/topics/fundamentals/processes-and-threads.html#Threads

[63] Threads: Worker Threads. Developer Android [online]. © 2012 [cit. 2012-05-07]. Dostupné z: http://developer.android.com/guide/topics/fundamentals/processes-and-threads.html#WorkerThreads

[64] Publishing on Google Play. Developer Android [online]. © 2012 [cit. 2012-05-07]. Dostupné z: http://developer.android.com/guide/publishing/publishing.html

[65] Versioning your Applications. Developer Android [online]. © 2012 [cit. 2012-05-07]. Dostupné z: http://developer.android.com/guide/publishing/versioning.html

[66] Preparing for Release. Developer Android [online]. © 2012 [cit. 2012-05-07]. Dostupné z: http://developer.android.com/guide/publishing/preparing.html

[67] Using your own SQLite database in Android applications. Reign Design [online]. 2009 [cit. 2012-05-08]. Dostupné z: http://www.reigndesign.com/blog/using-your-own-sqlite-database-in-android-applications/

[68] Handling Runtime Changes. Developer Android [online]. © 2012 [cit. 2012-05-08]. Dostupné z: http://developer.android.com/guide/topics/resources/runtime-changes.html

[69] Handling progress dialogs and orientation changes. Do it yourself android [online]. 2010 [cit. 2012-05-08]. Dostupné z: http://blog.doityourselfandroid.com/2010/11/14/handling-progress-dialogs-and-screen-orientation-changes/

[70] VGA. TechTerms [online]. © 2012 [cit. 2012-05-08]. Dostupné z: http://www.techterms.com/definition/vga

[71] SQLite Database Browser. SQLite Database Browser [online]. © 2012 [cit. 2012-05-09]. Dostupné z: http://sqlitebrowser.sourceforge.net/

[72] ŠAFAŘÍK, Jiří, Prof., Ing, CSc. Počítače a programování 2: Grafy. Západočeská Univerzita v Plzni, 2005.

Přílohy

Vzhledem k rozsáhlosti zdrojových kódů aplikací, jsou veškeré zdrojové kódy spolu s ostatními daty aplikací uloženy na CD, přiloženém k této práci.

Zdrojové kódy aplikace jsou také k dispozici na vývojářském webu Google Code: http://code.google.com/p/zcu-map-system-for-android/

Obsah CD

- Kompletní projekt mobilní aplikace
- Zkompilovaný projekt mobilní aplikace do souboru .apk připravený pro instalaci do mobilního telefonu
- Dokumentace k mobilní aplikaci (javadoc)
- Kompletní projekt aplikace pro údržbu mobilní aplikace
- Spustitelný JAR soubor aplikace pro údržbu
- Dokumentace k aplikaci pro údržbu mobilní aplikace (javadoc)

Evidenční list

Souhlasím s tím, aby moje bakalářská práce byla půjčována k prezenčnímu studiu v Univerzitní knihovně ZČU v Plzni.

Datum: 31.5.2010 Podpis:

Uživatel stvrzuje svým čitelným podpisem, že tuto bakalářskou práci použil ke studijním účelům a prohlašuje, že ji uvede mezi použitými prameny.

Jméno	**Fakulta/katedra**	**Datum**	**Podpis**